INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

University of Alberta

CONTENT ADDRESSABLE MEMORY-BASED CIRCUITS FOR INTERNET PROTOCOL ROUTERS: A CACHE AND A LOOKUP TABLE

by

Soraya Kasnavi

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2005

Abstract

Packet forwarding is a fundamental task for an Internet router. A routing lookup table (LUT) is used to decide where to forward a packet at each router. The routing lookup is a rather complicated and slow process and the lookup delay is a bottleneck in high throughput routers. An effective strategy to speed up routing lookup is to use a cache to store recent routing results for reuse. This thesis proposes an efficient forwarding mechanism with a hardware-based LUT and a Multizone Pipelined Cache.

The LUT is implemented with a pipelined TCAM (Ternary Content Addressable Memory). Our TCAM employs a novel Hardware-based Longest Prefix Matching (HLPM) to completely eliminate table management requirements and to reduce the power consumption for short matching prefixes. The cache is a multizone non-blocking pipelined cache for IP routing lookup that achieves lower miss rates compared to previously reported IP caches and reduces the effective miss penalty by using a very small non-blocking buffer. The simulation results of our forwarding mechanism, based on real traffic, demonstrate the efficiency of the design.

Acknowledgements

I would like to especially thank Dr. Vincent C. Gaudet for his guidance, supervision and funding for the work presented in this Thesis. I would also like to thank Mr. Paul Berube, Dr. José Nelson Amaral and Dr. Mike MacGregor from the Department of Computing Science, University of Alberta for providing data (IP traces and lookup tables), priceless discussions and consulting.

I would like to thank my parents and my brothers for their non-stop support and patience throughout my life, specially during past three years. I would like to thank my husband, *Hanif*, for his positive and encouraging comments and his care and support for my academic goals. Finally, I would like to thank the International Center of the University of Alberta for helping me enjoy my social life as well as my academic life at the University of Alberta and feel like home in Canada.

Thank you all.

Contents

List of Tables

List of Figures

Acronyms

Acronyms	Definition
AHAL	Active High Active Low
CAM	Content Addressable Memory
CIDR	Classless Inter Domain Routing
DL_i	Data Line *i*
FIFO	First Input First Output
FPGA	Field Programmable Gate Array
Gbps	Giga Bit Per Second
HLPM	Hardware-based Longest Prefix Matching
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider
LAN	Local Area Network
LFU	Least Frequently Used
LPM	Longest Prefix Matching
LRU	least Recently Used
LSB	Least Significant Bits
LUT	Look Up Table
ML	Match Line
MPC	Multizone Pipelined Cache

Mpps	Mega Packet Per Second
MSB	Most Significant Bits
OC	Optical Carrier
OCU	Out-of-order Cache Update
OMB	Outstanding Miss Buffer
QoS	Quality of Service
RAM	Random Access Memory
SL_i	Search Lines
Tbps	Tera Bit Per Second
TCAM	Ternary Content Addressable Memory

List of Symbols

Symbol	Definition
C	Capacitance
f	Operation Frequency
$O(N)$	Order of N
V	Voltage
V_{DD}	Source Voltage
V_{Tn}	Threshold Voltage for the NMOS Transistor
V_{Tp}	Threshold Voltage for the PMOS Transistor

Chapter 1

Introduction

With increasing use of the Internet, a more robust, fast, reliable and secure backbone network is required. Some applications such as Voice over IP or QoS, generate huge amount of complexity as well. The backbone network is required to transfer and process Internet data as fast as possible. Internet routing is one of the fundamental tasks done in a Network. Routers receive a packet of data and decide where to forward it. Since this routing process is done at each router the packet visits from its source to its destination, the routing speed is an important parameter of the whole network performance.

In this thesis a high throughput-power efficient forwarding mechanism is proposed, discussed and simulated. The forwarding mechanism includes a routing lookup table and a routing cache. Content Addressable Memory (CAM) is desirable for the cache and the lookup table implementations. The CAM based devices are desirable due to their high speed (they can perform a parallel search for a specific pattern, in all entries in a single memory access), but are disadvantageous in terms of power consumption and area requirements. The hardware presented in this thesis, both for the main lookup table and for the cache, saves power through smart search operations. The lookup table is implemented with a pipelined Ternary CAM (TCAM). It resolves the Longest Matching Prefix with no table management and requires less area compared to other TCAM-based lookup tables. The lookup table saves power for short matching prefixes as well as for non matching prefixes.

The other part of the design is the routing cache which is a half-prefix half full address cache. Our cache is a Multizone Pipelined Cache (MPC) which has lower miss rates compared to a full address cache and requires less table expansion compared to a full prefix cache. We propose two novel table expansion methods in order to reduce the table size as well as to increase the locality of the data stored in the cache. Short Prefix Expansion (SPE), fully expands the table for short prefixes. MPC uses SPE for table expansion. Also, we propose an Expansion Free (EF) method for software based lookup tables to generate cacheable prefixes during lookups and forward those prefixes to the cache. The EF method fully eliminates table expansion requirements for prefix caches with software based lookup tables. MPC also employs a very small non-blocking buffer and reduces the effective cache miss penalty.

This thesis dissertation is organized as follows. A brief background on IP forwarding and Internet router architectures is given in Chapter 2. Also the motivations for designing a more efficient forwarding mechanism are discussed. Chapter 3 relates to CAM based solutions for fast routing lookups. The main CAM functionality is explained and a brief description of several previously reported low power CAM designs is given. Chapter 4 describes the novel hardware solution proposed in this thesis for a high throughput routing lookup. Chapter 5 describes the novel cache design used in our forwarding mechanism. The complexities of routing caches and their impact on routing table implementations are discussed in detail. The simulation and evaluation results of our forwarding mechanism are given in Chapter 6. Finally Chapter 7 concludes the thesis dissertation and presents future research directions.

Chapter 2

Background and Motivation

The Internet has become a part of everyday life for many people since the early 1990s. The Internet mainly provides information exchange and communication services such as web browsing and e-mail, from one point to another. The backbone network is built of high capacity transmission, multiplexing and signal switching facilities to provide transmission paths for logical connectivity and requirements for all Internet services. Figure 2.1 depicts an example of an Internet network. Internet routers are responsible to route packets from their sources to their destinations. The backbone routers (core routers) are responsible for routing the aggregated traffic of all users connected to the network. The increase in Internet traffic over the last decade has necessitated faster and faster backbone networks. First generation routers could support up to 0.5-Gbps. Currently, 160-Gbps to 20-Tbps routers are in development [5]. An OC-48 (Optical Carrier) backbone network can support up to 2.5-Gbps of aggregated data. Each light-path could be formed as 4 OC-48s to carry an 192c (10-Gbps aggregation of traffic) [6]. Enterprise routers (edge routers) aggregate the local user traffic (*e.g.* a LAN of a Company). Internet Service Providers (ISP) are the connection points of users to the high capacity backbone network. Routers closer to the backbone require higher capacity, are more complex and more expensive.

The data is carried in the form of packets routed across the network. The most popular inter-network transport protocol is the Internet Protocol (IP). An IP packet consists of a

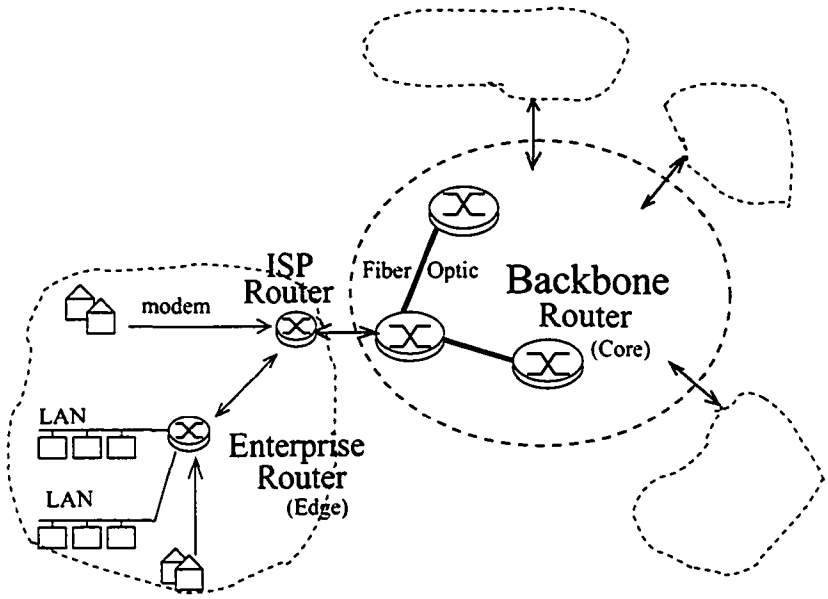
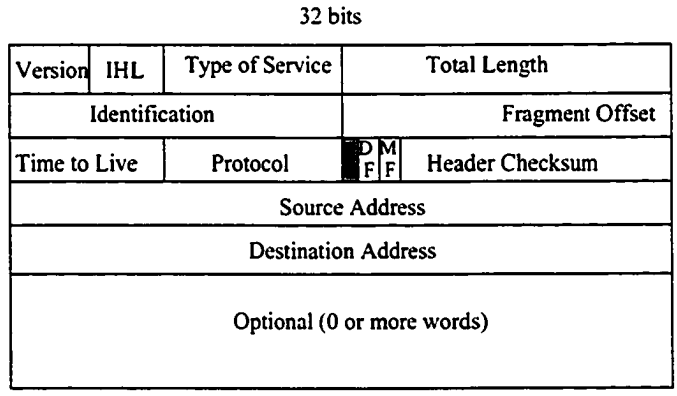


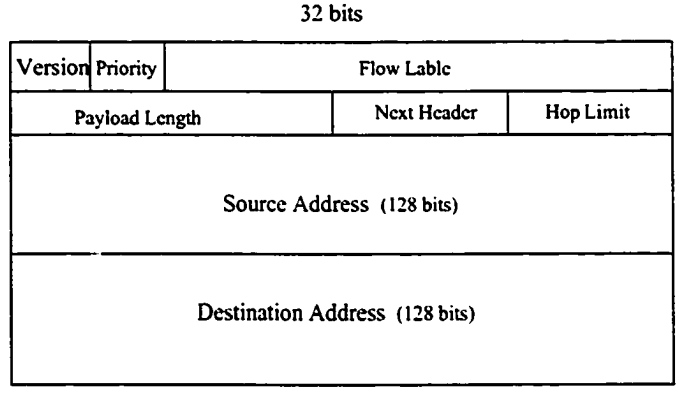
Fig. 2.1. An Example of a Network

header part and a payload part. The header format of IP version 4 (IPv4) is shown in Figure 2.2(a). Each host connected to any public Network has a unique IP address. All IPv4 addresses are 32 bits long and are used in the *Source Address* and *Destination Address* fields of the header of IP packets. Usually IPv4 addresses are broken into 4 groups, 8 bits each, represented as four decimal numbers separated by dots (*e.g.* 129.128.0.0). Each IP address is composed of a Class identifier, a Network number and a Host number. IPv4 addresses have three main classes (Classes A, B and C). The class of an address is coded in the Most Significant Bits (MSB) of the address and represents the size of a Network. Network size is determined by the number of bits used to represent the network and host parts. Thus, networks of class A, B, or C consist of an 8, 16, or 24-bit network part and a corresponding 24, 16, or 8-bit host part. Thus, the decimal presentation, as explained above, is suitable for presenting addresses in all classes. With this addressing scheme, known as *Classful Addressing*, Internet routing is simply a two-level hierarchy with three possible network sizes. All the hosts connected to a single network share the same Network Number in their IP address. This hierarchy in IP addresses allows Internet Protocol to interconnect networks.

