SYSTEM ARCHITECTURE AND HARDWARE IMPLEMENTATIONS FOR A RECONFIGURABLE MPLS ROUTER

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Master of Science in the

Department of Electrical Engineering

University of Saskatchewan

By

Li Sha

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Dept of Electrical Engineering
57 Campus Road
Saskatoon, Sask. S7N 5A9 Canada
Phone: (306) 966-5380
Fax: (306) 966-5407

ABSTRACT

With extremely wide bandwidth and good channel properties, optical fibers have brought fast and reliable data transmission to today's data communications. However, to handle heavy traffic flowing through optical physical links, much faster processing speed is required or else congestion can take place at network nodes. Also, to provide people with voice, data and all categories of multimedia services, distinguishing between different data flows is a requirement. To address these router performance, Quality of Service /Class of Service and traffic engineering issues, Multi-Protocol Label Switching (MPLS) was proposed for IP-based Internetworks. In addition, routers flexible in hardware architecture in order to support ever-evolving protocols and services without causing big infrastructure modification or replacement are also desirable. Therefore, reconfigurable hardware implementation of MPLS was proposed in this project to obtain the overall fast processing speed at network nodes.

The long-term goal of this project is to develop a reconfigurable MPLS router, which uniquely integrates the best features of operations being conducted in software and in run-time-reconfigurable hardware. The scope of this thesis includes system architecture and service algorithm considerations, Verilog coding and testing for an actual device. The hardware and software co-design technique was used to partition and schedule the protocol code for execution on both a general-purpose processor and stream-based hardware. A novel RPS scheme that is practically easy to build and can realize pipelined packet-by-packet data transfer at each output was proposed to take the place of the traditional crossbar switching. In RPS, packets with variable lengths can be switched intelligently without performing packet segmentation and reassembly. Primary theoretical analysis of queuing issues was discussed and an improved multiple queue service scheduling policy UD-WRR was proposed, which can reduce packet-waiting time without sacrificing the performance. In order to have the tests carried out appropriately, dedicated circuitry for the MPLS functional block to interface a specific

MAC chip was implemented as well. The hardware designs for all functions were realized with a single Field Programmable Gate Array (FPGA) device in this project.

The main result presented in this thesis was the MPLS function implementation realizing a major part of layer three routing at the reconfigurable hardware level, which advanced a great step towards the goal of building a router that is both fast and flexible.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. Eric Norum and Dr. J. Eric Salt for their guidance and financial support throughout the course of this project. Especially, she would like to show her great appreciation for Dr. Norum's timely thesis reviewing after he left University of Saskatchewan. A wonderful work environment was setup at Trlabs. Jack Hanson, Garth Wells, and the rest of the staff at Trlabs, Saskatoon, are all thanked for their time and help. Also, the author would like to say thank you to Dr. Ronald J. Bolton and her first supervisor at University of Saskatchewan, Dr. Gul Khan.

The Department of Electrical Engineering, University of Saskatchewan and Dr. J. Eric Salt provided financial assistance. Their support is gratefully acknowledged.

Finally, the author would like to thank her parents, Li Xianlin and Zhang Xiurong for their much needed support and her husband, Hu Song for providing a great deal of patience and support during some difficult times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ATM	Asynchronous Transfer Mode
BGP	Boarder Gateway Protocol
CoS	Class of Service
CRC	Cyclic Redundancy Check
CR-LDP	Constraint-based Routing LDP
DA	Destination Address
DiffServ	Differentiated Services
DLCI	Data Link Connection Identifier
DMA	Direct Memory Access
ESB	Embedded System Block
FEC	Forward Equivalent Class
FPGA	Field Programmable Gate Array
GPS	General Processor Sharing
HDL	Hardware Description Language
HDLC	High Level Data Link Control
HOLB	Head of Line Block
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IP	Internet Protocol
ISA	Industry Standard Architecture
ISP	Internet Service Provider
LDP	Label Distribution Protocol
LER	Label Edge Router
LFIB	Label Forwarding Information Base
LIB	Label Information Base
LSB	Left Significant Bit
LSP	Label Switched Path

MPLS	Multi-Protocol Label Switching
OSI	Open System Interconnection
OSPF	Open Shortest Path First
PCI	Peripheral Component Interconnection
PIM	Protocol-Independent Multicast
PING	A utility to determine whether a specific IP address is accessible. Often believed to be short for Packet Internet Groper
PNNI	Private Network-to-Network Interface
POS	Packet Over SONET/SDH
PPP	Point to Point Protocol
QoS	Quality of Service
RHFE	Reconfigurable Hardware Functional Element
RISC	Reduced Instruction Set Computer
RPS	Real Packet Switching
RSVP	Resource Reservation Setup Protocol
SA	Source Address
SDH	Synchronized Digital Hierarchy
SONET	Synchronous Optical Network
SOPC	System-On-a-Programmable-Chip
TCP	Transmission Control Protocol
ToS	Type of Service
UDP	User Datagram Protocol
UD-WRR	Unit Data Weighted Round Robin
VCI	Virtual Circuit Identifier
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VPI	Virtual Path Identifier
VPN	Virtual Private Network
WFQ	Weighted Fair Queue

Chapter 1 Introduction

1.1 Motivation

Over the last ten years, the Internet has evolved into a ubiquitous network. New extranet services, network-enabled intranet applications, and much more powerful PCs are turning the Internet rapidly into an electronic agent for information retrieval, commerce, entertainment, and communication. As well, there is exponential growth in the number of users who have diverse demands for more reliable and differentiated services. Therefore, Class of Service (CoS) and Quality of Service (QoS) issues must be addressed in order to support the diverse requirements of the wide range of new applications and network users. Both large and small Internet Service Providers (ISPs) constantly face the challenges of adapting their networks to accommodate new services and meeting more diverse customer requirements. In many situations, software updates are not enough to achieve this goal. Meanwhile, due to extremely high costs, physically replacing or upgrading network infrastructure constantly is not feasible, either. As a result, Multi-Protocol Label Switching (MPLS) that can address all these issues was proposed.

MPLS [2, 3] is a direct and elegant industrial solution to improve the controllability, efficiency, and reliability of the current worldwide IP networks. It gives the network better extensibility and also provides more flexibility to routing services, which means that it allows the addition of new routing services without changing the original packet-forwarding mode. MPLS is not confined to any particular link layer technology; it can use any medium to transmit packets between any two entities of the network layer. However, though MPLS is now taken as a crucial standard technology that offers new capabilities for large-scale IP networks, the concept of label switching was originally proposed as a way of improving the forwarding speed of routers only.

Routers can easily become places where network bottlenecks are formed, and fast node processing speed is extremely important to avoid these bottlenecks to achieve

good network performance. Meanwhile, from the industrial point of view, the flexibility that can reduce the cost when new services need to be added to the router later is of the same importance [7, 9]. Naturally, routers that are both fast and flexible are desired, but these two characteristics are generally considered a contradiction in terms. The reason is that maintaining high throughput requires fast but fixed-configuration application specific integrated circuits (ASICs) while flexibility requires slower though flexible configuration of general-purpose processors. Is there an ideal compromise? The answer is positive. The solution is a reconfigurable router that is fast and flexible at the same time, by integrating the best features of both hardware and software processing through the efficient use of Field-Programmable Gate Array (FPGA) technology, hardware description language, and hardware/software partitioning and scheduling technique. Detailed reasons for why FPGA instead of ASIC or software technologies are chosen in this project is fully described in section 1.3.2

In the project described by this thesis, a partial fulfillment FPGA for the next generation fully reconfigurable IP routers adopting MPLS is implemented. The thesis is organized as follows. Chapter 1 presents a general introduction to MPLS and typical router architecture, then clearly explains why a reconfigurable router is desirable and realizable, and ends with a brief overview of some current commercial router products. Chapter 2 gives a more detailed literature review on MPLS standards. Chapter 3 begins with a brief description of the switch/router evolution over the years, then talks about the software/hardware partitioning for MPLS implementation, finally presents the architecture design proposal for a fundamental reconfigurable MPLS router and an improved multiple queue scheduling policy, Unit Data Weighted Round Robin. Chapter 4 depicts the full details of the MPLS logic circuit design completed within a FPGA device in this project. Chapter 5 introduces the test equipment first, then presents the test methodology development and the interface design, finally presents the test parameter selection. Chapter 6 deals with practical test procedures and result demonstration and analysis. At the end of the thesis, conclusions for work having been done and suggestions for future work are given.

1.2 What is MPLS?

MPLS [4, 5] provides a new technical foundation for today's multi-user, multi-service IP-based networks and can effectively address the bandwidth and quality of service requirements. According to the TCP/IP model, there are 4 layers: the transportation layer (layer 4), the network layer (layer 3), the logic link layer (layer 2) and the physical layer (layer 1). MPLS can be deployed directly over current ATM-based wide area networks without any hardware modification on ATM switches. Meanwhile, by inserting an MPLS shim layer between layer 2 and layer 3, MPLS can be used over different layer 2 protocols other than ATM to transport different Layer 3 protocols such as IPv6, IPX, or AppleTalk in addition to Ipv4 traffic.

With software or hardware implementation, MPLS supports service differentiation by using traffic-engineered path setup, helps achieve fine-grained service-level guarantees and incorporates provisions for constraint-based and explicit path setup. MPLS can improve and simplify packet-forwarding performance by enabling routing in Layer 2 switching that operates at wire-line speeds with hardware implementation. MPLS also helps in building interoperable networks due to its layer 2 independency and in building scalable Virtual Private Networks (VPNs) due to its traffic-engineering capability.

MPLS is significantly different from the hop-by-hop processing methods used by traditional networks. The essence of MPLS is the generation of a 'label' that acts as a shorthand representation of an IP packet's header. The MPLS ingress edge router selects the appropriate label that is to be inserted between layer 2 and layer 3 headers after it analyzes the contents of the packet's IP header. Part of the great power of MPLS comes from the fact that, compared to conventional IP routing, this analysis can be based on more than just the destination address carried in the IP header. The label is a short, fixed length, locally significant identifier, which distinguishes the route the packet should take to reach the required egress node of the MPLS-enabled network. Each label corresponds to a Forward Equivalence Class (FEC), which is a group of packets that are forwarded in the same manner (i.e., over the same path, with the same forwarding treatment). FECs can be defined at different levels of granularity. Each

Label Switching Router (LSR) must keep track of how packets should be forwarded by containing this FEC information in a Label Information Base (LIB) that includes FEC-to-label bindings. Conventional routing protocols, such as OSPF, BGP and PIM, provide the LSRs with the mapping between the FEC and the next hop addresses.

The basic operation of an MPLS network involves switching that is based on these labels, instead of the IP headers. Full IP header analysis occurs at every node in conventional IP routing, while in an MPLS cloud this analysis occurs only once at the network edge when the label is assigned. When a labeled packet is received at an LSR, the input port and label information are read and the output port is determined. Then an outgoing label in context for the next hop's label switching operation replaces the incoming label.

The MPLS standard allows for MPLS-enabled networks to be nested within each other. To accommodate this nesting, packets may have multiple labels, which form a label stack. The number of labels that need to be stored in a LSR depends on the type of label mapping policy that is used in the MPLS network.

A standard label distribution method is required when a LSR assigns a label to a particular FEC and conveys this information to its peers in the MPLS network. The MPLS standard does not dictate which signaling protocol should be used for such label distribution. The most popular protocol is called Label Distribution Protocol (LDP), which uses TCP and UDP over layer 4 to send messages; however, other signaling protocols do exist, such as the Resource Reservation Protocol (RSVP). In addition, extensions to LDP and RSVP have been created and are currently being considered to support traffic engineering. They are Constraint-based Routing LDP (CR-LDP) and RSVP Traffic Engineering (RSVP-TE) respectively.

In MPLS, a Label Switched Path (LSP) can be created by using different signaling protocols mentioned above, conforming to explicit network administrator's requirements. A LSP is functionally equivalent to a virtual circuit and is defined by a set of labels that are used from the ingress of the MPLS domain to the egress.

One of the most important advantages of MPLS is to allow traffic flows to be moved away from the shortest path calculated by say, the Interior Gateway Protocol (IGP), and onto potentially less congested physical paths across the network when necessary, which results in better utilization of the network.

Another advantage is that, MPLS is beneficial when realizing differentiated services (DiffServ). Users are motivated to use the Internet as a public transport for a number of different applications ranging from traditional file transfer to delay-sensitive services such as real time voice and video. To meet such diverse requirements, not only traffic engineering techniques but also traffic classification technologies have to be adopted. There are two approaches to support MPLS-based class of service forwarding. The first approach is that traffic flowing through a particular LSP can be queued for transmission on each LSR's outbound interface on the setting of the precedence bits carried in the MPLS header. The second approach is that an Internet Service Provider can provide multiple LSPs between each pair of edge LSRs. Each LSP can be traffic engineered to provide different performance and bandwidth guarantees. The head end LSR could place high-priority traffic in one LSP, medium-priority traffic in another LSP, best-effort traffic in a third LSP, and less-than-best-effort traffic in a fourth LSP. MPLS offers tremendous flexibility in the different types of services. The precedence bits are used to classify packets into one of several classes of service.

MPLS is also valuable in providing a more complete separation between inter- and intra-domain routing. This improves the scalability of routing processes and, in fact, reduces the route knowledge required within a domain because on some networks there may be a large amount of transit traffic. Meanwhile, with a clean separation between its control and forwarding functions, MPLS can evolve each part without impacting another part, which in turn enables the network evolution easier, less costly, and less prone to errors.

The last but not the least advantage of MPLS to mention here is providing a simple solution to VPN-related issues. VPN allows the public Internet to be used as a method for connecting various networks to form a private WAN. The VPN service provider must provide data privacy and support private IP addressing use where the IP address

space overlaps other network domains. Since forwarding decisions are based on MPLS labels and not destination IP addresses, traffic between (and even within) VPNs can be easily isolated.

On the whole, MPLS provides significant improvements in the packet forwarding process by simplifying the processing, avoiding the need to duplicate header processing at every step in the path, and creating an environment that can support controlled QoS and traffic engineering.

1.3 Introduction to Reconfigurable Routers

In this section an introduction to the typical router architecture and an overview of today's router products are given.

1.3.1 Typical Router Architecture

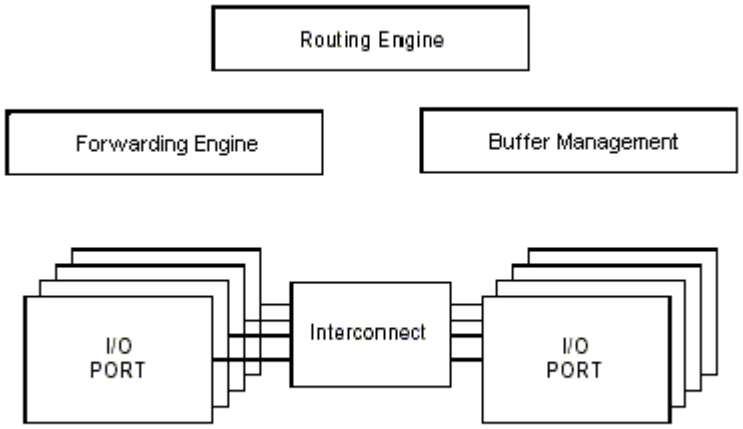


Figure 1-1 Architecture of a Typical Router

A typical router does three fundamental jobs [11, 14]. The first is to compute the best path that a packet should take through the network to its destination. This computation accounts for various policies and network constraints. The second job is to actually forward packets received on an input interface to the appropriate output interface for transmission across the network. Forwarding relies on the best-path information pre-computed in the route processor. The third job is to temporarily store

packets in large buffer memories to absorb the bursts and temporary congestion that frequently occur and to queue the packets using a priority-weighted scheme for transmission. Figure 1.1 shows the basic logical architecture of a router. The basic functional components carrying out these three jobs are named *the Routing Engine*, *the Forwarding Engine*, and *the Buffer Management system* respectively. A set of input and output ports is interconnected via some interconnection architecture.

The Routing Engine is dedicated to communicating with adjacent routers in order to build a comprehensive route database for the forwarding engine to send packets across optimum paths through the network. The routing engine runs software algorithms executing routing protocols, which enable the sharing of network status information among routers.

The Forwarding Engine examines the content of the packet's header, then searches for corresponding route information provided by the routing engine to find a match and finally direct the packet from the input port to the output port across the system's switching fabric.

If multiple packets arriving at different input ports simultaneously need to be forwarded to the same output port, a buffer must be available as a temporary waiting area in which packets queue up for transmission. The order in which they are transmitted is determined by the queuing scheduling policy pre-selected.

1.3.2 Why Reconfigurability?

As a commercial infrastructure providing differentiated services, the Internet has to be constructed with routers that can meet the massive demand increases in both bandwidth and processing speed. It is very important that these core elements of any networks be extensible and reconfigurable to support the ever new, ever evolving protocols and be able to provide third-party software vendors or value-added service providers with opportunities to develop applications. While many powerful routers with high processing speeds and throughputs have been manufactured already, they are not flexible and thus make it impossible for potential new protocols and services to be added without incurring large costs.

It has been thought that the time spent to process a packet on an IP switch should not exceed 0.27 ms [39], which clearly shows that maintaining high-throughput is a problem. Some researchers (Keshav and Sharma in [8]) also note that the reduction of port cost is currently a tradeoff between application specific integrated circuits (ASICs) and general-purpose processors. In this dichotomy, the only solution that provides flexibility is the use of a general-purpose processor. However, new software technologies deployed within the router operating system with the potential of offering increased flexibility in the router may not increase its performance. Software instructions themselves ask for processing time. Thus it is not clear if these software technologies will be practical, especially considering the current problems with maintaining high port performance. As line speeds continue to rise and the upper bound on processing time continues to fall, as on-demand scheduling of hardware resources is required to assist in the development of flexible network, some solution to provide reconfigurability at the hardware layer has to be found.

One way of providing this low-level reconfigurability is through configurable computing technology. With suitable hardware-level configurable computational units, stream processing has the potential to allow packets to be processed at line speeds. It becomes more practical since reconfigurable hardware technology has made several compelling performance advances recently, identifying it as a possible solution to the reconfigurable network node problem. New reconfigurable hardware devices contain approximately 110K logic elements (millions of application logic gates), an internal clock about 420 MHz, and over 10MB of on-chip RAM.

Contemporary Field Programmable Gate Arrays, which serve as the flexible fabric in configurable computing platforms, are already being used to provide field-upgrades of firmware in some industrial and research switches ([35], [38]). FPGAs provide an intermediate operating point between the relative slowness, flexible configuration, and low cost of a general-purpose processor and the high-performance, fixed configuration, and high cost of an ASIC. A modular and configurable set of functional units can be strung together and implemented within FPGA devices quite easily. Also, it is relatively easy to add, remove, modify and interconnect modules since they can be developed

independently, which greatly simplifies the implementation of a design. It is true that the cost of modularity brings an increase in the number of gates required to implement a particular function while some computational resources available in the module may not be used at all. However, because of the significant increase in FPGA resources, this is not expected to be a problem. Before long, one could expect a reconfigurable router composed of FPGAs, custom ASICs, and custom general-purpose processors to obtain the optimal combination of performance, flexibility, and cost.

1.4 Industrial MPLS Router Products Overview

The routers that power the Internet are evolving architecturally to keep pace with the escalating use of the Web and the requirement for a whole new generation of innovative, revenue-generating application services. Certain high-end router architectures that support ultra fast fiber-optic interfaces of up to 10 Gbps speeds and achieve system throughput in excess of 350 million packets per second (pps) already exist. Also, large router manufacturers claimed that they had implemented routers supporting MPLS, such as Alcatel IND, Cisco Systems, Juniper Networks, Marconi FORE Systems, etc. All these manufacturers stated that they had implemented or planned to implement both CR-LDP and RSPV-TE signaling protocols. In the following sections, these MPLS router products will be investigated according to product data sheets provided in [30] – [34].

1.4.1 Cisco 12000

The major components of the 12000 Gigabit Switch Router (GSR) are the switch fabric, the gigabit route processor (GRP), and the line cards (LCs). The packet-forwarding functions are performed by each of the LCs. Each LC performs an independent lookup of a destination address for each datagram received on a local copy of the forwarding table, and the datagram is switched across a crossbar switch fabric to the destination LC.

At the heart of the Cisco 12000 GSR is a multi-gigabit crossbar switch fabric. The switch fabric includes two card types: switch-fabric cards (SFCs) and clock and scheduler cards (CSC). The CSC handles requests from LCs, issues grants to access the

fabric, and provides a reference clock to all the cards in the system to synchronize data transfer across the crossbar. The SFCs receive the scheduling information and clocking reference from the CSCs and perform the switching functions.

The GRP is dedicated to determining the network topology and calculating the best path across the network. It creates and maintains the routing table (up to one million route entries), also distributes and updates express forwarding (EF) tables on the LCs and maintains copies of the tables of each LC for card initialization.

Line cards connect the GSR to other devices via electrical or optical media. The LCs are designed for the transmission of IP packets over Dynamic Packet Transport (DPT), PPP, Frame Relay, Packet over Sonet/SDH (POS) or ATM interfaces. The features and functions of the LCs are interface-specific.

The system of this series delivers scalable traffic engineering features by adopting Multi-protocol Label Switching (MPLS). Meanwhile, the design of this series supports virtual output queues (VOQs) that eliminate head-of-line blocking (HOLB) and increase overall system efficiency. Micro programmable application-specific integrated circuits (ASICs)-based queuing provides line speed forwarding for unicast and multicast traffic that fills SONET/SDH transmission facility.

1.4.2 JUNOS M40

As shown in Figure 1-2, there are two key components of the M40 architecture: the packet forwarding engine (PFE) and the routing engine. The PFE is responsible for packet forwarding performance. It consists of the flexible PIC concentrators (FPCs), physical interface cards (PICs), system control board (SCB), and state-of-the-art ASICs. The routing engine maintains the routing tables and controls the routing protocols. It consists of an Intel-based PCI platform running JUNOS software.

The M40 ASICs deliver a comprehensive hardware-based system for packet processing, including route lookups, filtering, sampling, rate limiting, load balancing, buffer management, switching, encapsulation, and de-encapsulation functions. To

ensure a non-blocking forwarding path, all channels between the ASICs are oversized, dedicated paths.

The Internet Processor II ASIC delivers high-speed forwarding performance with advanced IP services, such as filtering and sampling, enabled. The distributed buffer managers ASICs allocate incoming data packets throughout shared memory on the FPCs.

Each FPC is equipped with an I/O Manager ASIC that supports packet parsing, packet prioritizing, and queuing. The media-specific ASICs perform physical layer functions, such as framing. Each PIC is equipped with an ASIC or FPGA that performs control functions tailored to the PIC's media type.

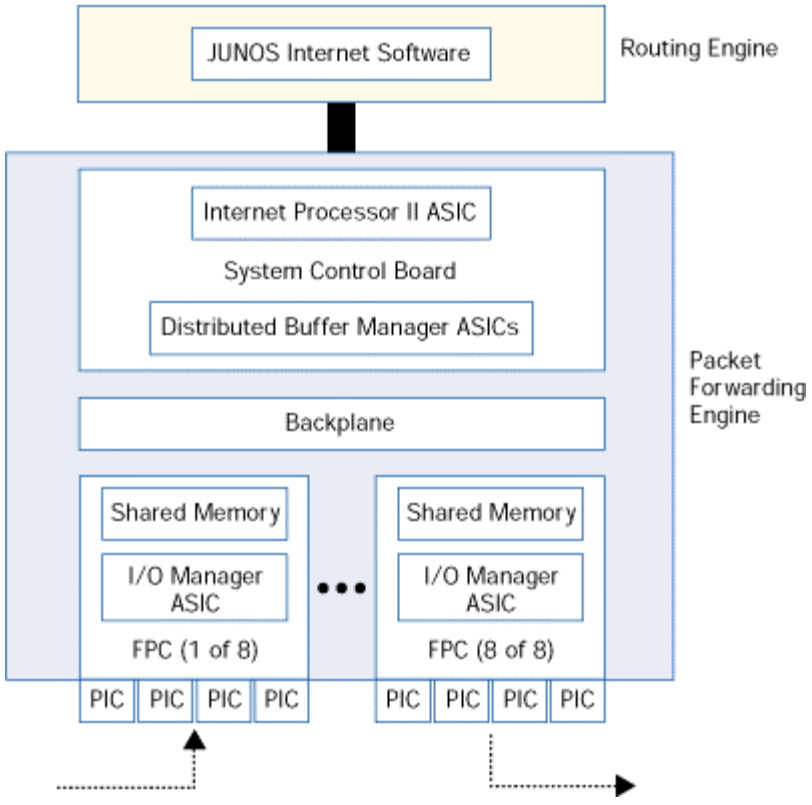


Figure 1-2 Logical View of M40 Architecture

The packet-forwarding engine (PFE) provides Layer 2 and Layer 3 packet switching, route lookups, and packet forwarding. The PFE supports ASIC-based features, for example, class-of-service features include rate limiting, classification and priority queuing, etc.

The enhanced flexible PIC concentrators (FPCs) house PICs and connect them to the rest of the PFE. Each FPC supports up to four PICs in any combination. Each FPC contains shared memory for storing data packets received. The physical interface cards (PICs) provide a complete range of fiber optic and electrical transmission interfaces to the network. The system control board (SCB) performs sampling, filtering, and packet forwarding decisions. It processes exception and control packets, monitors system components, and controls FPC resets.

The routing engine maintains the routing tables and controls the routing protocols, as well as the JUNOS software processes that control the router's interfaces, the chassis components, system management, and user access to the router. These routing and software processes run on top of a kernel that interacts with the PFE.

1.4.3 Alcatel 7670 Routing Switch Platform

The Alcatel 7670 Routing Switch Platform (RSP) is an MPLS-enabled ATM core switch designed for networks, integrating ATM multi-service capability and MPLS/IP switching into a unified scalable platform.

Per-VC queuing and shaping at ingress and egress, and buffer management with frame discard are adopted. All ATM service categories and most IP routing features are supported. To provide MPLS/IP, the switch platform can act as both edge LSR and core LSR and support Permanent LSP (P-LSP) and signaled LSP (S-LSP). The switch supports point-to-point and point-to-multipoint PVCs and SVCs, point-to-point SPVCs. Since Alcatel 7670 is an ATM switch, line cards are designed mainly for optical interfacing.

1.4.4 Marconi ASX4000

The ASX-4000 is a backbone switch that features the architecture to support low speed multi-service connections including ATM, Frame Relay DSL, Circuit Emulation and Ethernet. With the IP routing (IPR) module, the ASX-4000 can also operate as an MPLS gateway device.

1.4.5 Conclusion: New Products Desired

All the products mentioned above were designed to have redundancy in all key system components---processors, switch fabric, line cards, and power---to minimize network disruption in the event of a failure. This provides some kind of flexibility since components can be added or removed without service disruption.

All of these companies claimed that MPLS was supported. However, most of them stated this with only one or two sentences in their product data sheets, just as when they stated they could of course support software implemented BGP and OSPF, etc. No description of streamlined hardware dedicated for label switching was provided. This vagueness might due to the companies' confidential policy, but might also due to the more likely fact that most of them implemented MPLS in software. Only Marconi described very briefly that MPLS was supported by an IP routing module and this simple function description certainly led to the conclusion that it was implemented in software. With a powerful microprocessor, it may be realistic and meaningful to implement MPLS in software, providing both flexibility and better QoS guarantees. However, software implementation cannot take full advantages of what MPLS brings for layer 3 routing, which is critical to gain the overall faster processing speed at network nodes.

Some ATM switch products seemed to have fulfilled MPLS in hardware. But MPLS hardware implementation over ATM is quite straightforward and totally different from the implementation done over other layer 2 protocols. There is no need to do label binding or removing physically over an ATM based network when realizing MPLS, since the labels can reside in VPI and VCI fields that already exist in the ATM frame structure. However, a shim layer to hold MPLS labels is necessary if MPLS is to be deployed over PPP, Ethernet and Frame Relay networks in hardware, because their

frame structures contain no or not enough existing fields for MPLS labels to reside in correspondingly.

Finally, these companies all based their router design on powerful ASICs, which could not adopt new system parameters when needed, such as buffer space, routing table scale, etc; let alone the extensibility for potential protocols or other value added features that people want the routers to support in the future without any hardware modification or replacement. Generally speaking, routers existing in today's market are not reconfigurable at all.

Since almost all router manufacturers tend to stress the MPLS features of their products to make their routers look more competitive in the market, it can be inferred that MPLS routers are really the trend. By implementing the MPLS functions in hardware and making the router architecture reconfigurable, this project is meaningful.

Chapter 2 Multi-Protocol Switching

2.1 Main MPLS Components

An MPLS node can obtain all the information it needs to forward a packet as well as determine resource reservations needed by a class of traffic using a single memory access through its specially designed software and hardware components. In this section the main MPLS components are introduced.

- ***LSRs and LERs***

There are two categories of node equipments that participate in the MPLS working mechanisms. One is called the Label Switching Router (LSR), which is a high-speed MPLS-enabled router in the core of an MPLS domain; the other is the Label Edge Router (LER), which operates at the boundary between access networks and the MPLS domain. LERs can perform all the functions executed by LSRs besides handling issues of packets' entering and leaving the MPLS domain.

With the aid of an appropriate label signaling protocol, LSRs cooperate to establish Label Switched Paths (LSPs) and perform high-speed switching of the data traffic according to MPLS labels attached to packets. A fundamental step in label switching is that LSRs have to agree on the MPLS labels they use to forward traffic. They come to this common understanding by using the dedicated Label Distribution Protocol (LDP), Constraint Routing-Label Distribution Protocol (CR-LDP) or extensions to other protocols, such as PIM, BGP, RSVP. Since the current Internet consists of all kinds of networks which may not support MPLS traffic but only traditional IP traffic, to make the backbone MPLS router backwards compatible with other ordinary routers, LSRs are also able to forward native Layer 3 packets and routing packets without MPLS labels.

LERs support multiple ports connected to dissimilar networks (such as frame relay, ATM, and Ethernet). A LER can act as an ingress node or an egress node or both, for the MPLS domain. When acting as an ingress node, the LER forwards the traffic on to

the MPLS network after establishing LSPs using the label signaling protocol; when acting as an egress node, the LER distributes the traffic back to the access networks. The two very important MPLS functions, the label assignment and removal as traffic enters or exits an MPLS domain, take place at ingress LERs and egress LERs respectively. Like all LSRs, LERs can also perform a conventional IP forwarding function.

- **Forwarding Equivalent Class (FEC)**

A forwarding equivalent class is defined for a set of packets that receive the same treatment during transmission. In the context of MPLS, a packet is assigned to a FEC when it enters the MPLS network. The ingress router may use, in determining the FEC assignment, any information it has about the packet, even if that information cannot be gleaned from the network layer header, which is why labels that represent corresponding FECs contain considerably more information than just destination/source addresses for longest prefix match in IP routing. For example, a packet that enters the network at a particular router can be labeled differently than the packet from/to the same source/destination entering the network at a different router.

Insofar as the forwarding decision is concerned, different packets that get mapped into the same FEC are indistinguishable. All packets that belong to a particular FEC and travel from a particular node will follow the same path (or if certain kinds of multi-path routing are in use, they will all follow one of a set of paths associated with the FEC).

- **Labels and Label Bindings**

Since each FEC has associated labels according to some policy, once a packet is classified as a new or existing FEC, the associated fixed length labels are assigned to the packet. The events that result in such label assignments can be either *data-driven bindings* or *control-driven bindings*. The latter one is preferable because of its advanced scaling properties that can be used in MPLS.

Policies according to which label assignment decisions are made may be based on forwarding criteria such as destination unicast routing, traffic engineering, multicast, virtual private network (VPN), and QoS. Under some circumstances identifiers for underlying data link layers (such as frame relay or ATM) can be used directly as MPLS labels, such as Data Link Connection Identifiers (DLCIs) in the case of frame-relay networks or Virtual Path Identifiers (VPIs)/Virtual Channel Identifiers (VCIs) in case of ATM networks.

The generic label format is illustrated in Figure 2-1. Figures for different label formats are shown in next page. If layer 2 is ATM, the label is placed into the VPI/VCI field of the ATM cell header, as shown in Figure 2-2. Similarly, if layer 2 is frame relay, the label can be placed into the data link connection identifier (DLCI) field in the frame header, as shown in Figure 2-3. If Ethernet or point-to-point protocol (PPP) is running in layer 2, a shim header is inserted between the layer 3 header and the layer 2 header. The shim header contains the MPLS label, as shown in Figure 2-4. Support for the shim header requires that the sending router have a way to indicate to the receiving router that the frame contains a shim header. This is facilitated differently in various technologies.

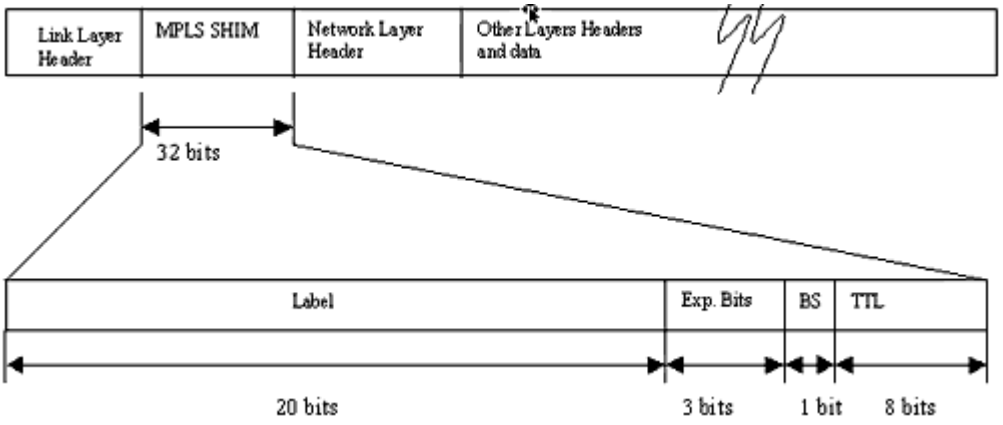


Figure 2-1 MPLS Generic Label Format [1]

A set of labels, in its simplest form, identifies the path a packet should traverse. Once a packet has been labeled, the rest of the journey of the packet through the backbone is based on label switching. At the edge router, the MPLS label will be

attached to the front of layer-3 header before the packet is transferred to Layer-2 for data link layer header encapsulation. Then each of the flowing receiving LSRs examines the packet for its label content to determine the next hop and then assigns a new label to replace the old one. The label values are of local significance only, which means they pertain only to hops between neighboring LSRs.

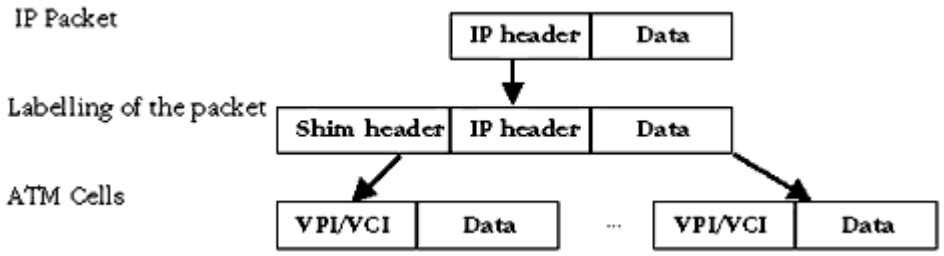


Figure 2-2 ATM as the Data Link Layer [1]

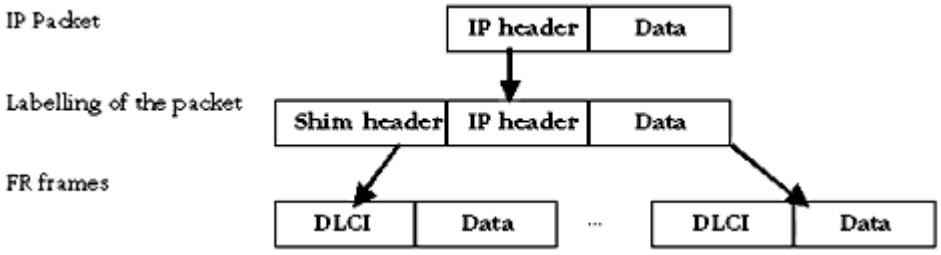


Figure 2-3 Frame Relay as the Data Link Layer [1]

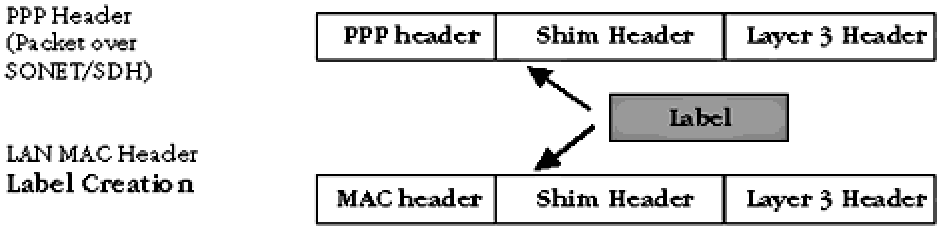


Figure 2-4 Point-to-Point (PPP)/Ethernet as the Data Link Layer [1]

MPLS defined two categorized label scopes for the uniqueness of different FEC–label bindings at each LSR. When a LSR can tell which peer-LSR adopts the particular

label value, it can use the “per-interface label space”, which indicates from the name “per-interface” that the label ranges are associated with interfaces. Multiple label pools are defined for interfaces, and the labels provided on those interfaces are allocated from the separate pools. The label values provided on different interfaces could be the same. Otherwise, the labels must be unique over the LSR that has assigned them, and the LSR is said to use a “per-platform label space”. The labels are allocated from a common pool and no two labels distributed on different interfaces have the same value.