With the exponential growth in the number of hosts and networks, classful addressing



(a) IPv4



(b) IPv6

Fig. 2.2. Internet Protocol Header format.

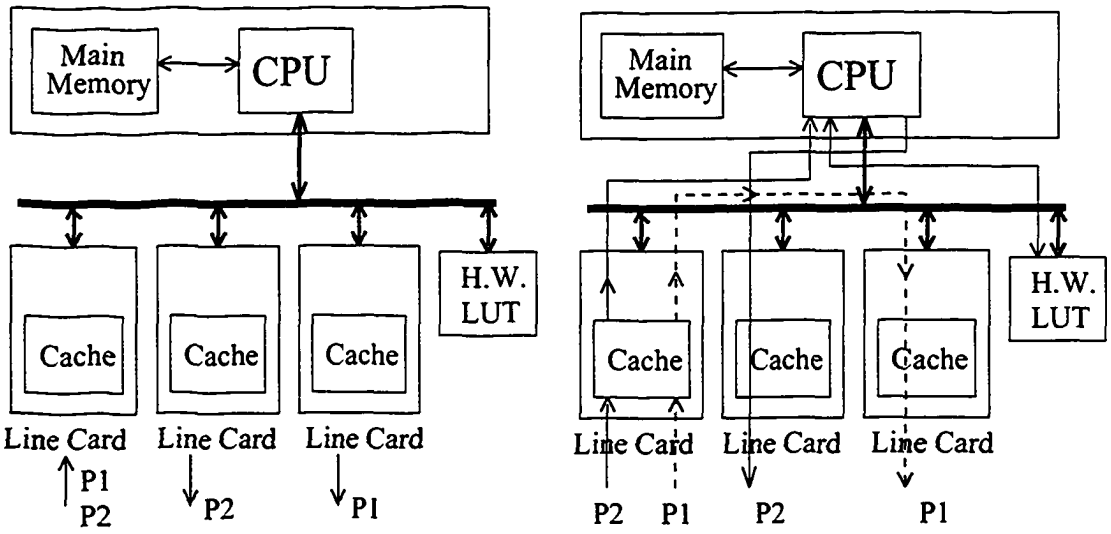
appears to be very inefficient. Since only three network sizes are allowed (Classes A, B and C), IP addresses are not used efficiently. Even the smallest network size allowed might be too large for a network. Thus there might be many IP addresses that are assigned to a network that cannot be assigned to any other host but are not used at all. It was observed that Internet Protocol was running out of IP addresses, even though only a small fraction of allocated addresses were actually used. Also, routing tables stored in routers were getting very large. This is due to an exponential increase in the number of networks. *Classless Inter Domain Routing (CIDR)* has since been introduced to allow more efficient use of IP addresses and slow down the growth of backbone forwarding tables [7]. In CIDR the

network size can be variable to match with a network. Assume a network requires 156 addresses. In classful addressing scheme, a class C with 256 addresses must be assigned to this network. CIDR assigns a subset of 128 addresses and a subset of 32 addresses to this network, providing 160 addresses. Thus CIDR uses IP addresses more efficiently. To provide more IP addresses, IP version 6 (IPv6) has been adopted. It can support a lot more addresses even with inefficient space allocations using a 128-bit addressing scheme. Figure 2.2(b) depicts the IPv6 header. A detailed description of other parts of the header is given in [8]. However, CIDR allows address aggregation at several levels. The address aggregation reduces the number of entries in the router forwarding tables as explained in detail in Section 2.1.

Internet routers receive a packet from an input line, extract the destination address, perform packet processing (*e.g.* packet forwarding and classification) and finally, forward the packet to its destination. Figure 2.3(a) depicts the architecture of a general bus-based router. In this example, Packet 1 (P1) and Packet 2 (P2) arrive at the first port from the left. P1 is forwarded to the third port and P2 is forwarded to the second port.

Packet forwarding is a fundamental task in routing IP traffic. Routers lookup the destination address of each packet in their Look Up Table (LUT) and resolve which interface the packet should be forwarded to. Routers either store LUT information in the main memory and use a software method to perform the lookup or use dedicated hardware. More details on routing lookup are given in Section 2.1.

However, the routing lookup is a rather complicated and slow process. An effective strategy to speed up routing lookup is to use a cache to store recent routing results for reuse. Figure 2.3(b) describes the cache impact on forwarding. Assume that the cache on the first port stores the forwarding information of P1. When P1 arrives at the first port, it hits the cache. The forwarding information (the output port identifier) is fetched from the cache and the packet is directly forwarded to its output port (third interface). Since the cache does not have the forwarding information for P2, it misses the cache. The processor fetches the missing information from the lookup table (either by software methods or hardware) and then, updates the cache and forwards packet 2 to its output interface.



(a) (b)

Fig. 2.3. A General Description of an Internet Router. (a) depicts an example of the IP forwarding in a router in general, and (b) depicts the impact of the routing cache on IP forwarding.

Clearly, the cache hit ratio and the main lookup speed directly impact the packet forwarding efficiency. Thus it is quite beneficial to design efficient and fast lookup methods and to improve the cache hit ratio at the same time. In this Thesis, the lookup process and the previous implementations of routing caches and routing lookup tables are investigated. Finally, an efficient forwarding mechanism is designed, studied and simulated.

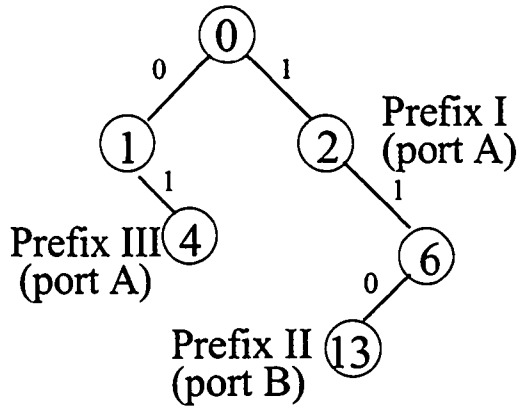
2.1 Routing Lookup

An Internet router performs a routing lookup in a Look Up Table (LUT) to forward a packet to its next hop to get it closer to its destination. The routing table stores routing prefixes rather than full destination addresses, in order to reduce the table size. A routing prefix corresponds to a number of Most Significant Bits (MSB) of an IP address followed by *don't cares*. For example, a Classful IPv4 prefix is comprised of the Class Number and the Network Number of a network followed by don't cares. Thus the address of all the

hosts of that network are covered by a single prefix. An address is searched in the LUT and the matching prefix is found. The corresponding forwarding information is fetched from the table. As mentioned before, in classful addressing architectures, the prefixes represent the networks. Prefixes in the forwarding table are organized in three separate tables and an exact match of the class part of an IP address resolves which table to access. With Classless Inter-Domain Routing (CIDR), routing prefixes (Network sizes) may have variable lengths. In the case where multiple prefixes match with an address, the longest matching prefix is the correct result. Consequently, routers must perform Longest Prefix Matching (LPM) when searching the routing table. For example, consider the networks represented by the network numbers from 208.12.16/24 through 208.12.31/24. 208.12.16/24 represents a prefix of size 24 bits. This prefix covers all the addresses from 208.12.16.0 to 208.12.16.255. Suppose that all these network addresses are reachable through the same router. From the binary representation we can see that the leftmost 20 bits of all the addresses in this range are the same (11010000 00001100 0001). Thus, these 16 networks can be aggregated into one supernet represented by a 20-bit prefix of 208.12.16/20. While a great deal of aggregation can be achieved if addresses are carefully assigned, in some situations a few networks can interfere with the process of aggregation. For example, suppose now a customer owning the network 208.12.21/24 changes its service provider and does not want to renumber its network. In this situation, either 16 prefixes (representing 208.12.16/24 to 208.12.31/24) must be stored in the lookup table or these prefixes could be aggregated in spite of the exception networks and additionally storing entries for the exception networks. In our example, this will result in only two entries in the forwarding table: 208.12.16/20 and 208.12.21/24. In this case, some addresses match with both prefixes because prefixes overlap. In order to always make the correct forwarding decision, routers need to find the most specific match, which is the longest matching prefix [7]. Since LPM is performed in every router along a packet's path from source to destination, routers require a fast mechanism to perform the lookup in order to maintain high throughput and low latency under load.

A simple example of a LUT for a 4-bit addressing scheme is given in Figure 2.4(a). The lookup result for address 1001 is prefix I and the corresponding port ID is port A. Address

Prefix	Port ID
Prefix I (1xxx)	Port A
Prefix II (110x)	Port B
Prefix III (01xx)	Port A



(a) (b)

Fig. 2.4. An Example of Longest Matching Prefix

1101 matches with both Prefixes I and II. Since prefix II is the LPM, the port ID is port B. Figure 2.4(b) depicts the LUT organized as a *trie*. A trie presentation of a lookup table is a tree-based scheme where the root of the trie corresponds to the most significant bit of the address. Branching right indicates that a bit is 1, while branching left indicates that a bit is 0. A complete trie enumerates every possible address. In order to reduce the space requirement of the trie, only the nodes required to form a path to each prefix are stored. Nodes are sequentially numbered from top to bottom and from left to right. Gray nodes represent prefixes stored in the lookup table.

The key elements in routing lookup efficiency other than the lookup speed include routing table update delay, power consumption and routing table memory footprint. Several lookup methods have been proposed to increase the efficiency of routing lookup [7]. In general, schemes that achieve high lookup speed or smaller memory footprint size, require more complicated implementations and suffer from slower routing table updates. Software solutions are slow (at least 4 to 6 memory accesses required for a lookup) compared to hardware solutions and do not easily scale up to 10-Gbps processing. Content Addressable Memory (CAM) is one of the available hardware solutions for IP lookup. A CAM is a fully associative binary memory capable of matching a specific pattern of data (a key) against all its entries in parallel. A TCAM can store and search for *don't care* values as well

as 0s and 1s. A *don't care* matches with both 0s and 1s during a search process. CAM based solutions are desirable due to their high speed (a single memory access resolves the lookup) but are not efficient in terms of power consumption and on-chip area compared to RAM. On the other hand to find the LPM of multiple matching prefixes, complicated table maintenance or management is required in CAM-based lookup tables. The details of CAM-based lookup table designs are discussed in Chapter 3.

2.2 Routing Cache

Caching forwarding information for IP addresses is an effective method to speed up IP forwarding in Internet routers. However, the performance of the cache depends on the characteristics of the IP traffic, such as its *temporal* and its *spatial* locality. Greater temporal locality increases the probability that destination addresses are frequently used, and thus increases the utility of a cached address. Spatial locality means referencing addresses in the same numerical range. When prefixes are cached, a single cache entry can cover a large number of destination addresses in the same numerical range. Therefore, the spatial locality in the traffic stream is converted to temporal locality in the cache access stream.

Figue 2.5 depicts a functional description of a routing cache designed by Berube *et al.* [1]. IP addresses or prefixes are stored in a *Destination Address Array* (DAA). Next hop information is stored in the *Next Hop Array* (NHA). The DAA and NHA are co-indexed, with one entry in the NHA corresponding to a single entry in the DAA. The NHA is implemented using standard SRAM technology. The DAA is implemented using a CAM or a TCAM if prefixes are cached.

A routing cache searches for an IP address in all entries of the DAA in parallel. If the address is found, a cache hit occurs. The Next Hop identifier (output port) is read from the NHA, and the packet is directly forwarded to the output port. If no entry in the DAA matches the IP address, a cache miss occurs. In this case, a lookup in the full routing table is performed (a software or a hardware lookup) and the cache is updated with the new destination address/next hop pair.