- **Label Creation and Control**

MPLS defines several methods to create labels: *topology-based method* uses normal processing of routing protocols (such as OSPF and BGP); *request-based method* uses processing of request-based control traffic (such as RSVP); *traffic-based method* uses the reception of a packet to trigger the assignment and distribution of a label. The topology- and request-based methods are examples of control-driven label bindings, while the traffic-based method is an example of data-driven bindings.

Also, there are two ways to control the label creation. In the *independent* mode, an LSR recognizes a particular FEC and makes the decision to bind a label to the FEC independently to distribute the binding to its peers. The new FECs are recognized whenever new routes become visible to the router. In the *ordered* mode, an LSR binds a label to a particular FEC if and only if it is the egress router or it has received a label binding for the FEC from its next hop LSR. This mode is recommended for ATM–LSRs.

- **Label Stack**

More than one label header can be attached to a single packet and are managed by the label stack mechanism that allows for hierarchical operation in the MPLS domain. There is a stack bit in a standard MPLS label helping to implement label stacking. The label is indicated to be at the bottom of the stack if the stack bit contained within it is 1. All stack bits in other labels are set to 0. In packet-based MPLS, the top of the stack appears right after the link layer header, and the bottom of the label stack appears right before the network layer header. Packet forwarding is accomplished using the label

values of the label on the top of the stack. The stack bit becomes one when the corresponding label moves to the top of the stack.

Basically, tunneling operation can be facilitated by adopting the label stack mechanism, which allows MPLS to be used simultaneously for routing between individual routers both within an Internet service provider (ISP) and at a higher domain-by-domain level. Each level is indicated by a label in the stack that pertains to some hierarchical level.

- **Label Merging**

Resource usage can be increased if different traffic flows can be merged together and switched at a LSR when possible. This is known as stream merging or aggregation of flows. It can be done when the incoming streams of traffic are from different interfaces but toward the same final destination; or when traffic streams have to travel a same period of journey before they can reach their different final destinations separately. Label merging can be achieved by using a common outgoing label for several different incoming labels.

If the underlying transport network is an ATM network, LSRs could employ virtual path (VP) or virtual channel (VC) merging. In this scenario, cell-interleaving problems, which arise when multiple streams of traffic are merged in the ATM network, need to be avoided.

- **Label Retention**

There are two modes defined in MPLS for the treatment of label bindings received from LSRs that are not the next hop for a given FEC. They are *liberal* mode and *conservative* mode.

In the former mode, the bindings between a label and an FEC received from LSRs that are not the next hop for a given FEC are discarded. This mode requires an LSR to maintain fewer labels and thus is recommended by IETF.

In the latter mode, the bindings between a label and an FEC received from LSRs that are not the next hop for a given FEC are retained. This mode allows for quicker adaptation to topology changes and switching of traffic to other LSPs in case of such changes, but it requires larger memory at each MPLS node.

- **Label Forwarding Algorithm**

Label swapping is the base on which packet switching is performed in a MPLS domain. MPLS uses only a label swapping based forwarding algorithm to do packet switching for all traffic types such as unicast, multicast, and unicast packets with ToS bits set, which conventionally require multiple forwarding algorithms.

Each MPLS node maintains a Label Information Base (LIB). Most frequently used labels are formed into a smaller Label Forwarding Information Base (LFIB) for actual packet switching. Label values are extracted from the label field found in incoming packets and used as an index in the LFIB. After a match is found, the MPLS node replaces the label in the packet with the outgoing label from the subentry and sends the packet over the specified outgoing interface to the next hop specified by the subentry. If the subentry specifies an outgoing queue, the MPLS node places the packet in the specified queue. If the MPLS node maintains multiple LFIBs for each of its interfaces, it uses the physical interface on which the packet arrived to select a particular LFIB, and then performs label swapping according to this LFIB.

- **Label-Switched Paths (LSPs)**

Through an MPLS network, a traffic path along which packets belonging to a certain FEC travel is specifically defined over a set of LSRs prior to data transmission and is named the Label-Switched Path. MPLS allows a hierarchy of labels known as the label stack. It is therefore possible to have different LSPs at different levels of labels for a packet to reach its destination. The LSP setup for an FEC is unidirectional in nature, which means the return traffic must take another LSP. MPLS provides the following two options to set up an LSP

Hop-by-hop routing/Independent control--- This methodology is similar to that currently used in IP networks. Each LSR uses any available routing protocols, such as OSPF or ATM PNNI (Private Network-to-Network Interface), to independently select the next hop for a given FEC.

Explicit routing (ER)/Ordered control--- This methodology eases traffic engineering throughout the network, and differentiated services can be provided using flows based on specific service level policies or network management methods. The ingress LER specifies the list of nodes through which the ER–LSP traverses and then propagates such information to other nodes contained in the list. This kind of LSP could be non-optimal, say, not the shortest, because its primary goal is to ensure QoS to the data traffic through appropriate resources allocation and reservation along the path.

The hop-by-hop routing method provides faster convergence and establishment of LSPs due to the fact that label bindings can be established and advertised at any time by the LSR, while explicit routing method introduces the delay of waiting for messages to propagate in order across the network before the LSP can be established. However, the latter provides a better traffic engineering control and better loop prevention capabilities. And the good thing is, these two types of LSP establishments may coexist on the same network without any special considerations for architecture or interoperability issues.

- **Label Distribution Protocol**

For label distribution, MPLS architecture allows several signaling methods, which are either stemmed from existing routing protocols or newly proposed ones. For example, Border Gateway Protocol (BGP) has been enhanced to piggyback the label information within the contents of the protocol for external (like between VPNs) label exchange. Another currently used protocol RSVP has also been extended to support piggybacked exchange of labels and becomes RSVP-TE. Meanwhile, IETF has defined a new protocol known as the label distribution protocol (LDP) dedicated for MPLS label signaling and label space management. As well, extensions captured in the constraint-based routing LDP definition have also been defined to support explicit routing based on QoS and CoS requirements. Here the LDP is introduced briefly.

LDP has a set of signaling messages destined for the distribution of label binding information to LSRs in an MPLS network. LDP peers in the MPLS network, adjacent or not, establish LDP sessions between them and exchange certain LDP messages to map FECs to labels, which, in turn, create LSPs. There are basically 11 types of LDP messages, among which the most important ones are shown as below.

DISCOVERY--- used for finding LSRs and maintaining their existence.

ADJACENCY--- initialize, maintain, and shut down LDP sessions between LSRs.

LABEL ADVERTISEMENT---distribute label-binding, binding reverse and label release information by using *Label Mapping, Label Withdrawal and Label Release* messages respectively.

NOTIFICATION--- used for advisory and error signaling.

Due to the critical nature of the information being transferred, LDP runs on transmission control protocol (TCP) in order to ensure reliable data transport between LSRs, except for DISCOVERY messages that are run on UDP.

There are two types of label distribution strategies allowed in the MPLS architecture: *Downstream-on-Demand Mode* and *Unsolicited Downstream Mode*. The first mode allows an LSR to explicitly request a label binding for a particular FEC from its next hop. Label Request messages are used to request label mappings from downstream LSRs. Label Request Abort messages are used to abort the Label Request message during or prior to the completion of the request. The second mode allows an LSR to distribute bindings to LSRs that have not explicitly requested them.

2.2 MPLS Operation

When routing a packet, choosing the next hop can be thought of as a composition of two functions. The first function classifies the entire set of possible packets into a set of "Forwarding Equivalence Classes (FECs)". The second function maps each FEC to its corresponding next hop. In MPLS, the assignment of a particular packet to a particular FEC is done just once as the packet enters the network. The FEC to which

the packet is assigned is encoded as a short fixed length value known as a “label”. Each data packet is “labeled” before they are forwarded. At all subsequent hops, further analysis of the accompanied label instead of the network layer header, is used to decide the next hop until the packet reaches its destination. Indicated by a sequence of labels, LSPs are established either prior to data transmission (control-driven) or upon detection of a certain flow of data (data-driven). High-speed switching of data occurs on such LSPs is possible because the fixed-length labels are inserted at the very beginning of the packet or cell and can be used by hardware to switch packets quickly between links.

MPLS brings the advantage that, not all of the traffic between a certain pair of source and destination is necessarily transported through the same path within an MPLS domain. Depending on the network congestion status and specific traffic characteristics, different LSPs could be created for packets with the same source and destination addresses but with different QoS or CoS requirement.

Next, the step-by-step MPLS operations that occur on the data packets as the packet is transported across the MPLS domain to its destination are illustrated with reference to Figure 2-5. The LSP is set up between LER1 (the ingress LSR) and LER 4 (the egress LSR) through two inner nodes LSR1 and LSR3. The broken red lines indicate the actual data path followed by the packet.

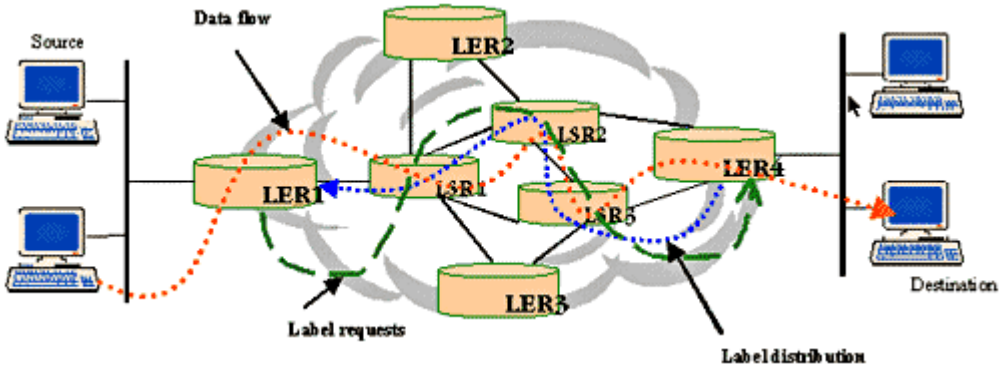


Figure 2-5 LSP Creation and Packet Forwarding through an MPLS Domain [3]

Step 1 Label creation and label distribution

The ingress router LER1 does not always have a label for a packet, as it may be the first occurrence of the FEC to which this packet belongs. Thus the ingress router requests labels for this FEC from its downstream peer to build a label information table. This has to be done before any traffic begins. In LDP, downstream routers initiate the distribution of labels and the label/FEC binding. In Figure 2-5, LSR1 is the next hop for LER1, thus LER1 initiates a label request toward LSR1. This request will propagate through the network as indicated by the broken green lines. The reliable and ordered transport protocol, TCP, should be used for the signaling protocol LDP. In addition, traffic-related characteristics and MPLS capabilities are negotiated and CR–LDP may be used in determining the actual path setup to ensure the QoS/CoS requirements are complied with.

Step 2 Table creation

Each intermediary router will then receive a label from its downstream router starting from LER2 and going upstream till LER1. On receipt of label bindings each LSR creates entries in the label information base (LIB) specifying all the mapping between a label and an FEC, that is, mappings between the input port and input label table to the output port and output label table. The entries are updated whenever renegotiation of the label bindings occurs. Another table named LFIB, which is a subset of the labels extracted from the LIB, will also be created for actual packet forwarding.

Step 3 Label switched path creation

As shown by the dashed blue lines in Figure 2-5, the LSPs are created using LDP or any other signaling protocol in the reverse direction to the creation of LIB entries. More detailed establishing procedure has been introduced in the first section.

Step 4 Label insertion/table-lookup

The ingress router LER1 inserts the label corresponding to a specific FEC to the packet and then forwards the packet to its next hop LSR. Subsequent LSRs use their LFIB tables to find the next hop for the packet. As shown in Figure 2-5, LSR2 and LSR3 examine the label in the received packet, replace it with the outgoing label and forward

it on. The label is removed once the packet reaches the egress LSR (LER4) because it is departing from the MPLS domain. Then the packet is supplied to the destination.

2.3 Tunneling in MPLS

By adopting the label stack to create tunnels through the intermediary routers that can span multiple segments, a great unique feature of MPLS used in provisioning MPLS–based VPNs can be achieved. The entire path of a packet can be controllable without explicitly specified intermediate routers.

Consider the scenario in Figure 2-6. BGP is used between all the LERs (LER1, LER2, LER3, and LER4), and a first level LSP, LSP1, is created between them. These LERs will use the LDP to receive and store labels from the egress LER (LER4 in this scenario) all the way back to the ingress LER (LER1).

For LER1 to send its data to LER2 (one segment of the LSP1), it must go through several LSRs, in this case there are three. Therefore, a separate second level LSP, LSP 2, is created between these two LERs, LER1 and LER2, that spans LSR1, LSR2, and LSR3. This, in effect, represents a tunnel between LER1 and LER2 in the view of level 1 LSP. The labels for this LSP2 are different from the labels that the LERs created for LSP1. The same holds true for the LSP1 segment between LER3 and LER4 as well. Thus a second level LSP, LSP 3, can be created for this segment. Note that in this scenario, LER2 and LER3 are communicating directly, which means there is no tunnel between LER2 and LSR3. In more complicated scenarios, there can be even more levels of LSP between the source and destination LERs.

When the packet is transported through more than one network segments, the concept of the label stack is the foundation on which tunneling is realized. Take the scenario in Figure 2-6 as the example. Since a packet must travel through LSP 1, which contains two tunnels, LSP 2 and LSP 3, it has to carry two complete labels at a time. The pair used for each segment is (1) pair for the first segment, labels for LSP 1 and LSP 2 and (2) pair for the second segment, labels for LSP 1 and LSP 3. When the packet exits the first network segment and is received by LER3, it will remove the label for LSP 2 and replace it with LSP 3 label, while swapping LSP 1 label within the packet

with the next hop label. LER4 will eventually remove both labels before sending the packet to the destination.

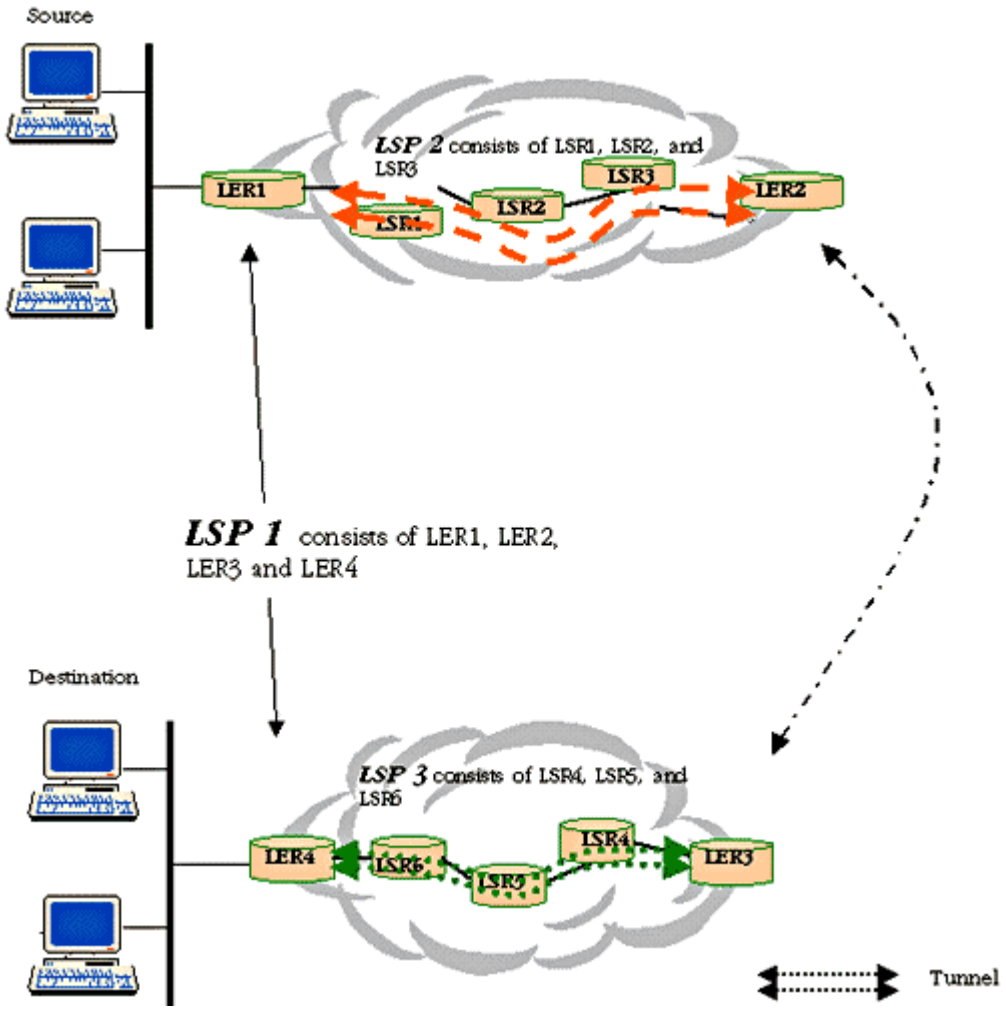


Figure 2-6 Tunneling in MPLS [3]

2.4 Traffic Engineering and QoS

In normal IP routing, the data path is calculated from some measurement of efficiency. The common metrics for IP routing and forwarding decisions, including next hop, hop count, and cost, are useful in predicting the "shortest path" through the network. However, those metrics cannot be assumed to be reliable at all times, or to be the best for a given flow that requires some fixed or guaranteed amount of bandwidth.

Traffic engineering is a process that enhances overall network utilization by attempting to create a uniform or differentiated distribution of traffic throughout the network. TE enables the network to quickly and automatically re-route traffic when failure or congestion conditions are detected by ensuring that all available network resources are optimally used. A network that maximizes its resources and capacity during normal operation is thus achieved through avoiding network hot spots and areas of hyper-aggregation, which means that traffic engineering does not necessarily select the shortest path between two devices. It is possible that packets may traverse completely different paths even though their originating node and the final destination node are the same. In this way, the less-exposed or less-used network segments can be used and differentiated services can be provided. Links between any two points in a network are relatively fixed and quantifiable, and the cost to increase that capacity, in many cases, is high, so effective traffic engineering and higher utilization of available links can provide both long- and short-term cost savings.

"Constraint-based" and "congestion-aware" routing are terms used to describe networks that are fully aware of their current utilization, existing capacity and provisioned services at all times. While traditional IP routing protocols, including OSPF, IS-IS and BGP, are not inherently congestion-aware, and have to be modified to enable such awareness, CR takes into account parameters, such as link characteristics (bandwidth, allocation multiplier, current bandwidth reservation, resource class, packet loss ratio, and link propagation delay, etc.), hop count, and QoS, etc. And the resulting data path can also ensure that none of the constraints that have been set are violated along the path. Once connections have been configured (either by dynamic signaling or by static provisioning), the Layer 2 and Layer 3 network becomes fully aware of the amount of bandwidth being consumed, as well as the parts of the network being used to route the connections. This information can then be propagated to the accompanying IP routing protocols that are exchanged by all IP routers, creating a truly congestion-aware view of the network and its current topology. Then, all future network requests can be directed to their destination by not only the "shortest path first" (as defined by OSPF), but by a path that will guarantee the bandwidth requirements of the IP application or service. This means when using CR, it is entirely possible that a longer (in terms of cost)

but less loaded path is selected. And there is another side effect that while CR increases network utilization, it adds more complexity to routing calculations, as the path selected must satisfy the QoS requirements of the LSP.

CR–LSPs set up with explicit hops or QoS requirements can be realized easily in MPLS architecture. Explicit hops dictate which path is to be taken. QoS requirements dictate which links and queuing or scheduling mechanisms are to be employed for the flow. A CR–LDP component to facilitate constraint-based routes has been defined by the IETF and its more detailed description has been introduced earlier in this chapter.

In MPLS, traffic engineering is inherently provided using explicitly routed paths. The LSPs are created independently, specifying different paths that are based on user-defined policies. However, this may require extensive operator intervention. RSVP and CR–LDP are two possible approaches to supply dynamic traffic engineering and QoS in MPLS.

RSVP-TE and CR-LDP are now two competing protocols used for MPLS that perform CR. RSVP is an existing protocol, standardized by the IETF, which has been extended to RSVP-TE. Similarly, CR-LDP is an extension of LDP, which has been designed for MPLS especially. There are advantages and disadvantages to both protocols. One side, CR-LDP sits on top of TCP to ensure reliability. For RSVP, refreshing that must occur in the steady state is required to ensure reliability while refreshing consumes bandwidth and processing resources. Also, TCP requires some handshaking before an LDP session can begin and results in a moderate amount of overhead while RSVP does not require connection establishment before label distribution occurs. Because of such advantages and disadvantages of RSVP-TE and CR-LDP, designers need to keep their systems flexible enough to accommodate future changes to the protocols.

2.5 Protocol Architecture

Figure 2-7 depicts the protocols that can be used for operations on a MPLS node. The LDP module utilizes transmission control protocol (TCP) for reliable transmission of control data from one LSR to another during a session. But the LDP uses the user

datagram protocol (UDP) during its discovery phase of operation. In this phase, the LSR tries to identify neighboring elements and also signals its own presence to the network. This is done through an exchange of hello packets.

There are two tables relevant to MPLS forwarding at an MPLS node: the LIB and the LFIB maintained by LDP. The LIB (not indicated in Figure 2-7) contains all the labels assigned by the local MPLS node and the mappings of these labels to labels received from its MPLS neighbors. The LFIB uses a subset of the labels contained in the LIB for actual packet forwarding.

The MPLS forwarding module matches a label to an outgoing port for a given packet. The IP Routing module performs the classic function that looks up the next hop by matching the longest address in its tables. The IP Routing module can run any popular industry protocol available depending on the operating environment, such as OSPF, BGP, or ATM's PNNI, etc. Though this IP routing function can be done at LERs only, any MPLS node should also take into account that ordinary unlabeled IP traffic may traverse over it. Aside from the process shown as green arrows that packets with MPLS labels go along, the more complex process for packets without MPLS labels should also be supported, which is indicated by pink arrows.

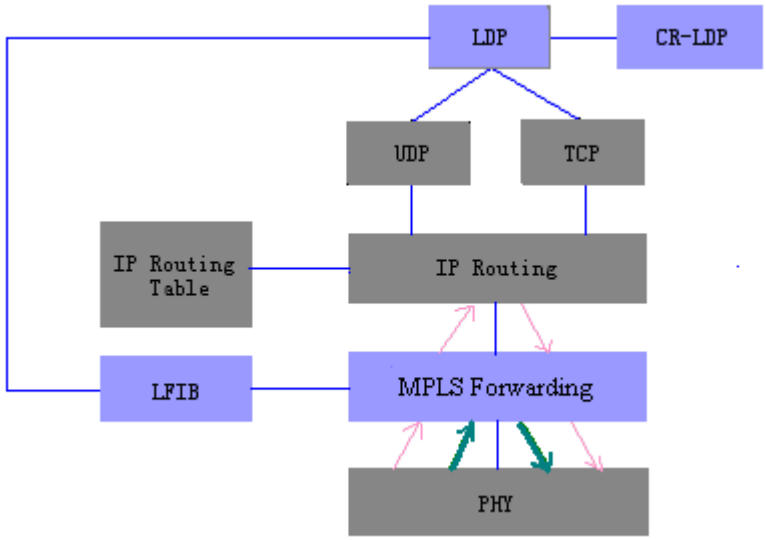


Figure 2-7 MPLS Protocol Stack

Chapter 3 Reconfigurable MPLS Router Design Issues

This chapter begins with a brief description of the switch/router evolution; then section 3.2 talks about the MPLS reconfigurable router design at the architecture level; section 3.3 introduces CAM technique used for lookup table implementation for an MPLS router; the last section proposes a modified multiple queue scheduling policy UD_WRR that is implemented in hardware for this reconfigurable MPLS router prototype.

3.1 Switch/Router evolution

Networking devices have been developed at a rapid pace for many years. According to the hardware used and the level of integration, the evolution of the switch/router can be roughly separated into different phases [28, 37]. In this section, a brief description of different generations of the switch/router is illustrated and the trend of the router design in the near future is introduced as the 4th generation.

3.1.1 The First Generation

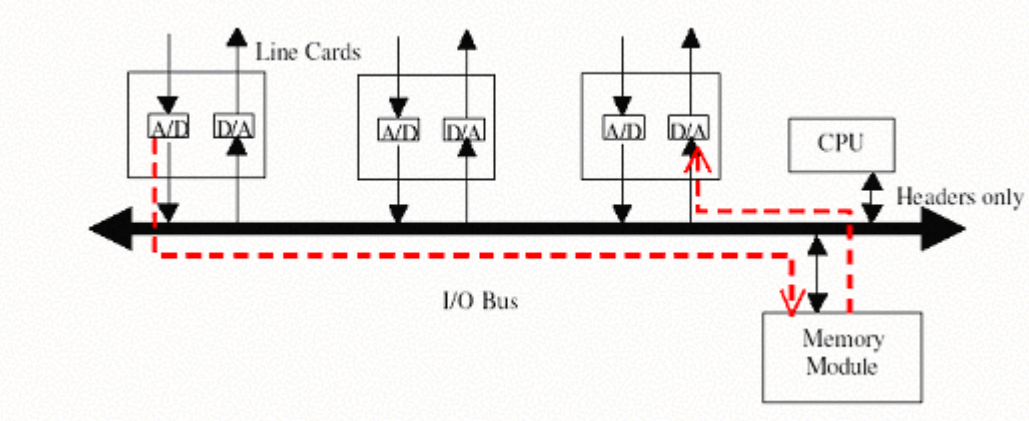


Figure 3-1 First Generation Switch/Routers [28]

The switches of the first generation included a CPU that hosted all the routing software, a main memory, and an optional DMA module. Figure 3-1 depicts the

architecture of first generation devices. CPU power, memory throughput and I/O bus bandwidth are three bottlenecks in this architecture.

3.1.2 The Second Generation

As seen in Figure 3-2, each line card shown contains a separate memory module and a small CPU. Input queuing or output queuing or both can be implemented. The sole purpose of the central CPU is to arbitrate the usage of the bus, the exchange of routing information between the local cards and the programming and maintenance of the whole system. Now the only bottleneck is the I/O bus bandwidth that fails to scale along with the number of high-speed line cards and the port count.

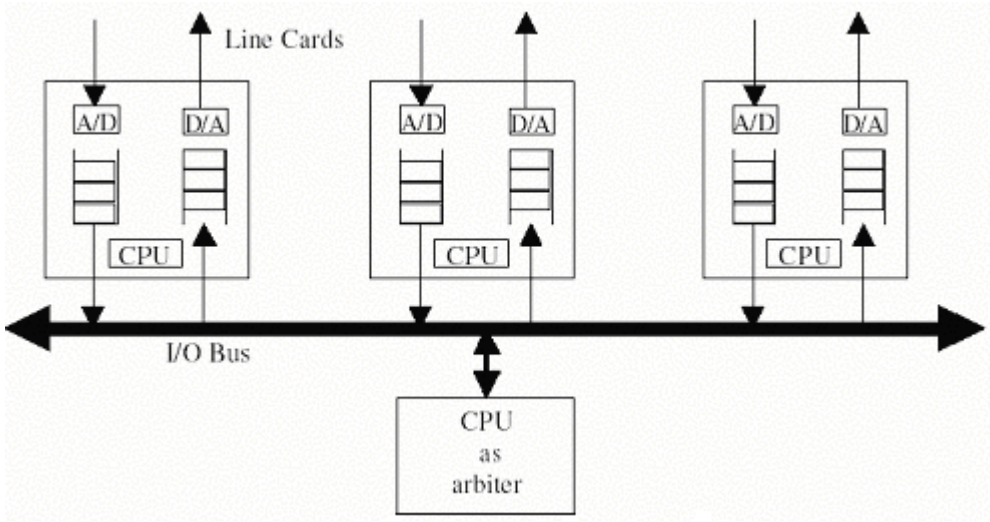


Figure 3-2 Second Generation Switch/Routers [28]

3.1.3 The Third Generation

This generation introduced switching fabrics to replace the I/O bus as the medium to relay packets between cards. Buffering and routing of data packets are performed inside the line cards while specialized hardware is provided to give the line cards access to the switching fabric. Switching fabrics can accept multiple simultaneous transfers of packets with a maximum of N transactions when N Line cards are connected to the fabric. Most of the current network devices employ ASIC large-scale integration to

implement SoC (System on Chip) architectures. Except for the analog components, all the line hardware (buffers, routing) for all ports is stored inside the same chip along with a crossbar (the most effective but less scalable switching fabric), a scheduling unit and a CPU. Existing chips can accommodate up to 32 input/output ports and are sufficient for a low-end switch/router. They can also be used as a building block of a much larger high-end switch/router. In the latter case, they are organized in switching fabric topologies such as Banyan, Benes, and Batcher-Banyan networks [36, chapter 8], as depicted in Figure 3-3.

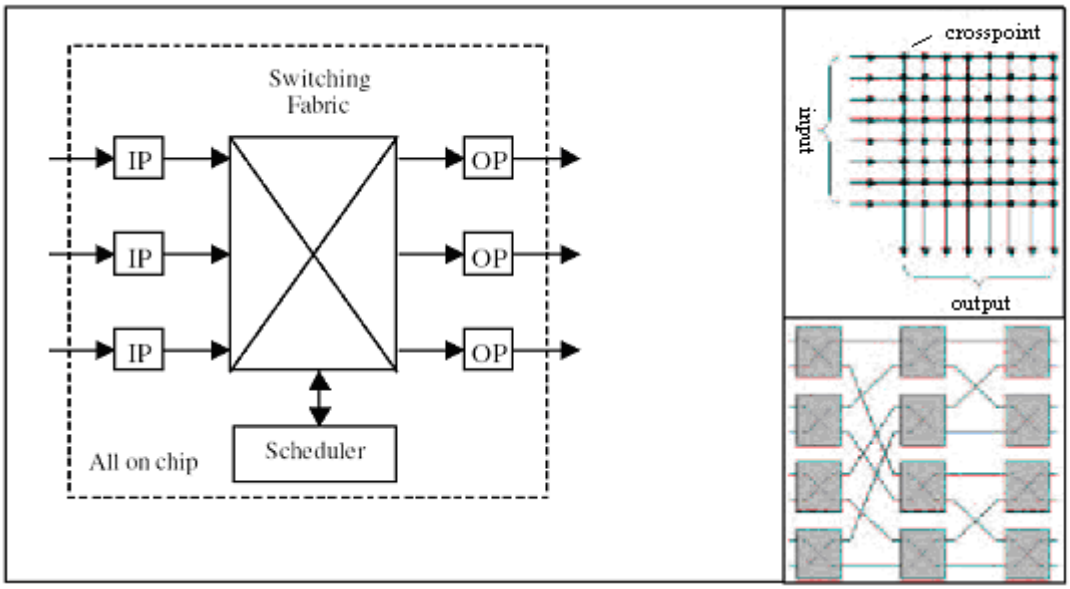


Figure 3-3 Left: Third Generation Switch/Router; Top-Right: A Crossbar; Bottom-Right: An 8x8 Banyan Fabric made of small 2x2 switch blocks. [28]

Switches/routers of the third generation that adopt cross bar to realize the point-to-point connection actually perform cell switching. For traditional single-stage, high-bandwidth packet switches, crossbar fabrics have been recognized as potentially providing the best architecture for a long time. In these switches, though from the angle of the layer 3, it is packets that enter and leave the switch, what the switch fabric core sees are cells. All kinds of data types are transported in optimally sized fixed-length fabric cells to address the QoS problem. And this implies a need for segmentation and reassembly that brings extra time delay and hardware source consumption.

3.1.4 The Fourth Generation To Be Developed

In [10,12,13], it was investigated that the diversity of networking applications and data flows calls for a new generation of switch/routers that have dynamically reprogrammable processing environment to cover the potential design space. Meanwhile, the development of flexible network software technologies also asks for some other solution than the third generation switch/router to assist in on-demand scheduling of hardware resources. While some applications performing limited processing at low data rates readily lend themselves to software implementation, a vast array of applications map well to hardware implementations due to their requirements for high data rates, parallel operations, and data regularities. Routers that are capable of aggregating forwarding rates of terabits per second and link speeds of 2.4 Gb/s and 10 Gb/s set the current standard for high-performance. To be considered commercially practical, programmable routers need to achieve comparable performance with scalable mechanism for data flow processing at router ports [6].

Traditionally used for low-volume prototyping and testing purposes, the reconfigurable hardware employed in FPGAs now provides a flexible hardware platform. Also, continuing advances in integrated circuit technology are making it possible to implement several complete subsystems on a single chip, which can result in scalable processing mechanisms at a reasonable per-port cost. The architectural optimizations and silicon fabrication improvements bring much impressive progress rate: usable logic gate count has increased by 10 times in two years; system clock frequency doubled in one year; I/O bandwidth quadrupled in two years; block and distributed on-chip memory capacity quadrupled in one year. Reconfigurable hardware devices are obviously positioning themselves as viable options for flexible, high-performance systems.

The third generation switches/routers adopting crossbars fall short in delivering higher levels of intelligence in the edge switching architecture to improve QOS and some new switching scheme is desired. On the whole, this generation is expected to have a scalable architecture capable of robust flow-specific processing at line speeds to

meet the demand of growing sophistication of networked applications and more complex network services without prohibitively high per-port costs. In the following sections, the concept of a MPLS reconfigurable router, a powerful candidate for the 4th generation realizes intelligent data transfer throughout the system architecture is expanded.

3.2 System Design Strategy

The final goal of this project is to develop a fundamental prototype of a fourth generation router through the adoption of MPLS standards, reconfigurable hardware, novel switching idea and improved multiple queue service scheduling.

To keep up with fast link speeds, most modern commercial high-performance backbone IP routers, such as what we have introduced in the first chapter, typically use ASICs on each port and have high-bandwidth access to the local table of routes. They are capable of forwarding standard datagrams (without special features like IP options) entirely in hardware. However, with more amount of processing spent on a single packet and since the processing is application-specific for a potentially significant variety of applications, it is impossible to implement all of them in ASICs. This means that both flexible protocols and hardware are needed and it is the very place that the concept of an MPLS reconfigurable router applies perfectly.

Generally speaking, MPLS nodes have two architectural planes: the routing plane and the forwarding plane. As describe in earlier chapters, in order to be backwards compatible, MPLS nodes can also perform ordinary Layer 3 IP routing for packets without MPLS labels. MPLS can take advantage of all the routing information obtained by protocols that run in software above layer 3 and then decide the optimal network path to maximize network efficiencies, deliver the fastest possible response times to users, minimize bandwidth usage costs, and meet some other criteria.

The first step to start the hardware design of an MPLS node prototype is to do software and hardware partitioning to decide which parts of the MPLS standards are possible and desirable to be implemented in hardware.

3.2.1 Protocol software and hardware partition

Much explanation has been given to show that it is possible to enhance both the router processing speed and QoS guarantees at the same time by implementing MPLS partly in hardware. The software/hardware partition is the first part of the practical work completed in this project. The block diagram of a logical label switching router (LSR) architecture is given in Figure 3-4. Since MPLS was originally proposed for today's largest network, Internet, which is based on TCP/IP model, MPLS routers supporting IP make the most sense and the word "IP" is used in the figure to represent the protocol at layer 3. However, MPLS can of course support any other layer 3 protocols.

It is already known that implementing MPLS routing and switching functions both in software contributes nothing to the throughput and node processing speed. Then is it possible for both functions to be implemented in hardware? Though with the rapid development of silicon fabrication, some protocols used to be carried out over higher layers are now possible to be realized in hardware to bring super-fast network node processing speed, in this project, only operations taking place below layer 3 are considered. The reasons are as follows. Not like other higher layer applications, the routing function contained within the routing plane has to deal with a very-large-scale routing table and may have to perform extremely complex routing algorithms to pick up suitable routes for LSP setup for all kinds of traffic, which consumes too many hardware resources. Hence it is now neither realistic nor cost effective to implement the routing plane in reconfigurable hardware. But for the forwarding plane that performs actual packet switching along LSPs that are already set up, it is quite suitable for hardware implementation. At the same time, since MPLS inherently removes a significant part of the burden from layer 3 routing to layer 2 switching, the throughput and node processing speed increases can be achieved by just implementing this forwarding plane in hardware. As a result, MPLS label distribution protocols that run over layer 4 (using UDP or TCP) are still supposed to be implemented in software. This software implementation will not affect the router performance adversely because LDP is only used at the time of LSP setup. During the much longer data transfer procedure

that takes place after the LSP is set up, the LDP messages are only used occasionally to keep the LSP active.

In conclusion, the current work to be done in the hardware implementation is contained only within the forwarding plane, as shown below. They are: on-chip LFIB, a subset of LIB; MPLS IP switching; off chip memory holding the LIB; and an embedded microprocessor maintaining the LIB and doing further packet processing. All these functional blocks to be implemented in hardware will be fit into a single FPGA device.

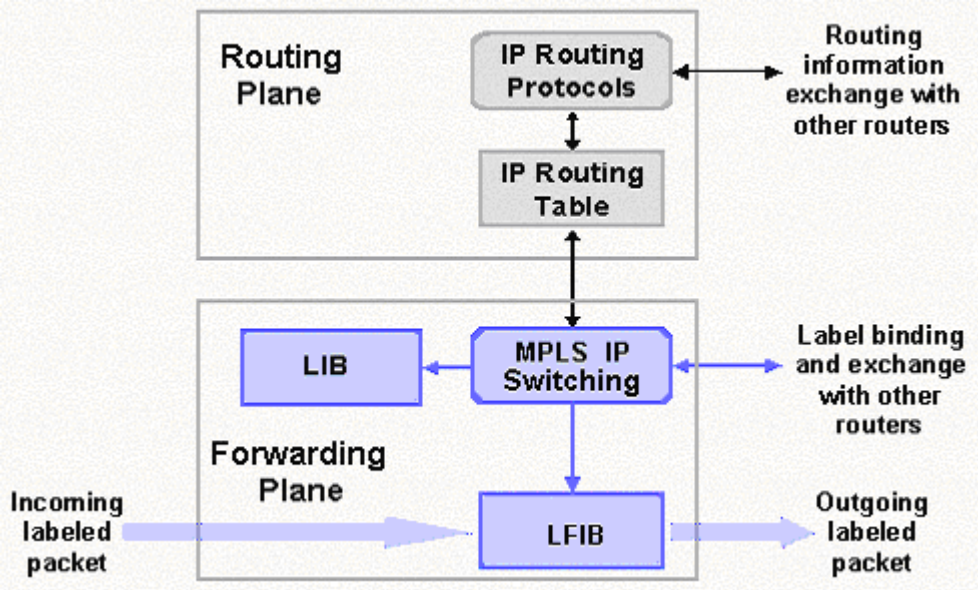


Figure 3-4 Logical Architecture of the LSR

3.2.2 Hardware Architecture of the Reconfigurable MPLS Router

As shown in Figure 3-5, the programmable router consists of several reconfigurable line cards interfacing different layer 2 materials, a scalable switching fabric that connects to an external super-power CPU through a high-bandwidth PCI bus. The switching fabric can be implemented to perform Real Packet Switching (RPS) instead of cell switching. The RPS implementation is illustrated in section 3.4.

For traditional cell switching, the difficulty of the arbitration and scheduling task increases exponentially as more line cards are added. One solution would be to use distributed arbitration on each line card. The arbiters must communicate with one

another and coordinate their switching decisions because cells from different queues are transferred in an interleaved way. This process will inevitably take more time than the required arbitration rate while introducing inefficiencies throughout the switch fabric. A global arbiter can eliminate a lot of communication overhead, but it asks for more complex functionality and consumes more hardware resources. As for the RPS, distributed arbitration on each group of queues (one group corresponds to one output) is adopted and all arbiters can work in parallel and independently with one another. In the view of each connection, data traverse the switch fabric packet by packet. The same architecture can be deployed in building the first level switching fabric that interconnects physical inputs and outputs within each line card.

Due to the reconfigurability of each line card, the router architecture presented in Figure 3-5 aids greatly in providing a scalable processing environment for high-level software administration over hardware resources. Implementation of specific new service functions or protocols can be downloaded into the reconfigurable hardware device any time on demand. The line card architecture is introduced in the next section.

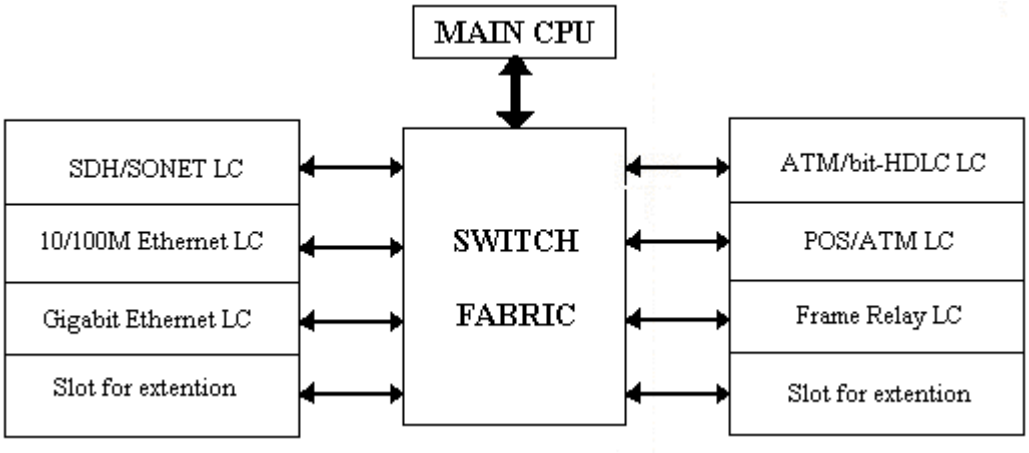


Figure 3-5 Hardware Architecture of a Reconfigurable MPLS Router

3.2.3 Single-chip RHFE design for Line Cards

A basic reconfigurable line card architecture supporting MPLS switching is illustrated below in Figure 3-6. The MAC interface block can be designed to enable the MPLS router to interface all kinds of physical layers as indicated in Figure 3-5.

In this project, components that make up a programmable, multi-port switch/router are named as the Reconfigurable Hardware Functional Element (RHFE). They employ reconfigurable hardware to provide a flexible hardware-processing environment. RHFE allows multiple hardware configurations for variable protocols and applications to be dynamically loaded into a single device and run in parallel, providing a substantial amount of per-flow processing. With dedicated on-chip logic and memory resources provided for each functional element, as well as arbitrated access to off-chip memory resources, RHFE supports a broad spectrum of protocols and applications.

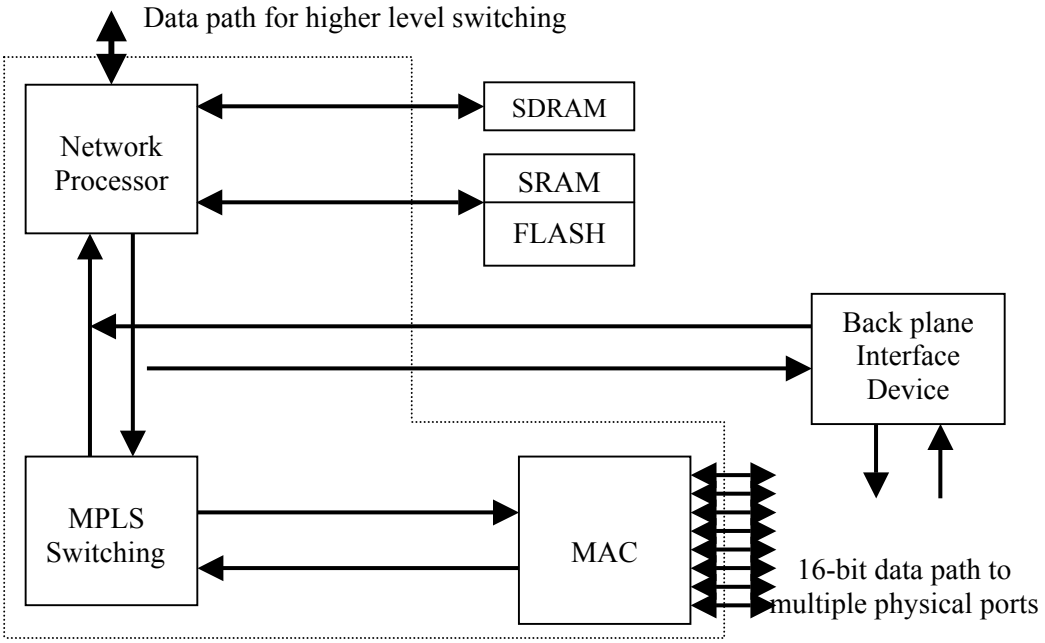


Figure 3-6 Single-Chip RHFE Design for Line Cards

As shown in Figure 3-6, we can see that the basic reconfigurable unit RHFE for the purpose of MPLS switching contains an embedded network processor, one/several (there is just one MPLS switching functional block here) protocol(s) or application specified functional block(s), and an integrated MAC block. This is actually a system-on-chip design and the integration of MAC can save much memory space for packet storage compared to the case that separate MAC chip is used. An integrated design brings faster data transfer speed and smaller product size. The embedded network microprocessor only deals with packets that cannot be switched within the local line card. The RHFE can be used as a general-purpose building block to form larger

switches/routers of arbitrary topology. According to the accommodation of the FPGA used, one or more RHFEs could reside within one chip. When these chips are to be next to each other, the connection can be made directly between their pins and a bit-parallel, clock-synchronous link could be used.

Several modules are there to form a complete MPLS IP switching functional block that sits between layer 2 and 3 as a shim layer. They are: the UD-WRR scheduler that controls the service order granted to traffic flows with different priorities; on-chip buffers; a lookup table; a label binder; a label remover; and the RPS switch fabric.

3.3 Dealing with Queuing Issues

At any network node, there is the problem of how to queue the incoming packets when traffic arrives faster than what the node can immediately handle. Also, the queuing scheduling policy is critical in providing guaranteed service for network applications with strict and diverse QoS requirements. In the following subsections, hardware implementation for several queuing algorithms are introduced and compared. Then the UD-WRR queuing scheduling policy is proposed for this project.

3.3.1. Background

This section provides some background information summarized from [18], [19], [20], [21], [24] about the queuing issues.

3.3.1.1 Priority Queue Scheduling

Current switches/routers realize the priority queue by assigning priority numbers to packets after analyzing their layer 3 or/and layer 4 headers. The priority number can represent a deadline, a virtual finishing time, or a sequence number, depending on which the link-scheduling algorithm takes into consideration. In these schemes, all packets contained in a certain queue are sorted according to their priority values pre-assigned and are transmitted in a highest-priority-first order. In the following paragraphs of this section, implementations of the four priority queue scheduling

algorithms --- FIFO priority, binary tree, shift-register and systolic array--- are briefly introduced.

a) FIFO Priority

First-in-first-out operation makes for the simplest priority queue. Clearly, no priority number and no queue resorting are needed in this case, which results in extremely easy hardware implementation. But FIFO is only meaningful to packets of the same priority. When diverse service levels are demanded, the FIFO policy is far from sufficient.

b) Binary Tree of Comparators

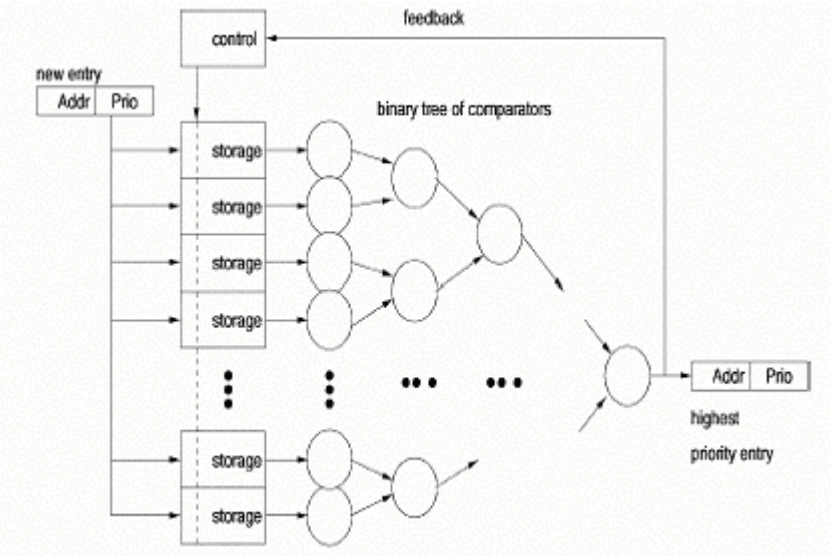


Figure 3-7 Binary Tree of Comparators Priority Queue [25]

An N-entry storage block and a comparator tree of log2 N depth make up the binary tree comparator architecture as shown in Figure 3-7. The comparator tree logic can be shared among several storage blocks to reduce hardware costs. When N increases, the depth of the comparator tree is increased by log2 N and bus loading can become a problem since a new entry has to be distributed to each storage element.

c) Shift Register

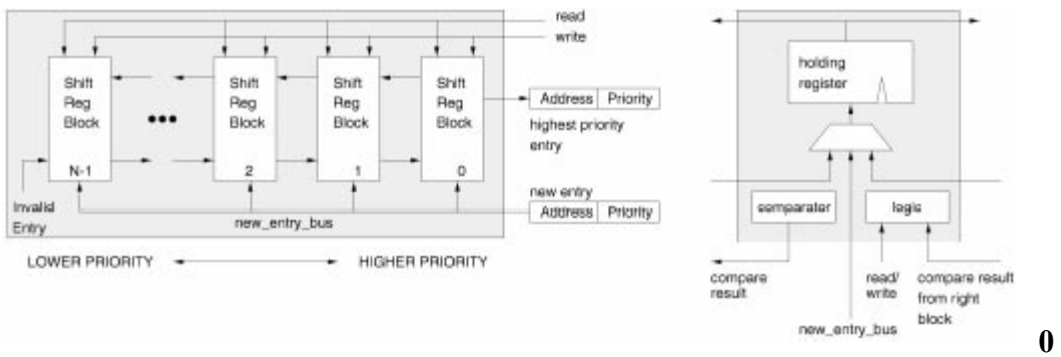


Figure 3-8 Shift Register Priority Queue and Shift Register Block [25]

In this priority queue architecture made up of shift registers, there is an array of blocks, each of which stores a single entry and communicates with its immediately adjacent block on both right and left in order to sort the queue. As shown in Figure 3-8, the zeroth block contains the current highest-priority entry. When a new entry comes, it is broadcast to all the blocks via the new_entry_bus, but only one block will latch it. The effect is that the new entry forces all entries with lower priority to shift one block to the left and places itself to the left of the entries with higher and equal priority. The lowest priority entry is discarded if the queue is full. With the increase of entry port, bus-loading problem can decrease the performance.

d) Systolic Array

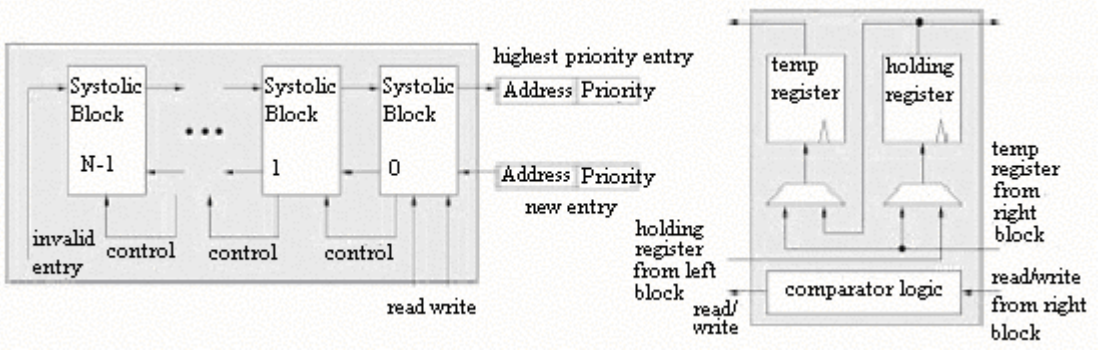


Figure 3-9 Systolic Array Priority Queue and Systolic Array Block [25]

The systolic array priority queue shown in Figure 3-9 is similar to the shift register architecture in that each block holds only one entry. The difference lies in the fact that

the systolic array architecture doesn't use the new_entry_bus to broadcast the new entry to each block, instead, only the zeroth block has access to the new entry at its arrival to compare the priority of its own and that of the new entry. The lower priority entry is then passed to the left block and the higher priority entry stays within the zeroth block. The same process is repeated until the queue is fully sorted. This methodology promises that the zeroth block always holds the highest-priority entry in the queue while introducing no bus-loading problem at the cost of twice as much storage as the shift register architecture.

3.3.1.2 Multiple Per-Flow Priority-Queue Management

Since the offered traffic less than the network's capacity can have all the packets eventually get through without QoS requirements, it used to be that only when congestion existed, the network had to make bandwidth allocation decisions, i.e., it had to arbitrate among all the links that tried to use more throughputs than existed. However, even if there is no congestion, with the dramatic increase of requirements for diverse QoS guarantees in today's IP networks, isolation among different data flows and bandwidth allocation both become necessary. When incoming packets belonging to different data flows (each of which corresponds to one of the resulted multiple per-flow queues) contend for a certain given output link, a more sophisticated scheduler is needed to serve these queues in an order that fairly allocates the available throughput to each active flow. Much research work on hardware implementations for multiple per-flow priority-queue management has been done in [25-27, 29].

Commercial switches/routers can support multiple queues per output at present, but the number is limited (a few tens), so their schedulers are relatively simple. When higher throughput and finer granularity of service level are desired, more queues have to be maintained, and specialized hardware architecture to manage these queues has to be adapted accordingly. Per-flow queuing typically requires the implementation of a large number of logical queues inside one or a few physical memories. Most advanced scheduling algorithms for per-flow queuing over QoS networks rely on the common concept of priority queues. The link-scheduling algorithm sorts the priority queue and

then interleaves the packet transmission from various sessions such that each connection's QoS requirements are satisfied. In another word, all the sessions (one priority queue per session) are multiplexed onto a single link that transmits data for different flows in each time slot. For stability, the link rate should exceed the sum of the sessions' sustainable traffic arriving rates.

Such link sharing as described in the paragraph above can be modeled by the ideal Generalized Processor Sharing (GPS), which provides a useful paradigm for governing the interaction between competing sessions. Assume a system that can be characterized by positive real numbers $\phi_1, \phi_2, ...\phi_N$, which represent the traffic, queued in the system for each session. A GPS scheduler is set to work conserving and operates at a fixed data processing rate r. This means that the scheduler keeps busy whenever there are packets waiting in the system. With the above assumptions, GPS models the link sharing abstraction by continuously dividing link bandwidth among the backlogged sessions, in proportion to the ϕ_i's. Each session *i* is guaranteed a rate of $r_i = \frac{\phi_i}{\sum_j \phi_j} r$ under GPS.

Though ideal, GPS is not feasible in practice, because it requires preemption of the link resource on an arbitrarily small time scale. A good feasible algorithm, Weighted Fair Queuing (WFQ) was presented by some researchers to approximate this idealized GPS model by ranking packets with the time they would complete service under GPS, in the absence of future arrivals [15]. In each time slot, a WFQ scheduler transmits the packet with the smallest *Service Finish* value, among the packets already queued for service. This approach closely tracks the underlying GPS reference model in terms of both throughput and delay. WFQ never lags more than one packet behind GPS in servicing a connection; similarly, a packet never completes service more than one packet time slot later than it would under GPS [16]. However, it needs non-trivial computation to sort the queue according to the priority of sessions [17], which makes it not very suitable for hardware implementation. Another good algorithm that is much easier to implement in hardware is Weighted Round Robin (WRR). Under this policy, each priority queue at each session is served in a round-robin fashion and a "fair"

allocation is achieved. The idea of round robin scheduling, in general, is that a scheduler circularly and repeatedly serves a number of clients and performs one job for each of them that has such a need during its service interval. However, to be really fair, the mechanism should not treat all sessions as exactly equal, but rather as equal within the range of a given set of weight factors. This means that the available throughput should be distributed to them in proportion to their different weight factors. Thus classic Round Robin evolves into Weighted Round Robin (WRR). WRR assigns weight factors to all sessions, and then circularly scans all of them and transmits a number of packets in the queue from each of those found to be "ready" according to the session's weight. "Ready" means that the queue has enough data and asks for service. The major advantages of WRR include guaranteed allocated bandwidth, intrinsic fairness and simple hardware implementation ([22], [23]). Therefore, the WRR technique attracts the most attention from researchers and is the basis of the service policy here.

3.3.2 An Improved UD-WRR Policy

Assume that there are N input sessions to a MPLS network node and the maximum packet length is $L_{\max}$. Since up to $L_{\max}$ bits from a packet may have to be queued over any session before the packet has "arrived" and can be processed, at least $L_{\max}$ bits of buffer space should be allocated to each session. The convention adopted in this thesis is that a packet has arrived only after its last bit has arrived.

The bit-by-bit round robin is not desirable since each session can only have one bit processed after waiting for N-1 bits of other sessions being served. Also, from the viewpoint of hardware, it is not feasible as well due to the fact that most systems are working in parallel instead of in serial now. Then it seems that the packet-by-packet WRR is the only choice if people want to use WRR. However, there is a waiting time problem inherent in a WRR system on a packet-by-packet basis. Though the scheduler can move on to serve the next session in the order instantaneously if an empty queue is encountered, when an arriving session i just misses its service interval unluckily, it cannot be served until the next service interval for session i comes. In the worst case, if the system is heavily loaded in every service interval, a packet of session i will have to

wait $\sum_{j=1}^{i-1} P_j^t + \sum_{j=i+1}^{N} P_j^t$ of packet processing time before it can be processed, where P_j^t stands for different packet processing times of an arbitrary session at time *t*. Since the maximal length of an Ethernet packet can be 1526 bytes or even longer, the waiting time will be unacceptable for most applications in the future. Thus the required buffer space to prevent buffer overflow from happening becomes unacceptable as well.

In this project, the service scheduler was supposed to be implemented in hardware; hence it is possible that the system is designed to not work on a packet-by-packet basis but on an adjustable data unit basis to alleviate the defects existing in both bit-by-bit and packet-by-packet round robin policies. The data unit is much smaller than the maximal length of an Ethernet packet and the value can be optimized according to system parameters of the network nodes, such as the number of bytes that can be transferred at a time at each rising edge of the system clock. The performance of the system adopting such WRR policy can be easily adjusted by defining the weight value of each session to be different integers that are times of some basic data unit value. So far, this modified WRR policy is named as Unit Data -WRR (UD-WRR) in this thesis. It is quite obvious that UD-WRR cannot be implemented using Java, C/C++, etc.

In the past, data flow classification is simple and strict. Therefore scheduling policies of the WRR family were ever supposed to function only among sessions that belong to the same strictly defined priority class. For example, the priority of real time traffic is absolutely higher than data traffic, which means, so long as there is real time traffic, no bandwidth will be allocated for data traffic. This can lead to service starvation for traffic with lower priorities. However, due to the demand for much finer data flow classification, it is already very common that over one physical link, there can be several logic links, or sessions. In a practical MPLS reconfigurable router, multiple physical ports and multiple logic links at each port are supposed to be supported at the same time. The mappings between input and output physical ports, as well as the definition of logic links over some physical link are both reconfigurable according to corresponding LSP setup and changes. Therefore, the UD-WRR policy is to be applied under MPLS to serve sessions with arbitrary levels of priorities.

Each session may have many conversations with time passing by, and each conversation may contain different number of packets. Though analysis done below is mainly based on what a packet perceives, the UD-WRR policy itself does not consider detailed packets or conversations within each session.

A. Leaky Bucket

Before the analysis can be presented, characteristics of the traffic supposed to arrive at the node should be introduced first. The traffic shaper adopted in this thesis is Leaky Bucket, which imposes some special constraints on the traffic before they can enter the network. The Leaky Bucket scheme works through the usage of tokens or permits, which are generated at a fixed rate, ρ. Packets can be released into the network only after the tokens of a required number are removed from the token bucket. There is no bound on the number of packets that can be buffered, but there is an upper bound on the number of bits worth of tokens, which is defined as σ. In addition to securing the required number of tokens, the traffic is further constrained to leave the bucket at a maximum rate of C, which is greater than ρ.

It is said that session *i* conforms to (σ_i, ρ_i, C_i) if

$$A_i(\tau,t) \le \min\{(t-\tau)*C_i, \sigma_i + \rho_i*(t-\tau)\}, \forall t \ge \tau \ge 0, \qquad (3.1)$$

for every session *i,* where $A_i(\tau,t)$ is the amount of session *i* traffic that leaves the leaky bucket and enters the network in time interval $(\tau,t]$. This model for incoming traffic is attractive for its arrival constraints that restrict the traffic in terms of average sustainable rate (ρ), peak rate (*C*), and burstiness (σ and *C*).

B. Analysis for the Hardware Implemented UD-WRR Policy

In this section, a simple performance analysis of a single-node UD-WRR system for sessions that operate under Leaky Bucket constraints is provided. Assumes that there are *N* sessions, and the incoming traffic A_i of each session has already been shaped by a Leaky Bucket traffic shaper, conforming to (3.1) for *i = 1,2,...N*. The system is empty before time zero. The UD-WRR service scheduler is supposed to work

conserving (e.g. it is never idle if there are data in the system), operates at a fixed system clock speed and serves all *N* sessions circularly. The total time duration for the UD-WRR scheduler to serve each of the *N* sessions once is defined as a *service cycle*. The length of each cycle is not a constant because each session may have different amount of data in queue to be served during each service cycle.

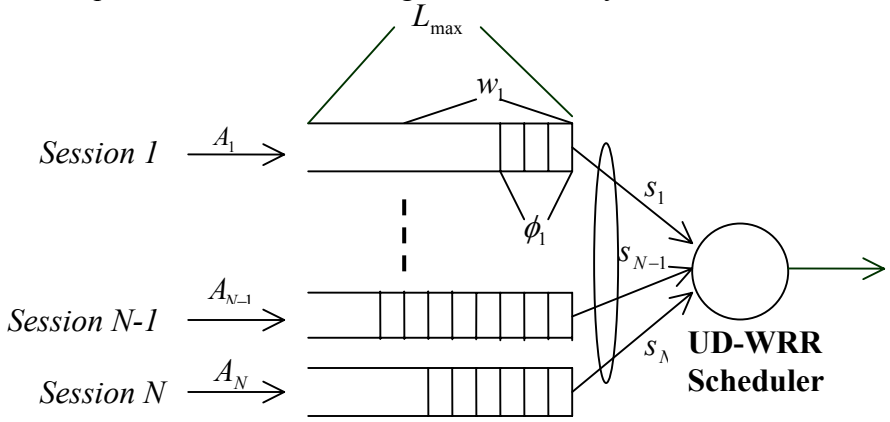


Figure 3-10 UD-WRR Scheduling Policy

Figure 3-10 depicts a basic idea of how the UD-WRR works. An integer weight w_i is associated with each session *i* and tells the UD-WRR scheduler that the session *i* can have maximally w_i data units processed during its service interval within one service cycle. It does not hurt if it is assumed that one data unit is processed within one time unit. Thus the number of data units being processed within an arbitrary time interval $(\tau, t]$ by the UD-WRR scheduler also represents the length of time needed for this amount of traffic to be processed, which is $t-\tau$. Both τ and *t* are positive integers

Within each service cycle, the scheduler polls the *N* sessions according to some pre-computed sequence, say, in order *1,2,...N*, in an attempt to serve the session *i* at a guaranteed average service ratio of $\frac{w_i}{\sum_j w_j}$, j = *1,2,...N.* This lower bound of service ratio achieved by arbitrary session *i* under the UD-WRR scheduler will be proved to be true in the following paragraphs. It is thus apparent that different QoS guarantees can be provided for each session by adjusting the value of w_i properly.

During a certain service cycle c_k, within which the *N* sessions are served exactly once, the number of data units waiting to be served at session *i* are represented by

positive integers $\phi_i(c_k)$, $k=1,2,...\propto$. Let $S_i(c_k)$ and $S(c_k)$ be the number of data units of session i and all N sessions served within the service cycle c_k respectively. Let $S_i(\tau,t)$ and $S(\tau,t)$ be the number of data units of session i and all N sessions served within an arbitrary time interval $(\tau,t]$. The time interval $(\tau,t]$ may include several service cycles.

Under normal cases, there are

$$S_i(c_k)=\max\{\phi_i(c_k),w_i\} \tag{3.2}$$

$$S(c_k)=\sum_j \max\{\phi_j(c_k),w_j\}, j = 1,2,...N. \tag{3.3}$$

When the scheduler is working conserving and all the sessions are active during cycle c_k ("active" means the session asks for as much service as possible, with the maximum of w_i), (3.2) and (3.3) can be always reduced respectively to $S_i(c_k)= w_i$ and $S(c_k)=\sum_j w_j$, if the packet will not be finished processing within the current cycle c_k. This reflects the fact that: over any session i, before a packet under service is completely served (no matter receive or transmit or other processing procedures), the actual number of data units waiting to be processed during a service cycle c_k, $\phi_i(c_k)$, equals w_i. At this time, the length of the cycle reaches its maximum and there is $c_k=S(c_k)=\sum_j w_j$.

Thus for a session i packet $l_i(\tau,t)$, which has an arbitrary length of l_i, starts getting service at time τ, and finishes its processing at time t, it is always true that

$$S_i(\tau,t)= l_i(\tau,t)= \sum_{k=1}^{K} w_i + \phi_i(c_{K+1})=K w_i+\phi_i(c_{K+1}), \text{k}=1,2...K. \tag{3.4}$$

K stands for the number of *complete* service cycles experienced by the packet $l_i(\tau,t)$ between the time interval $(\tau,t]$.

Let $T_i(\tau,t)$ be the processing time needed by the session i packet with the length l_i during time interval $(\tau,t]$. Since UD-WRR scheduler only serves each session maximally w_i data units, which are much smaller than the packet length within each

cycle, the processing time of an arbitrary packet usually results in lasting for several service cycles. According to all the definitions introduced above, the expression for $T_i(\tau,t)$ can be obtained as follows:

$$T_i(\tau,t)=\sum_{j=i}^{N}S_j(c_1)+\sum_{k=2}^{K}\sum_{j}S_j(c_k)+\sum_{j=1}^{i-1}S_j(c_{K+1})+\phi_i(c_{K+1}) \tag{3.5}$$

$$=\sum_{j=i}^{N}\max\{\phi_j(c_1),w_j\}+\sum_{k=1}^{K}\sum_{j}\max\{\phi_j(c_k),w_j\}+\sum_{j=1}^{i-1}\max\{\phi_j(c_{K+1}),w_j\}+\phi_i(c_{K+1})$$

Since w_i is always greater than or equal to $\phi_i(c_k)$ as explained earlier, it follows that:

$$T_i(\tau,t)\le\sum_{k=1}^{1}\sum_{j=i}^{N}w_j+\sum_{k=2}^{K}\sum_{j}w_j+\sum_{k=K+1}^{K+1}\sum_{j=1}^{i-1}w_j+\phi_i(c_{K+1})$$

$$\le K\sum_{j}w_j+\phi_i(c_{K+1}) \tag{3.6}$$

The average service ratio perceived by a packet with arbitrary length over session *i* under discussion during the processing time $t_i(\tau,t)$ is:

$$\overline{R}_i(\tau,t)=\frac{l_i(\tau,t)}{T_i(\tau,t)} \tag{3.7}$$

Substituting (3.4) and (3.6) into (3.7) gives

$$\overline{R}_i(\tau,t)\ \ge\frac{Kw_i+\phi_i(c_{K+1})}{K\sum_{j}w_j+\phi_i(c_{K+1})}\ \ge\frac{w_i}{\sum_{j}w_j} \tag{3.8}$$

Hence, from the view of any packet, the service ratio the UD-WRR scheduler can provide for a certain session is guaranteed to be no less than $\frac{w_i}{\sum_{j}w_j}$. This is a worst-case service ratio a packet of session *i* perceives. If the processing speed of UD-WRR scheduler over the time period of this worst-case is set to be the same rate as GPS scheduler's fixed rate r, it is clear that when the data unit size is small, the UD-WRR approximates GPS pretty well, in comparison to the service rate seen by session *i* under the GPS system, $r_i=\frac{\phi_i}{\sum_{j}\phi_j}r$.

Another parameter needs to be considered is, at least how long each service interval should last to make the hardware-implemented UD-WRR scheduler work as

efficiently as possible. Let's take a look at the best-case service ratio a packet of session *i* can perceive first. Assume such an extreme situation: except session *i,* no other session has any packet to be served, which means, the scheduler will only serve session *i* each service cycle. Under an ideal GPS scheduling system, it is clear that all the bandwidth can be used up by any session *i* in the absence of traffic from other sessions, which means that the work efficiency for any particular session can reach 100% theoretically. However, for a practical UD-WRR system, this is a goal impossible to achieve due to reasons given below.

To implement the UD-WRR scheduling policy in hardware, the only way to jump over all the empty sessions without checking whether each session is empty or not per time unit is to build a very large scale "case" circuit to handle the service order explicitly for each possible combination of empty sessions. Before each service cycle begins, the combination of empty sessions is determined and service intervals are only granted to those not empty. Such design brings a circuitry complex of $\sum_{k=1}^{k=N} \frac{N!}{k!(N-k)!}$, which refers to the number of lines of Verilog code needed to implement a system with *N* sessions to be served. It is assumed that each line of Verilog code completes a basic logic function. Clearly, a design using so many hardware resources is not practical. Actually, even this exhaustive-search design cannot bring ideal 100% work efficiency due to the extra one time unit used for service order determination before each service cycle starts.

A pragmatic and very simple way to realize the UD-WRR scheduling policy is to always permit one time unit stay, named here as the "checking" time unit, for each service interval (not just each service cycle). This is to enable the scheduler to check whether the current session is empty or not. If empty, the scheduler enters the next state to serve the next object immediately; if not empty, the scheduler can start serving the current non-empty session from the very first time unit. Thus for non-empty sessions, the "checking" time unit is utilized at the same time for data processing. Therefore, according to what has been defined above, the best-case service ratio a session *i* packet can experience under the assumption that no session except session *i* has data to process

is: $\frac{w_i}{(N-1)+w_i}$. It is clearly from this expression that there is no possibility for session *i* to obtain 100% service efficiency because the value of *N* under discussion is always greater than 1.

Let $\frac{w_i}{(N-1)+w_i}=e_i$ (3.9), where e_i stands for the best-case service ratio requirement for session *i* and $\sum_i e_i = 1$. It is clear that $e_i=\frac{1+(w_i-1)}{N+(w_i-1)} \geq \frac{1}{N}$ (3.10), since w_i is always greater than or equal to 1. Then with any specific $e_i \geq \frac{1}{N}$, the minimal value of w_i obtained should be:

$$\text{Min } \{w_i\} = (N-1)\min\{\frac{e_i}{1-e_i}\} \qquad i=1,...N \tag{3.11}$$

With the maximal and minimal values of w_i that can be derived from (3.8) and (3.11), the concrete weight value for each session can be decided according to relevant service time ratio between each session, which has been predefined according to different QoS demands. In other words, processing delays experienced by a session *i* packet can be reduced by increasing the value of w_i for that session when higher QoS requirements are set for the corresponding LSP *i,* along which session *i* packets travel through. The following is a brief summary for why UD-WRR is an attractive multiplexing scheme:

- Extremely easy hardware implementation.
- Taking the full system throughput as "1", a session *i* packet is guaranteed to have a throughput always greater than or equal to $\frac{w_i}{\sum_j w_j}$, independent of any other session.
- With the above guaranteed worst case throughput, the delay experienced by a session *i* packet due to necessary processing time can be bounded as a function of the session *i* queue length and all the sessions' weight values w_i's, $K\sum_j w_j + \phi_i(c_{k+1})$, independent of the queues and arrivals of the other sessions.

- Each session might have different traffic characteristic, some may experience longer packets and others may experience shorter ones. Thus different service efficiency calculating methods should be used in different cases. By varying the w_i's, the flexibility of treating the sessions in a variety of different ways can be achieved in a straightforward manner. For example, when all w_i's are equal, the system reduces to uniform service sharing. UD-WRR is flexible enough to provide service on the basis of data byte number or on the basis of packet number, simply by assigning appropriate values to w_i's.
- Data processing can be done continuously even if the packet data have not arrived completely, when the packet length is provided at the same time as the first data unit of the packet arrives.