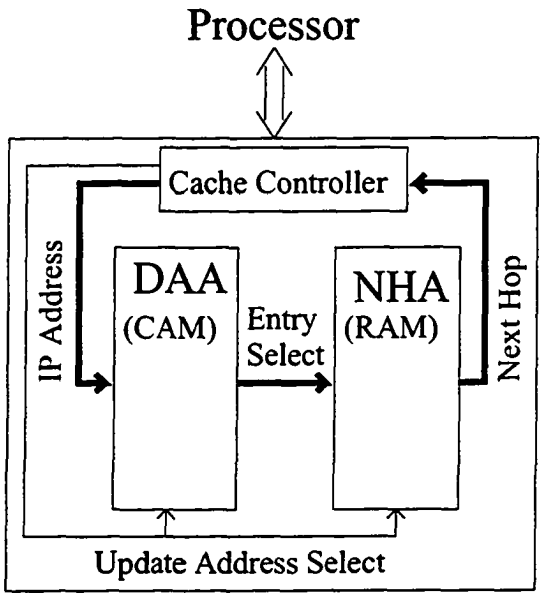


Fig. 2.5. A Functional View of A Routing Cache [1]

Generally the cache impact on the total performance of a system is directly dependent on the miss rate and the miss penalty of the cache. The lower the miss rate the better is the total performance. Several designs have been proposed to reduce the miss rate of routing caches. The next section briefly describes the related work to routing cache design.

2.2.1 Related Work

Many researchers have addressed the efficiency of routing caches. Some studied existing locality in IP traffic [9–11]. Others designed more efficient caches for IP routing [12–15]. In 1988, Feldmeier demonstrated that a routing-table cache could reduce the lookup time in network gateways by 65% [9]. He believed that a fully associative cache provides the best performance and ran his simulations for a fully associative cache. He also observed that a packet from host A to B is very likely to be followed by a packet from B to A. Thus he suggested two different schemes of storing data: Destination/Source or only Destination addresses and demonstrated his observation was correct.

Chiueh *et al.*designed a CPU style IP caching scheme and demonstrated that general-purpose processors can serve as a powerful platform for high performance IP routing [13].

However, the data streams presented to the network processors have very different characteristics than the streams accessed by general-purpose CPUs. Thus, the cache design must be considerably different and the cache coverage must be improved to achieve acceptable performance [16]. This study shows that although enough temporal locality exists (based on real packet traces), the cache coverage should be improved to achieve acceptable performance such as caching address ranges rather than individual addresses. Also they demonstrate, due to very poor spatial locality cache block sizes should be small, preferably one entry. Talbot *et al.* [11] demonstrate that caching IP destination addresses is an effective speedup for IP lookup in high speed routing. They also show the lowest bits of the IP addresses are the most random suitable for indexing their CPU style cache. Shyu *et al.* [10] also analyze the temporal locality in IP addresses and observe strong locality in backbone routers.

Liu [14] completes the idea of caching address ranges instead of caching full destination addresses (e.g. 32-bit destination address for IPv4) by using the term *IP Prefix Caching* and demonstrates higher locality in prefixes resulting in lower miss ratio for *IP Prefix caching* compared to full IP caching. One potential problem with this scheme is multiple prefix hits. The choice of what routing prefix to return is critical because if only one of those matches is cached wrong lookup results may occur. Thus an algorithm must be applied to the table to make sure this problem is avoided. Liu suggests 3 different possible algorithms of which one of them increases the size of the lookup table dramatically by expanding the lookup tree completely. His second solution simply looks like the conventional IP full address caching and the third is a hybrid of the two. The details of his method are further discussed in Chapter 5, Section 5.2.

Improving the *Replacement Policy* of an IP cache is another way to reduce the miss rate [10, 17]. Feldmeier demonstrated that FIFO (First Input First Output) performance is almost as good as LRU (Least Recently Used) when large caches are used. But FIFO is very poor for small caches [9]. An LFU (Least Frequently Used) replacement policy is found to outperform FIFO and LRU [10]. H. Liu presents the multi-segment LRU (mLRU) replacement policy applied to prefix caches. mLRU aims to combine LRU and LFU together but

it is complicated to implement and maintain [17].

2.2.2 Multizone Cache

A cache naturally exploits temporal locality. However, routers manage traffic from a large number of hosts. In some cases, only part of the traffic has high locality. In a router with a single cache, low-locality traffic pollutes the cache with low-utility entries. These entries reduce the effectiveness of the cache for all traffic, and may cause thrashing. However, if the cache is split then the cache performance is improved [12, 15]. One portion of a split cache stores addresses or prefixes associated with shorter routing prefixes, and the other portion caches the addresses of prefixes associated with the longer routing prefixes. Such a *multizone* cache prevents the lack of locality in one portion of the traffic from polluting the locality in the rest of the traffic. The new cache design in [12] shows miss ratios approximately one-half those of conventional caches.

2.2.3 Cache Miss Penalty

IP caches have very large miss penalties because a miss requires a rather slow main table lookup. Complicated lookup techniques can be applied to the main table to increase the lookup speed. However, these techniques dramatically increase the table updating delays. Thus, improving the cache miss ratio could compensate for the large cache miss penalty and allow a simple main lookup table to provide fast table updates.

Non-blocking general purpose processor caches hide memory latency by overlapping the processor computations with memory data accesses [18]. Special registers are used to hold information about each cache miss. The processor can then overlap the service of a cache read miss with the execution of subsequent instructions [19, 20]. Bhuyan *et al.*used execution-driven simulation to study the impact of instruction level parallelism (ILP) and cache architectures on the performance of routers [21]. They observed up to 37% improvement for their traces due to multiple issues, out of order execution and non-blocking loads.

2.3 The Proposed Forwarding Architecture

In this thesis an efficient forwarding mechanism for Internet routers is designed, studied and simulated. Our design goals are to: (1) reduce the cache miss rate; (2) reduce the effective cache miss penalty; (3) provide simple and fast table updates; (4) lower the power consumption and (5) decrease the storage area requirements. Our design has two main parts:

1. An efficient lookup table (Figure 2.6(a)). We propose a pipelined TCAM with a novel Hardware-based Longest Prefix Matching (HLPM) technique to provide an efficient and fast hardware solution for routing tables. Chapter 4 describes our HLPM in detail.

2. An efficient forwarding cache (Figure 2.6(b)). We propose MPC, a Multizone, non-blocking, Pipelined Cache. MPC uses prefix caching in multiple zone caches to improve cache miss ratio. MPC adopts a non-blocking buffer to reduce the effective cache miss penalty. A pipelined design implements a novel search and reduces power consumption. The details of the MPC architecture and features are further described in Chapter 5.

2.4 Summary

In this chapter, the background on Internet routing has been discussed. The high speed requirements of Internet Routers motivates us to design an efficient forwarding mechanism in terms of throughput, power consumption and are requirements. The next chapter describes the basics of a Content Addressable Memory (CAM) which is used in several parts of an Internet router. The most important disadvantages of a CAM, the high power consumption and the management requirements, are described and a variety of existing solutions are presented in detail.

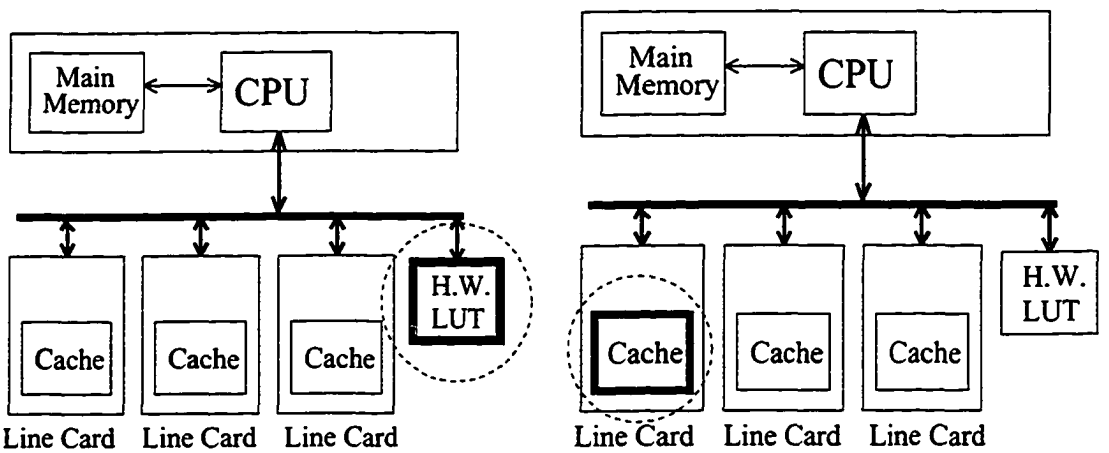


Fig. 2.6. The proposed forwarding scheme (a) Hardware-based Look Up Table and (b) the routing cache.

Chapter 3

Content Addressable Memory

3.1 Overview

A Content Addressable Memory (CAM) is a binary memory device that accelerates any application requiring fast searches of a pattern (or key) in a database or list such as database machines, voice or image recognition, fully associative caches or computer and communication networks. Specifically, a CAM is suitable for routing lookup implementations due to its fast and simple searching operations. Ternary CAM (TCAM) is capable of storing a third value as well as zeros and ones. This third value is a *don't care* value which matches with both zeros and ones during a search process. However, CAM-based design is challenging due to the high power consumption, cost and on-chip area requirements. This chapter gives a brief description of a CAM and its challenging design issues. Also, some previously reported low power CAM designs are discussed. Complications of applying a TCAM to CIDR lookups are explained as well, later in this chapter.

3.2 Content Addressable Memory (CAM) Architecture

Figure 3.1 compares a CAM to a RAM. A RAM provides the data stored in the location of the address given to the RAM. A CAM receives data (key or pattern) and provides the