In fact, when the unit of w_i is set to "bit" and each session has the same w_i value 1, the UD-WRR policy reduces to the bit-wise round robin; when the unit of w_i is set to "packet" and each session also has the same w_i value 1 (no matter how long the packet is), the UD-WRR policy reduces to the packet-based round robin policy that is usually implemented in software.

3.4 RPS and UD-WRR Implementation in a MPLS System

The implementation of the switching fabric is challenging. In a typical crossbar fabric, cells are firstly queued on the input side of the switch fabric. The state of all the input queues is visible to the crossbar arbiter. On the basis of these states, knowledge of the QoS required for each flow and feedback from the output queues, the arbiter decides which connection to make in the memory-less crossbar and thus determines the order in which cells get forwarded to their respective egress ports. However, the input queuing has the Head of Line Blocking (HoLB) problem. When the cells at the head of several inlet queues happen to be destined to the same output port, the fabric can accept only one of them. In this scenario, all the other queues remain idle, although cells behind the head of those idling queues are actually destined to other outputs that are not busy at all.

Hence usually, the cells in the input queues are presorted on the basis of destination address and class, which forms many VOQs. This prearrangement brings great freedom and flexibility to the arbitration algorithms to manage the QoS and to maximize the efficiency of the fabric aside from avoiding the HOLB problem. Therefore, VOQs are also adopted in the RPS scheme.

Assume that there are N inputs and N outputs. Mapping to each input, there are N VOQs, each of which represents an output. As mentioned earlier, a mapping between inputs and outputs can be determined in advance and be created through LSP setup in the MPLS environment. In addition, it is very straightforward to relate the MPLS label assignment to the virtual link weight assignment since the group of MPLS labels assigned to a LSP stands for the service priority of the packets traveling along this LSP. This is to say, the virtual link weight that is used for providing service scheduling by the UD-WRR policy can be obtained once the corresponding LSP has been setup.

All priority queue scheduling algorithms mentioned in the section 3.3.1.1 are methods that serve within the same priority queue; however, they can be utilized together with UD-WRR to realize multiple per-flow priority queue scheduling. Among all of these queuing techniques, FIFO is still the simplest and most straightforward method for hardware implementation, and inextricably intertwines three allocation issues of bandwidth, promptness and buffer space occupation. At the same time, when packets at the same priority level arrive in the order that they were sent, maintaining FIFO ordering among entries with the same priority is necessary. Therefore, the FIFO scheme is adopted to buffer packets within each flow, and the UD-WRR that approximates the GPS system pretty well on the basis of a small data unit is adopted to serve these flows with different priorities circularly. The QOS-aware UD-WRR ensures that the outputs are never starved of packets that are already waiting in the input queues.

So far, a feasible $N \times N$ Real Packet Switching architecture adopting UD-WRR can be constructed as shown in Figure 3-11, which is practically easy to build and can realize pipelined data transfer at each output packet by packet.

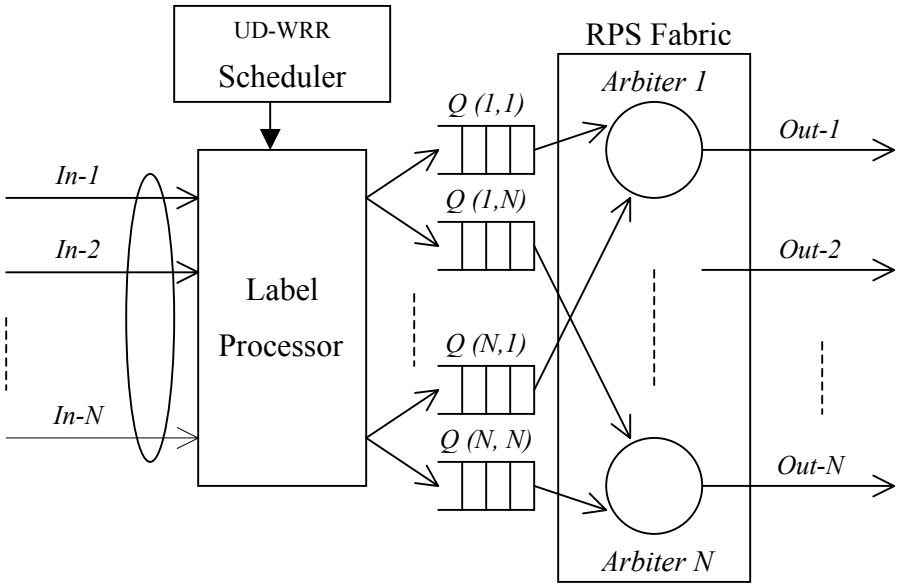


Figure 3-11 $N \times N$ **RPS Architecture Adopting UD-WRR for MPLS**

In-i represents the input *i (i=1,2…N)*; *Q (i, j)* represents a VOQ temporarily storing packets from the input *i* to the output *j*; and *Out-j* represents the output *j (j=1,2…N)*. According to their weights respectively, the UD-WRR scheduler serves all *N* input queues for label processing in a circular manner. After label analysis, incoming packets are transferred to VOQs corresponding to their destination outputs, where they wait for their turn to be output. The Arbiter *k (k=j)* controls the order in which the head packets from *Q (1, j)* to *Q (N, j)* are transferred. All arbiters work independently and in parallel. In the view of a certain output, data are transferred packet by packet, instead of cell by cell, which is different from the traditional crossbar switching, and is why this switching scheme is given the name "Real Packet Switching". Packets with variable lengths can be switched intelligently without performing packet segmentation and reassembly.

Chapter 4 Reconfigurable MPLS Hardware Implementation

Based on the investigation and analysis performed in the previous chapters, the essential part of hardware implementations for a primary reconfigurable MPLS router is finished. In this chapter, the first section gives a brief overview of hardware implementation strategy drawn from practical considerations; the following sections first present a block diagram of the top-level hardware architecture, and then the details of the MPLS functional block implementation, which includes 6 sub-modules.

Verilog HDL was used for the whole logic circuit design that was later all downloaded into a single Altera FPGA device for the tests. Due to space limitation, the lengthy Verilog codes are not provided in this thesis. Simulation, tests and results will be illustrated in Chapter 5 and Chapter 6.

4.1 Implementation Strategy Considerations

A commercially practical IP reconfigurable MPLS router would likely be required to support Ethernet, ATM and Frame Relay, and other protocols, at layer 2. However in this project, only Ethernet has been taken into consideration. The reasons are as follows. Firstly, this project was focused on MPLS hardware realization and so supporting different layer 2 protocols was not the critical point. Secondly, it was already clear enough to demonstrate the advantages of MPLS for packet forwarding in the environment of Ethernet. Finally, Ethernet is the most popular layer 2 protocol at present, which makes it comparatively cheaper and easier to find suitable equipment from the market to set up a practical test bed.

As described in Chapter 3, it is with a flexible architecture and the reconfigurable hardware units that more efficient or newer value-added functions can be added to the system later without causing too much hardware modification or replacement. Also, with the scale of networks becoming larger and larger, traffic on each link and the number of links at each network node both increase dramatically, thus multi-port, multi-

service and multi-user switches/routers become desirable. It is expected that more and more ports should be integrated on a single chip, since on-chip delay is much less than off-chip delay. This integration also brings products with better performance, less fabrication cost and smaller product size.

According to the RHFE architecture introduced in Chapter 3, an integrated MAC is supposed to be included within the RHFE. However, due to the time limitation and the availability of existing separate MAC chips, integrated MAC circuit design was not included within the scope of the current project. As a result, dedicated circuitry to interface the separate MAC chip CS8900A was designed and temporary data buffering was provided within the MPLS block.

4.2 Design and Implementation

The design and implementations of the MPLS hardware are illustrated in this part.

4.2.1 Top Module Design

In this single-chip system, there are 8 sets of buffer integrated, each of which corresponds to a LSP and is assigned the particular priority number for that LSP. A multiple queue service scheduler adopting the UD-WRR policy and maintaining this set of prioritized buffers is implemented.

Increasing the number of queues requires adding more buffer space, which brings added hardware cost and increased complexity of the priority encoder at the same time. Therefore logically linked lists instead of physical buffers should and can be utilized to accommodate more queues conceptually. Though currently the necessity of using multiple logical links does not exist in this first step design, modification for such a purpose is straightforward and simple based on the original design.

For the system concerned here, 32-bit wide buses are adopted. The reason not to use 16 or 8 bit-wide buses is that to provide a certain data processing speed, a wider data bus operating at a lower system clock speed helps to maintain the system more stable. The reason not to use a 64 bit-wide bus is that the 32 bit-wide bus only uses up half of the I/O pins while being able to provide adequate data processing speed.

The main functions implemented in hardware that can increase the node processing speed and efficiency should include: 1) table lookup function using CAM technique for packet forwarding; 2) MPLS label removing and binding, which enables fast layer 3 routing through layer 2 MPLS switching; 3) 8 sets of transmit and receive buffers for 8 physical ports integrated on a single chip to reduce both cost and product size. 4) Standard I/O interfaces for both material access layer and the embedded microprocessor.

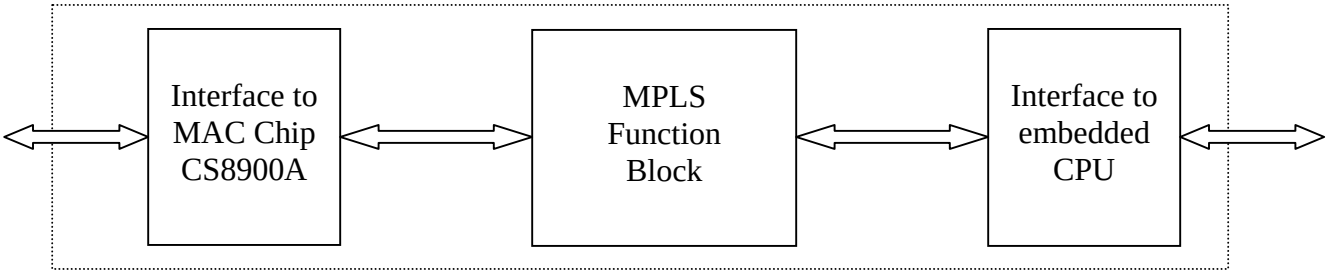


Figure 4-1 Top Module Block Diagram

Figure 4-1 illustrates the block diagram of the top-level design of a simplified MPLS node performing label switching, which includes the MPLS functional block, the interface to the Media Access Controller CS8900A from *Cirrus Logic*, and the interface to the embedded local microprocessor. Interface design for CS8900A and glue circuitry between the second layer modules in the MPLS functional block are illustrated in full in Chapter 5. As for the interface to the local embedded microprocessor, it is left for future work. In the following sections, circuit design of different modules that make up the MPLS functional block is illustrated in details.

4.2.2 Second Layer Modules

Figure 4-2 shows the second layer block diagram within the MPLS functional block, which contains 6 functional modules: Transmit Buffers for 8 outgoing ports, Receive Buffers for 8 incoming ports, Label Removing, Label Binding and Switching, Lookup Table, and State Machines/Service Schedulers.

The MPLS functional block has two dedicated unidirectional 32-bit wide data buses for transmit and receive respectively, to support dual communications. It also provides good architecture flexibility when in the future there is a need to reconfigure the MPLS

functional block to support 64-bit parallel data transfer. The 32-bit receive input port and the 32-bit transmit output port can be redefined into bi-directional ports, which is not covered at present though.

When outputting a packet, the MPLS functional block can provide the packet length, indicators of the start/end of the packets, and signals indicting if current data on the data bus are valid or not. Similarly, the MPLS block has to be provided with the same information when there is a packet coming in.

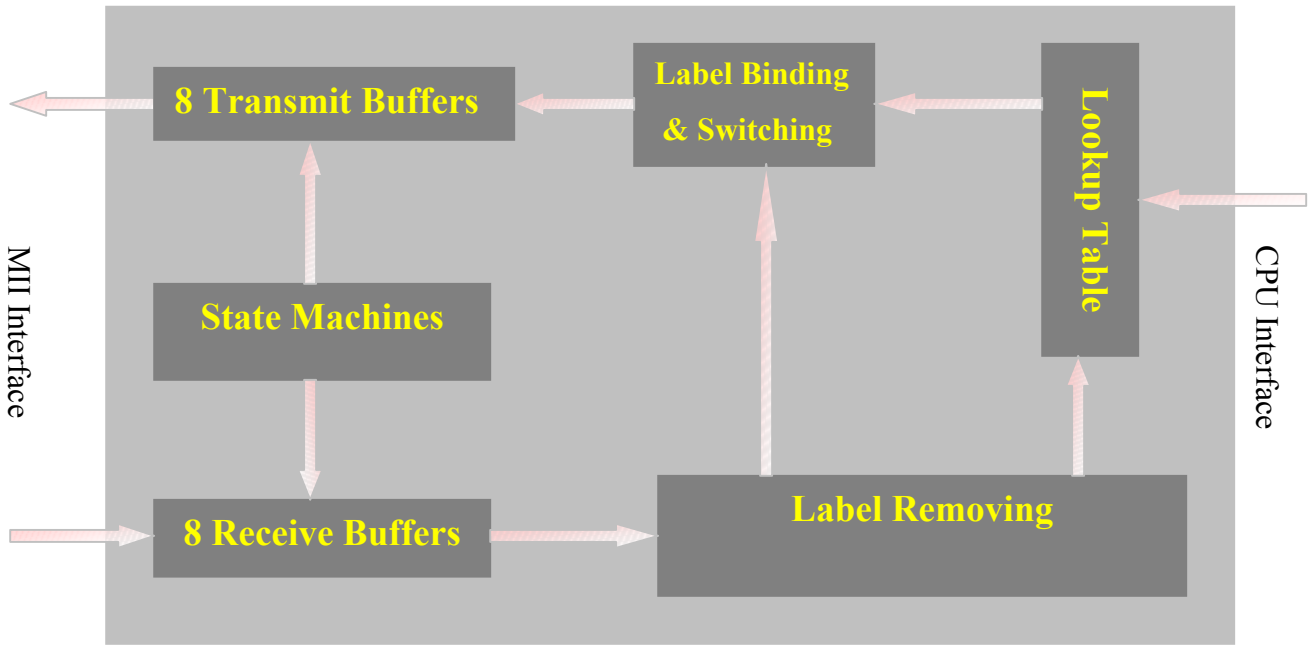


Figure 4-2 MPLS Functional Block Diagram

Due to time limitation and as the first-step simplified implementation, it is assumed that there are not data flows entering from different input ports heading for the same destination at a MPLS node in this project. This is to say that there is no LSP merging under discussion and thus there is no need for intermediate buffers at present. In the future work when intermediate buffers are added, the number of packets held within each buffer can be computed by setting constraints in packet delay time while controlling the probability of buffer overflow under a required level.

Eight transmit and eight receive buffers are integrated to make the design more cost effective and suffer less data transfer delay. Each set of buffer space, for both receive and transmit, can be taken as the extension of that of a corresponding physical port in the media access controller CS900A. All transmit/receive buffers are realized in FIFO whose length is currently set to be 1600 bytes, which can hold several short 802.3 packets.

The interface for the local microprocessor, named as the ninth port, does not have its own buffer. This microprocessor is embedded and it can itself buffer the packet generated there. IP packets from local layer 3 are sent through this ninth port, to the Lookup Table module directly. There, a corresponding MPLS label is assigned to the packet according to its IP header and/or other additional service requirements specified for a certain FEC the packet belongs to. The specifications are settled between customers and Internet service providers in advance.

Packets entering the node from the network side are firs buffered at one of the 8 receive buffers waiting for their turns for further processing. At the Label Removing module, the label of the packet is stripped off and then this label is fed into the Lookup Table module as an index to find a new appropriate outgoing label for the packet.

After the new outgoing label is ready and the outgoing port is determined, the Label Binding module binds this label to the packet coming from either the network side or the local microprocessor, and sends the packet to the corresponding transmit buffer, where the packet waits for its turn to get transmitted onto the Ethernet.

The State Machine module is designed to control the service order and duration time for each port according to the UD-WRR policy introduced earlier. Together with other signals, it regulates the working procedure of the whole system and keeps the other 5 second-layer modules cooperating together with a proper time schedule. Detailed tasks it completes include manipulating the procedure of checking 8 receive buffers and the microprocessor interface to see if there is any data ready for processing, and then having each port served to finish its label switching in an pre-determined order within its weighted service interval. Values of the weighted factors used by the UD-

WRR policy are temporarily taken as equal and the port number “8” can be adjusted according to application requirements in the future.

The Lookup Table module takes advantages of techniques of both CAM and RAM. This module only talks with the Label Removing module to get necessary information and is completely separate from other modules. This organization provides a clear distinction between functional modules.

Due to the reprogrammable characteristic of the FPGA device, the number and depth of FIFOs and the scale of the lookup table can be extended in the future when more table items are to be added and more traffic flows have to be distinguished. There are system status registers that store all the current status parameters and can be read out for debugging or administration purposes. However, they are read only and will be overwritten once the next packet starts receiving its service.

4.2.3 Implementation Details of the Third-Layer Modules

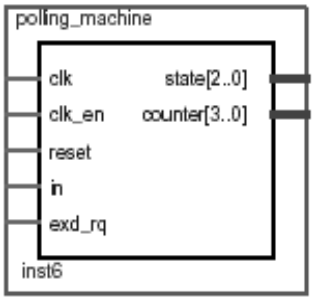
The following sub-sections describe the 6 3rd layer modules in full detail.

4.2.3.1 State Machines

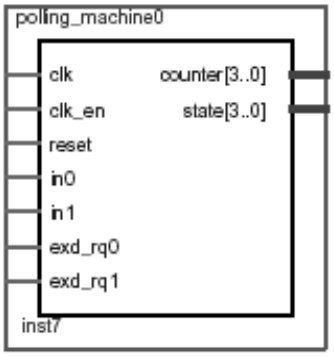
Three separate state machines, State_Machine_1,2,3 are instantiated within this module. The name of the prototype of State_Machine_1 and 2 is Polling_Machine; the name of the prototype of State_Machine_3 is Polling_Machine0. In the following paragraphs, functions and signal description of the state machines are given, according to what is shown in Figure 4-3.

State_Machine_1 generates 8 states, each of which represents the service interval granted to a certain physical port for data reception. Each service interval allows the corresponding weighted number of writes executed on 8 MPLS receive FIFOs. In another word, State_Machine_1 determines the sequence of reads performed on 8 receive buffers of the media access controllers to obtain received packets by transiting from state 0 to state 7 in turn. State0 stands for the service interval granted to MAC receive_buffer0/MPLS receive FIFO0… state7 stands for the service interval granted to MAC receive_buffer7/MPLS receive FIFO7.

State_Machine_2 also generates 8 states, each of which represents a certain service interval granted to a certain physical port for data transmission. Each service interval allows the weighted number of reads executed on 8 MPLS transmit FIFOs. This is to say that State-Machine-2 determines the sequence of writes performed on 8 MAC transmit FIFOs to transfer the packets to be transmitted by transiting from state 0 to state 7 in turn. State0 stands for the service interval granted to MAC transmit buffer0/MPLS transmit FIFO0…state7 stands for stands for the service interval granted to MAC transmit buffer7/MPLS transmit FIFO7.



a) Prototype of State_Machine_1 and 2



b) Prototype for State_Machine_3

Figure 4-3 State Machine Block Symbols, prototype names: a) polling_machine b) polling_machine0

The third state machine named State_Machine_3 is a little bit different from those introduced above. Except serving the 8 receive FIFOs, it also takes the responsibility of deciding if the local layer 3 has any data waiting for processing. Hence it generates 9 states and transits from state 0 to state8 to have reads performed on 8 receive FIFOs plus the local host. State0 stands for the service interval granted to receive FIFO0, state1 stands for the service interval granted to receive FIFO1 …state8 stands for the service interval granted to the local layer 3 interface, the microprocessor interface. Each state has the same weighted length of service time as that of State_Machine_1/2. In each state, data from one receive FIFO or the local layer 3 is read and then processed. If it is found a suitable outgoing label, the packet is transferred to the transmit FIFO corresponding to its destined outgoing port.

To prevent the FIFOs from overflowing or running short of data, the three State machines should not work at the same clock speed if multiple ports need to be served smoothly in the UD-WRR manner. To achieve successful service multiplexing, State_Machine_3 has to work at a clock speed 9 times faster than that at which the other two State machines work. The reason that it is 9 times faster instead of 8 times faster is that State_Machine_3 is not only responsible for 8 MPLS receive FIFOs, but also responsible for the local microprocessor.

At any time of each state, if State_Machine_1/2 detects that the corresponding receive/transmit FIFO has no data to be received/ transmitted, or the corresponding MAC port has no receive data to provide or has no room to hold any more transmit data, it will leave the current state and enter the next one right away without idling for its full length of the service interval. For State_Machine_3, the state transition can take place right away as well, whenever it detects that there is no packet ready in the corresponding receive FIFO (or in the microprocessor) for processing; or, the required transmit FIFO currently lacks enough space to hold any more data. In this way, the unnecessary waiting time experienced by each service object is reduced. In cases other than that mentioned above, each state will last for the full length of its weighted service time. However, it can also transit to the next state in the middle of the service interval right away once the task undergoing (such as packet transmission/ receiving or label removing/binding) is finished.

At the state transition, some important signals can lose the correct timing relationship between each other, which will lead the whole system into a malfunction state and so asks for special consideration to prevent this from happening. The method used is to have the state machine able to extend its current service interval to finish all necessary processing once some state extension requirement signals become active. The signals *in* and *exd_rq* of the prototype polling_machine and the signals *in0/1* and *exd_rq0/1* of the prototype polling_machine0 are all for the state length adjustment. The *state* signal of both prototypes outputs the current service state the system is in; the *counter* outputs the service timer value of this state. The maximal value of the service timer is the service weight granted to current service state.

4.2.3.2 Receive Buffers

This module holds received data temporarily and generates delimiter signals for the packet. State_Machine_1 takes the responsibility of enabling data transfer from MAC chip *i* to receive FIFO *i*. State_Machine_3 takes the responsibility of enabling data transfer from some receive FIFO *i* to the corresponding transmit FIFO *j*.

For any receive FIFO *i*, when the last read behavior taken in a regular service interval brings the penultimate data unit of a packet (this means that the final data unit of the packet will still show up on the data bus when the service state has changed into the next one to serve the packet of receive FIFO (*i*+1)), the last several bytes of data from receive FIFO *i* originally destined for transmit FIFO *j* will be mistakenly written into some other transmit FIFO *k*, where actually the packets of receive FIFO (*i*+1) should go. Such a malfunction can be prevented by keeping some dedicated signals low during the time period that is originally for the last three normal data unit fetches to be performed on the receive FIFO *i*, which makes it as if there were no buffer space available in transmit FIFO *j* and thus the receive FIFO *i* knows that it should not output any more data. However, in the case that the head packet of the receive FIFO *i* has been almost finished, this method will force the head packet to wait for a whole service cycle to complete its processing in its next service interval. This means that, the session *i* packet experiences a longer than necessary processing time delay. To alleviate this unnecessary performance degradation, a low-active service extension demand signal is issued by the Receive Buffers module to State_Machine_3 to realize service interval extension when required. It holds true as well for the other two state machines.

Except for the case of end packet data transfer, a special condition occurs when there is an immediate state transition following the first 4 bytes of packet data transferred from the receive FIFO *i* to the Label Removing Module. The first 4 bytes of packet data is defined as the MPLS label in this project, and due to the immediate state transition, no label processing for the packet from the receive FIFO *i* can take place and dedicated buffer space has to be allocated to store such incoming MPLS labels from all the 8 receive FIFOs. To reduce hardware consumption and avoid the circuit complexity brought by this kind of buffer space management, service interval extension is also

applied to make sure each incoming MPLS label is processed right away after it is stripped off from the packet.

This module contains 8 36-bit wide data FIFOs, which are actually the expansions of the on-chip buffer space of the 8 separate MAC chips. The lower 32 bits are for packet data; the higher 4 bits form the valid-data-indicator, which indicate the validity of the 4 bytes of data made up of the lower 32 bits. Not every packet has a length of integer times 4 and when there is only start and end of packet indicators available, valid-data-indicator bits becomes necessary. Besides this, it also provides a possible method of simple data encryption. "1" means the byte is valid and "0" means not. For example, if bit 35 is high, it means the most significant byte made up of bit 24 to bit 31 is valid; if bit 35 is low, it means that this most significant byte is invalid and should be discarded. Bits 34 to 32 indicate the validity of the bytes made up of bit 16 to bit 23, bit 8 to bit 15 and bit 0 to bit 7 respectively.

Since it is very likely that many Ethernet packets are with different short lengths, each data FIFO may have more than one packet buffered in the queue from time to time. To output the buffered packets later correctly, the length of each packet has to be recorded along as well. Therefore a separate FIFO named pkt_length_fifo is instantiated in this module to buffer the length of each packet staying in the data FIFO correspondingly. Please refer to Figure 4-4 for a clearer picture.

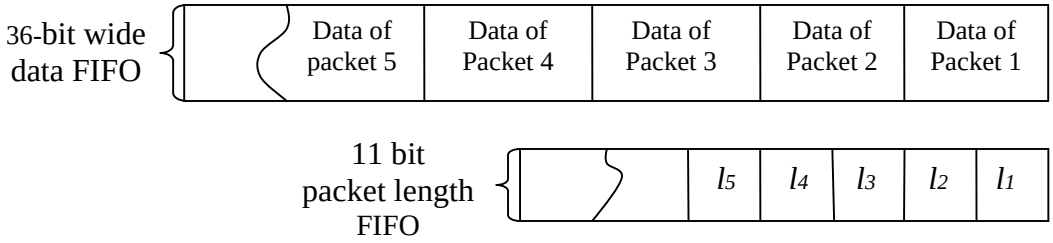


Figure 4-4 Relationship Between the Data FIFO and Packet Length FIFO: *l*n (n=1,2,…) represents the length value of the corresponding packet

Figure 4-5 depicts the functional block diagram of the Receive Buffers module that has been implemented. Two modes are supported by this Receive Buffers module to

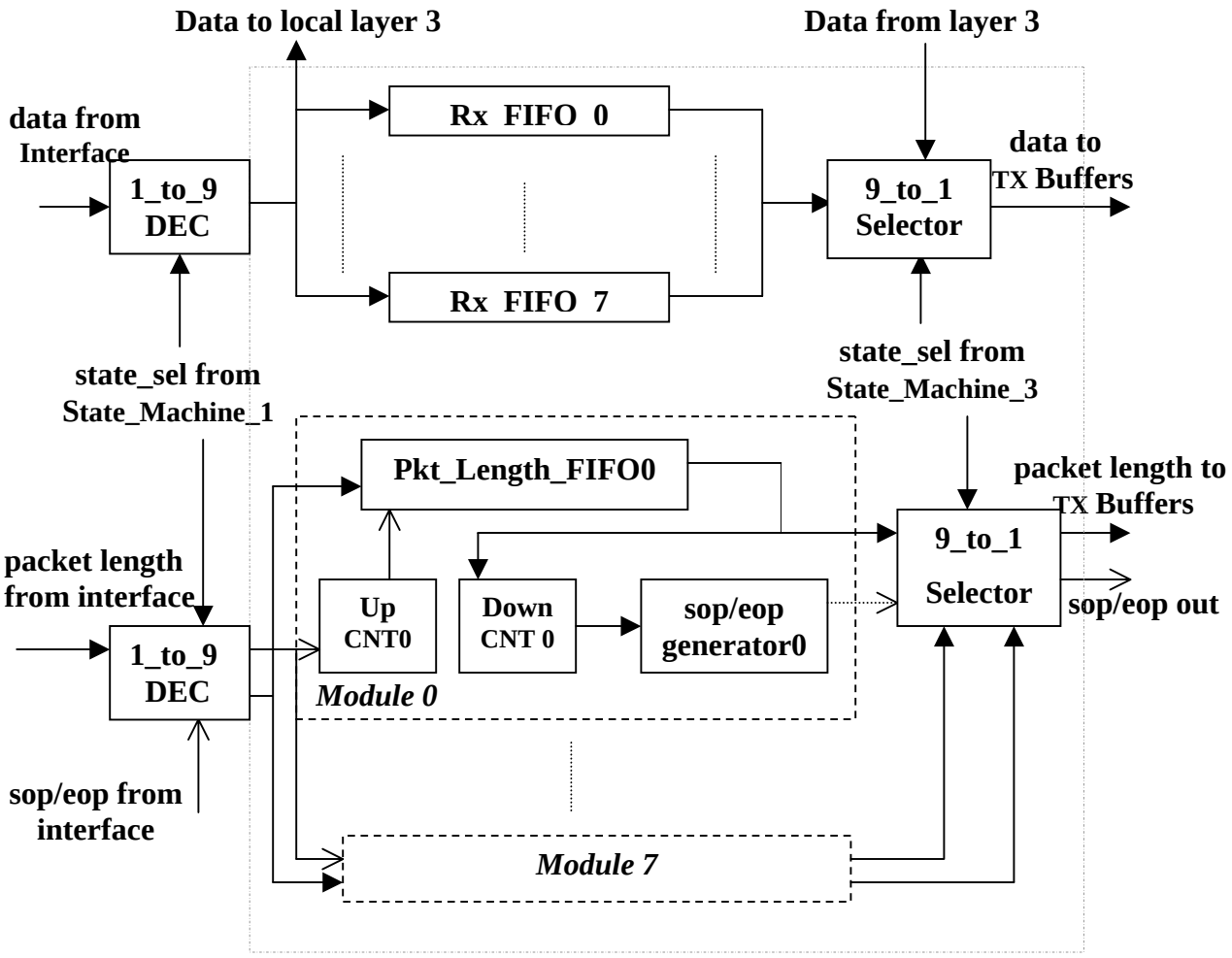


Figure 4-5 Receive Buffers Block Diagram (CNT means Counter)

obtain the length of an incoming packet. If the MAC chip can provide packet length in advance (before real packet data transfer begins), the Receive Buffers module receives it and buffers it into the pkt_length_fifo directly; if the MAC chip only provides start/end of packet (sop/eop) and valid-bytes indicators, then the packet length can be calculated by a simple up counter with the aid of these indicators provided that the whole packet is received correctly. The up counter is cleared synchronously each time reset is high or the incoming eop signal is high. In either case, the packet length is written to the pkt_length_fifo at the rising edge of the incoming end-of-packet indicator. When the Receive Buffers module outputs the stored packet with its stored length to Label Removing module, corresponding sop/eop indicators are required to be generated for the Label Removing module to function correctly.

When the packet data are ready to be transferred to the Label Removing module, the down counter loads the packet length value at the output of the pkt_length_fifo at the rising edge of the output indicator, sop, and starts counting down. When the down counter reaches the value of 1, it generates the end-of-packet indicator to indicate that there are no more data of the current packet to be transferred.

The packet data can be output correctly only with a correct packet length. In the case that the length of the packet has to be calculated by the Receive Buffers module itself, the incoming packet cannot be transferred to the next module before the complete packet has been buffered in the Receive Buffers module. When there is less than 1 packet in the receive FIFO, which is the case that the FIFO may be empty indeed or contains only a part of the packet, an active high signal indicating that the FIFO is empty will be driven high. No read is allowed to execute on such an “empty” FIFO.

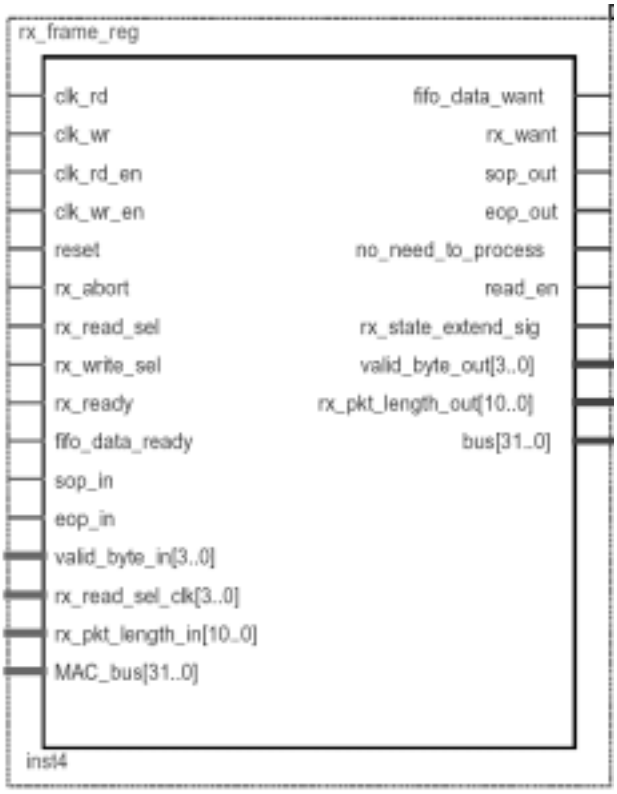


Figure 4-6 Single Receive Buffer Block Symbol, Prototype name: rx_frame_reg

This prevents the packet length calculation procedure from being interrupted and being resulting in a wrong packet length. In addition, for the purposes of testing and

system maintenance, the Receive Buffers module also provides the numbers of packets currently buffered at each receive FIFO at any time.

Figure 4-6 on the last page indicates the block symbol of a single receive FIFO. There are two clocks adopted for this module; *clk_rd* is 9 times faster than *clk_wr*. This conforms to the fact that there are two state machines controlling the reads and writes performed on the FIFOs respectively. When the two signals, *rx_read_sel and rx_write_sel* sent from State_Machine_1 and 3 respectively, stay high and if other related active high signals are also high, the module is enabled for reads/writes to be performed. When the *rx_ready* and *read_sel* both stay high, data can be transferred to either the Label Removing module or to the transmit FIFO correspondingly in different data processing phases, so long as there are data waiting in the FIFO. When the *fifo_data_ready* from the MAC side stays high and the *write_sel* is high, data transfer from the MAC to the receive FIFO is performed. If the *rx_abort* signal from the MAC side stays high, then the current packet being transferred is supposed to be dropped by the MPLS functional block later. The active high *sop_in* and the *eop_in* signals indicate the start and the end positions of the packet being transferred from the MAC to the receive FIFO. The *read_sel_clk* provides the receive FIFO with necessary timing information when reads are performed. The r*x_pkt_length_out* is the packet length sent from the MAC Chip. MAC_bus is for received data from the MAC chip to be buffered at the receive FIFO while *valid_byte_in* indicates the validity of each byte. When the *rx_want* signal stays high, the Label Removing module knows that the Receive Buffers module has at least one packet received and data processing is required. When the *fifo_data_want* signal stays high, the MAC chip learns that now the receive FIFO has some free space to hold more data. When the *no_need_to_process* signal stays high, no further processing for the packet currently being transferred through the data bus towards the Label Removing module should be done. The *sop_out* and *eop_out* signal the Label Removing module when to start and stop accepting the packet from the Receive Buffers module. The r*x_pkt_length_out* is the packet length calculated by the Receive Buffers or received directly from the MAC Chip. After being inserted with some time delay, the *valid_byte_in* and *MAC_bus* become the *valid_byte_out* and *bus* respectively. The *valid_byte_in* and the *read_en* work together to inform other modules

about the exact time duration of the data read from the receive FIFO that should be accepted. The *read_en* signal is very important because the receive and transmit FIFOs are communicating with each other at a much faster clock speed than what the MAC chip works at. The last signal to be mentioned is *rx_state_extend_sig*. It is low active and is sent to the State_Machine_3 in the case that the Receive Buffers module is likely to lose the last 2 bytes of a packet due to state transition. Thus unnecessary idling time suffered by the receive FIFO to finish its head packet transfer can be avoided

4.2.3.3 Label Removing

The Label Removing module accepts packets from the 8 receive FIFOs (not from the local host), removes their MPLS labels and then sends the labels to the Lookup Table module. After new outgoing MPLS labels are found and bound to the packets, the Label Removing module signals the corresponding receive FIFOs to transfer the rest of the packet data to their destined transmit FIFOs. The following parts of this section will introduce the signals and functions of this module, as shown in Figure 4-7.