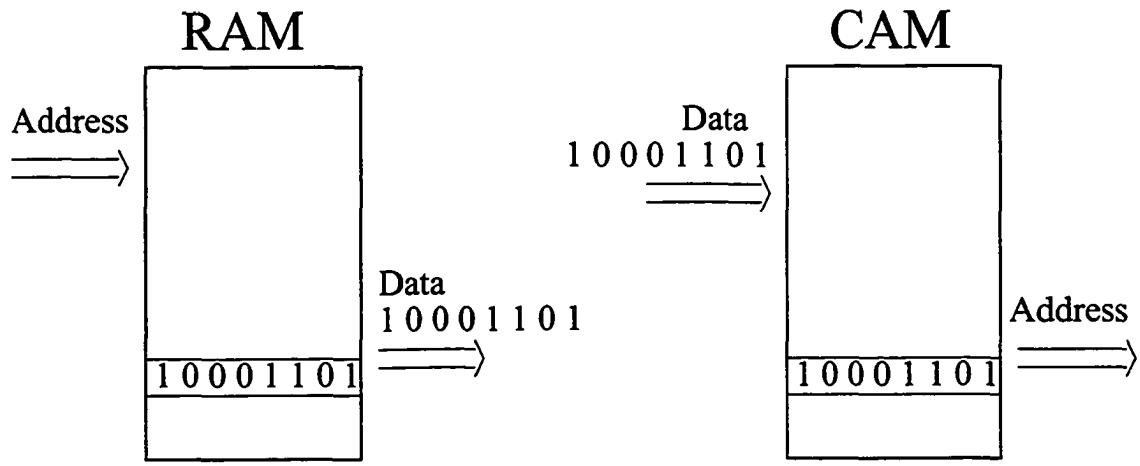


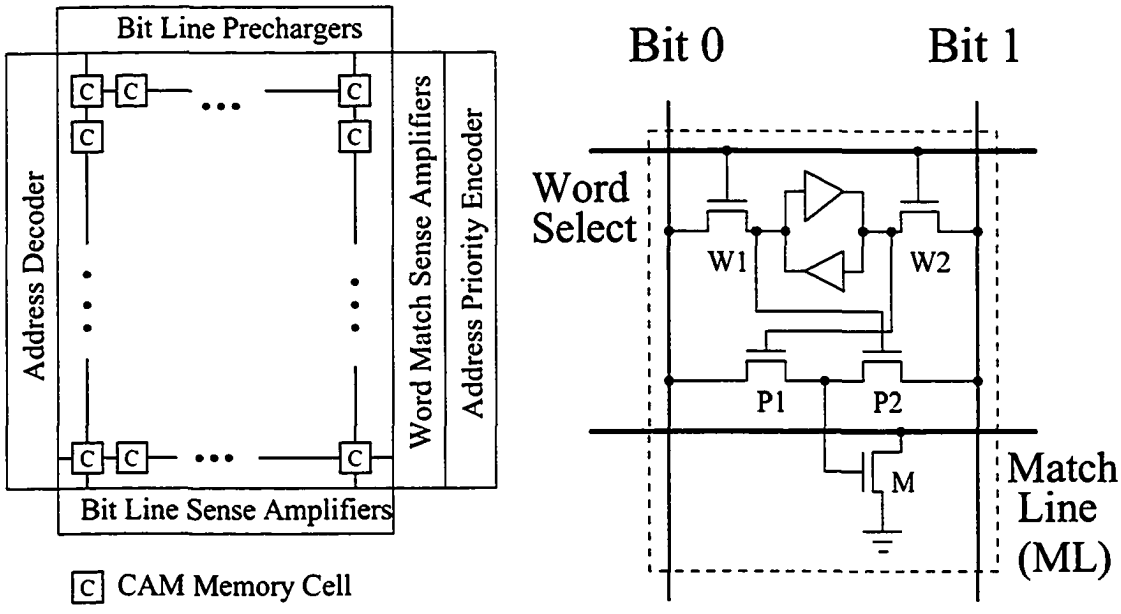
Fig. 3.1. CAM vs RAM

address of the location where data is found or generates a non-matching signal in case data does not match with any entry in the CAM.

A CAM performs a write and a read similar to a RAM, but it allows parallel searching for a specific pattern in all its entries as well. The pattern (key) is stored in a comparand register and is compared with the data stored in the CAM bit by bit. All entries are searched in parallel. During a search, if all the bits of an entry in the CAM match with the corresponding bits of the key, a match flag is set. An encoder generates the address of the location of the matching data. In a Ternary CAM, multiple different entries might match with a key, simply because *don't care* bits match with everything. In this case, a priority encoder is used to resolve the matching entry. Thus, a CAM/TCAM gets the content and produces the location.

Figure 3.2(a) depicts a block of CAM cells. The peripheral circuits of a CAM or a TCAM are similar to RAM as well. Bit Line Prechargers and Bit Line Sense Amplifiers are used to write or read a data to or from a CAM. The position of the entry to be written into or read from is selected by the Address Decoder. After each search operation, the Word Match Sense Amplifiers, detect the matching entry and the Address Priority Encoder generates the address of the matching entry.

A CAM cell is implemented by modifying a RAM cell. There are both static and dynamic CAM cells. A standard static CAM cell is depicted in Figure 3.2(b). The cross-



(a) Memory Block (b) Standard Static Cell

Fig. 3.2. Content Addressable Memory (CAM).

coupled inverters form a latch to store data statically. A CAM uses the pass transistors (W1 and W2) to perform read and write operations identically to those in a static RAM [22]. To write in a CAM entry, the location is selected by an address bus or data is simply written to the first empty entry. To write a new value in each cell, the value is asserted on the complementary bit lines (Bit 0 and Bit 1) while the Word Select Signal is raised. To read a data from a cell, the bit lines are precharged, then the Word Select Signal is raised and finally the Bit Line Sense Amplifiers detect a discharge on one of the bit lines, representing either a one or a zero value stored in the cell.

A CAM performs a search operation with an exclusive OR comparator formed by two pass transistors (P1 and P2) and a pull-down transistor (M). The value to be searched (key) is asserted on the Bit Lines and the Match Line is precharged. If the key and the data stored in the cell do not match, a path from the Match Line to ground is created and the Match Line discharges. When a pattern is searched in a row of memory cells (*word line*), the Match Line discharges to ground if one or more cells do not match with their corresponding bits

in the pattern. If a Match Line of a row, does not discharge at the end of a search operation, the Match Sense Amplifiers sense a Match and the priority Encoder generates the location address of the match. A priority encoder is used to resolve the match in case more than one entry matches with the data. This is specifically useful for Longest Prefix Matching in CIDR, where multiple prefixes might match with an IP address. The priority encoder can easily resolve the longest matching prefix, if the prefixes are sorted based on their lengths.

As described in Figure 3.2(b), a standard 9-transistor CAM static memory cell is 50% larger than a standard 6-transistor SRAM cell (9 vs. 6). Charging and precharging long memory lines (the Bit Lines and the Match Lines) of all entries results in high power consumption of a CAM compared to a RAM. Note that when a pattern is searched, the precharged match lines of all non-matching entries discharge to ground. Only one (or possibly a few more) entry actually matches with the pattern. Thus, a lot of power is wasted for non-matching entries. Also when the Match Line is precharged before a search operation, the Bit Lines should be discharged to ground. All these transitions on long memory lines with large capacitances, consume a lot of power. Thus, although CAM devices are desirable due to the parallel searching of a pattern in all CAM entries, high power consumption, cost and area requirements are serious limiting issues. These problems worsen when Ternary-CAM (TCAM) is used.

Although the first binary CAM, brought to market in the early 1990s, suffered from various performance limitations such as high power consumption, high cost and slow search rate, now many vendors claim their product employs techniques to lower power consumption with very fast search rates. CAM based search hardware has become increasingly attractive due to fast table look-ups and being well suited for high-speed applications. TCAM with three states per cell (0, 1, X (dont care)) opened new possibilities, particularly for longest prefix match problems. Also, TCAM devices prioritize search results in such a way that multiple search matches, corresponding to different prefix lengths, could be resolved in accordance with Classless Inter Domain Routing (CIDR) requirements. TCAM technology is rapidly being adopted by networking equipment vendors.

Figure 3.3 depicts a conventional static TCAM cell with 15 transistors. Two SRAM

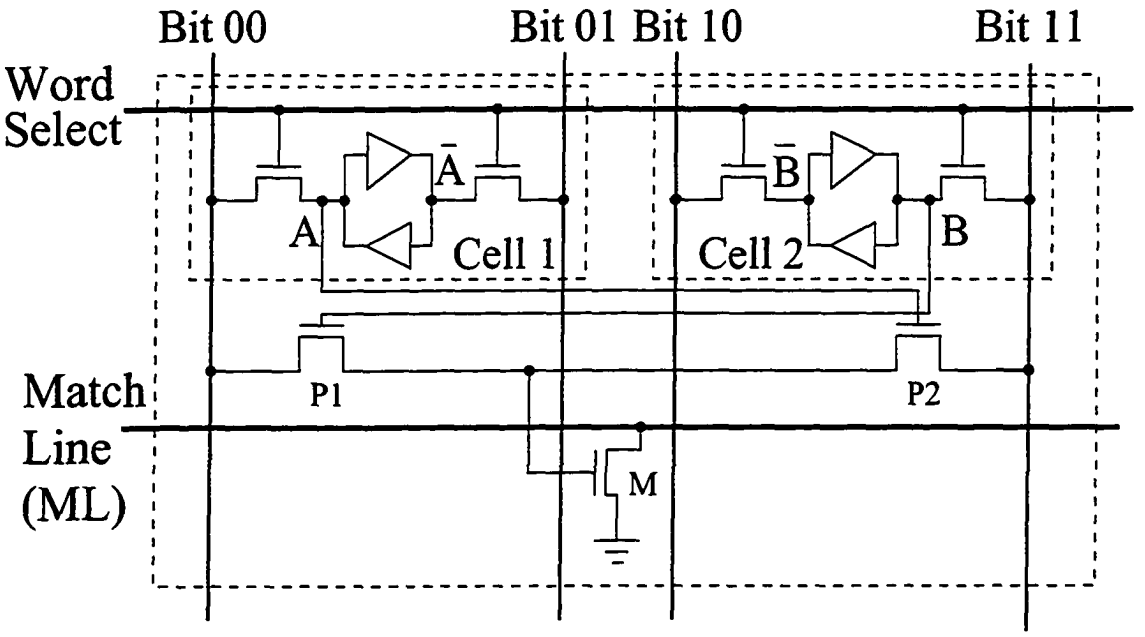


Fig. 3.3. A Conventional TCAM cell

cells (Cell 1 and Cell 2 in Figure 3.3) provide four different storage states. These states are presented with a two-bit binary value. One bit represents the data stored in point A and the other bit represents point B. If cell 1 stores a zero at point *A* and Cell 2 stores a zero at point *B*, both pass transistors (P1 and P2) are always closed. Thus no matter what is applied to the bit lines (Bit 00 and Bit 11) during a search operation, the match line does not discharge. This state is the *don't care* state of the TCAM cell. The cell provides a zero and a one state by storing a *01* or a *10* in *AB*. Note that the fourth possible state (*11* in *AB*) is not desirable at all, because this state always keeps both pass transistors open. This means that the entry always mismatches with the data. Thus three states are easily provided with this TCAM cell. Note that the complementary values of *A* and *B* also exist in the cell.

A TCAM cell is even larger than a CAM cell (66% with a conventional 9-transistor CAM and a 15-transistor TCAM cell). Although more complicated designs reduce the number of transistors down to 12 transistors in each cell [23], a TCAM cell is much larger than a RAM cell and consumes more power. Beside the power consumed during the search operations in a TCAM, the larger memory cells require longer memory lines with larger capacitances, which results in higher power consumption. Thus in a TCAM cell, the power

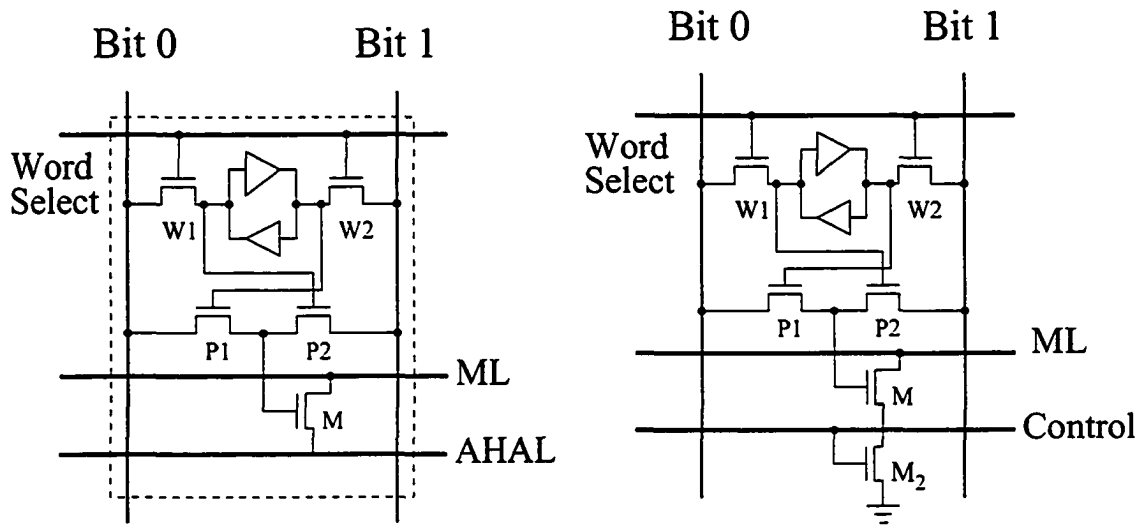
consumption is a limiting issue. Several low power methods have been proposed to decrease the power consumption in a CAM based device. In the next Section, some of those methods are presented.

3.3 Low Power CAM Design

Since power consumption is a challenging issue in CAM-based circuits, this section specifically describes and analyses previously reported low power CAM solutions. The first step to design a low power CAM is to find the sources of power consumption. Hsiao *et al.* model the power consumption of a standard CAM [24]. The power models are originated from the fCV^2 formula. The main sources of power consumption are (1) the *Evaluation* power which is the power consumed on the match lines, (2) *Input Transition* power and (3) *Clocking* power. Each of these sources of power consumption are modeled based on the architecture of a CAM. To decrease the power consumption, designers reduce fCV^2. This is achieved through either (1) fewer transitions on memory lines, (2) less capacitance or (3) lower power supply. Some low power CAM architectures modify each cell to save power. Some designs use the standard cell but apply modification to the whole system to avoid unnecessary power consuming tasks. Since all power consumption models are linearly dependent on the number of entries searched during a search process, some low power CAM devices save power by reducing the effective number of entries. This can be achieved through smart search operations. In this section some of the previously reported low power CAM architectures are discussed.

3.3.1 Power Savings in a CAM Cell

To perform a search operation in a CAM cell (See Figure 3.2(b)), the Match Line (ML) is precharged to VDD and discharged to ground (in case of a mismatch) through transistor M. One simple way of saving power is to reduce the voltage swing on ML. An *Active Low* match line connected to V_{DD} instead of the ground, reduces the voltage swing to $V_{DD} - V_{Tn}$.



(a) AHAL (b) A CAM cell with *Controlled* Match Line

Fig. 3.4. Some Low Power CAM Cell Design.

Note that an active low match line requires a pre-discharge circuit instead of a precharge circuit [25]. If transistor M is replaced with a PMOS transistor, the voltage swing on ML is reduced to $V_{DD} - V_{Tp}$ with the same *Active High* method [26]. If transistor M is connected to an *Active High / Active Low (AHAL)* Signal (is switched on alternate cycles to identify the activity of ML), the switching activity on ML is reduced by half [25]. This cell modification is depicted in Figure 3.4(a). However, in the standard CAM cell both bit lines must be discharged to ground during the precharge phase of a search. A *Controlling* pass transistor can disconnect ML from ground during the precharge phase [27]. This modified cell is depicted in Figure 3.4(b).

One of the sources of power consumption is *Input* power. The transitions on bit lines dissipate a lot of power due to their large capacitances. For every search operation in a standard CAM, the key is applied to bit lines (See Figure 3.2(b)). Thus bit line transitions are unavoidable. Figure 3.5 depicts a CAM cell which avoids bit line transitions by adopting two extra transistors and *Data Lines* (DL0 and DL1) [28]. During the precharge phase of a search, the DL0 and DL1 are set to ground. Thus transistor M_3 and M_4 are turned off. During the evaluation phase of the search, the data (key) is applied to the Data Lines (DL0

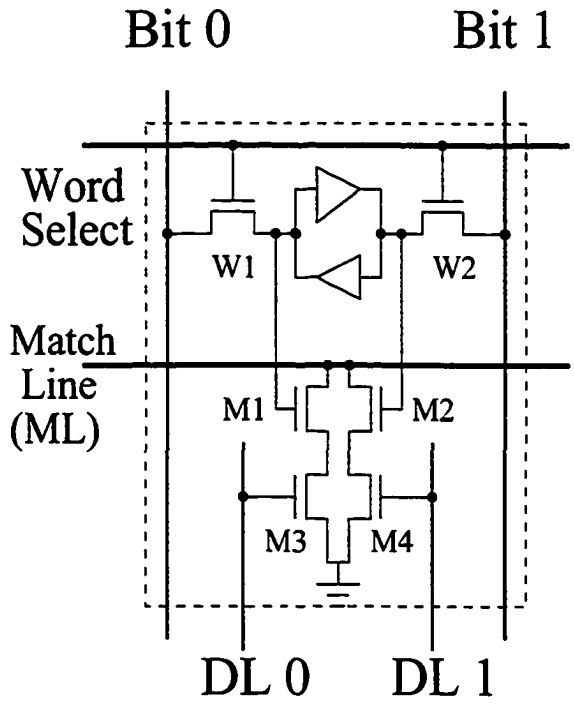
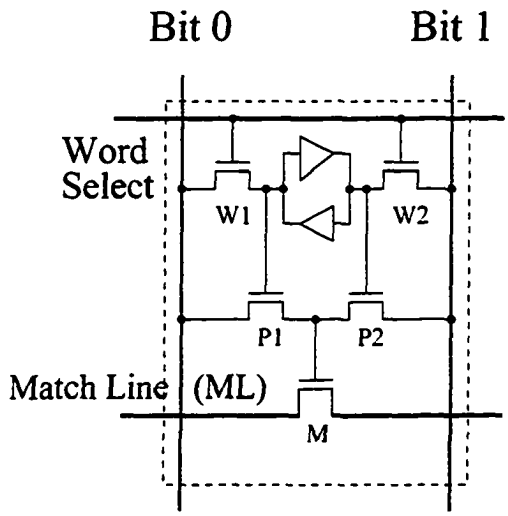


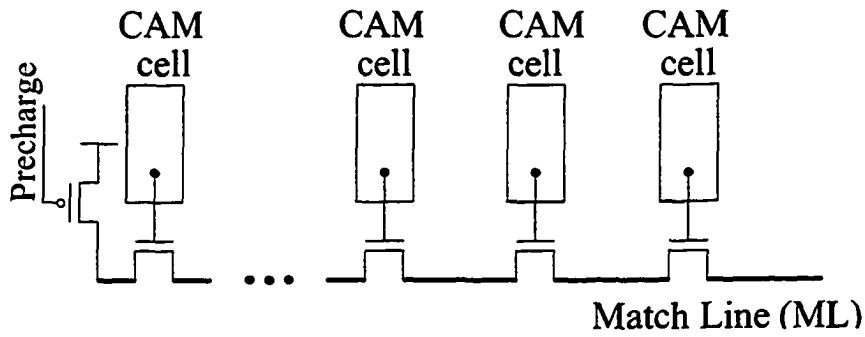
Fig. 3.5. A CAM Cell with Data Lines

and DL1). The use of separate bit line pairs (bit lines and data lines) reduces the load on bit lines and thus, lowers the *Input* power [24]. These Lines are also called Search Lines (SL0 and SL1).

In the standard implementation of a CAM, NOR type comparison circuits are used. Two transitions occur on ML during a search in a non-matching entry (precharging during the precharge phase and discharging during the evaluation phase). The fact that very few entries might actually match with the key, results in excessive power dissipation. A smart design of a low power CAM would use the power only for matching entries. One easy solution is to use *NAND* type of comparison circuits compared to conventional XOR ones. In a NAND type match line, the precharging path is cut down if any non-matching bit is found anywhere in the word [29]. Thus the match line is precharged only if it is a matching entry. Figure 3.6(a) depicts a CAM cell with a NAND type match line and Figure 3.6(b) depicts a CAM entry (Word Line). NOR type match lines, provide parallel and fast search operations while NAND type match lines perform a serial and slow search. The word line can be divided in half to allow faster search operations [29]. However, NOR type match



(a) A CAM Cell with NAND type ML



(b) A CAM Entry with NAND type ML

Fig. 3.6. NAND Type Match Lines.

lines are preferred when the search speed is considered. Some CAM designs are based on a combination of serial and parallel search choices such as [30]. In this serial-parallel CAM, the first few bits of each entry are searched with the corresponding bits of the key serially. If those match, the rest of the bits are searched in parallel. However, this serial-parallel CAM is 25% slower than a conventional parallel CAM with NOR type match lines [30].

In order to allow parallel and fast search operations on the ML as well as save power for non-matching entries, a mismatch dependent power allocation technique is designed in [2]. The sensing circuit in [2] is combined with the precharging circuit through pos-

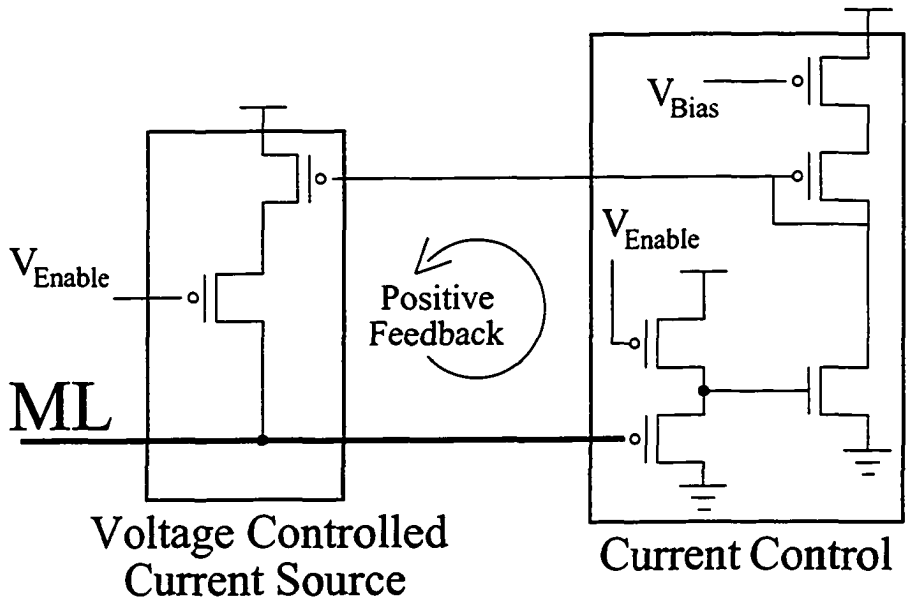


Fig. 3.7. A MisMatch Dependent Sensing Match Line [2]

itive feedback. Figure 3.7 depicts the design in [2]. Before a search starts, the ML is precharged to ground. When the search operation starts, the Voltage Controlled Current Source precharges the ML. If the entry matches with the key, the ML is precharged very fast. A higher voltage on the ML, pushes the Voltage Controlled Current Source to generate more current and the positive feedback loop is closed. If there is at least one non-matching bit in the word line, a path from ML to ground is created and the ML does not precharge a lot. This positive feedback results in higher voltage on the ML of matching entries and lower voltage on non-matching MLs. Thus power is mostly used for matching entries. Note that a matching entry has no path from ML to ground. Thus it is precharged faster. A *Threshold* voltage can be defined based on a dummy matching entry (all don't care values). If an entry reaches the threshold voltage the search operation stops. Thus the maximum voltage swing is the threshold voltage which is less than V_{DD}. A 60% power saving is reported in [2] compared to a CAM with conventional sensing circuits.