This module takes the responsibility to signal receive FIFOs if any more packet data for further processing can be accepted after it analyzes all the feedback information sent by the Label Binding and Switching module, the Lookup Table module and the Transmit Buffers module. If all necessary conditions are met, an *rx_ready* signal is asserted high to inform the Receive Buffers module about this. The first 32 bits of a packet is always taken as the MPLS label by the Label Removing module and thus these 32 bits are stripped off once the Label Removing module receives the start-of-packet indicator coming with the data. The *rx_ready* signal is driven low right after the label is received, telling the receive FIFO to wait until the decision is made to either forward this packet to its next hop (represented by a certain transmit FIFO) or to transfer it immediately to the upper layer for further IP header analysis. The removed label is fed into the Lookup Table module as an in-coming label item immediately after being stripped off, and a new outgoing label with the corresponding outgoing port may be found 5 clock cycles later. Then the *rx_ready* signal is asserted high again and the remaining bytes of the packet can be read from the receive FIFOs, so long as the receive FIFOs are not empty. To ensure that all the functions work correctly, there is a 1-bit

register for each of the 8 receive FIFOs to record the label removing status: label having been stripped off or not.

If the in-coming label cannot be found a match within the mappings contained in the lookup table and the local microprocessor says it is ready for packet analysis, the packet will be sent to local layer 3 to see if it should be discarded or if the local host is just the destination. This helps in implementing the penultimate hop function of MPLS and enabling the system to handle the packets with an erroneous label at the same time.

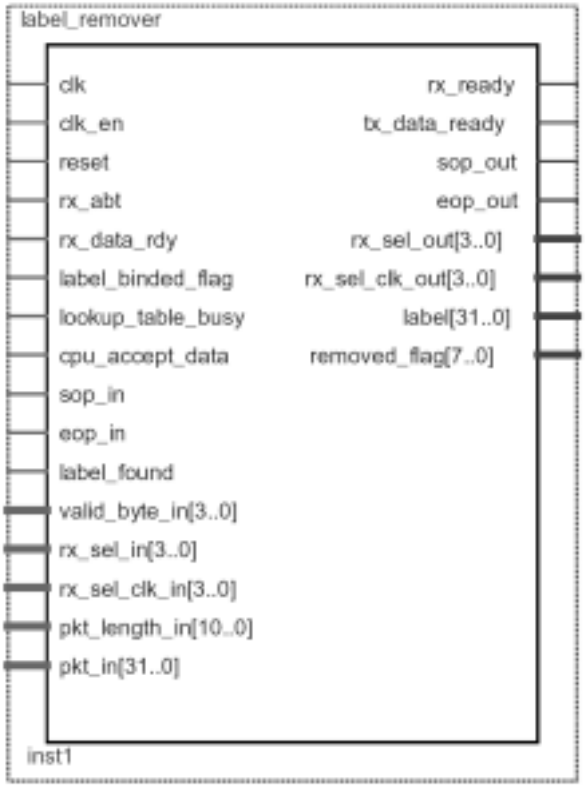


Figure 4-7 Label Removing Block Symbol, Prototype name: label_remover

Only when the *lookup_table_busy* signal from the Lookup Table module is low, can the removed MPLS label be fed into the Lookup Table module for processing. Then, if the *label_found* from the Lookup Table module becomes high after 7 clock cycles, and the *label_bound_flag* from the Label Binding and Switching module also becomes high, the output *rx_ready* will be driven high to enable directly data transfer between the Receive Buffers and the Transmit Buffers. If the *label_found* becomes low but the

cpu_accept_data is high at this time, the *rx_ready* is also driven high to enable data transfer between the receive FIFOs and the local host. Otherwise, the *rx_ready* is set to be low. The *sop_out* and *eop_out* signals are generated by the Label Removing module for the packet whose MPLS label has just been removed. The *rx_sel_out* and the *rx_sel_clk_out* are delayed *rx_sel_in* and the *rx_sel_clk_in* by one clock cycle, which are for the Lookup Table module to record the relevant outgoing port number and the memory overflow status for each incoming port. This inserted delay is to avoid system malfunctions due to a timing difference between different modules. The *tx_data_ready* is to tell the Transmit Buffers that there are packets waiting to be transmitted from the time the new outgoing labels are bound to the incoming packets. The *label* is a 32-bit wide bus used to send the removed MPLS label to the Lookup Table module. Descriptions of other signals that are straightforward to understand (either from their names directly or from previous introduction to signals of similar functions) are omitted.

4.2.3.4 Lookup Table

i) CAM Technique

For most memory devices, data storage and retrieval are done through specific memory location addressing. With conventional indexing schemes, the data content is used with a hash or index to produce the address location of the data. The address has no real or direct relationship with the information contained in the data. A typical example is a system utilizing RAM or ROM, which searches through memory to locate data sequentially. However, the address indexing, or any other conventional indexing, can slow system performance since the search may require many clock cycles to complete.

With content-addressable memory (CAM), the data is its own key, which differentiates CAM from a traditional index. The time required to find an item stored in memory can be considerably reduced by identifying stored data by content, rather than by its address. This type of distributed memory has the advantage of allowing greater flexibility of recall and is more robust. It is able to work its way around errors by reconstructing information that may have been damaged from the system.

In this project, Content Addressable Memory (CAM) is adopted together with traditional RAM technology to build the MPLS LFIB in hardware. LIB is still left for software implementation. Mappings from incoming MPLS labels to local MPLS labels and from IP headers to local MPLS labels are both taken into consideration since the design target is for an edge router. For an ordinary label switching router inside a MPLS cloud, mapping between IP headers and local MPLS labels is not necessary.

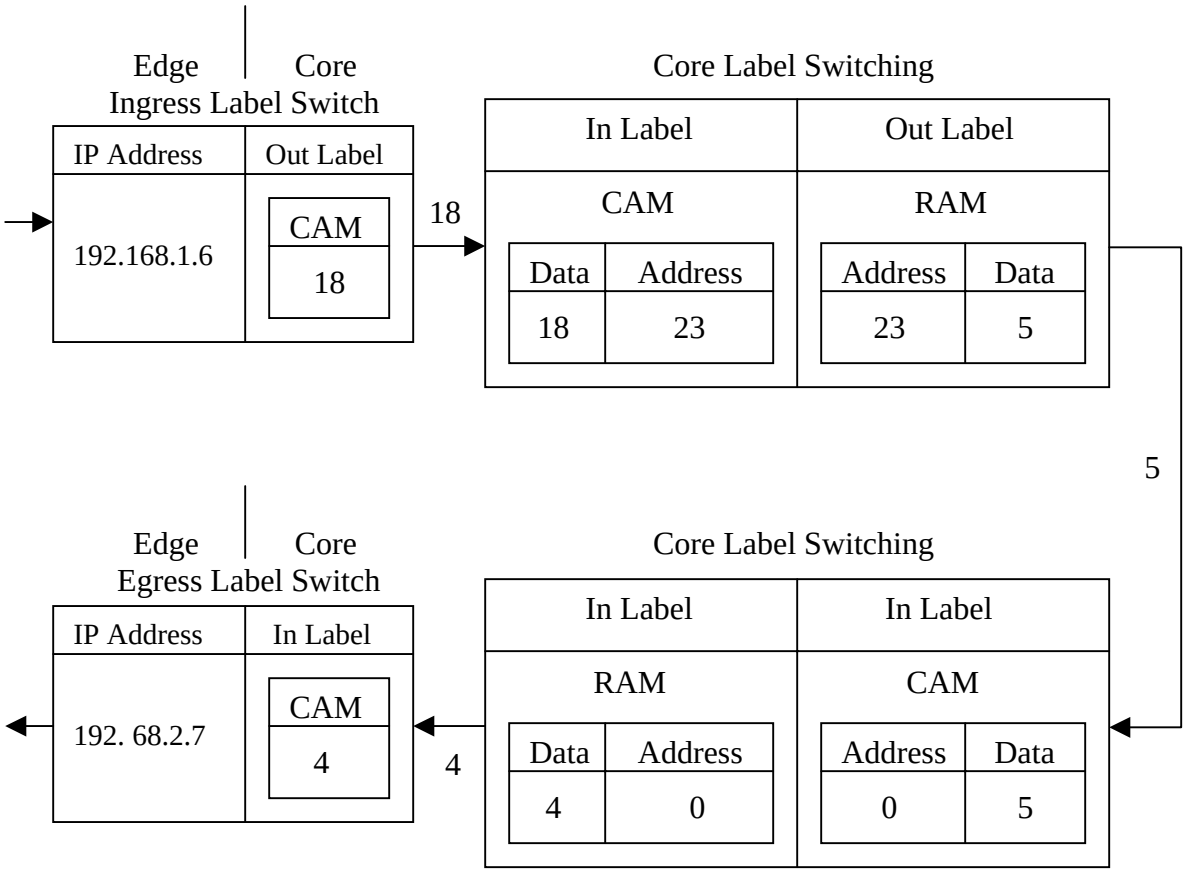


Figure 4-8 CAM and RAM Combination for MPLS

As shown in Figure 4-8, a combination of CAM and RAM can be used to implement the MPLS lookup table. The incoming label is used as an index by the CAM block to specify the next hop and the appropriate new label in the ingress label switch. Then the packet is forwarded to its next hop with the new label attached. At the last edge of the network or egress label switch section, a CAM block can again efficiently implement the table to find its corresponding IP address for the label from the incoming packet and then forward the packet using IP forwarding.

Typical multi-protocol label routers store up to 1,024 items at a time, requiring a 1,024×32 CAM block. This CAM block requires only 32 embedded system blocks (ESBs) and can be efficiently implemented within an FPGA device. The outgoing labels are stored in the RAM, which consists of 1,024×32 bit locations consuming 16 ESBs.

ii) Practical Lookup Table implementation

To avoid that packets from different incoming ports directed to the same outgoing port are buffered at one transmit FIFO in an interleaved way, there are dedicated registers recording the status of each transmit FIFO: whether the transmit FIFO is in the middle of accepting a packet from the receive FIFO *i* at present. If it is, then the head packet at some other receive FIFO *j* also destined for it is asked to wait until the status register shows that the last packet has been completely written into the transmit FIFO already. There are two ways to handle the header packets from receive FIFOs other than FIFO *i* but destined to the same transmit FIFO *j*. The first method is: right after the transmit FIFO is found to be busy, the outgoing label assigned to the header packet at receive FIFO *j* is sent to the local microprocessor, where it is allocated some memory space of the external RAM for temporary storage. After a proper waiting time, this packet will be transmitted in the normal way as if it were originated from the local host. The other way to handle this issue is to allocate dedicated on-chip buffer space to hold the outgoing labels found for the packets from FIFOs other than the receive FIFO *i* within the same FPGA chip. Since each MPLS label is just 32 bits, it does not cost much to store a number of such labels in on chip memory. However, to make sure that theoretically no packet loss due to buffer space overflow takes place, the two methods described above are adapted to work together. When there is contention at some transmit FIFO, the system will send to the host microprocessor the outgoing labels of the head packets from the receive FIFOs other than receive FIFO *i* in the case their corresponding on-chip memory is experiencing overflow. At the same time, the system finishes buffering the head packet from the receive FIFO *i* to the transmit FIFO where contention is taking place as soon as possible.

Here the lookup table is made up of three CAM and one RAM, whose architecture and behaviors were described in the last section. For ordinary LSRs that work within an

MPLS domain, only the mapping between incoming labels and out-going labels is needed. In order to implement the layer 2 switching, the table containing the mapping between local MPLS labels and physical outgoing ports has to be included as well. Since this project is focused on edge router design, an extra table doing mapping between the IP header and the MPLS label is included in the architecture of the lookup table, too.

Usually an entire packet cannot be transferred completely within one service interval and data will not know where to go when the next service interval arrives if no outgoing port information is available. Therefore the outgoing port information needs to be saved. To handle this, a dedicated set of status registers is adopted within the Lookup Table module. This set contains nine 3-bit wide registers, which record the outgoing ports for the head packets of the 8 receive FIFOs and the local microprocessor under service. These registers are cleared once the corresponding packets have left their receive FIFOs completely. Since currently LSP *merging* is not considered, the case of output port contention taken place among several input ports is neglected.

As shown in Figure 4-9, the *sop*, the *eop*, the *rx_sel* and the *rx_sel_counter* signals are used to set and clear all status registers recording necessary packet information. The *wrdelete*, the *wren*, the *wraddr* and the *update_data* are for lookup table content updates. The *label_in* carries the incoming MPLS label from the Label Removing module. The *IP_header* carries the IP header of the packet from the host microprocessor. The output signals *label_out* and the *fifo_sel* provide the new MPLS label to be bound to the packet and the outgoing port number indicating where should the packet be switched. The signal *extend_rx_state_rq* is sent to State_Machine_3 when the label searching task cannot be finished within one service cycle, thus the service time can be extended as needed.

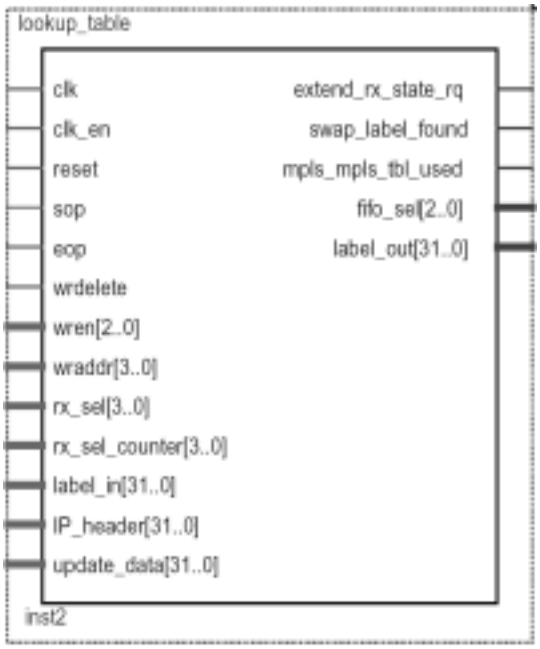


Figure 4-9 Lookup Table Block Symbol, Prototype name: lookup_table

4.2.3.5 Label Binding and Switching

With the outgoing MPLS label and outgoing ports provided by the Lookup Table module, label binding and switching can be performed now.

As indicated in Figure 4-10, if the *tx_data_want* from the Transmit Buffers module is high, the binding task to be done can be completed with two steps. Firstly, once the *label_removed_flag* and the *fifo_rdy* both become high, the Label Binding and Switching module outputs the label as the first 4 bytes of the packet data to be switched to the *pkt_out* port. This behavior accomplishes the function of "label binding". Meanwhile, the *tx_sop_out* and the *label_bound_flag* are set high to indicate this completion. The former one is just a pulse with the width the same as that of the *data_valid*, while the latter one has to always stay high until a pulse of *tx_eop_in* appears. Then as the second step, after the label is bound, the Label Binding and Switching module directs the remaining part of the incoming packet data to the *pkt_out* port. Along with the *pkt_out* data, the *tx_reg_sel_out* is sent to the Transmit Buffers module to identify the destination transmit FIFO for the currently being transferred packet. The *cpu_data_rdy* and *rx_data_rdy* help in generating the *tx_data_ready*, which

is set to be high so long as there is data required to be sent to the Transmit Buffers module, regardless of whether it is from the local host or from one of the 8 receive FIFOs. Also, there are 9 registers to record whether the label has been bound or not for the header packet at each receive FIFO. Description of other signals of this module is omitted, because their functions are apparent from their names.

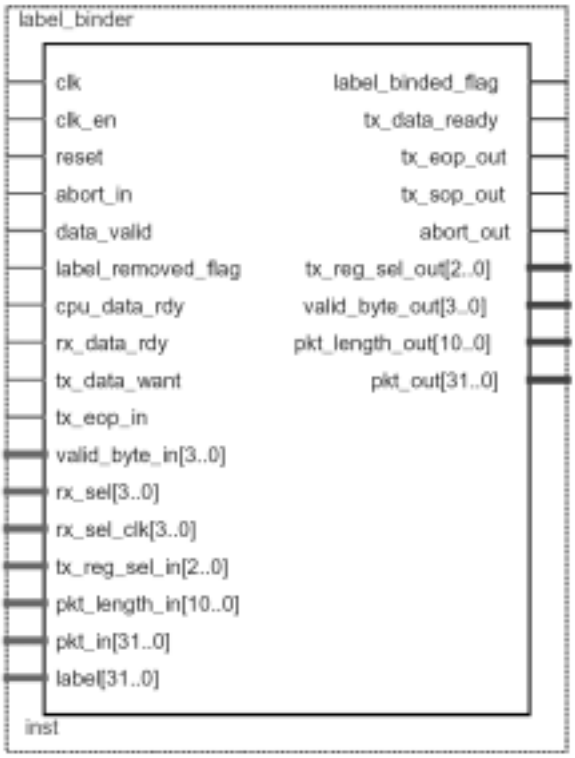


Figure 4-10 Label Binding and Switching Block Symbol, Prototype name: label_binder

4.2.3.6 Transmit Buffers

As shown in Figure 4-11, this module is very similar to the Receive Buffers module, but it is simpler since the packet length is never computed in this module. Another difference is that for the packet length FIFO, the packet length is written when the *sop_in* is high, instead of when eop_in is high as in the Receive Buffers module.

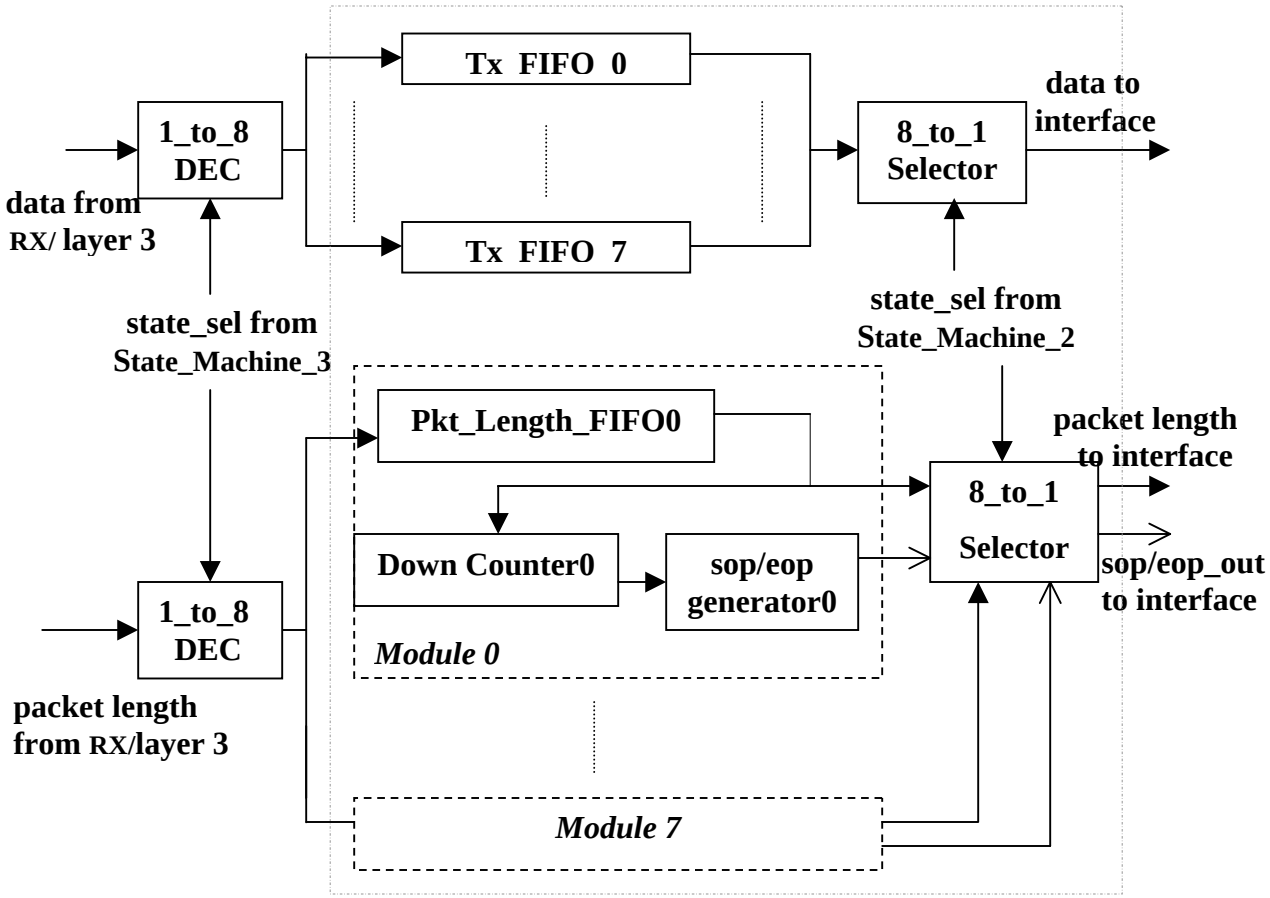


Figure 4-11 Transmit Buffers Block Diagram

Figure 4-12 is the block symbol of a single transmit FIFO. There are also two clocks adopted for this module. The *clk_rd* is 8 times faster than the *clk_wr*. When the two signals, the *tx_read_sel* and the *tx_write_sel* generated by the State_Machine_2 and 3 respectively, stay high, the module is enabled for reads/writes to be performed. When the *MAC_ready* and the tx_*read_sel* both stay high, data in the buffer can be transferred to the MAC chip through the MPLS_MAC interface module. When the *tx_data_ready* from the Label Binding and Switching module stays high and the *tx_write_sel* is also high, data transfer from the Receive Buffers module or the local host to the Transmit Buffers module is performed. The MPLS_MAC interface module knows that the Transmit Buffers module has data to transmit when the *tx_want* signal stays high. When the *tx_reg_ready* signal stays high, the Transmit Buffers module indicates that now it has some free space to hold more packets that have already been bound with a new label. The *sop_out* and the *eop_out* signals tell the MPLS_MAC interface about the start and

the end positions of the current packet. The tx_state_extend_sig is low active and is sent to the State_Machine_2 to prevent from happening the case that the last 2 bytes of the packet get lost due to state transition during the course of data transfer from the Transmit Buffers to the MAC-MPLS interface.

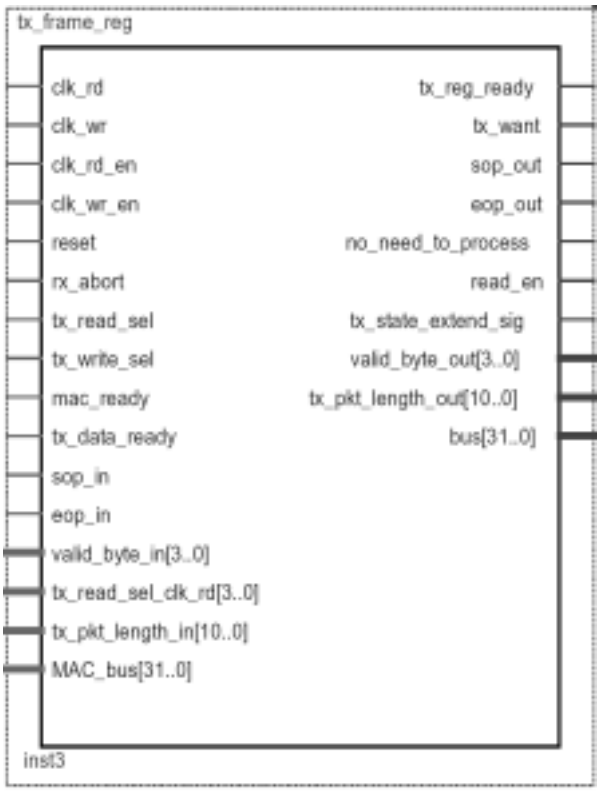


Figure 4-12 Single Transmit Buffer Block Symbol, Prototype name: tx_frame_reg

Chapter 5 Test Development and Procedure

5.1 Introduction

It is ideal if the whole design presented in the previous chapter be tested in a real MPLS network, which means several completely finished MPLS switches/routers would have to be built for the tests to be carried out. However, such a task involves work over all layers of the TCP/IP model and is beyond the scope of this project. Since the project is focused on digital hardware circuit design, it is sufficient to demonstrate that the MPLS functional block implemented within an actual FPGA device can perform MPLS label binding and removing according to requirements set in advance and can realize packet transmission and reception over the physical layer by directing incoming packets to their outgoing ports correctly. Real layer 3 routing is not considered in the tests of this project. It is assumed that all necessary LSPs have been set up successfully already and that the only remaining task is label switching. Therefore, high-level software programming for the FEC definition and the LDP is not needed in this test.

With the simplified testing methodology, an Ethernet Development Kit (EDK) from Altera Corporation can be utilized to build the test bed. The most important hardware component included in the EDK is a network-interface daughter card containing the MAC chip CS8900A, which can be plugged directly into the motherboard of the development kit. Though this EDK is made up of both hardware and software components that provide network connectivity and operation utilities for a Nios-based embedded systems, only the hardware components will be introduced in section 5.3 since software utilities running on an embedded Nios microprocessor are not used in this project.

The MPLS functional block is designed to support 8 sets of integrated FIFOs, and each set corresponds to a certain physical port. However, the MAC chip CS8900A provided on the board only supports one physical port. This problem has to be

considered before the EDK can be put into use since it must be made certain that the tests being done under such a situation can still be meaningful. To surmount this problem without imposing more requirements on the testing environment, it can be assumed that the other 7 MAC chips do exist but currently have no data to transmit or receive, and thus by driving relevant signals inactive in a normal working mode, the service intervals granted to the 7 fake physical ports can be saved by the system once these relevant signals are found to be inactive. With this assumption, the goal of the tests can still be reached with only one MAC port available for a node. The normal operation of the system with multi-port integration can still be demonstrated. A detailed explanation of the test procedure is given in section 5.4.1.

Yet another problem exists. The CS8900A is designed to communicate directly with a microprocessor instead of other hardware circuits; while in this project, the MPLS must be interposed between the CS8900A and the microprocessor, which means that the MPLS functional block is required to take the place of the microprocessor in communicating with the CS8900A. To handle this, a special interface has to be designed to aid packet transfer between the CS8900A and the MPLS functional block (in another words, between the MAC layer and the MPLS shim layer).

In the following sections, general test methodology development is presented first; then the main hardware equipment used to build the test bed is introduced; in the third section, a detailed description of the interface design for the MPLS block to cooperate with the CS8900A MAC chip is depicted; finally, the detailed procedure of the practical tests is described.

5.2 Test Methodology Development

In the real world, the MPLS network can be arbitrarily large, consisting of parts that are separated by considerable physical distance from each other and are connected with each other via links (usually of bit-serial nature) like coaxial cables, optical fibers, microwave links, etc. One part of the network, which resides at one physical location, may be as small as a few chips on a small printed-circuit board or as large as thousands of chips on many boards in several boxes all located physically close to each other.

Naturally ideal tests are supposed to be taken over such a real MPLS network, where the edge nodes interface different types of physical mediums. However, such a perfect condition is not truly necessary when the purpose of the tests is only to show how an edge LSR functions, and the test methodology can be simplified as illustrated in the following subsections without affecting the desired results.

Another concern is the FPGA capacity. In today's market, there are FPGAs with millions of gates and over 10MB RAM space, which are very suitable for on-chip switch/router design. However, the FPGA device available for this project is limited in EBS blocks, which makes it impossible to fit in the complete integrated 8-port design. Also, due to the limited number of available CS8900A chips representing the number of physical ports (one CS8900A can only talk to one 10BaseT physical port), only one set of receive FIFO, transmit FIFO and interface module really consumes the hardware resources within one FPGA device in the tests. However, the service scheduler still takes the other 7 ports as existing conceptually and this one port implementation still can demonstrate the performance of the 8-port integrated design. The detailed reasons will be given in Chapter 6.

After the top module consisting of the MPLS functional block and the interface between MAC and MPLS was fully compiled, a programming file was generated by QuartusII (A digital circuit design software tool provided by Altera corporation) and then loaded into the APEXII FPGA device mounted on the mother board of the EDK through a download cable named ByteBlasterMV. Or, the programming file can be stored in the FLASH memory incorporated on the mother board and be loaded automatically into the FPGA device at each reset or power-up. The APEXII FPGA device and the CS8900A mounted on the daughter card make up the essential hardware part of an MPLS edge node operating over Ethernet, as illustrated in Figure 5-1.

Because it makes no difference if the packet simply travels through a line or across several internal networks before it arrives at its destination, the test bed can be built simply with two sets of the EDK (acting as a simplified MPLS LER and a LSR respectively), an ordinary desktop computer, and a hub, as shown in Fig 5-2.

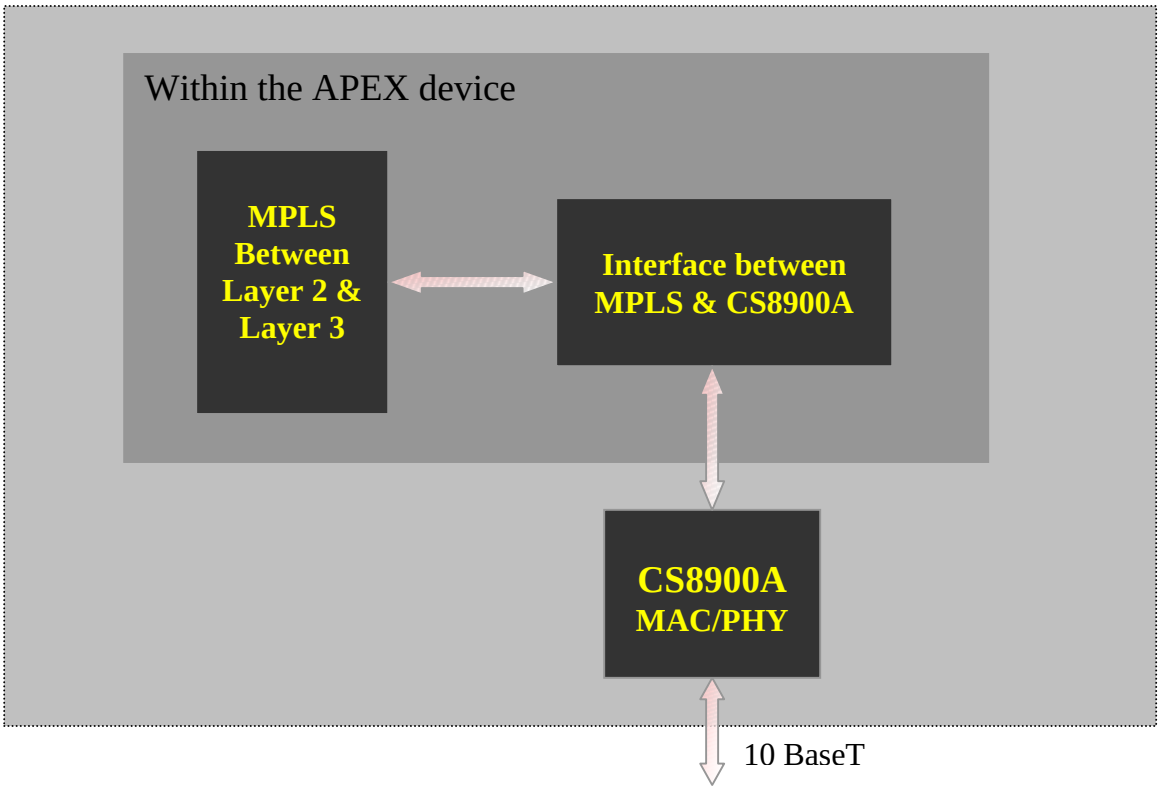


Figure 5-1 MPLS Edge Node Hardware Architecture

Although in this project, there is no real routing occurring over layer 3 and only label switching is concerned, it is desirable that the CS8900A grasps Ethernet traffic with individual destination MAC address successfully to demonstrate real communication at layer 2. It is well known that within a LAN each node should have a unique MAC address to ensure that there is no confusion for packet reception. Since there is no universally unique MAC address assigned to each CS8900A chip when it is shipped, the CS8900A has to be configured with an MAC address unique within the scope of the LAN into which it is to be plugged. This simplified test bed architecture is still capable of demonstrating the performance of the hardware-realized part of an MPLS edge node.

According to the design introduced in Chapter 4, the Receive Buffers module and Transmit Buffers module communicate with each other through the Label Removing module and Label Binding and Switching module, at a clock speed 9 times faster than

the clock speed at which the Receive/Transmit Buffers communicate with the MAC chip through the MPLS_MAC Interface module. However, since the tests are only done between two physical nodes, where there is no real service multiplexing happening, only one system clock is applied for the whole FPGA system.

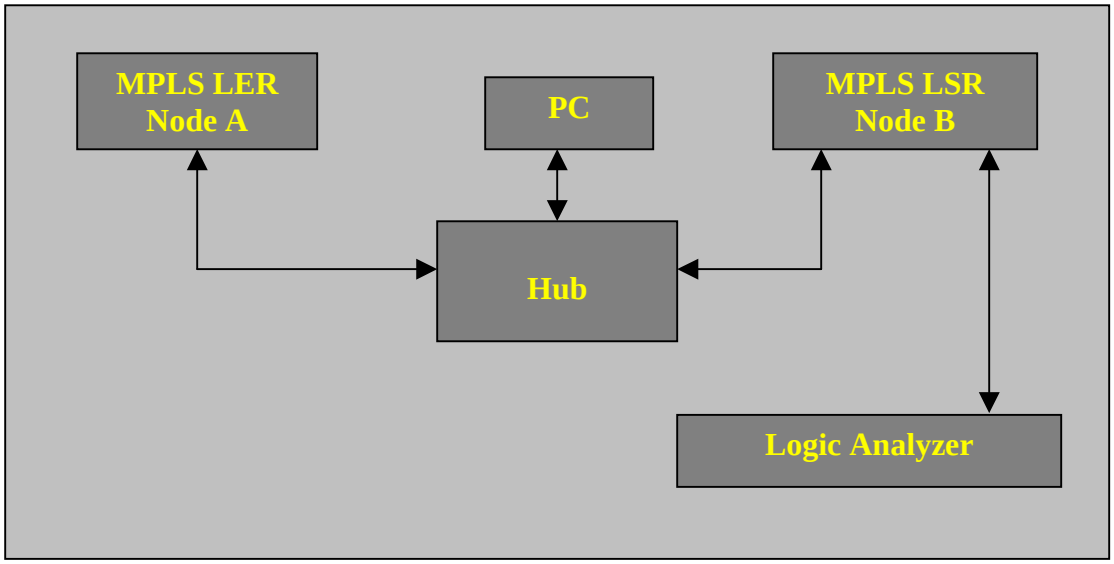


Figure 5-2 Test Bed Architecture

After the APEXII FPGA devices on the two boards are programmed, Node A and Node B have been built and can then operate independently. They are plugged into the small LAN each with a unique MAC address. Currently there is no need to create any mapping between IP address and MAC address due to the lack of higher layer communication. Neither is the networking setting needed for now. This part of the task is only desirable in future work. Label switching is the only thing that needs to be checked here. So long as it can be seen at one node that an incoming packet with label A is transmitted onto the Ethernet again with a new label B, as expected, the tests are said to be successful.

5.3 The Test Equipment

5.3.1 The Motherboard

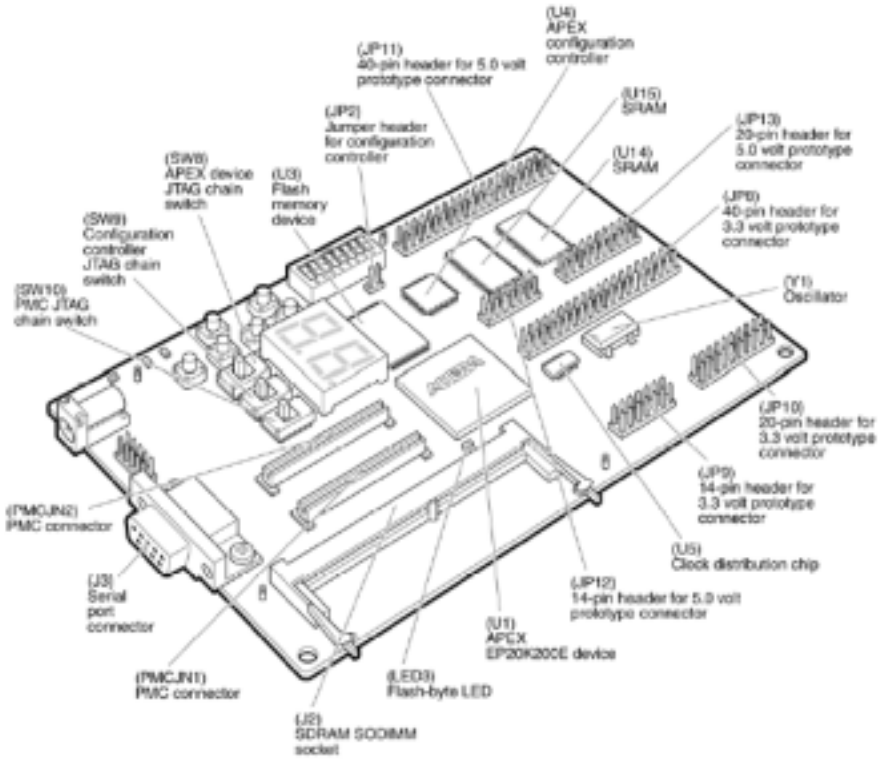


Figure 5-3 The Mother Board

The motherboard of the development kits is a board that features an APEX™ 20K200EFC484-2x device; 1 Mbytes (512 K x 16-bit) of flash memory; 256 Kbytes of SRAM (in two 64 K x 16-bit chips);on-board logic for configuring the APEX device from flash memory, etc. The APEX 20K200E device is in a 484-pin FineLine BGA™ package. It has 8,320 Logic Elements, 52 ESBs, and 106,496 RAM bits.

The 1 Mbytes flash memory chip is an Advanced Micro Devices (AMD) AM29LV800BB. It is connected to the APEX device so that it can be used for two purposes. Firstly, the flash memory can be used as general-purpose readable memory and non-volatile storage by the Nios processor implemented on the APEX device. Secondly, the flash memory can hold an APEX device configuration file that is used by the configuration controller to load the APEX device at power-up. For this project the flash memory is only used for the latter purpose.

5.3.2 The Daughter Card

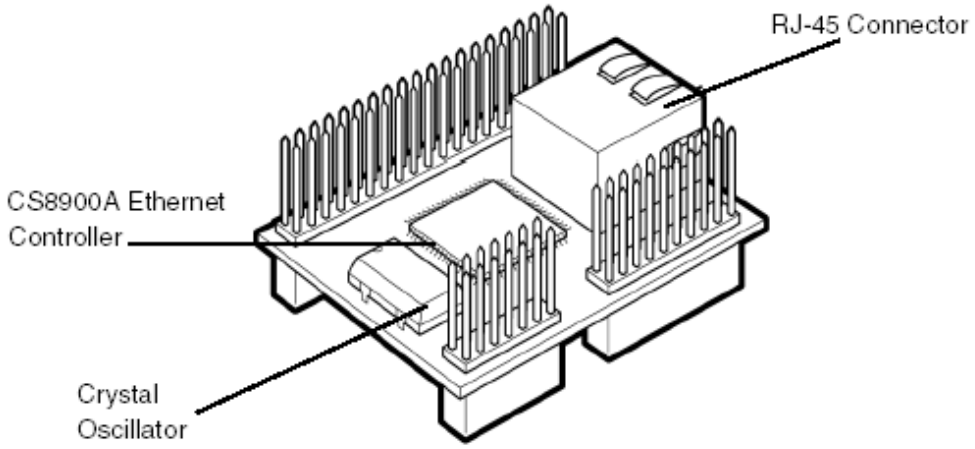


Figure 5-4 The Daughter Card

An EDK daughter card works fine with the motherboard and only one daughter card will be used along with one development kit in the test. However, a motherboard supports at most two EDK daughter cards that form a two-level daughter card stack. As illustrated in the figure above, the daughter card is a circuit board with the following components:

- A Cirrus Logic CS8900A integrated Ethernet 10 Mbit PHY/MAC chip
- A RJ-45 network connector with integrated transformer magnetic and Link/LAN LEDs
- Three female connectors to mount the daughter card on the Nios development board
- Three male headers for stacking two daughter cards
- A 20 MHz crystal oscillator that is used by the CS8900A chip
- All necessary resistors and capacitors

The EDK includes an SOPC Builder library component that provides all logic and I/O signals necessary for using the daughter card as the peripheral of an embedded RISC CPU. However, currently this is not used since in this case the MAC/PHY chip does not talk with the host CPU, but with the MPLS functional block through some hardware glue circuitry.

5.3.3 Introduction to the CS8900A

In the tests, only one EDK daughter card that located at the lower level of the stack is used. The main functional component on this daughter card is a CS8900A integrated PHY/MAC chip. The CS8900A chip presents an ISA-bus interface to the host CPU (here the MPLS functional block). The necessary electrical-interface signals are provided on the set of female connectors. These connectors are compatible with the *expansion prototype connector* groups on the motherboard. In this project, the daughter card is connected to the 3.3-V *expansion prototyped connector* group.

5.3.3.1 CS8900A Work Mode

The CS8900A is a single-port Ethernet solution incorporating all of the analog and digital circuitry needed for a complete Ethernet circuit. It mainly includes: a direct ISA-bus interface, an 802.3 MAC engine, integrated buffer memory, and a complete analog front end with 10BASE-T.

The CS8900A can work in both memory mode and I/O mode and the latter is the default mode. According to the way the Ethernet daughter card is connected to the motherboard, I/O mode is adopted for the tests. In this mode, the on-chip memory space of the CS8900A can be accessed through eight 16-bit I/O ports that are mapped into sixteen contiguous I/O locations in the host system's I/O space. Therefore the interface only needs to have a 4-bit wide address bus and a16-bit wide data bus. However, since all registers are accessed as words only, the least significant bit of the address can be always tied to low. The CS8900A I/O mode mapping is shown as Table 5-1.

Receive/Transmit Data Ports 0 and 1 are used when transferring 32-bit transmit data to the CS8900A and 32-bit received data from the CS8900A Real traffic carrying information in practice is not concerned here. For fake MPLS traffic assumed to run between the CS8900A and the MPLS functional block, simple 16-bit MPLS labels can be used in the test. Therefore, though the MPLS functional block is designed for 32-bit traffic, it makes no difference if the higher 16-bit data are always assigned 0. Finally, because the CS8900A is designed optimally to work in 16-bit mode, the CS8900A is set to do 16-bit operations and thus only Port 0 is needed.

Table 5-1 CS8900A I/O Port Descriptions

Offset	Type	Description
0000h	Read/Write	Receive/Transmit Data (Port 0)
0002h	Read/Write	Receive/Transmit Data (Port 1)
0004h	Write-only	TxCMD (Transmit Command)
00006h	Write-only	TxLength (Transmit Length)
00008h	Read-only	Interrupt Status Queue
000Ah	Read/Write	MAC_RAM Pointer
000Ch	Read/Write	MAC_RAM Data (Port 0)
000Eh	Read/Write	MAC_RAM Data (Port 1)

It is through the MAC_RAM Pointer Port and MAC_RAM Data Port that the MPLS hardware can access the internal registers of the CS8900A in I/O Mode. Whenever such an access is needed, the MAC_RAM Pointer has to be setup first by writing the MAC ram address of the target register to the MAC_RAM Pointer Port (I/O base + 0001Ah). Among the 16 bits written to the pointer port, the first 12 bits (bits 0 through B) provide the internal address of the target register to be accessed during the current operation; the next three bits (C, D and E) are read-only and will always read as 011b, thus any convenient value may be written to these bits; the last bit (Bit F) indicates whether or not the MAC_RAM Pointer should be auto-incremented to the next word location. The contents of the target register are then mapped into the MAC_RAM Data Port (I/O base + 000Ch). In most cases, MAC_RAM Data Port 1 is not used in this test, since most internal registers are just 16 bits wide.

For faster access, the internal Tx Command Register at MAC_RAM base + 0144h is mapped to TxCMD Port and the internal Tx Length Register at MAC_RAM base + 0146h is mapped to TxLength Port. These mappings save the write needed to setup the MAC_RAM pointer for each normal internal register access. The interrupt Status Queue Port is not used in the tests since polling, instead of interrupts, is adopted to control the CS8900A.

5.3.3.2 CS8900A Configuration

Before any packet transmission and reception are possible, the CS8900A must be configured properly. Various configuration parameters have to be determined, such as I/O Base Address, Ethernet Physical Address, what frame types to receive, and which media interface to use. Usually this is done at power-up or software/hardware reset. All the parameters are fed into the internal *configuration* and *control registers*, which are an integrated part of CS8900A on-chip memory. Specific configuration parameters selected to carry out the real test are illustrated in section 5.5.

5.4 Interface design

There is currently no microprocessor involved, so the CS8900A is controlled by the MPLS hardware through a dedicated interface circuit.

5.4.1 Functions to Be Performed

The Ethernet frame header components, Destination MAC address, Source MAC address, Type/Length field, Payload, Pad and CRC are supposed to be provided before the packet can be sent to the MAC chip. Also, after being captured from the network side, the complete MAC frame is sent out by the CS8900A, without having the DA, SA and type/length fields removed. As described in Chapter 2, the MPLS label has to be inserted between the MAC header and the Layer 3 header. Only after the MAC header is stripped off, can the MPLS functional block begin processing the incoming packet. Meanwhile, only after the MAC header indicating the next hop is appended in front of the outgoing MPLS label, can the frame be sent to the CS8900A for transmission. Therefore, it is the task of the interface circuit to strip off the entire MAC header before transferring the received frame to the MPLS hardware and to encapsulate the layer 3 packet before feeding it to the CS8900A. Dedicated registers are provided to hold the removed MAC header and packet type/length information that can be accessed by the MPLS hardware before the packet is finished processing.

An oscillator on the daughter card provides the CS8900A with a system clock of 20 MHz, while the oscillator on the motherboard provides the APEXII FPGA device with a

system clock of 33 MHz. Asynchronous communications is required between the two devices.

Originally, the MPLS hardware was supposed to interface an Intel MAC chip IXF440, which only provides signals indicating the start and end of the packet instead of the packet length when outputting received packets, and requires the same signals from other circuitry while accepting packets to be transmitted. Due to some constraints on equipment availability, the Intel IXF440 was abandoned after the MPLS hardware design had been almost finished. Instead, the project used the Cirrus CS8900A chip for the tests later. In order to make the least modification of the MPLS function design, signals that are exactly the same as those from Intel IXF440 are need to be generated by the interface. Such packet delimiter signals also aid in some flag setting and clearing used by the Label Removing module, Table Lookup module and Label Binding and Switching module within the MPLS functional block.

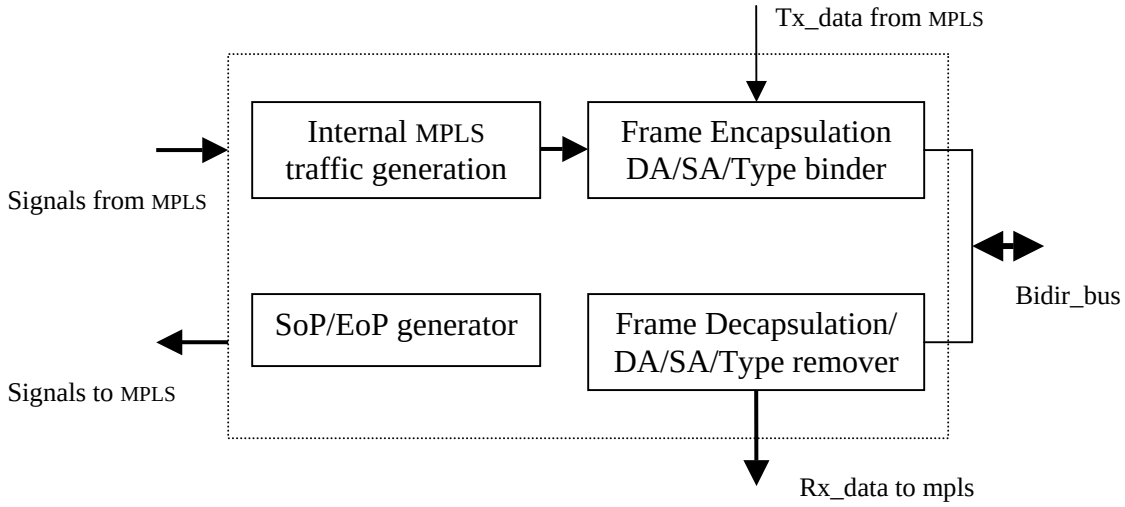


Figure 5-5 Block Diagram of the Interface Between MPLS and MAC

Since no higher-level software design for MPLS is done, it is impossible to have MPLS traffic generated by an ordinary desktop computer. A straightforward solution is to provide an internal MPLS packet generator within the interface module. This generator can be used in LER Node A to produce the initial MPLS traffic. Figure 5-5 indicates the basic diagram block of the interface functions introduced in above paragraphs.

5.4.2 Flow Chart of Interface Functions

Please refer to Figure 5-6. Since there is only one set of 16-bit bi-directional I/O pins for data transfer, the CS8900A chip can only transmit and receive packets alternatively rather than in parallel. The interface has to poll between the two states, transmit or receive, to decide what to do next. The default state after each reset is reception. In polling mode, the RxEvent register of the CS8900A at MAC_RAM base + 0124h is checked repetitively until the bits indicating a complete packet reception are set. Then the RxStatus register at MAC_RAM base + 0400h and the RxLength register at MAC_RAM base + 0402h are read. Actually the RxStatus register contains the same value as that of the RxEvent register, and the CS8900A data sheet says that the former can be skipped if the latter has been read. However, in order to make sure that the receive buffer of the CS8900A can be released completely, the RxStatus register is always read. The number of reads needed to fetch the data of the whole frame can be calculated in the interface after the RxLength register is read. Then repetitive reads are performed by the interface to retrieve data from the receive frame location of the on chip memory of the CS8900A.

After the last byte of data is received, the interface can transit to the transmission state. If no packet has been transmitted yet since the last reset, the interface issues a transmit command directly to bid for buffer space of the CS8900A for the transmit frame data to be held. Otherwise, before the transmit command can be issued and the transmission state is timeout, the bits of the TxEvent register at MAC_RAM base + 0128h are continuously monitored until the last packet has been transmitted by the CS8900A.

As part of a complete transmit command, the length of the packet is written to the TxLength port that is mapped to the TxLength register at MAC_RAM base + 0146h, immediately after the transmit command word is written into the TxCMD port that is mapped to the TxCMD register at MAC_RAM base + 0144h. After that the interface starts polling the BusStatus register at MAC_RAM base + 0138h to see if the bid is successful. If not, the interface issues the transmit command again; if yes, repetitive writes are performed to transfer the transmit data from the MPLS hardware to the

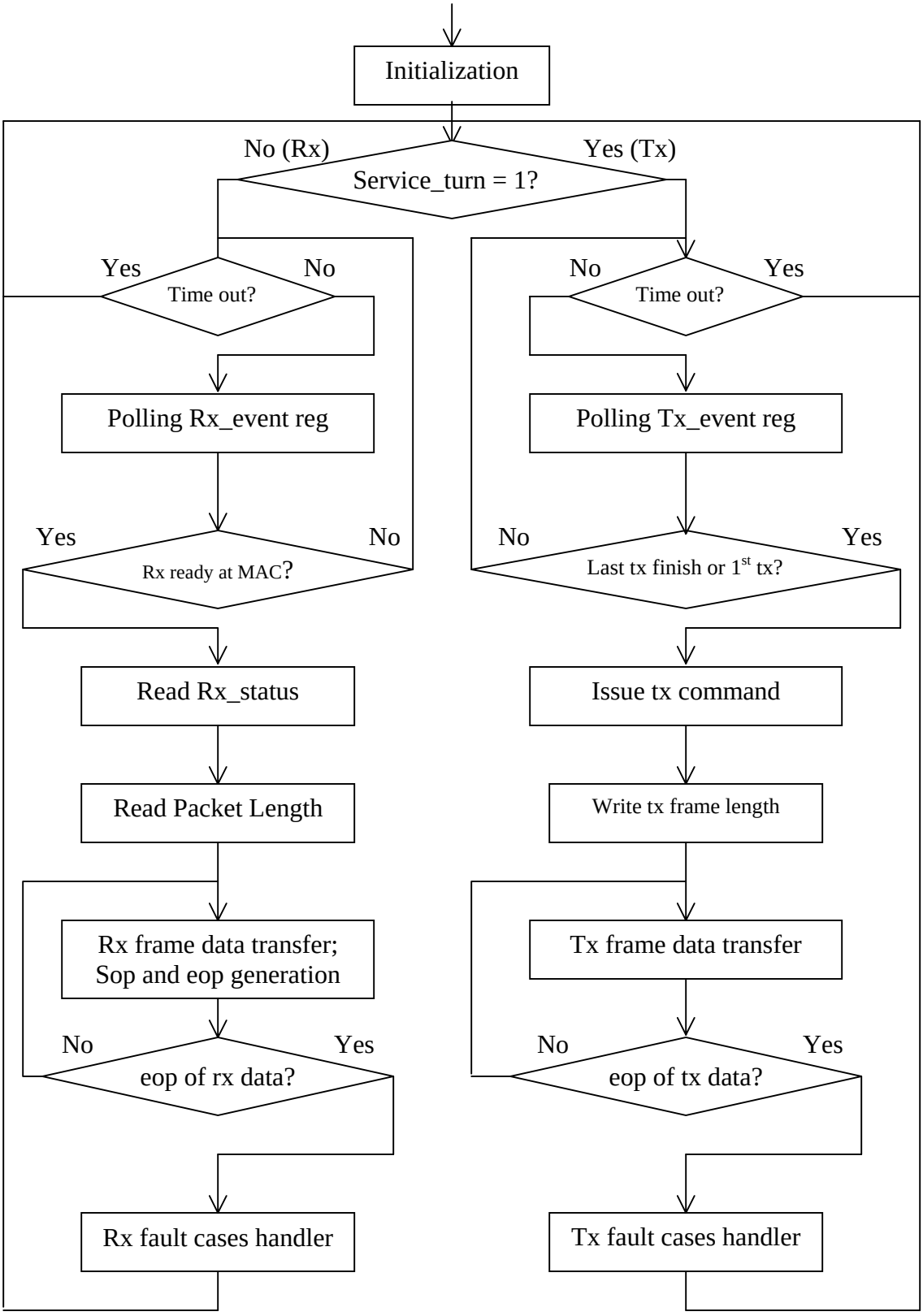


Figure 5-6 Flow Chart of Interface Functions

CS8900A. After the last byte of data is transferred to the transmit frame location of the CS8900A, the interface can enter the receive state again.

There are two situations requiring for special care. The first is that, after the transmit command and transmit packet length are written to the CS8900A, the CS8900A has to take some time to find out if it is now able to do the job. If the state changes too quickly, the packet to be transmitted has to wait until its next turn and the bid for transmit buffer space on the CS8900A has to be done all over again. Similarly, sometimes the MPLS functional block may not be ready at the beginning of the service turn for the reception state. If the state changes too quickly, the packet already waiting in the CS8900A has to wait until its next turn, too. Therefore, the system is designed to only leave the current state and enter the other one after some predefined time of waiting, which is set to be 8 clock cycles in the tests.

The second situation is that, during transmission there may be collisions on the Ethernet or something wrong taking place physically at the 10Base-T port; during reception, packets with bad CRC or illegal lengths occupying the buffer space on the CS8900A may prevent new valid packets from being received. These fault cases are irrelevant to the design and do not need to be handled right now, but they cannot be ignored, either. So by monitoring associated event bits and then setting some indicators accordingly within the CS8900A internal registers, the CS8900A can come out of such fault cases and go on with its regular operations. Therefore, the system will not be stuck in a dead cycle.

As shown in Figure 5-6, the operations that are required for the interface between the MPLS hardware and the CS8900A to perform are:

1) Power on reset and wait for the CS8900A to finish its self-initialization;
2) Configure the CS8900A with the required parameters for the tests;
3) Before timeout, check if the CS8900A has successfully received any packet: If yes, go to 4); if no go to 7). If timeout, go to 7) directly;
4) Begin reading RxStatus and RxLength registers;

5) Strip off the MAC header and calculate the number of reads to fetch the whole frame of data;
6) Start reading the I/O data port repetitively for the number of times obtained in step 5) to free the receive buffer space on the CS8900A;
7) If the CS8900A has not transmitted any packet since last reset, go to 9); If CS8900A has transmitted some packets, go to 8). If timeout, go back to 3); If none of the above happens, poll the TxEvent register to see if the last packet has been sent out successfully by the CS8900A. If successful, go to 9); if not successful, go back to 7);
8) Issue a transmit command;
9) Before timeout, poll the BusStatus register to see if the CS8900A has any buffer space available to hold the transmit packet, if it has, stay in 9); if not, go to 10);
10) Transfer the transmit packet to the CS8900A, and then go back to 3).

5.4.3 Input/Output Signal Description

Parameter	Symbol	Min	Typ	Max	Unit
16-Bit I/O Write					
Address, AEN, $\overline{SBHE}$ valid to $\overline{IOCS16}$ low	t_{IOW1}	-	-	35	ns
Address, AEN, $\overline{SBHE}$ valid to $\overline{IOW}$ low	t_{IOW2}	20	-	-	ns
$\overline{IOW}$ pulse width	t_{IOW3}	110	-	-	ns
SD hold after $\overline{IOW}$ high	t_{IOW4}	0	-	-	ns
$\overline{IOW}$ low to SD valid	t_{IOW5}	-	-	10	ns
$\overline{IOW}$ inactive to active	t_{IOW6}	35	-	-	ns
Address hold after $\overline{IOW}$ high	t_{IOW7}	0	-	-	ns

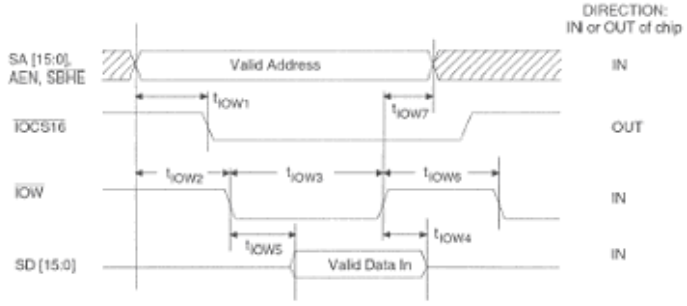


Figure 5-7 16-bit I/O Write to the CS8900A [34]

Parameter	Symbol	Min	Typ	Max	Unit
16-Bit I/O Read, IOCHRDY Not Used					
Address, AEN, $\overline{SBHE}$ active to $\overline{IOCS16}$ low	t_{IOR1}	-	-	35	ns
Address, AEN, $\overline{SBHE}$ active to $\overline{IOR}$ active	t_{IOR2}	10	-	-	ns
$\overline{IOR}$ low to SD valid	t_{IOR3}	-	-	135	ns
Address, AEN, $\overline{SBHE}$ hold after $\overline{IOR}$ inactive	t_{IOR4}	0	-	-	ns
$\overline{IOR}$ inactive to active	t_{IOR5}	35	-	-	ns
$\overline{IOR}$ inactive to SD 3-state	t_{IOR6}	-	30	-	ns

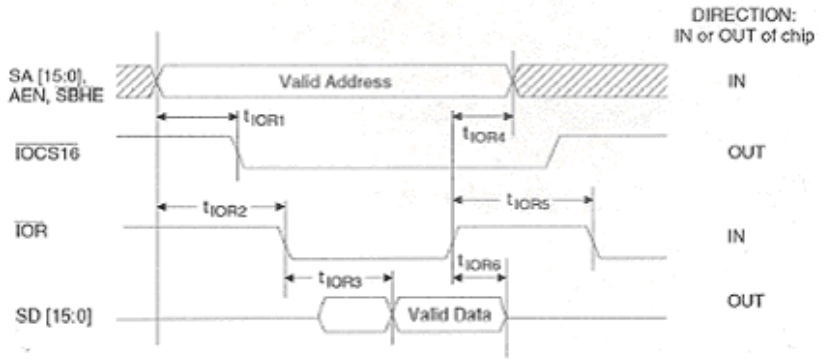


Figure 5-8 16-bit I/O Read from the CS8900A [34]

As mentioned in the last section, the CS8900A should be visited asynchronously, which can be achieved by repetitively toggling the write/read enables. As shown in Figure 5-9, the **clk** input is assigned to the pin of the APEX device that is connected to the on-board 33.33 MHz oscillator. According to Figure 5-7, the **io_w** signal is designed to stay high for 120 ns at first and then go low for another 120 ns. During the time **io_w** is high, an address pointer pointing to the targeted internal register to be accessed is set and once **io_w** goes low, the data on the bi-directional data bus can be written into the register at the targeted address. Similarly, the **io_r** is designed to meet the timing requirement of the CS8900A, according to Figure 5-8. The time from address and **sbhe** active to **io_r** active is required to be at least 10 ns. To take advantage of the circuitry used for **io_w** generation, and satisfy this requirement, **io_r** is set high for the same amount of time (120 ns) as **io_w**. However, the time **io_r** has to stay low is longer than that of **io_w**, which is 250 ns in this case.

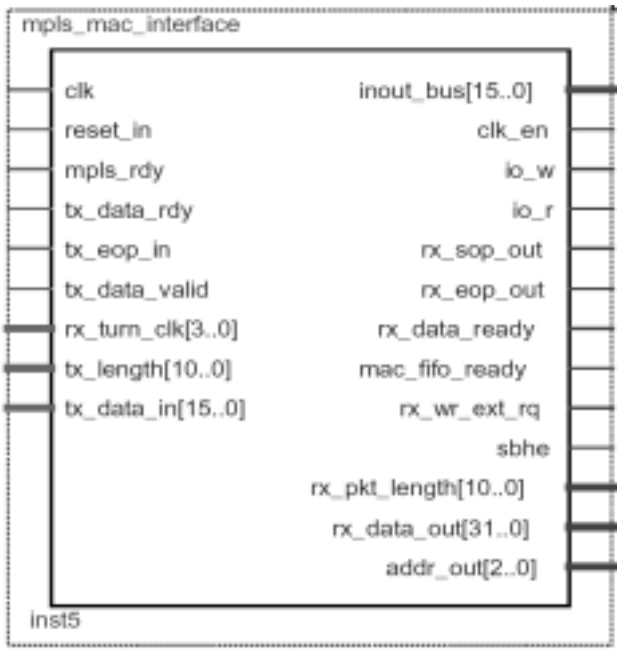


Figure 5-9 Interface Module Block Symbol, Prototype Name: mpls_mac_interface

The **reset_in** is connected to the hardware-reset pin on the motherboard, which drives the APEX FPGA device's reset pin low when pressed. Thus this active low **reset_in** can reset both the MPLS functional block and the interface module residing in the APEX device. A NOT gate is connected to the **reset_in** to provide an active high reset signal for the CS8900A according to its requirement.

After each reset, the CS8900A checks to see if an external EEPROM is present through an EEDataIn pin. If the EEDataIn pin is high, an EEPROM is present and the CS8900A automatically loads the configuration data stored in the EEPROM into its internal registers. If EEDataIn is low, an EEPROM is not present and the CS8900A comes out of reset with the default configuration. Since no EEPROM is used in this project while the CS8900A must be configured in a certain way as wanted, there has to be 10 ms spent waiting for the CS8900A to finish its self concatenation before any writes to the internal control and configuration registers can be done. A hardware delay is used, though continuously polling a self-status register to check if an INIT_rdy bit becomes high is an alternative. The INIT_rdy bit goes high once the self-concatenation is done.

The CS8900A works in 8-bit mode at power up but it has to work in 16-bit mode as required by this project, thus a **sbhe** signal fed to the CS8900A must be toggled to put the chip into 16-bit mode. Therefore after the hardware delay is ended and the INIT_rdy bit is checked to be also high, the **sbhe** line is toggled once and then always kept low until the next reset or power down. Before the CS8900A finishes its self-initialization, the **sbhe** is kept high to disable any read or write.

At this time, parameters for the tests can be written to the internal control/ configuration registers of the CS8900A. Once such configuration is done, the **clk_en** signal can be driven high to enable both the MPLS functional block and other part of the interface circuitry.

The bi-directional **inout_bus** of the interface module interfacing the CS8900A is 16-bit wide. The CS8900A assumes a little-endian ISA-type system. However, the network byte order is always big-endian. Therefore to minimize manipulation of frame data in ISA systems, the CS8900A byte-swaps frame data internally (The control and status registers are not byte swapped). In this design, the data lines are byte swapped, which means the interface takes the 7-0 bits of data as 15-8 bits of data from the CS88900A. By swapping the data lines, only the configuration/control/status values but not the frame data have to be swapped. This is more efficient due to the fact that most of the reads/writes are done for frame data.

The **tx_data_rdy, tx_data_valid** and **tx_data_in** are provided by the MPLS functional block. So long as there are data waiting for transmission, **tx_data_rdy** is set high, while **tx_data_valid** is only high for half clock cycle when there are data on the bi-directional **inout_bus [15:0]**. This **tx_data_valid** from the Transmit Buffers module of the MPLS functional block is used by the interface to generate write enable signal **io_w** for data transfer to the CS8900A. One thing has to be stated is how the **tx_data_valid** signal works. Actually since the time duration of each access (either read or write) to the FIFO (either rx or tx) is defined as one time unit and since data will stay on the bus much longer than one time unit, to prevent the same data being processed twice, the **tx_data_valid** signal is needed to indicate the availability of the data.

The r**x_sop_out** and **rx_eop_out** indicate the first and last one or two bytes of a received packet. They can stay high only when the **tx_data_valid** is also high. When a received frame is ready at the CS8900A and the MPLS functional block indicates that it is ready for packets processing, the MPLS packet generator can generate packets with certain MPLS labels as required. These MPLS packets then are transferred to the MPLS functional block through the interface. In other words, the actual received frame at the CS8900A is read but then discarded by the interface. In the case that the MPLS packet generator is not used, the actually received frame is sent to the MPLS functional block for label processing.

When the MPLS functional block does not have any room to hold more data or has no more data to transmit, the packet data generation or transfer (receive or transmit) are stopped right away and related information about the state is recorded for reference when this suspended state has to be resumed later. Each time when a packet is received or transmitted completely and successfully, the interface enters the other working state.

5.5 The Tests

5.5.1 Test Configuration

In this section, the real test procedure carried out is introduced. As mentioned earlier, the CS8900A chip has to be configured properly before it can receive and transmit packets. Table 5-2 shows the configuration parameters selected for the CS8900A in the tests. Other internal control or configuration registers not mentioned are set to keep their default values.

Table 5-2 a) CS8900A Configuration for Node A

Register Name	Register Address	Register Content	Register Content Description
RXControl	0104h	0180h	Accept packets with broadcast address
BusControl	0116h	1000h	Not to use IOCHRDYE signal
LineControl	0112h	C000h	Enable xmit and receive
TxCommand	010b	C900h	Xmit only after delivering the whole frame

Table 5-2 b) CS8900A Configuration for Node B

Register Name	Register Address	Register Content	Register Content Description
RXControl	0104h	0500h	Accept individual packet with MAC address saved in the register at address 0158h.
BusControl	0116h	1000h	Not to use IOCHRDYE signal
LineControl	0112h	C000h	Enable xmit and receive
TxCommand	010b	C900h	Xmit only after delivering the whole frame

Nodes A and B are similar except that node A is configured to enable the MPLS packet generator while Node B is not. Node A and Node B are also configured with different RxControl parameters. This is to enable Node A to receive any packet appearing on the LAN and then generate MPLS traffic accordingly but to enable Node B to receive only the generated MPLS traffic destined for it. Node B will forward the received MPLS packets onto the LAN again after assigning new MPLS labels to them. Thus a complete MPLS packet transmission, MPLS labels switching and packet receiving can be demonstrated.

The lookup tables residing in two nodes are initialized at the same time when the FPGA devices are programmed. It is very convenient to update the data afterwards in software through a simple CPU interface or in hardware with the aids of proper required interfacing signals. If the updates only happen to lookup table contents instead of the lookup table scale, software updates are more appropriate. Otherwise, hardware updates are preferred.

Typical LSRs that support QoS requirements should be able to store up to 1,024 labels at a time, requiring a 1,024×32 CAM block (The label is assumed to be 32 bits long). However in this project, no real routing is considered, and a final-stage commercial switch/router is not feasible for a single-chip implementation, thus it is not necessary trying to hold MPLS labels representing all kinds of EFCs. According to the test purpose, the lookup table is configured to have only 8 rows, just enabling switching between 8 sets of physical ports.

Table 5-3 to 5-5 describe the contents contained within lookup tables of Node A and Node B respectively, which include the mapping between IP headers and local

Table 5-3 Network Setting

Parameters	Nios1	Nios2
MAC Address	14.13.12.12.16.15	14.13.12.12.16.14
IP Address	192.168.129.216	192.168.129.215
Gate Way IP Address	192.168.129.254	192.168.129.254
DNS Server IP Address	192.168.129.254	192.168.129.254
Subnet Mask IP Address	192.168.129.0	192.168.129.0

Table 5-4 a) Node A Test Path Selection

Mappings	Input	Output
IP – MPLS	192.168.129.215	32’h0074
MPLS – MPLS	32’h00A4	32’h0074
MPLS - Outgoing Port	32’h00A4	3’b100

Table 5-4 b) Node B Test Path Selection

Mappings	Input	Output
IP - MPLS	192.168.129.216	32’h00A4
MPLS - MPLS	32’h0074	32’h00A4
MPLS - Outgoing Port	32’h0074	3’b100

Table 5-5 a) Node A Lookup Table Configuration

Mappings	Input	Output
IP - MPLS	192.168.129.211	32’h0070
	192.168.129.212	32’h0071
	192.168.129.213	32’h0072
	192.168.129.214	32’h0073
	192.168.129.215	32’h0074
	192.168.129.217	32’h0075
	192.168.129.218	32’h0076
	192.168.129.219	32’h0077
MPLS - MPLS	32’h00A0	32’h0070
	32’h00A1	32’h0071
	32’h00A2	32’h0072
	32’h00A3	32’h0073
	32’h00A4	32’h0074
	32’h00A5	32’h0075
	32’h00A6	32’h0076
	32’h00A7	32’h0077
MPLS - Outgoing Port	32’h0070	3’b000
	32’h0071	3’b001
	32’h0072	3’b010
	32’h0073	3’b011
	32’h0074	3’b100
	32’h0075	3’b101
	32’h0076	3’b110
	32’h0077	3’b111

Table 5-5 b) Node B Lookup Table Configuration

Mappings	Input	Output
IP - MPLS	192.168.129.211	32’h00A0
	192.168.129.212	32’h00A1
	192.168.129.213	32’h00A2
	192.168.129.214	32’h00A3
	192.168.129.216	32’h00A4
	192.168.129.217	32’h00A5
	192.168.129.218	32’h00A6
	192.168.129.219	32’h00A7
MPLS - MPLS	32’h0070	32’h00A0
	32’h0071	32’h00A1
	32’h0072	32’h00A2
	32’h0073	32’h00A3
	32’h0074	32’h00A4
	32’h0075	32’h00A5
	32’h0076	32’h00A6
	32’h0077	32’h00A7
MPLS - Outgoing Port	32’h00A0	3’b000
	32’h00A1	3’b001
	32’h00A2	3’b010
	32’h00A3	3’b011
	32’h00A4	3’b100
	32’h00A5	3’b101
	32’h00A6	3’b110
	32’h00A7	3’b111

MPLS labels, the mapping between in-coming MPLS labels and local MPLS labels and the mapping between local MPLS labels and local outgoing physical ports. Though real data flows from layer 3 do not exist in the tests, corresponding parts of the lookup table are still presented in this thesis. In practice, more dimensions in addition to the IP address can be considered for the selection of local MPLS labels to ensure specified quality of service. In order to show more clearly how an MPLS edge node works, the network setting is also listed though it is only needed in the future work.

5.5.2 Real Testing Procedure

After the test equipment are all correctly configured, the desktop computer continuously sends out PING packets evenly at a frequency of about 0.3 ms over this small Ethernet LAN. MPLS node A is set to grab those broadcast packets. After receiving a PING packet, Node A generates a packet with a predefined MPLS label (00A4h) and can have the length of this generated packet equal to that of the received one. Then after table lookup, the packet bound with the corresponding new outgoing MPLS label (0074h) and the destination MAC address representing Node B is driven onto the LAN. PING packets can be defined with various lengths, but for simplicity, all the PING packets are set to be 60 bytes by default.

Now there is internally generated MPLS traffic running over the Ethernet. MPLS Node B detects the existence of traffic destined to it and then receives the packets. The received packet is first buffered at its corresponding receive FIFO within the MPLS functional block of Node B. Then it is passed onto the Label Removing, the Label Binding and Switching, and the Lookup Table modules for label processing, where a new outgoing label, 00A4h (representing Node A here but could be anything else in a practical), is assigned to the packet. Then the packet is buffered at the corresponding transmit FIFO waiting for its turn to get transmitted.

A Tektronix TLA 700 series logic analyzer is connected between the CS8900A chip and the APEXII FPGA device of Node B to record what is taking place on the data bus, address bus, and I/O read and write strobe enables. The logic analyzer can hold 128K data samples, which is enough for the test demonstration.

Chapter 6 Test Results and Analysis

6.1 Overview

In this chapter, test results gathered by a digital analyzer are presented. All test results were obtained from Node B, which was defined as the receive node in Chapter 5. The results show that the MPLS functional block works properly as expected.