The large capacitance on the match line results in high power consumption through search operations. A recent low power TCAM design reduces the effective capacitance on the ML by providing a dual Match Line [31]. In this design, every TCAM cell is made of two SRAM cells and two comparison circuits. Each SRAM cell is connected to one Match

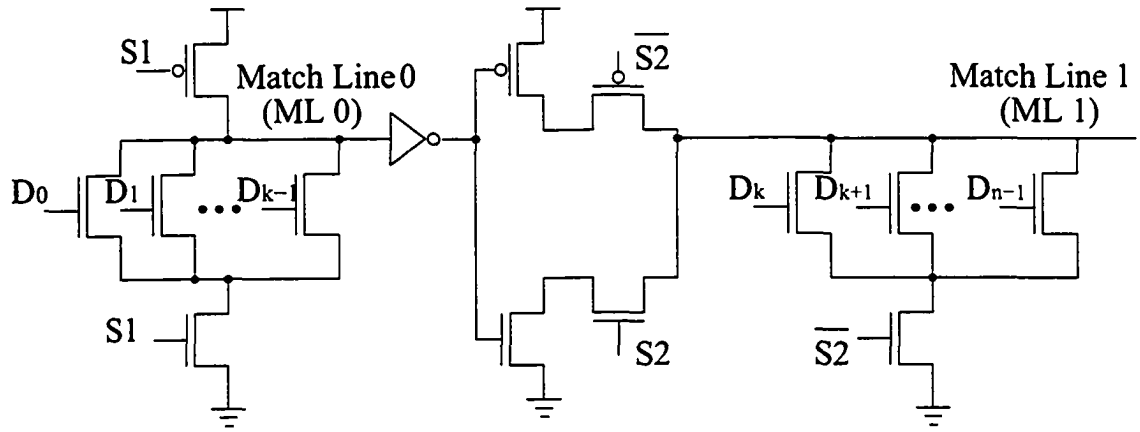


Fig. 3.8. A Selective Precharged Match Line [3]

Line (ML1 and ML2). The data applied to each TCAM cell is a two bit data to provide three states (one, zero and don't care). The search operation is a two step function. One bit of the data (key) is applied to one SRAM cell. If it matches, the second bit is applied to the second SRAM cell. Since fewer transistors are connected to each match line the effective capacitance is reduced in half. On the other if the first ML proves a mismatch, the second match line is disabled. Thus no power is used for the second one. Up to 43% power reduction with 4% penalty in the total search speed is reported in [31].

3.3.2 Systematic Power Savings

Beside cell improvements, systematic improvements can play an important role in saving power in CAM based devices. Some low power CAM designs use an adiabatic switching technique which uses an AC power source instead of a DC power source. An Adiabatic CAM recycles the charge stored in the Match Line capacitance. The AC power supply's slow transitions help to reduce the energy dissipation [32].

Some systematic power saving is achieved through smart search operations which avoid unnecessary searches. The power consumption in a CAM device is reduced if fewer entries or fewer bits in each entry are searched. A Selective precharge scheme precharges some of the bits of an entry and performs a search operation on those bits [3]. In case they match with the corresponding bit of the data (the key), the rest of that entry is precharged and

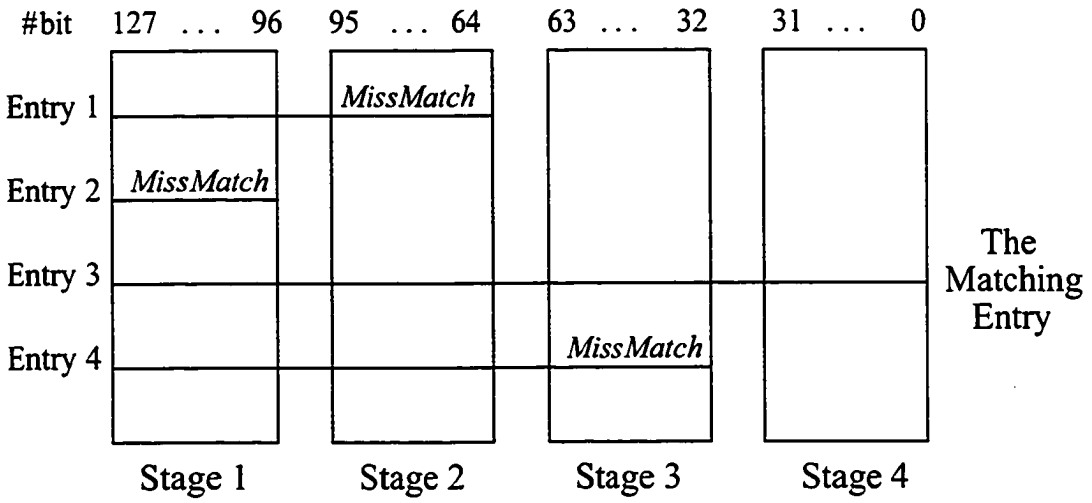


Fig. 3.9. A Pipelined TCAM Storing IPv6 Prefixes.

searched. If the few first bits do not match with the corresponding bits of the key, there is no need to search for the rest of the entry. Although there might exist some matching entries in the first few bits that mismatch in total, the most power is actually used for the matching entries. Figure 3.8 depicts a CAM entry of n bits with the selective precharge scheme. In the first Step, the first k bits are searched. *ML0* is precharged through the PMOS transistor controlled with *S1* signal. The D_i values come from each memory cell. During the evaluation phase, if any of the first k bits do not match with the corresponding bit of the key, the D_i equals to one for that memory cell. Thus the match line (ML 0) discharged to ground. If all first k bits match, then in the next step, the second match line (ML 1) is precharged and searched for the rest of the key (bit k to bit n-*1*) during the second evaluation phase. Note that *ML1* does not precharge at all if *ML0* senses a mismatch.

A selective precharge scheme can be used in a pipelined CAM [33]. If each CAM entry is divided into two or more sections, these sections can form the stages of a pipeline. In each stage, the entries that match with the data in their previous stage are searched. If one part of an entry does not match with the corresponding bits of the key, it is clear that the entry is not matching with the whole key. Thus there is no need for further searches. Figure 3.9 depicts an example of a four stage pipelined CAM/TCAM applied to IPv6 addresses. In the first stage, the Most Significant Bits (MSB) of the address are searched with all entries

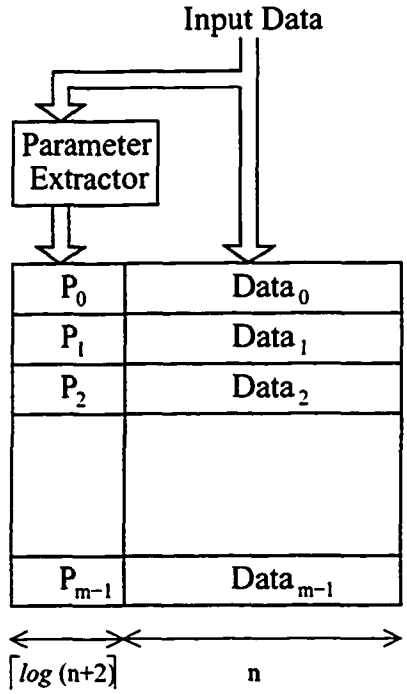


Fig. 3.10. A Pre-Computational CAM.

in parallel. Assume this results in three matching entries (entry 1, 3 and 4). It is clear that those entries that mismatch the key in this stage (*e.g.* entry 2), do not match with the key. Thus, in the second stage only entries 1, 3 and 4 are searched with the key. Thus, at each stage only matching entries of the previous stage are searched with the corresponding bits of the key. After the last stage the matching entry is found. Pipelined CAM devices save power by avoiding unnecessary search operations.

A pre-computational CAM is designed with the purpose of avoiding unnecessary search operations [34]. Figure 3.10 depicts an example of a pre-computational CAM. Each entry has a corresponding parameter section, V_i, stored for the data, $Data_i$. The design in [34] uses a *one count* function as the parameter extractor. With an n bit data length, there are n+1 possibilities for ones count (from 0 ones count to n ones count) and an extra case must be dedicated to clarify the availability of the stored data. Thus, a 30-bit entry has a 5-bit parameter. During a search operation, the parameter of an Input datum is extracted and searched will all parameters in the CAM. If any parameter mismatches with the parameter of the input, that entry mismatches with the input as well. Thus there is no need to perform

a full search on entries that do not match in their parameter part.

3.3.3 Numerical Examples of Power Consumption

To clarify why power consumption is a critical issue in a CAM design, a numerical example is given based on previously reported low power CAM designs. The energy consumption per bit per search for some of these designs are given in Table 3.1. Note that the energy consumption value for [33] is only for the Match line and the Search line. The clocking power is not considered. Also consider that the CAM presented in *Serial [30]* is 25% slower than a fully parallel CAM. To give a fair example, assume a CAM consumes 10 *fJ/bit/search* energy (a rough example based on Table 3.1) and has 10k entries, 128 bits wide each (suitable for IPv6). The size of the CAM represents the number of prefixes in the LUT. The core routers require larger lookup tables. Recently, tables with up to 150,000 prefixes are reported in [35]. However, we chose a rather small LUT with approximately 10,000 prefixes for the numerical example (10K entry CAM), based on our real traces as described later in Chapter 6. If the CAM is required to perform a search every 10 *ns* (for 100 Mpps processing), 1*w* power is consumed. With larger CAM sizes or faster searches (*e.g.2 ns* as in [2]), the power consumption increases dramatically. Note that a TCAM cell is inherently larger than a CAM cell. This would generate more capacitance on memory lines and result in higher power consumption. Thus it is very important to consider low power consumption for a CAM-based design.

3.4 Longest Prefix Matching in Ternary CAM

Since CAM based devices can perform a parallel search for a key in all their entries, they are suitable for lookup table implementations. Routing lookup tables store routing prefixes which are comprised of ones and zeros followed by don't cares. Thus a Ternary CAM (TCAM) is required for routing lookup implementations. Since CIDR requires a routing lookup to return the Longest Matching Prefix, the TCAM must resolve which of those

TABLE 3.1

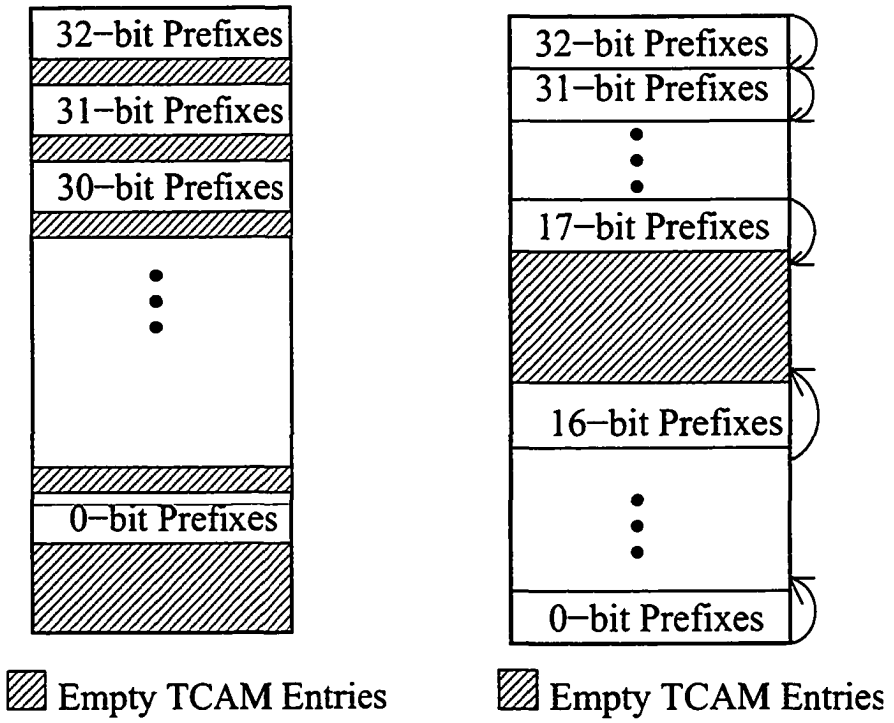
Comparison of Previously Reported CAM Energy Consumption.