One CS8900A Ethernet daughter card is mounted on the motherboard and connected to the FPGA device through the 3 pin headers for 3.3-volt prototype connector. Since data at the 10Base-T port cannot be probed (The pins are concealed within the package) and only the *lower* level of the two-level daughter card stack is used, it is natural to collect data through those pin header connectors preserved for the *higher* level daughter card that also locate between the CS8900A and the FPGA device but are not used for any function purpose in the design. Please refer back to Figure 5-3 and 5-4 in the previous chapter. These pin header connectors provide the *SD [15:0]*, the *SA [3:1]* (*SA [0]* has been set to be always low in the design), the I/O read enable $\overline{IOR}$, the I/O write enable $\overline{IOW}$ and the working mode selection signal $\overline{SBHE}$, which have been indicated in Figure 5-7 and 5-8 previously. These are all the signals required by the CS8900A to achieve successful communication with other circuitry and therefore they must be probed to verify the correctness of the design.

Also, several internal signals within the MPLS functional block are obtained through the pin header connectors for the 5.5 volt prototype connectors on the motherboard, for they help to present a better view of the whole design. The following is a brief description of the signals probed for results demonstration:

data_valid --- the signal that tells the Rx Buffers module when the data on the rx_data [7:0] bus should be buffered.

rxreg_read --- the read enable signal for the Rx Buffers module to transfer packet for further label processing.

rx_data [7:0] --- the lower 8 bits of the packet data transferred from the Rx Buffers module to the Label Removing module.

mac_sop_out --- start of the frame to be saved into the Tx Buffers module.

mac_eop_out --- end of the frame to be saved into the Tx Buffers module.

mpls_rdy --- the signal that indicates if the Label Binding and Switching module is ready to accept new packet.

turn --- the signal that indicates the service state of the interface module: high for transmission and low for reception.

save --- the signal that notifies the embedded microprocessor to accept the received data when it is high , and to ignore them when it is low.

Figure 6-1 exhibits 3 cycles of packet processing procedures. The time distance between two successive procedures on average is 0.3 ms, which is the time distance between two MPLS packets generated by the interface circuit.

6.2 Test Result Demonstration and Simple Analysis

In the following sections, detailed illustration for each phase during the packet processing procedure is given.

6.2.1 CS8900A Configuration

Upon each reset or power up, with the parameters descried in Chapter 5, the CS8900A is configured to work in 16-bit I/O mode, as required by the tests. The MAC_RAM base address and the I/O base address are configured by default to be 0000h and 0300h respectively.

After the hardware delay inserted for the CS8900A to do self-concatenation is finished, the SelfStatus register at MAC_RAM base + 0136 is checked to see if its INIT_rdy bit has been set. The CS8900A should set this bit after the 10 ms hardware delay completes. After this the CS8900A and the MPLS functional block enter their ordinary operation modes. The results following prove that the configuration is effective and correct.

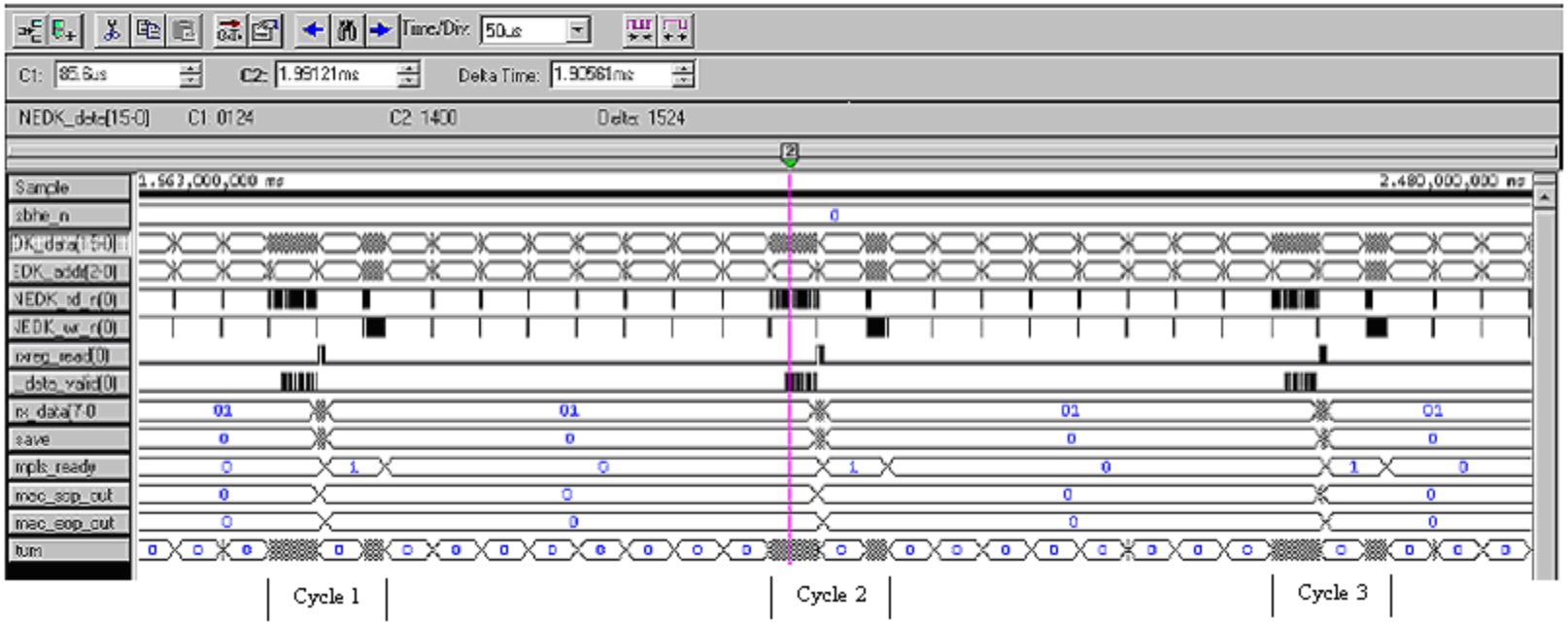


Figure 6-1 receive_&_transmit

Table 6-1 Bit Definition for SelfStatus Register

7	6	5	4	3	2	1	0
INIT_rdy	3.3V Active	0	1	0	1	1	0
F	E	D	C	B	A	9	8
						EEPROM present	

010110: These bits identify this as the Chip Self Status Register.

3.3v Active: If the CS8900A is operating on a 3.3v supply, this bit is set.

INIT_rdy: If set, the CS8900A initialization, including read-in of the EEPPROM, is complete.

EEPROMpresent: If the EEDataIn pin is low after reset, there is no EEPROM present, and this bit is clear. If the EEDataIn pin is high after reset, the CS8900A assumes that an EEPROM is present, and this it is set.

6.2.2 Packet Receiving and Label Swapping

Figure 6-2 receive_full shows the whole duration that a complete packet is being received. It can be seen that the *data_valid* signal only becomes active after certain data have been received. Those data are MAC header information and are saved to dedicated registers instead of being transferred to the Rx Buffers module of the MPLS functional block.

Figure 6-3 (a) receive_0 shows how the packet reception begins. Node B is configured to only accept packets with the individual destination MAC address 14:13:12:11:16:14. From this figure, it is clear that the internal register RxEvent at address 0124h (at point A in Figure 6-3 (a)) of the CS8900A is accessed by setting the address pointer at the MAC_RAM pointer port at I/O base + 000Ah; then the content of the RxEvent register, 0504h (at point B in Figure 6-3 (a)), appears at MAC_RAM data Port0 at I/O base + 000Ch, as expected. According to the bits defined within this register, it is learned that a packet with a destination address that matches the *individual address* found at 0158h has been received by the CS8900A successfully.

The frame data is then fetched by repetitively driving the $\overline{IOR}$ low, starting from the address MAC_RAM base + 0400h. The first word read is still 0504 ((point C in

Figure 6-3(a)), which is the content of the RxStatus register locating at MAC_RAM base + 0400h. The RxStatus is mirrored from the RxEvent and the only difference between them is that when the RxEvent register is read, RxStatus will not be cleared while the RxEvent will. The second word read is 003Ch (point D in the Figure 6-3 (a)), which is the length of the packet. The third word read is 1314h(point E in the Figure 6-3 (a)), which is the lower 2 byte of the standard IEEE 802 MAC address. It is 1314h, instead of 1413h, because bits 7-0 and bits 15-8 of the data bus connected to the CS8900A has been swapped due to different byte orders used at network layer and the physical layer.

Table 6-2 Bit Definition of RxEvent register

7	6	5	4	3	2	1	0
		0	0	1	1	0	0
F	E	D	C	B	A	9	8
					Individual Adr		RxOK

000100: These bits identify this as the Receiver Event Register. When reading this register, these bits will be 000100, where the LSB corresponds to Bit0.

RxOK: If set, the received frame had a good CRC and valid length. When RxOK is set, the length of the received frame is contained at 0402h.

Individual Adr: If the received frame had a Destination Address that matched the Individual Address found at 0158h, then this bit is set if, and only if, RxOK is set and Individual Adr (Register 5, RxCTL, Bit A) is set.

From Figure 6-3 (b) receive_1, it can be seen that the destination address contained within the frame data is 14:13:12:11:16:14, which belongs to Node B. From Figure 6-3 (c) receive_2, the source address 14:13:12:11:16:15 representing Node A can be seen. These two figures show that the packet is transmitted by Node A towards Node B and prove that the packet transmission has been successfully completed.

In both Figure 6-3(c) and Figure 6-3(d), the first word within the received frame is 0074h, which is the MPLS label assigned to the packet at Node A. It is from this point of the frame that the data start being saved to the Rx Buffers module of the MPLS functional block. This explains the absence of *data_valid* 's being high for the first part of the received frame. All the DA, SA and type/length data are saved to dedicated

registers for reference by the MPLS hardware when needed. However, these registers will all be overwritten automatically when the next frame start being received.

In Figure 6-3 (e) receive_4, the word 0100h (the actual data value is 0001h before byte swap) has been received and the whole receiving procedure is finished after the MissCounter register at 013Ch has been read. Its content turns out to be 0010h (shown in Figure 6-5 (a) save_to_mpls0), which means no packet has been missed. This read may not make much sense when nothing unexpected happens. However, since the CS8900A data sheet does not fully explain what should be done in polling mode to ensure that no event will be left unprocessed to cause some unknown problem, this MissCounter register is always read and cleared after each received frame is transferred to the MPLS functional block.

Still in Figure 6-3 (e) receive_4, the incoming label 0074h appears on the *rx_data[7:0*] bus for 120 ns and then the word 0016h lasts for about 1.9 us. This is because after the incoming MPLS label 0074h is read from the Rx FIFO and sent to the Label Removing module, the first two bytes of the frame content has to be read as well due to the internal structure of the FIFO function. Then there will be no more operations until a new outgoing label is found for this packet and bound to it. Label lookup behavior only needs 5 clock cycles to finish, which is much shorter than 1.9 us. However, due to the fact that packet length is always 60-byte long and the normal service interval lasts to perform exactly 16 reads/writes, the state always transits right after the incoming label is stripped off. This means that, the newly found outgoing label has to wait until the next service interval to be bound to the packet. That is why this 1.9 us exists and during which the data on the data bus is only 0016h. After this 1.9 us, the real first word 0016h, of the packet data will be written to the Tx Buffers module immediately, following the newly bound outgoing MPLS label.

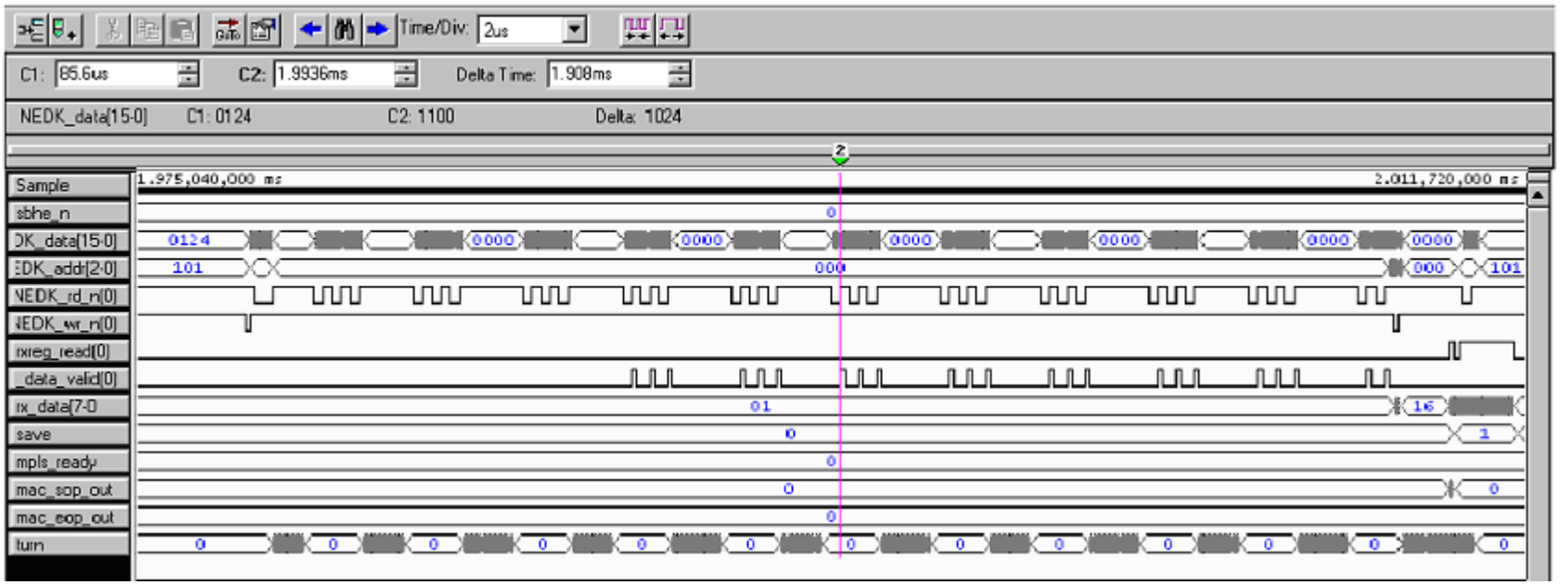


Figure 6-2 receive_full

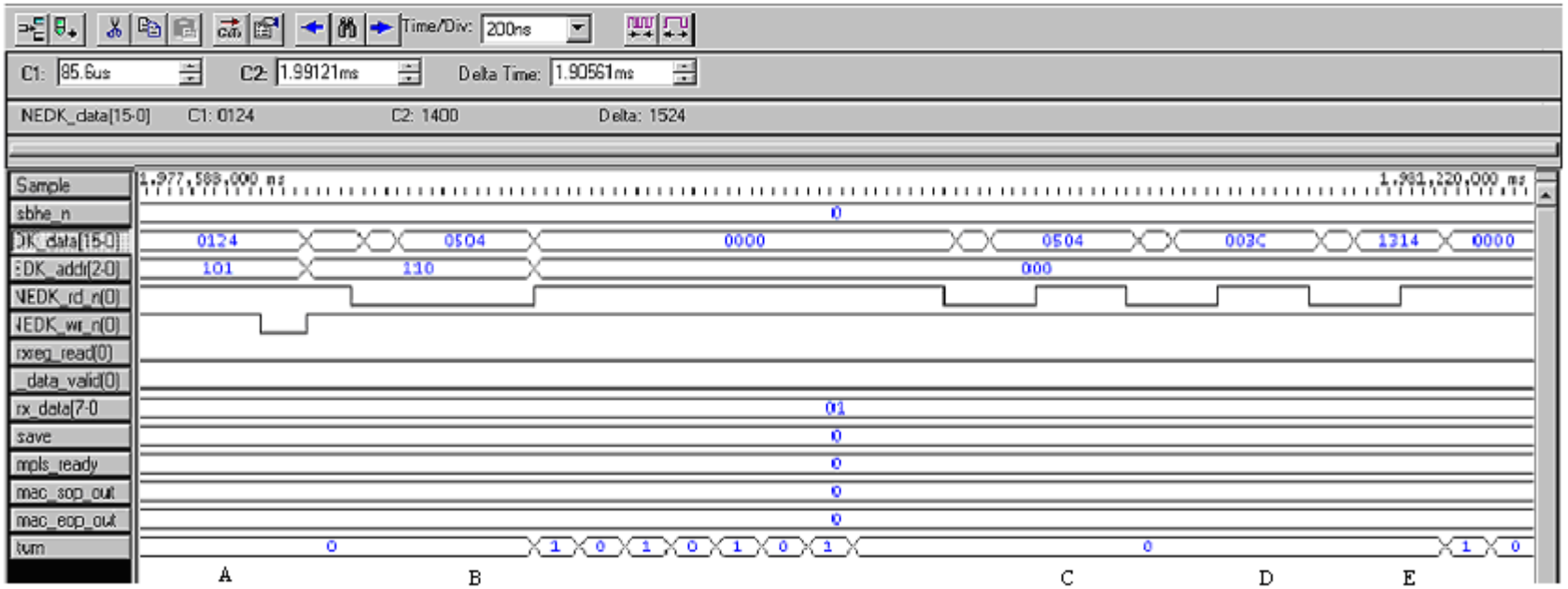


Figure 6-3 (a) receive_0

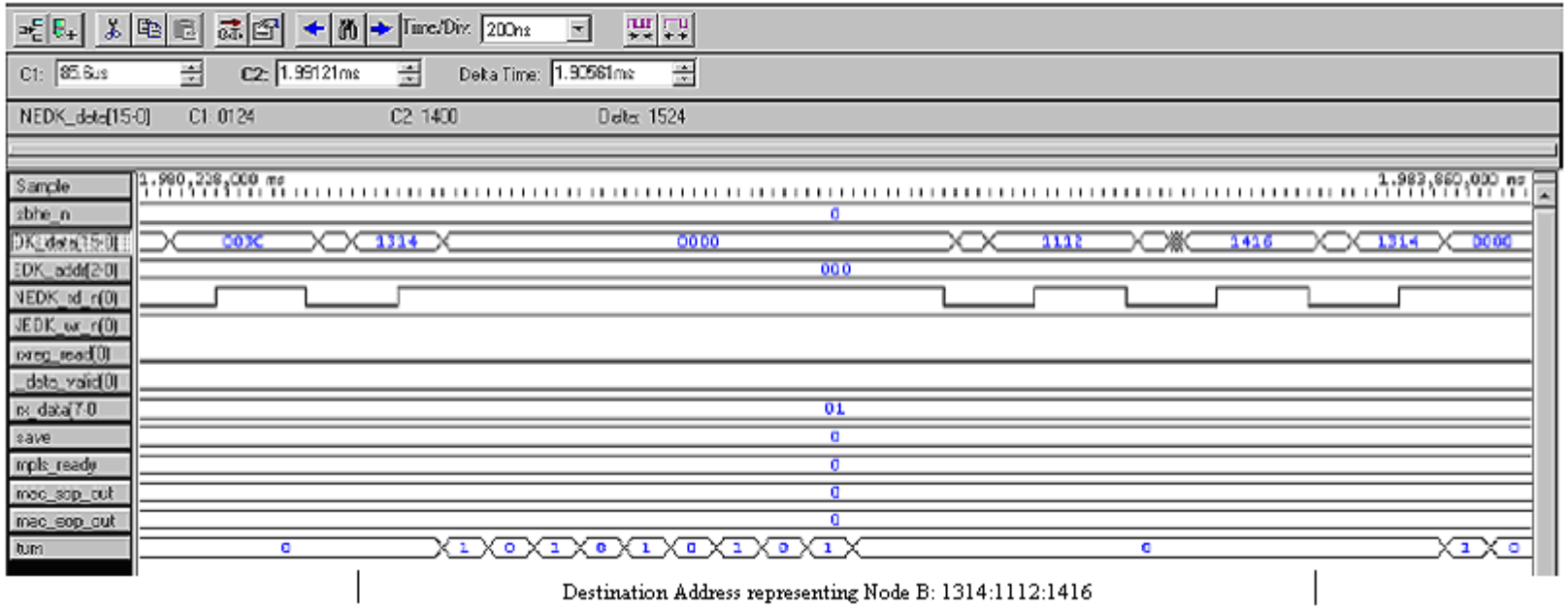


Figure 6-3 (b) receive_1

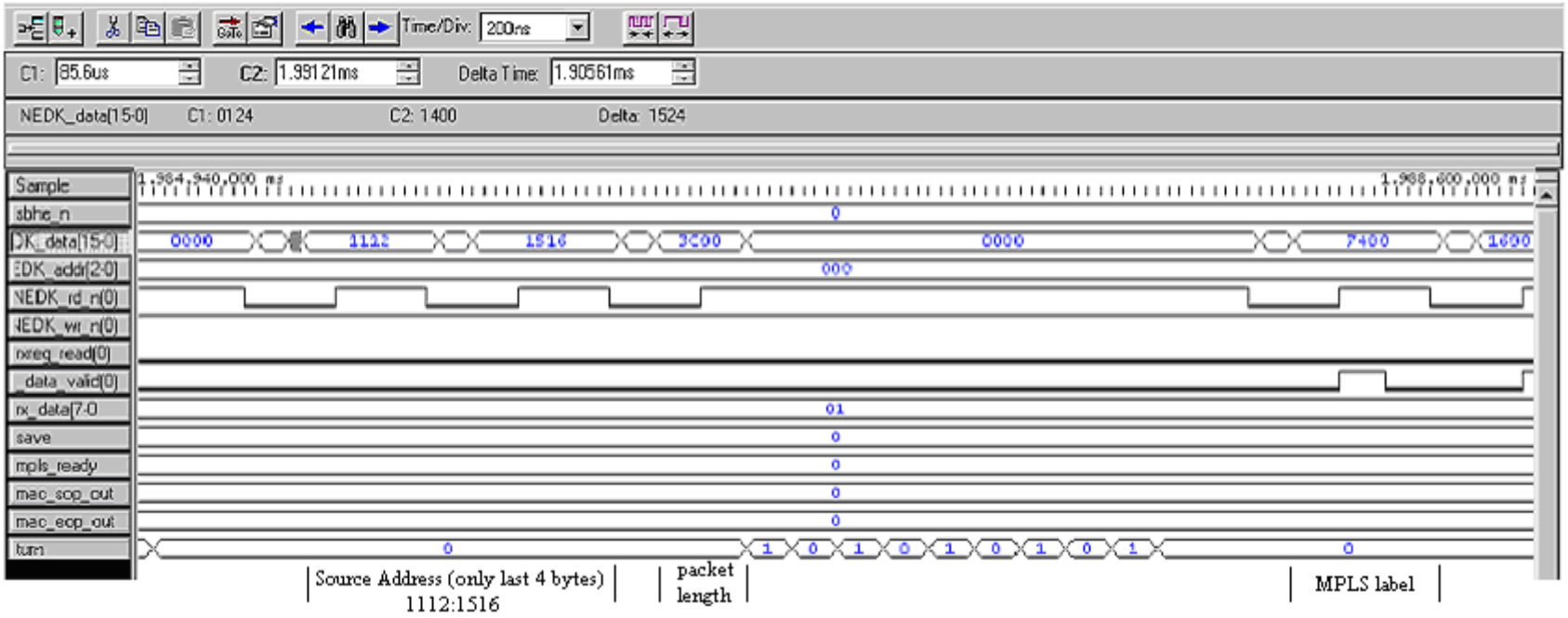


Figure 6-3 (c) receive_2

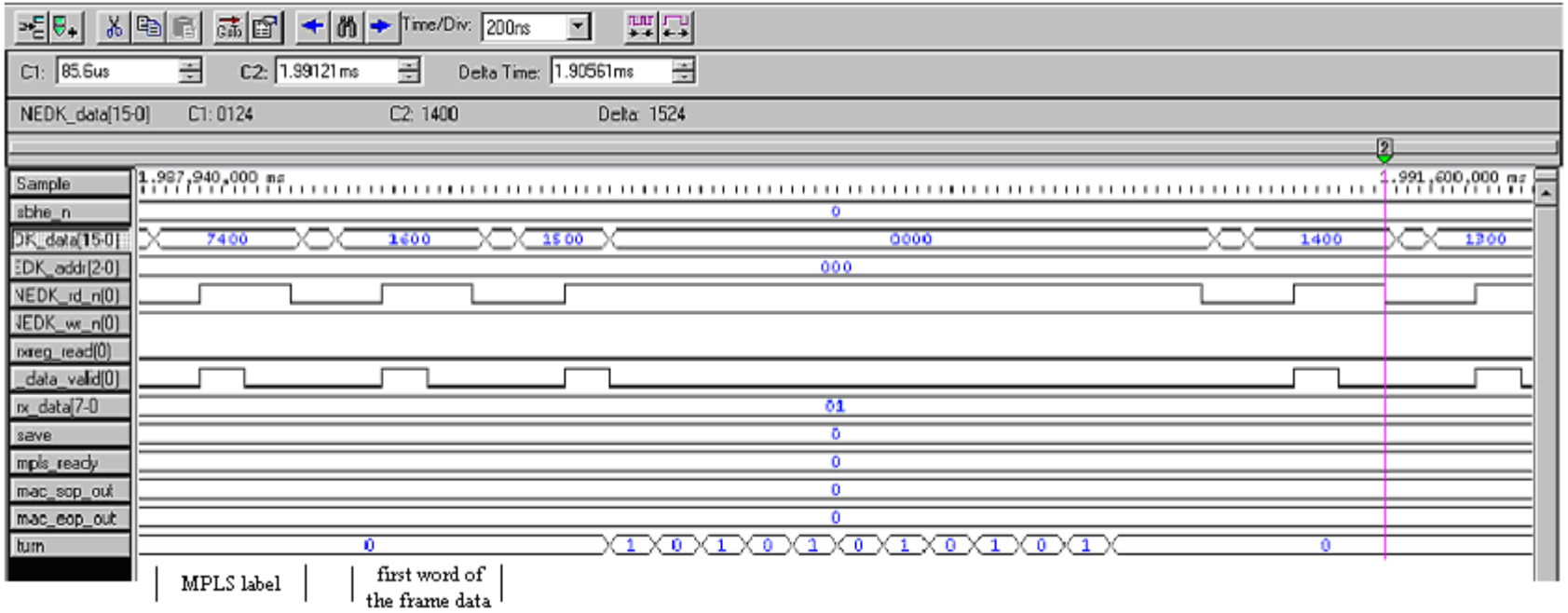


Figure 6-3 (d) receive_3

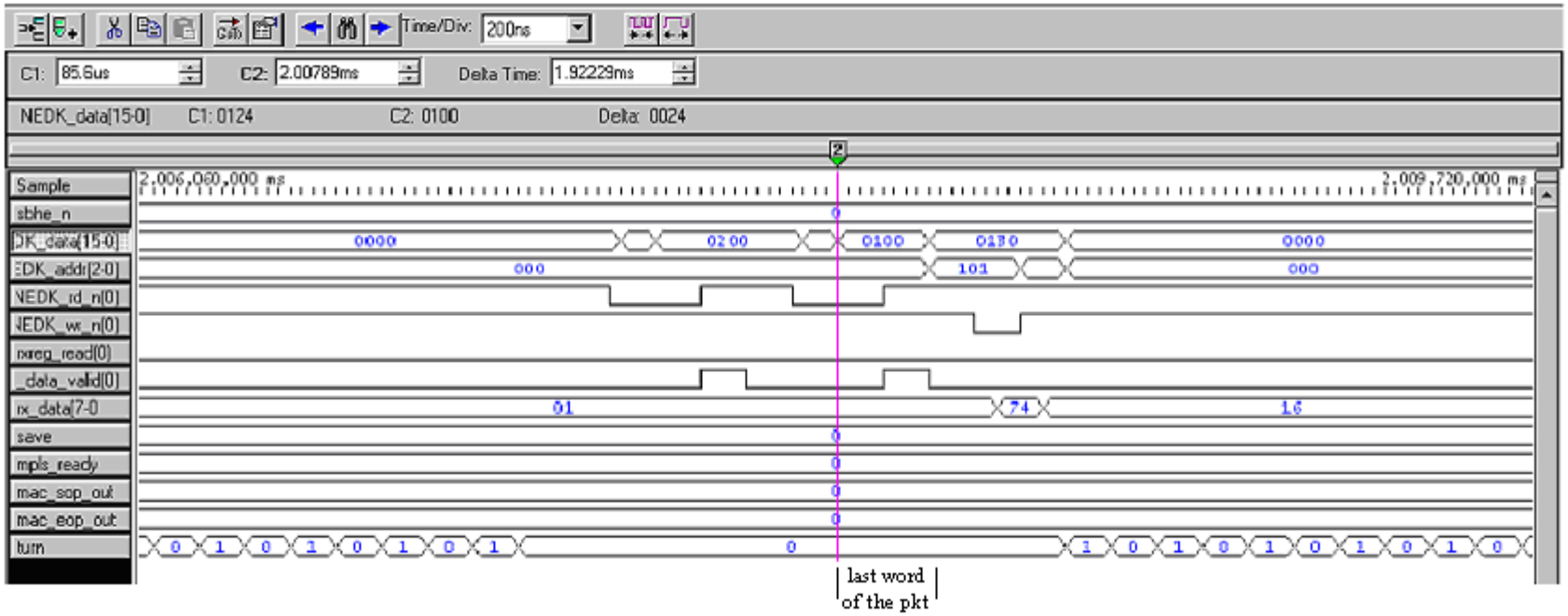


Figure 6-3 (e) receive_4

6.2.3 Label Binding and Packet Buffering

Figure 6-4 save_to_mpls illustrates the procedure whereby a complete packet is transferred from the Rx Buffers to the Tx Buffers and how the new MPLS label is bound to the packet. The individual steps making up this phase are presented in the following paragraphs.

In Figure 6-5 (a) save_to_mpls0, on the *rx_data[7:0]* bus, a new outgoing MPLS label 00A4h appears and is bound to the incoming packet that used to be with the MPLS label 0074h. After “00A4h”, the start of the packet to be forwarded is written to the Tx Buffers, frame data transfer from the Rx Buffers to the Tx Buffers can then be started. The figure shows that the *mac_sop_out* becomes high when the *rx_data[7:0]* bus has 00A4h on it (point A in Figure 6-5 (a)). The whole packet cannot be transferred completely within one service interval granted to this port, thus it has to take several service turns before all the transfer can be done.

In Figure 6-5 (b) save_to_mpls1, it can be seen that *mac_eop_out* becomes high when the last word of the packet, 01h(point A in Figure 6-5 (b)), appears on *the rx_data[7:0]* bus, which signals the end of the current packet transfer from the Rx Buffers to the Tx Buffers.

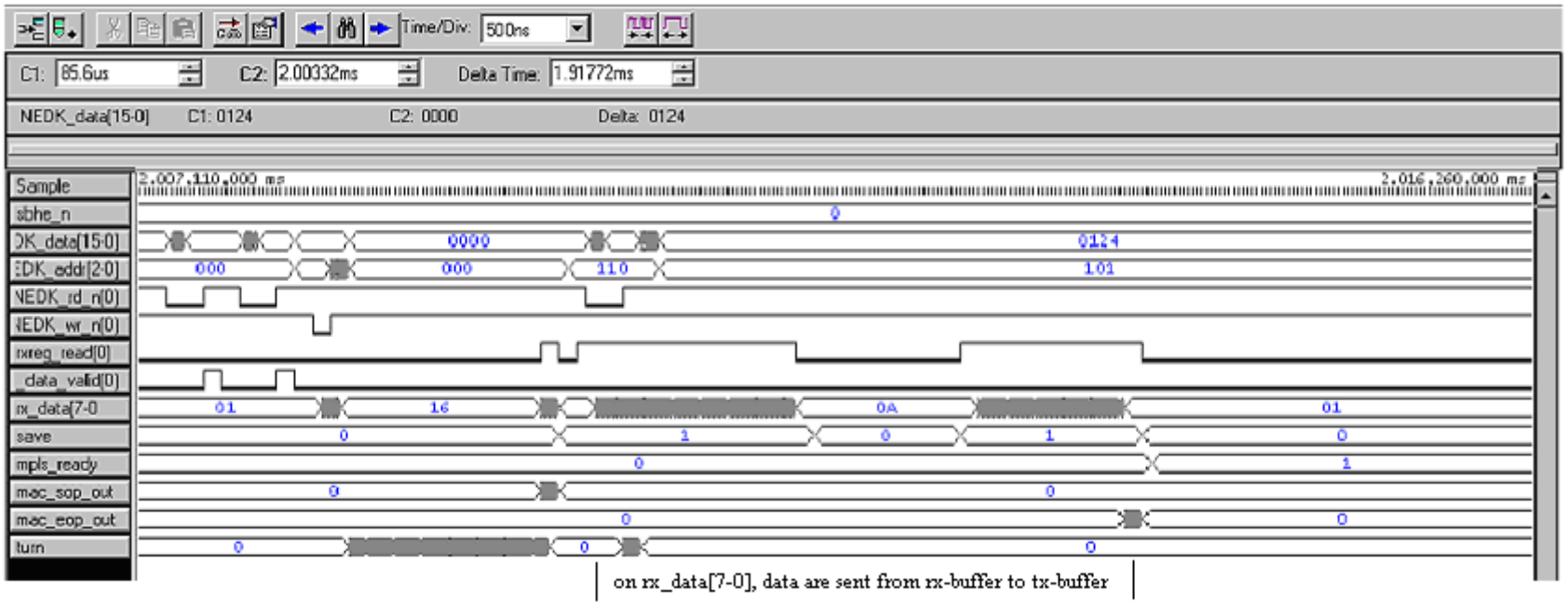


Figure 6-4 save_to_mpls_full

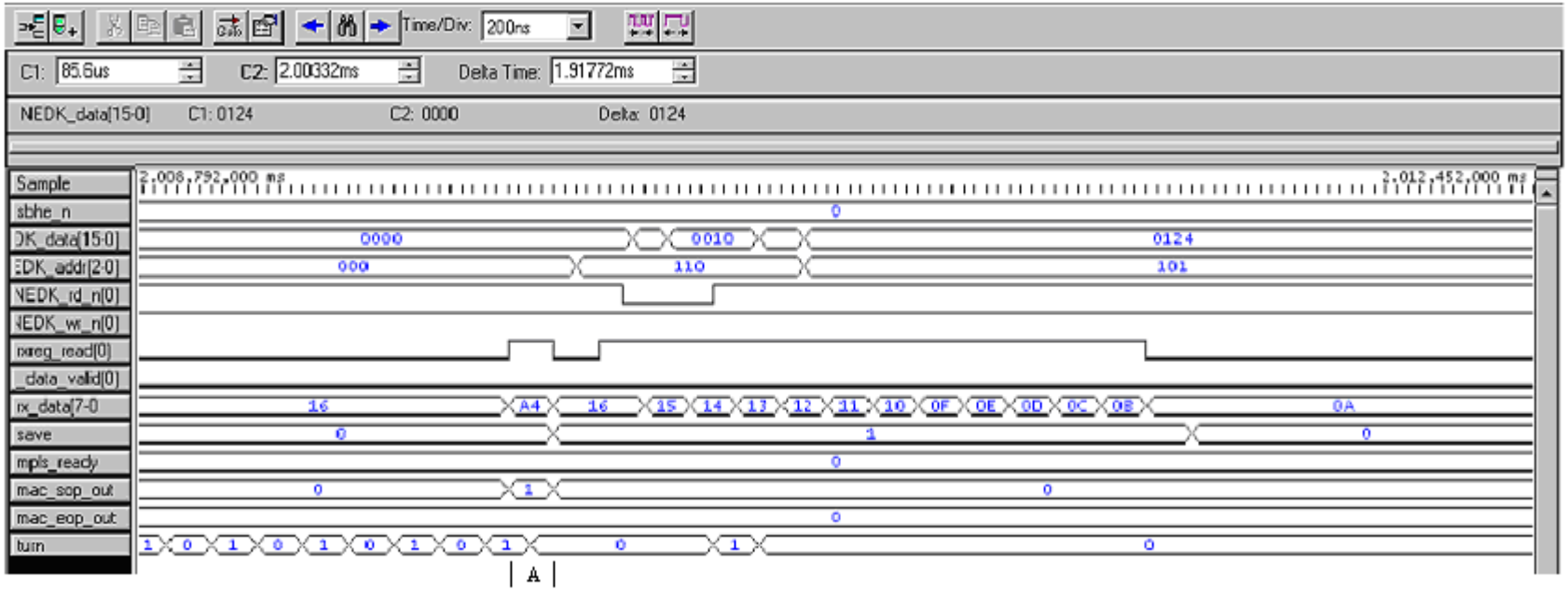


Figure 6-5 (a) save_to_mpls_0

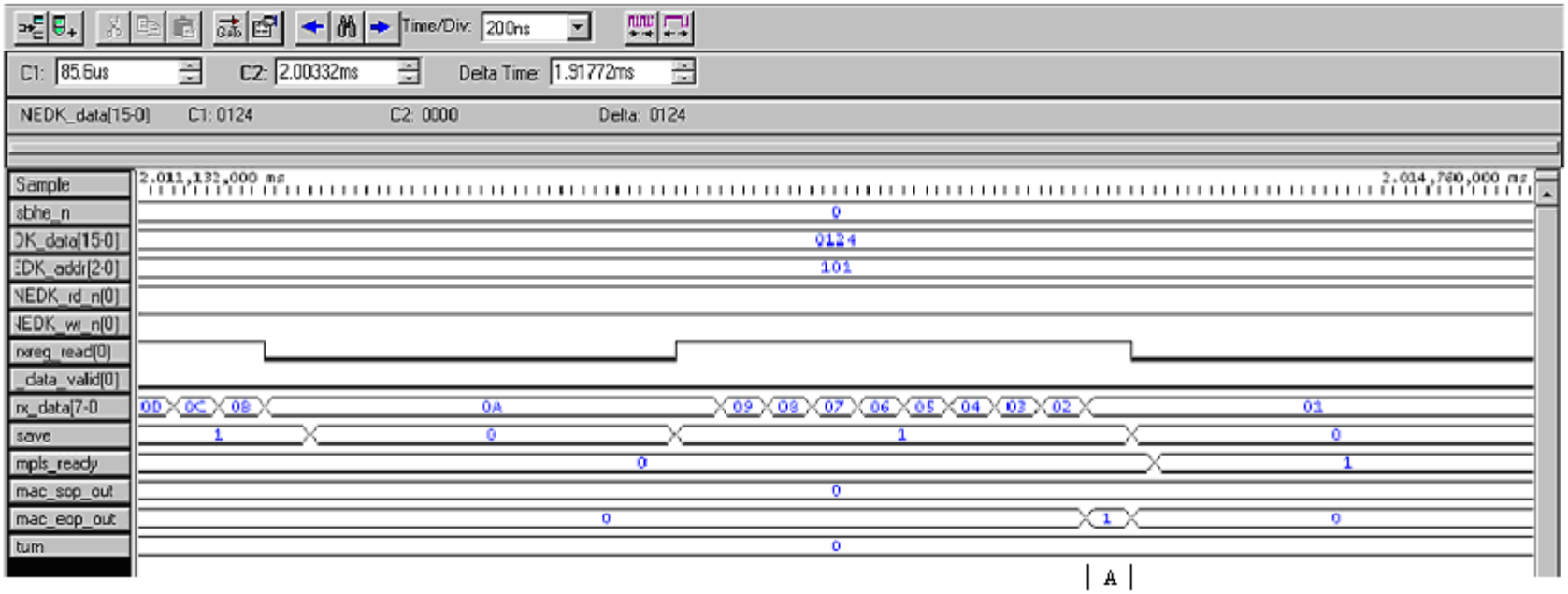


Figure 6-5 (b) save_to_mpls_1

6.2.4 Packet Transmission

The last phase demonstrates the packet transmission procedure, which is shown in Figure 6-6 transmit_full in a general view. Detailed explanation of each part of the figure is presented in the following paragraphs and figures.