	*f*J/bit/search
[29]	83.0
[36]	97.7
Serial [30]	3.1
Parallel [30]	7.5
[24]	45.5
[37]	13.9
[33]	4.5

matching entries is the longest matching entry, in the case of multiple matching entries.

Usually, finding the longest prefix match (LPM) during TCAM lookups requires maintaining the prefixes in a sorted length order which makes worst case updates very slow. For example, an insertion of a new entry in a TCAM storing N prefixes might result in moving (shifting) O(N) TCAM entries to create an empty space for the new insertion. This slow update is undesirable due to possibility of 100s to 1000s of updates per second in today's forwarding tables [38].

The routing table update delay is one of the key elements of routing lookup efficiency beside the lookup speed, power consumption and memory footprint. Several solutions have been proposed to decrease the routing table update delay such as reserving some empty entries between sets of different length prefixes. Figure 3.11(a) depicts a TCAM with some empty entries reserved between each set of prefixes with the same length. If the TCAM is required to insert a new prefix of size n, it will simply add this new entry to the reserved space. If all the reserved spaces are full, then O(N) shifts are required for the worst case. While some of these reserved entries might be used up very fast, some entries might remain untouched. This leads to under-utilization of the TCAM space while the worst case complexity of updates remains the same. Since there is no need to sort the



(a) (b)

Fig. 3.11. TCAM Space Management. (a) reserves some space between sets of prefix lengths, and (b) reserves the extra space in the middle only.

prefixes in a segment, the TCAM can reserve some empty space in the middle. Then an empty space can be provided anywhere with no more than *L/2* shifts (*L* represents possible prefix lengths (*e.g. L = 32* for IPv4 prefixes) [39]. Figure 3.11(b) depicts a TCAM with empty entries reserved only the middle of all entries.

The TCAM space management overhead and non-uniform update delays reduce the TCAM efficiency. These problems worsen with 128-bit IPv6 due to the much longer possible prefix lengths. Some applications avoid TCAM sorting requirements by manipulating the data before storing it. For example, in IP prefix caches implemented by TCAMs, the LUT is fully expanded to avoid multiple matches for correct cache results [14]. But not all applications have such a convenient solution to the problem. Many TCAM vendors employ a simple sorting technique and live with an *O(N)* worst-case update time solution.

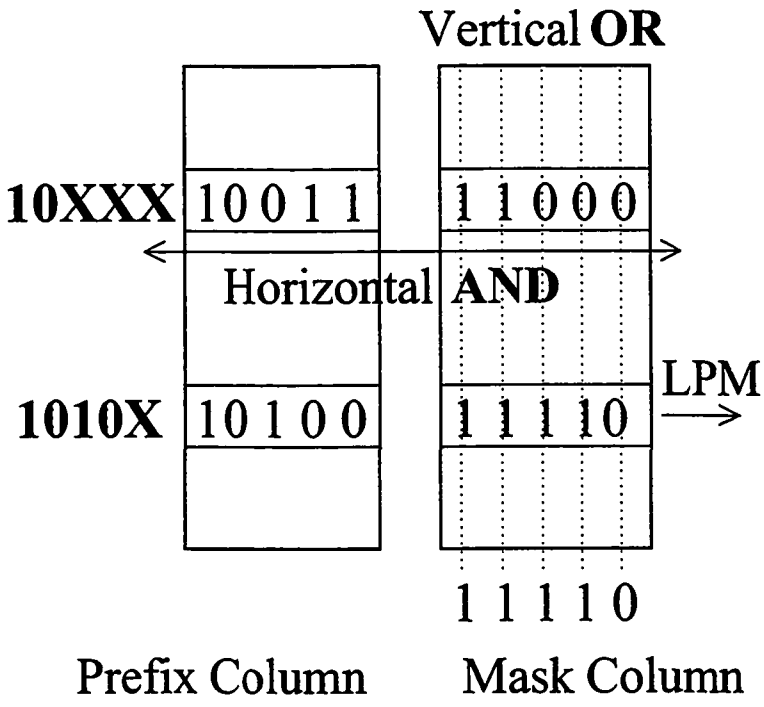


Fig. 3.12. Binary CAM with mask features presented in [4].

A TCAM can store the prefix lengths as well as the prefixes and use them to find the LPM. This is a simple and fast solution for LPM but it results in at least a 70% increase in the TCAM memory requirement [40]. Power consumption also increases because more search operations are required. However, binary CAMs, with no built-in mask circuits, can use this extra information to mask data as well as finding the LPM [4]. Faster updates are obtained at the cost of slower search times and lower memory density. Figure 3.12 depicts an example of searching *10100* in the table using the design in [4] for a 5-bit addressing scheme. A prefix (comprised of zeros and ones followed by don't cares) is represented by a binary prefix entry and a binary mask entry. Mask entries are sequences of ones followed by zeros, such that the number of consecutive ones represent the length of the prefix and zeros stand for *don't cares*. Horizontal *AND* circuits mask each prefix entry and vertical *OR* circuits in the mask column find the longest length among all matching entries. A second search of the longest length in the mask column resolves the position of the LPM.

All existing LPM solutions: (1) have long worst-case update delays, (2) slow down the lookup speed, (3) need complicated table management and maintenance or (4) require a

great amount of extra area. Thus, in a TCAM based lookup table design, the update delays and the table management complexities must be optimized as well as the look up speed, power consumption and memory footprint.

3.5 Summary

Although CAM based search engines are desirable due to their high speed, their high power consumption is an important disadvantage. Several solutions have been proposed to decrease the power consumption in a CAM device. Pipelined CAM devices avoid unnecessary search operations and provide high throughput. The Hardware-based Longest Prefix Matching (HLPM) technique proposed in this thesis provides a simple, fast and scalable LPM solution with very small increase in area as well as potentially reducing the power consumption. In the next Chapter HLPM is discussed in detail.

Chapter 4

Hardware-Based Longest Prefix Matching (HLPM)

4.1 Overview

As one major part of our forwarding mechanism, we adopt a pipelined TCAM as the main lookup table implementation, because TCAM based lookup tables perform parallel and fast search operations. As mentioned in Chapter 2, the main lookup delay directly impacts the cache miss penalty. The TCAM stores routing prefixes in its entries. When a lookup for an IP address is performed, the TCAM searches all prefixes in parallel. However a TCAM must be able to find the Longest Matching Prefix if multiple prefixes match with the IP address. As explained in Chapter 3, finding the longest matching prefix is a complicated task in a TCAM. In this Chapter a novel technique, Hardware-based Longest Prefix Matching (HLPM), is proposed for TCAM based lookup tables to resolve the LPM with no table management requirements as well as to maintain high search speeds and reasonable power consumption levels.

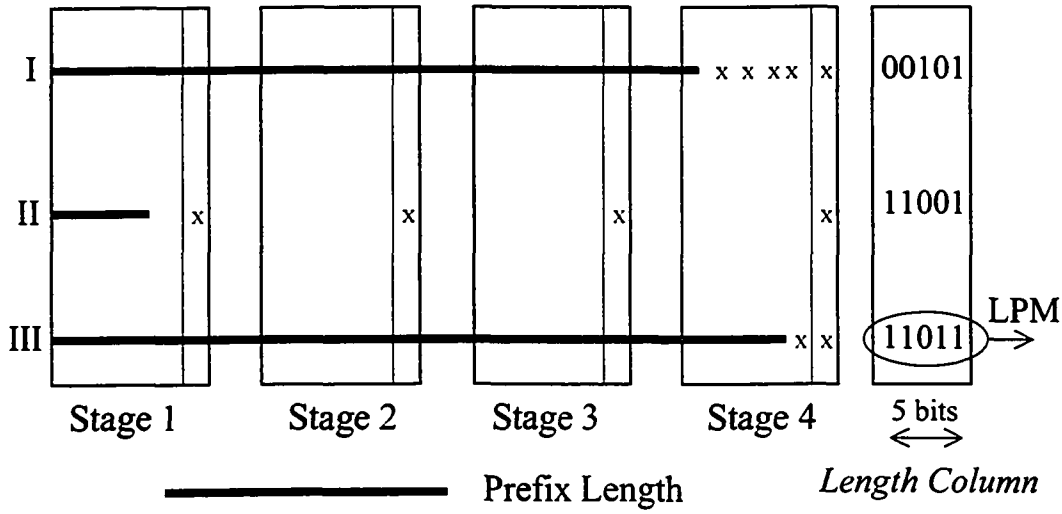


Fig. 4.1. The Proposed Four-Stage Pipelined TCAM.

4.2 HLPM

This Section describes our proposed HLPM for a four stage pipelined TCAM applicable to IPv6, but the technique is scalable to any pipelined TCAM. Later in Chapter 5 we scale HLPM to our two stage pipelined TCAM to store 32 bit IPv4 prefixes.

Figure 4.1 shows our four-stage pipelined TCAM with an extra SRAM stage named: *Length Column*. Every entry in each stage is 32-bits wide to provide the 128 bits required by IPv6 prefixes. HLPM stores the binary coded lengths of prefixes in their ending stages in the *Length Column*. Thus, 5 extra storage bits per entry are required for IPv6 prefixes (in a four-stage pipeline, 32 bits each). For example *00011* is stored in the corresponding *Length Column* entry of a prefix with size 35, because the prefix ends in the second stage, and there are only 3 bits in that stage. The same value will be stored for 3, 67 and 99 bit prefixes, because they all have 3 bits in their ending stages. However, with no sorting requirement for the prefixes, new prefixes can be inserted in any TCAM entry, regardless of the prefix length. Note that if the number of stages with the same length increases, the number of bits in the Length Column does not increase. Thus HLPM is easily scalable to any size or length.

As described in Section 3.4 Pipelined TCAM devices save power by lowering the power

consumption for non-matching entries. However, TCAM searches consume fixed power for different length matching prefixes. In CIDR prefixes can vary in size from 0 to 127 (IPv6) and many prefixes might match with an address. On the other hand, since short prefixes cover more addresses, there is higher probability of searching for a short prefix in a LUT than a long one. The HLPM technique saves power not only on non-matching entries through pipelining, but also on short prefixes which are more likely to be searched. The following sections describe search and update operations.

4.2.1 TCAM Search Operation

IP prefixes are formed as sequences of data bits (either 0s or 1s) followed by *don't care* bits (the number of 1s and 0s represents the length of each prefix). Since don't cares match with both zeros and ones, a TCAM search may result in multiple matching entries with an address. Figure 4.1 shows an example of multiple matching entries. Entries I,II and III are three matching prefixes with different lengths for a given address. Searching the IP address in the first stage results in matches in all those entries. These matches lead to further searching of the IP address in the following stages of the pipeline in those three entries. There is no need to search the rest of entry II, simply because the rest of the bits of that entry are *don't cares* and will always match. Thus, if the last bit of an entry in one stage of the pipeline stores a *don't care* value, there is no need to search the rest of the entry in the following stages. In our example, second stage searches are necessary only for entries I and III. On the other hand, after the fourth stage of the pipeline, it is clear that entry II is not the longest matching prefix due to the fact that prefix II ended in the first stage of the pipeline. These observations lead us to simplify the LPM in our TCAM. However, several matching prefixes might end in the same stage and a *Second Level Search* is necessary to find the LPM from those entries (e.g. entry I and III in Figure 4.1), by using the information stored in the Length Column.