In Figure 6-7 (a) transmit_0, the address 0128h (point A in Figure 6-7 (a)) of the TxEvent register is written to the MAC_RAM pointer port at 000Ah first; then at the MAC_RAM data port0 at address 000Ch, the content of the TxEvent register is read out and appears as 0108h(point B in Figure 6-7 (a)). The value of 0108h indicates that the last packet has been completely transmitted and the CS8900A is now ready to accept a new transmit frame storage bid issued by the MPLS hardware. This bid has to be done at the start of each transmit operation. The first step to issue the bid is to write the transmit command word (at point C in Figure 6-7 (a)) to the TxCMD register at I/O base + 0004h. The transmit command informs the CS8900A that the MPLS hardware now has a frame to be transmitted, as well as how that frame should be transmitted.

Table 6-3 Bit definition of TxEvent Register

7	6	5	4	3	2	1	0
	0	0	1	0	0	0	0
F	E	D	C	B	A	9	8
							TxOK

001000: These bits provide an internal address used by the CS8900A to identify this as the Transmitter Event Register;

TxOK: This bit is set if the last packet was completely transmitted.

Table 6-4 Bit definition of TxCommand Register

7	6	5	4	3	2	1	0
TxStart		0	0	1	0	0	1
F	E	D	C	B	A	9	8
			InhibitCRC				

001001: These bits provide an internal address used by the CS8900A to identify this as the Transmit Command Register;

TxStart: This pair of bits determines how many bytes are transferred to the CS8900A before the MAC starts the packet transmit process.

Bit 7	Bit 6	
0	0	Start transmission after 5 bytes are in the CS8900A

InhibitCRC: When set, the CRC is not appended to the transmission

Next, the transmit frame length is written to the TxLength register through the TxLength port at I/O base + 0006h to complete the bid for buffer space on the CS8900A. In Figure 6-7 (a) transmit_0, the length of the frame to be transmitted is shown as 003Ch(point D in Figure 6-7 (a)).

Table 6-5 Bit definition of Bus Status Register

7	6	5	4	3	2	1	0
TxBidErr	0	0	0	1	0	0	1
F	E	D	C	B	A	9	8
							Rdy4TxNow

001001: These bits provide an internal address used by the CS8900A to identify this as the Transmit Command Register;

TxBidErr: If set, the MPLS hardware has commanded the CS8900A to transmit a frame that the CS8900A will not send. Frames that the CS8900A will not send are:

1) Any frame greater than 1514 bytes, provided that InhibitCRC (TxCMD Register, Bit C) is clear;
2) Any frame greater than 1518 bytes;

Rdy4TxNow: Rdy4TxNOW signals the MPLS hardware that the CS8900A is ready to accept a frame from the MPLS hardware for transmission.

After the complete transmit command has been issued to the CS8900A, the state of the BusStatus register at MAC_RAM base + 0138h is checked to see if the bid has been successful or not. In Figure 6-7 (a) transmit_0, the BusStatus Register at 0138h (point E in Figure 6-7 (a)) returns the value of 0118h(point F in Figure 6-7 (a)), which means that the CS8900A is now ready to accept the frame with the required length as shown in

the transmit command issued by the MPLS hardware. It is apparent that the encapsulated MAC frame is transferred with 14:13:12:11:16:15 as its DA (point G in Figure 6-7 (a)) at the beginning of the frame. In Figure 6-7 (b) transmit_1, the SA 14:13:12:11:16:14 appears followed by the field of type/length, 003Ch. Then the following data transferred to the buffer of the CS8900A are the MPLS label 00A4h, and finally the remaining frame data. In Figure 6-7 (c) transmit_2, after the last word of the frame is transferred to the CS8900A, the interface module began polling the RxEvent Register at 0124h again, which represents the start of a new cycle of packet processing.

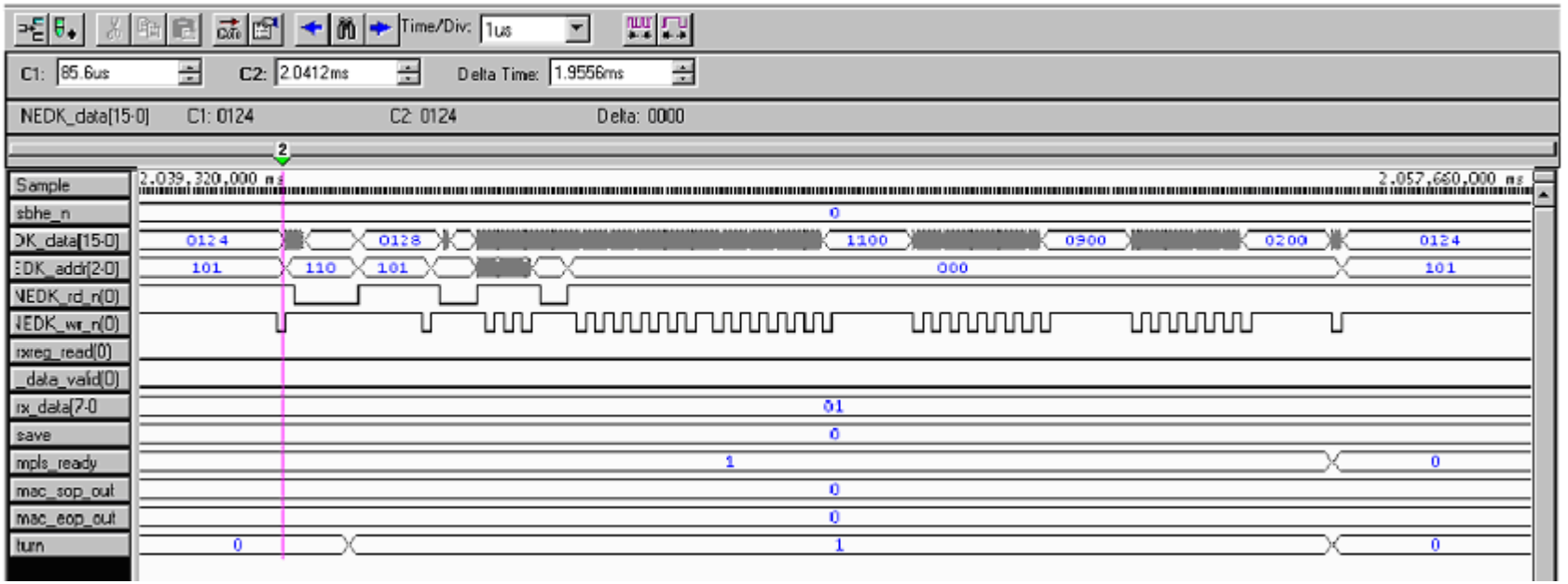


Figure 6-6 transmit_full

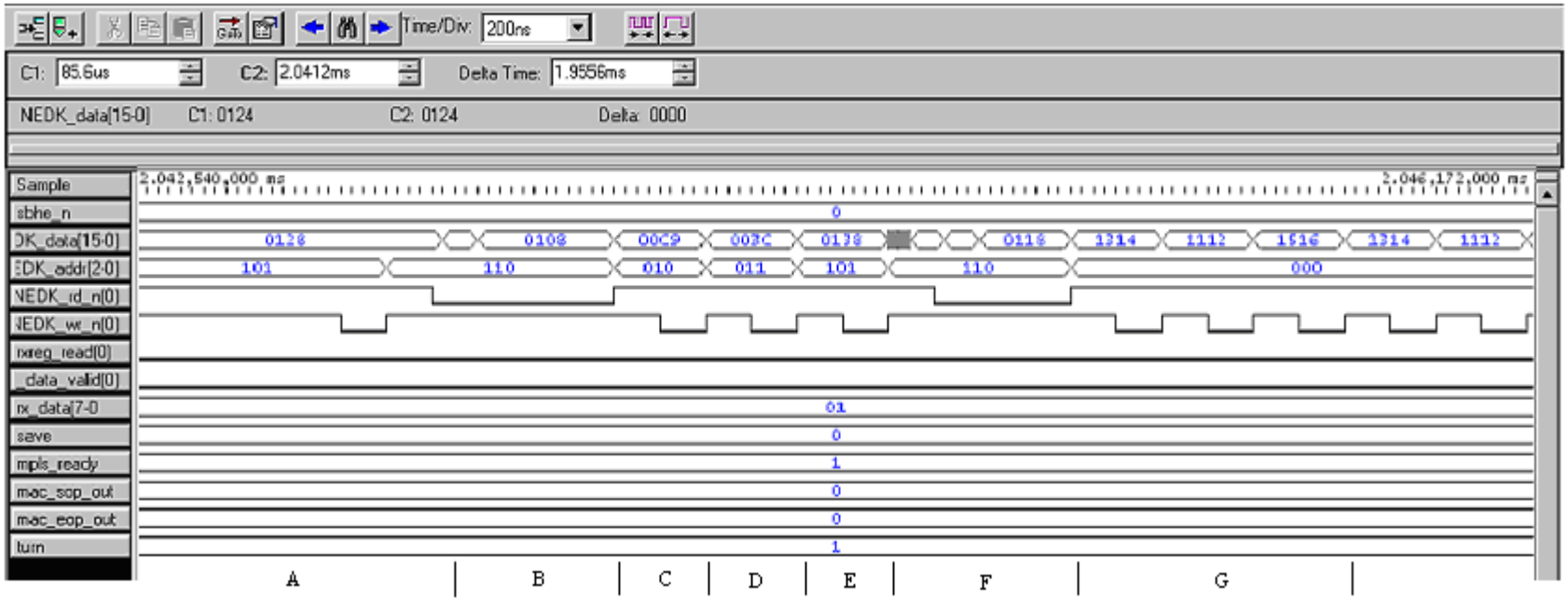


Figure 6-7 (a) transmit_0

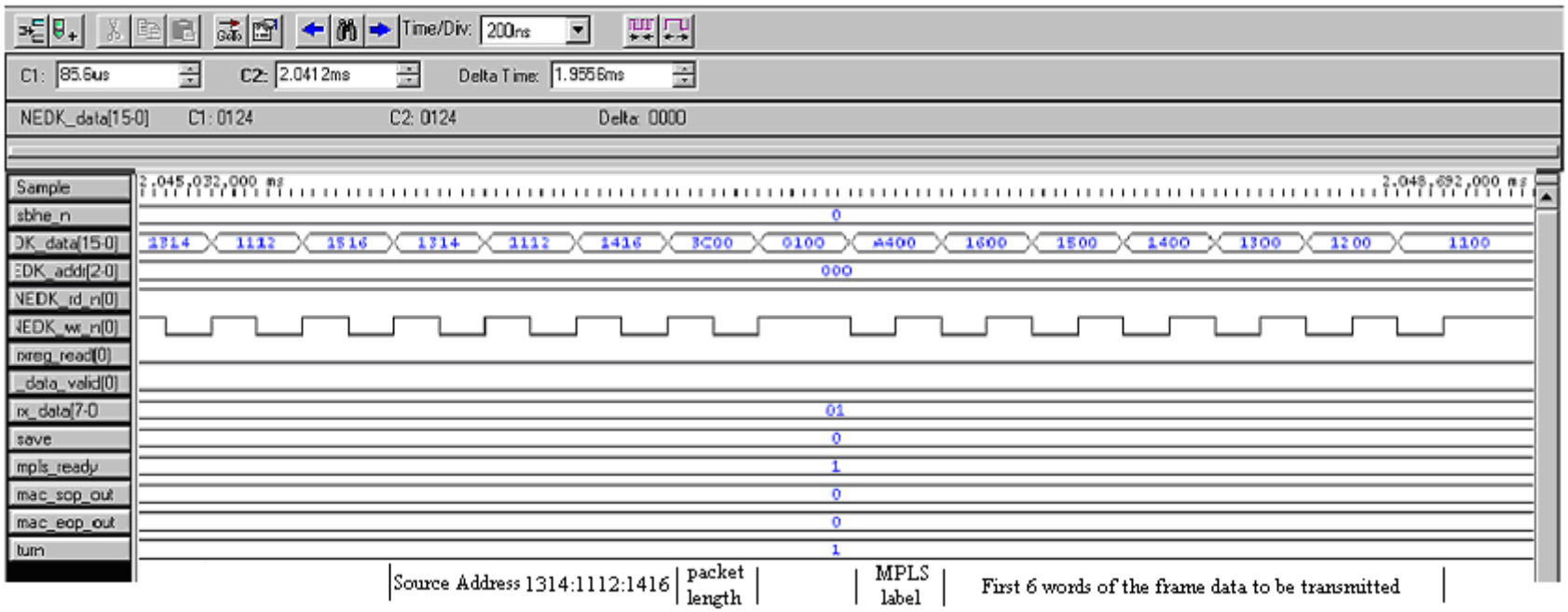


Figure 6-7 (b) transmit_1

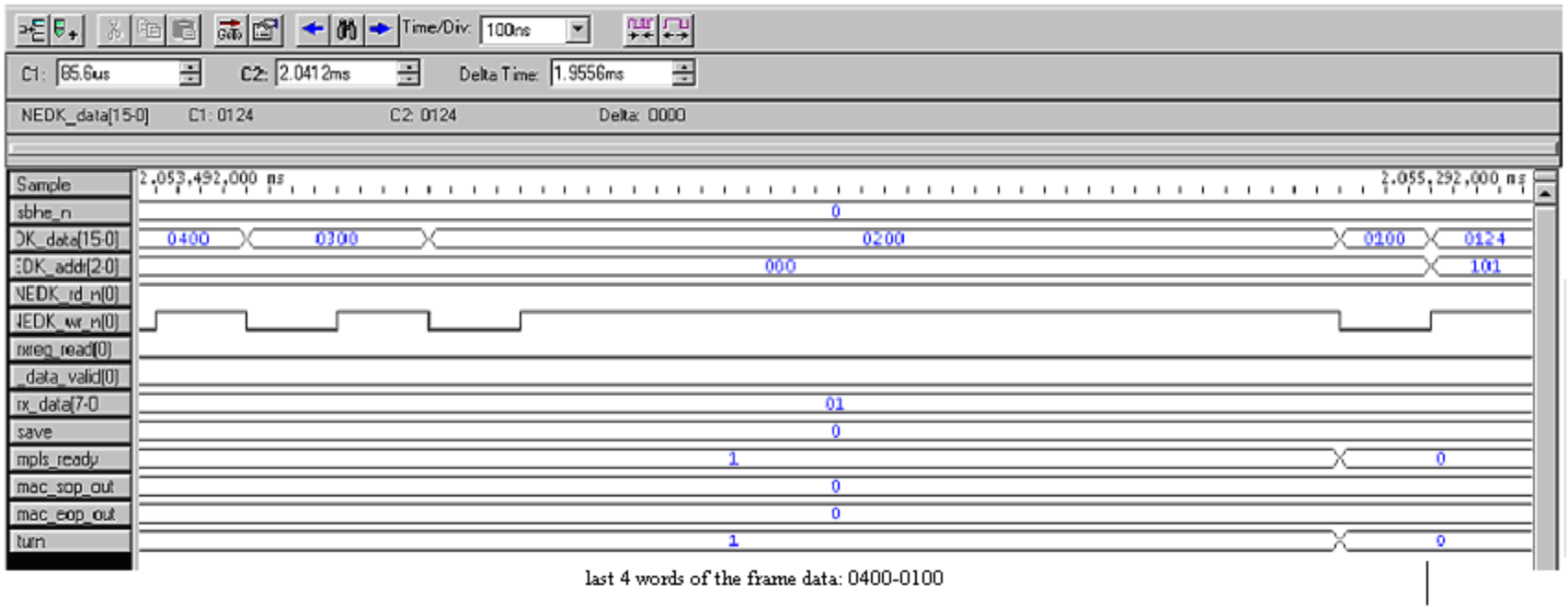


Figure 6-7 (c) transmit_2

The test results demonstrated above are exactly what were expected. The system was kept working continuously for one day and no disruption was found. Based on the appropriate test methodology described in the earlier chapter, the results prove that the CS8900A configuration is effective and correct, and the MPLS hardware design is successful.

In the tests, due to the fact that the packet length was defined to be always 60-byte long (standard PING packet length by default) and the normal service interval granted to each session was set to perform exactly 16 reads/writes, the service state would transit right after the incoming label is stripped off. In general, this kind of state transition can cause a bad effect on label swapping, such as data loss or wrong label being found and bound. According to the tests, no such errors occurred at the state transition (the most vulnerable situation in terms of packet lengths) and it can be inferred safely that under normal conditions (the packet length varies with time passing by and the state transition usually happens in the middle of the ordinary packet data transfer), the MPLS hardware can perform label processing just as wished. However, more tests can be carried out in the future to have MPLS packets generated with various lengths (simply by adjusting the PING packets' lengths) to provide more facts that can prove the correctness of the design.

Due to the time and energy limitation, no simulation has been done to compare the cost/performance between software implemented and hardware implemented MPLS. However, the size and complexity of many problems have quickly exceeded the power of conventional computer hardware in general [40, chapter1]. Also, software instructions are executed by a hardware implemented microprocessor, it is theoretically safe to say that if the same functions used to be performed serial by software now are performed parallel by hardware, faster processing speed can be achieved. For example, the process of packet receiving, label removing, table searching, label binding, label switching and packet transmitting can be completed within just 8 cycles by hardware. However, the process has to be translated into more steps and each step requires multiple clock cycles to finish in software. As a roughly estimation, pure hardware

implementation can bring a processing speed 5-10 times faster than pure software implementation in this project.

The whole design, including both the MPLS functional block and the interface between the MPLS hardware and the CS8900A chip, takes up to 7,143 logic elements (approximately 177,840 typical gates) and 113,500 RAM bits. With a 32-bit wide data path, the FPGA device supposed to operate at a system clock speed of 33.3 MHz, which was divided into 8 MHz only in the tests, can easily realize a system that provides 100 Mbits/sec data transfer with the proper MAC chip. For FPGA devices with even higher system clock speeds, such as Altera Stratix series that are said to be able to operate at 710 MHz, much faster data processing speed can be achieved.

6.3 Special Considerations

The CS8900A needs at most 135 ns before it can drive valid data onto the ISA bus. It also needs the active-low write enable to stay low for minimally 110 ns for one data fetch to be finished. After the write enable becomes inactive and before it can be active again, it has to stay high for minimally 35 ns. Since the CS8900A has its own system clock and works asynchronous, other circuitry cooperating with it cannot work at a faster speed. In the tests the FPGA device was actually working at a frequency of around 8 MHz by dividing the 33 MHz frequency of the supplied system clock by 4.

As mentioned before, the MPLS functional block was designed originally to cooperate with the Intel MAC chip IXF440, whose on-chip memory can be accessed at a clock speed provided externally. For MAC controllers as this IXF440, one multi-port integrated interface module is enough for all the 8 physical ports to be served since there are separate system clocks for IXF440 Media Independent Interface (MII) and FIFO interface. Thus IXF440 on-chip FIFOs can be accessed through its FIFO interface several times faster than IXF440 accessing each 10Base-T port through its MII. This ensures that each physical port can be served to its full bandwidth requirement easily.

However, for MAC controllers such as CS8900A, the fastest data fetching frequency that can be obtained is about 4 MHz only because each valid read/write performed on the CS8900A requires 250 ns or longer to complete. If the 8 physical

ports receive their services one by one in turn and each at a constant frequency of 4 MHz, taking the time each port waits in idle into consideration, the actual speed of data transfer service for either transmit or receive that each CS8900A can obtain is only 500 KHz. This means the bit rate each port can accommodate is only about 8 Mbit/s (500 K/s * 16 bit = 8,000,000 bit/s). Even though the traffic over a 10M Ethernet cannot be always at the peak of 10M bit/s, a node processing speed of maximally only 8 Mbit/s is far from acceptable. Therefore, when adopting MAC chips such as the CS8900A to build the test bed, a dedicated interface module for each physical port has to be adopted in order to transmit and receive packets at a speed without causing node processing performance decline. For an 8 port integrated system, 8 interface modules that are simply replicates of each other are required. Hence the feature of preventing any data loss caused by the service turn transition is still kept, which promises that each interface works independently.

Label processing service for each port was scheduled in a UD-WRR manner as described in Chapter 4, which took place completely within the MPLS functional block. The UD-WRR service scheduler was configured to work in its normal mode; the scheduler assumed that all the 8 ports existed and when they needed, they could be granted services regularly. Under the UD-WRR policy, when any one of the 8 receive/transmit buffers (each corresponds to a fixed physical port represented by a CS8900A) is served, the other 7 buffers have to wait in idle. Thus, the absence of the other 7 buffers appears to the service scheduler like they just don't have anything to be sent or have no more room to hold received data. The signals *tx_want* and *mpls_ready* signals from these 7 buffers are always driven inactive in order to enable the scheduler to only serve the port really that exists but to skip the service intervals granted to these absent ones who don't require any service. Hence, the complete service process is exactly the same as when there are 8 actual ports, which includes: reading a packet from one of the 8 receive buffers; removing the incoming MPLS label; finding out a appropriate new outgoing label; binding this new label to the packet under service; buffering the packet bound with the new MPLS label into the transmit buffer and finally starting the service interval for the next object.

Since there were only two CS8900A chips available to carry out the tests, the test results obtained were actually from the implementation containing only one interface module within each node. However, due to the design symmetry of the 8 integrated interfaces, each port is independent from the others and from the perspective of a particular physical port, the absence of other physical ports does not affect it.

According to what has been explained, if one port is proved to be served properly, the others can be inferred to be served in the same manner as well. Therefore, the tests performed on this one port implementation are theoretically convincing enough to demonstrate the performance of the integrated 8-port design. In fact, this saved a lot of extra money in building a test bed with 8 real physical ports, which does not necessarily provide more satisfying results.

Chapter 7 Conclusions and Future Work

7.1 Conclusions

In this thesis, MPLS standards and switch/router evolution were investigated first and then a novel idea named Real Packet Switching and its implementation were proposed. The RPS architecture can realize pipelined data transfer at the outputs. In the RPS, unlike the traditional crossbar switching that actually performs cell switching, packets are not segmented into cells at the inputs and then no cells have to be reassembled at the ouputs. Hence both processing time and hardware resources required by the traditional crossbar fabric can be saved. Over each connection, a packet can only be transfered upon the completion of the last packet.

The thesis then discusses the problem of the multiple queue service scheduling. Following a background introduction, an improved UD-WRR policy bearing several attractive attributes was proposed based on the WRR policy. The effective and easy-to-implement multiple queue service schedulng policy UD-WRR maintains a set of prioritized FIFO queueus to deal with the bandwidth allocation issue and diverse QoS guarantees in tomorrows' networks fairly and efficiently. After primary algorithm analysis was done, the two most significant parameter expressions for practical system implementations were developed.

Taking the full system throughput as "1", a session i (assigned the weighted factor w_i) packet is always guaranteed a throughput greater than or equal to $\frac{w_i}{\sum_j w_j}$. Also, the design is very flexible, since the values of w_i's can be modified to achieve different system performance if QoS requirements are changed. Finally, data processing may be done continuously even if the packet data have not arrived completely when the packet length is provided at the beginning of the packet.

In fact, when the unit of w_i is set to "bit" and each session has the same w_i value as 1, the UD-WRR policy reduces to the bit-wise round robin; when the unit of w_i is set to "packet" and each session also has the same w_i value as 1 (no matter how long the packet is), the UD-WRR policy reduces to the packet-based round robin policy that is usually implemented in software.

By adopting the hardware and software co-design technique, MPLS protocol partitioning and scheduling for execution on both a general-purpose processor and stream-based hardware were carried out. Accordingly, the MPLS data forwarding plane was implemented in hardware in this project and the data routing plane was left for future software implementation.

Based on all the investigations and analysis, a primary MPLS node concerning only the lower 2 layers of the TCP/IP model and running the UD-WRR scheduling policy was implemented in reconfigurable hardware.

As the major part of the system, the MPLS functional block contains 6 sub-layer modules: Receive Buffers; Transmit Buffers, Label Removing, Label Binding and Switching, Lookup Table, and State Machines/Service Schedulers. Together with the MAC-MPLS-Interface design, the complete design took up 7, 413 logic elements (approximately 177,840 typical gates) and 113,500 RAM bits. The adopted FPGA device can operate at a clock speed of 33.3 MHz. With 32-bit wide data path, the system can easily realize 100 Mbit/sec data transfer with the proper MAC chip. For FPGAs with even higher system clock speed, such as Altera Stratix series whose system clock is claimed to reach the frequency of 710 MHz, much faster data processing speed can be achieved.

Though this project was mainly focused on digital hardware design, a fundamental reconfigurable MPLS router architecture adopting basic RHFEs that could perform reconfigurable MPLS functions was also presented. This architecture is flexible in system upgrades of both new protocols and service add-ons.

To verify the correctness of the digital hardware design, appropriate tests have to be taken. As described in Chapter 5, a simplified test methodology was developed with limited available test equipment and was carried out successfully. The obtained test results demonstrated in Chapter 6 showed that all the circuits functioned properly as expected and realized line-speed switching that took over a great part of the burdens of traditional routing.

7.2 Future Work

Before the reconfigurable MPLS router can be put into practical use, there is still much work to do to uniquely integrate the best features of work being conducted in software and run-time reconfigurable hardware.

- ***More lower layer protocols to be supported***

In this project, a separated MAC chip CS8900A was used. Part of future work is to design an on-chip system that supports different lower layer protocols such as Frame Relay, SDH/SONET and ATM, in addition to Ethernet. According to the way the network is organized, various types and numbers of integrated MPLS-MAC interfaces could then be combined. With RHFEs that integrate MPLS and different lower layer interfaces on a single FPGA chip, more hardware reconfigurability, faster processing speed, lower fabrication cost and smaller product size can be obtained.

- ***Adoption of the embedded microprocessor***

A RISC microprocessor is needed to run some low-level software routines to enable communication between layer 2 and higher layers. This microprocessor is supposed to be embedded within the same FPGA device and communicate with the MPLS functional block directly.

The embedded microprocessor suggested for the future use is Altera Nios embedded system. The Nios development kit allows for a Nios embedded

microprocessor to interface other user logic designs within an FPGA device via a software-controlled parallel I/O port or via a hardware-realized *user-defined* interface.

The routines run in this embedded microprocessor could be compiled using C/C++ compiler, and then be downloaded into the on-chip ROM of the Nios microprocessor residing within the FPGA device.

- ***Routing Software Design and More Tasks***

Currently no dynamic routing has been considered. In future work, software programs performing label distribution to set up, maintain and tear down LSPs according to various QoS and traffic engineering requirements are to be designed.

More tasks to be completed include: line cards printed circuit board design, the concrete way in which all kinds of line cards are connected, and the back-plane design, etc.

References

MPLS

[1] MPLS Charter, IETF, http://www.ietf.org/html.charters/mpls-charter.html

[2] Rob Redford, "Enabling Business IP Services with Multiprotocol Label Switching", white paper of Multiservice Switching Business Unit, Cisco

[3] Multiprotocol Label Switching (MPLS), International Engineering Consortium, http://www.iec.org/online/tutorials/mpls

[4] MPLS and Next Generation Access Networks, Integral Access, Inc

[5] MPLS Guide, http://www.techguide.com Ennovate Networks, Inc

Switch/Router

[6] K. Toda, K. Nishida, E. Takahashi, N. Michell, and Y. Yamaguchi, "Design and Implementation of a Priority Forwarding Router Chip for Real-time Interconnection Networks", International Journal of Mini and Microcomputers, Vol. 17, No. 1, pp. 42-51, 1995

[7] David C. Lee, Scott F. Midkiff, Peter M. Athanas, " Reconfigurable Routers: A New Paradigm for Switching Device Architecture", April 27, 1998

[8] S. Keshav and R. Sharma, "Issues and Trends in Router Design", IEEE Communications Magazine, Vol. 36, No. 5, pp. 144-51, May 1998

[9] N. Yamanaka, E. Oki, H. Hasegawa, and T. M. Chen, "User-Programmable Flexible ATM Network Architecture Active-ATM-Experimental Results", Third IEEE Symposium on Computers & Communications**,** pp. 178-182**,** June 30-July 02, 1998**,** Athens, Greece

[10] David C. Lee, Scott J. Harper, Peter M. Athanas, and Scott F. Midkiff, "A stream-based Reconfigurable Router Prototype", IEEE International Conference on Communications, Vancouver, BC, pp. 581 – 585, June 1999

[11] Ranjita Bhagwan and Bill Lin, "Design of A High-speed Packet Switch With Fine-Grained Quality-Of-Service Guarantees", *IEEE International Conference on Communications (ICC'00)* 2000, New Orleans, Vol. 3, pp. 1430-1434, June 2000.

[12] Jun Gao, Peter Steenkiste, Eduardo Takahashi, and Allan fisher, "A Programmable Router Architecture Supporting Control Plane Extensibility", IEEE Communications Magazine, pp. 152-159, March 2000

[13] Scott Karlin and Larry Peterson, " VERA: An Extensible Router Architecture", IEEE OPENARCH 2001, pp. 3-14, April 2001

[14] Tilman Wolf and Jonathan Turner, " Design Issues for High Performance Active Routers", IEEE Journal on Selected Areas of Communications – Special Issue on Active and Programmable Networks, Vol. 19, No. 3, pp. 404-409, March 2001

Queuing Theory

[15] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm", J. Internetworking: Research and Experience, pp. 3-26, September 1990

[16] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case", IEEE/ACM Trans. Networking, Vol. 1, pp. 344 -357, June 1993

[17] Hui. Zhang and Jon. C. R. Benett, "Why WFQ is not good enough for integrated services networks", Proceedings of NOSSDAV 96, pp. 87-96, April 1996

[18] F. M. Chiussi and V. Sivaraman, "Achieving High Utilization in Guaranteed Services Networks Using Early-deadline-first Scheduling", Proceedings 6th IEEE IWQoS'98, pp. 209-217, May 1998

[19] Jorg Liebeherr and Dallas E. Wrege, "Priority Queue Schedulers with Approximate Sorting in Out-Buffered Switches", IEEE Journal on Selected Areas in Communications, Vol. 17, No. 6, pp. 1127–1144, June 1999

[20] Minseok Song, Naehyuck Chang, Heonshik Shin, and Kenji Toda, "A New Queue Discipline for Various Delay and Jitter Requirements in Real-Time Packet-Switched Networks", Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications, pp. 191-198, December 2000

[21] Aggelos Ioannou and Manolis Katevenis, "Pipelined Heap (Priority Queue) Management for Advanced Scheduling in High-Speed Networks", 2001 http://archvlsi.ics.forth.gr/muqpro/heapMgt.html,

[22] Manolis Katevenis, Stefanos Sidiropoulos, Costas Courcoubetis: ``Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip", *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 8, pp. 1265-1279, October 1991

[23] Ying Jiang and Mounir Hamdi, "A Fully Desynchronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture", 2001 IEEE Workshop on High Performance Switching and Routing, pp. 407-411, 2001.

Queuing Implementation

[24] Jennifer L. Rexford, Albert G. Greenberg, and Flavio G. Bonomi, "Hardware-Efficient Fair Queuing Architectures for High-Speed Networks", INFOCOM, pp. 638-646, March 1996

[25] Sung-Whan Moon, Jennifer Rexford and Kang G. Shin, "Scalable Hardware Priority Queue Architectures for High-Speed Packet Switches", IEEE Transactions on Computers, Vol.49, No.11, pp. 1215-1227, November 2000

[26] Ranjita Bhagwan and Bill Lin, "Fast and Scalable Priority Queue Architecture for High-Speed Network Switches", *IEEE Infocom*, Tel Aviv, Vol. 2, pp. 538-547, March 2000

[27] George Kornaros, Christoforos Kozyrakis, Panagiota Vatsolaki, and Manolis Katevenis, "Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow-Control", Proc. of 17th Conf. on Advanced Research in VLSI (ARVLSI'97), Univ. of Michigan, Ann Arbor, USA, September 1997
URL:ftp://ftp.ics.forth.gr/tech-reports/1997/1997.ARVLSI.Pipe MultiQueue.ps.gz

[28] Dimitrios S. Kapsalis, "Design and Implementation of a Per-Flow Queue Manager for an ATM Switch using FPGA technology", Technical Report 302, Institute of Computer Science (ICS), Foundation of Research & Technology ñ Hellas (FORTH)

[29] Aggelos D. Ioannou, "An ASIC Core for Pipelined Heap Management to Support Scheduling in High Speed Networks", Technical Report FORTH-ICS/TR-278 October 2000". Work performed as a M. Sc. Thesis at the Univ. of Crete

Data Sheets

[30] Cisco 10000 Edge Services Router Hardware Architecture, Cisco Systems, 2000

[31] Alcatel 7670 Routing Switch Platform data sheet, Alcatel, 2001

[32] M40 Internet Backbone Router data sheet, Juniper, 2001

[33] Marconi ASX4000 data sheet, Marconi, 2001

[34] Cirrus Logic CS8900A product data sheet, 2001

[35] IBM Corp., "IBM 8265 ATM Switch Overview", White Paper, September 1997

Other

[36] Fundamentals of digital switching / edited by John C. McDona

[37] James Aweya, "IP Router Architectures: An Overview", Nortel Networks

[38] S. Keshav: *An Engineering Approach to Computer Networking*, Addison-Wesley, 1997, ISBN 0-201-63442-2

[39] Steve Lin and Nick Mckeown, "A Simulation Study of IP Switching", Proceedings of ACM SIGCOMM, pp. 15-24, September 1997

[40] ANN Hardware Implementations, PhD thesis by Mike Craven, University of Nottingham, December 1993
http://www.crg.cs.nott.ac.uk/people/Mike.Craven/Mikepub.html