Thus the TCAM search operation is similar to any pipelined TCAM with one difference. During a search, an IP address is compared with every TCAM entry in parallel, stage

TABLE 4.1

Entry Evaluation in One Stage.

Entry Matches with the IP Address?	**Entry has a *don't care* in the Last Cell?**	**Search the Entry** in the Next Stage?
Yes	*No*	*Yes*
Yes	*Yes*	*No*
No	–	*No*

by stage (one prefix is stored in each entry). This means if the corresponding bits of the IP address do not match with the corresponding bits of a prefix (an entry) in one stage, that entry of the TCAM is not searched in the following stages of the pipeline. That entry is definitely a mismatch, no matter what is stored in the last bit of the entry in the current stage. If an entry matches with the corresponding bits of the IP address in one stage, the entry might match with the IP address at the end of the pipeline. Thus the entry must be searched in the following stages of the pipeline as well. However, if the entry has a *don't care* bit in its last cell, the entry is not searched in the following stages. This is because the rest of the entry are *don't cares* and definitely match with the IP address. This is the difference between conventional pipelined TCAMs and an HLPM-based pipelined TCAM. This entry evaluation per stage is given in Table 4.1.

Note that unnecessary searches are avoided for short matching prefixes as well as for non-matching prefixes. However, a two-level search is required to find the longest prefix amongst all matching prefixes. If multiple entries match with an IP address and those entries end in different stages of the pipeline, the LPM is simply found at the end of the pipeline search operation (*First Level Search*) as described before. In case multiple matching prefixes end in one common stage of the pipeline, the *Second Level Search* resolves the LPM. Section 4.2.3 describes the Second Level Search in detail. However, in order to find out if a prefix ends in one stage of the pipeline or not, the last cell of the TCAM in each stage should be modified. This modification is described in Section 4.2.2.

Since an HLPM-based pipelined TCAM avoids unnecessary searches for short match-

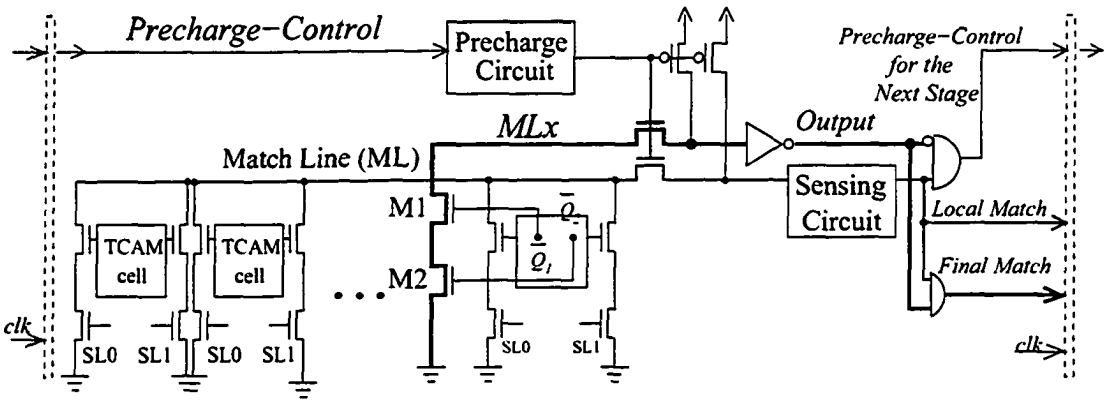


Fig. 4.2. A Modified TCAM Entry.

ing prefixes, the power consumption is reduced in comparison with previously reported pipelined TCAMs [33]. Meanwhile, since not all matching prefixes require the second level search (e.g. entry II), we achieve further power saving compared to previous designs such as [4].

4.2.2 TCAM Entry Modification

Figure 4.2 shows a TCAM entry in one stage of the pipeline. The last cell is modified by adding two extra transistors (M1 and M2) which are controlled by the complementary bits of the data stored in the cell. Each TCAM cell can store three states (zero, one and don't care). A two-bit binary state, Q_1 Q_2, represents these three states. A *01* or a *10* state represents a one or a zero and *00* represents a don't care. If a cell stores a don't care, its paths from ML to ground are closed. If the last cell stores a don't care, the path from MLx to ground is open, because M1 and M2 are controlled with the complementary values of Q_1 and Q_2.

At each stage, the *Precharge-Control* Signal comes from the evaluation results of the entry in the previous stage. The *Precharge-Control* signal of the first stage comes from the *valid bit* of the entry. A normal search operation of the entry includes searching for a *don't care* in the last cell in parallel with the conventional searching for a match or a miss-match between the data stored in the entry and the input pattern. The extra circuits for the last cell, including the precharging circuit and the sensing circuit, are similar to the

normal search circuits. A very short match line, shown by *MLx* in Figure 4.2, is precharged to logic high in the precharge state. In the evaluation phase, the last cell searches for a *don't care* which corresponds to storing *00* in the last TCAM cell. If the cell stores a don't care, the paths from *ML* to ground are closed but the path from *MLx* to ground is open. *MLx* discharges, and the *Output* senses a logic high. Although search operations of ML and MLx are similar, MLx evaluation is much faster, consumes less power, and does not require complicated sensing circuits due to the very short length of MLx. Besides, the MLx evaluations or sensing circuits are actually independent of the circuits used for the whole entry. This allows the system to use complicated sensing circuits for the whole entry to optimize the power consumption while a very simple sensing circuit is used for the last cell. Smart sensing and precharging circuits such as [2], precharge the *ML/MLx* only for matching entries, resulting in further power savings. However, after the search operation is complete at each stage, the pipeline decides whether search the rest of each entry in the next stage or not. As mentioned before, if one entry does not match the data at one stage, the rest of that entry is not searched in the next stage of the pipeline. In our pipeline, one entry is not searched in the next stage not only if it mismatches the data at the previous stage, but also if it ends with a *don't care*. As depicted in Figure 4.2, the *Local Match* signal is the result of the entry evaluation in each stage. If the entry does not match with the corresponding bits of the address in one stage, the *Local Match* signal senses a logic zero and the *Precharge-Control* signal notifies the next stage not to perform a search any more. However, if the entry matches with the address and stores a *don't care* in its last cell, the *Final Match* signal declares a matching prefix and the rest of that entry is not further searched in the following stages of the pipeline. The general evaluation of the entry is described in Table 4.2 Note that this change in the architecture not only saves power for short matching prefixes, but also resolves the first level search of our HLPM.

TABLE 4.2

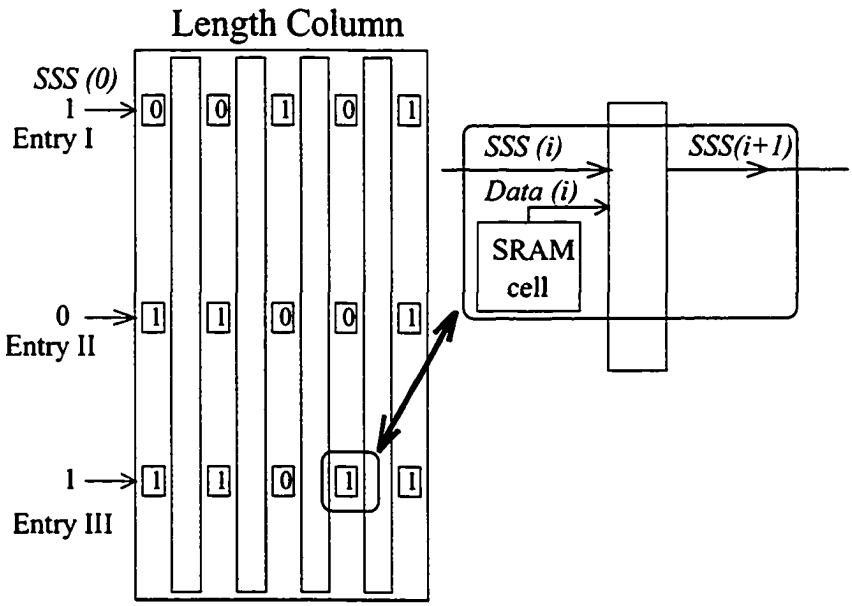
Entry Evaluation.

ML	*Output*	Search the Next Stage?
High	*Low*	*Yes*
Low	*High*	*No*
High	*High*	*No*
Low	*Low*	*No*

4.2.3 Second Level Search

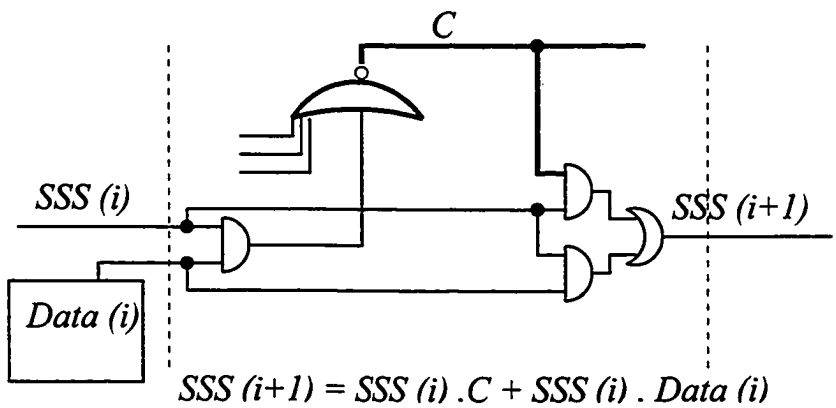
The second level search resolves the LPM for multiple matching prefixes ending in one common stage. Figure 4.3 depicts the length column in detail. The *Second Search Signal(0)* is the result of the last stage of the pipelined TCAM. SSS(0) is set only for matching prefixes requiring a second level search. In the example given in Figure 4.1, the second search signal is set only for entries I and III. Since the length column stores the binary lengths of prefixes in their ending stages (5 bits long), the entry storing the *max* value is the longest matching prefix. As depicted in Figure 4.3(a), we adopted a 5-stage pipelined *Bit-Serial* approach to find the *max* length. At each stage if a second search is required (*SSS(i) = 1*), one bit of the data in the corresponding entry of the Length Column is evaluated. If there is only one data equal to '1' among all entries, that entry is the *max* of all. But if no entry has a '1' or more than one entries store 1s, those entries should be searched in their next stages as well. Thus the SSS signal for the next stage of those entries will be set (*SSS(i+1) = 1*). Figure 4.3(b) depicts the logic required for the Second Level Search at each stage of the Length Column.

However, the simple length-column pipeline can be clocked faster (*e.g.*it can be sensitive to rising and falling edges of the clock) than the TCAM pipeline or a digit serial approach can be used to provide short latency.



Data (i): Data at stage i
SSS (i): Second Search Signal at stage i

(a)



(b)

Fig. 4.3. Length Column.

4.2.4 HLPM Advantages

The HLPM architecture has the following advantages:

1. HLPM saves power for short matching prefixes. HLPM does not search the rest of a matching entry, which has a *don't care* in its last cell in the current stage. This is important because all the bits of that entry are *don't cares* in the following stages and there is no need to perform a search on an entry full of *don't cares*. The details of the HLPM power savings is given in Chapter 6.

2. HLPM resolves the LPM without requiring LUT management or maintenance or sorting of the entries in the TCAM. Table updates require a single update operation.

3. HLPM is simple and scalable with TCAM width. The second level search does not change if the TCAM has more or less 32-bit stages. The first level search, as described in 4.2.1, is also independent of the TCAM size. Thus, the extra area and the complexity of HLPM remains the same if the TCAM size is scaled.

4. HLPM is efficient in extra storage area requirements. For example, a conventional TCAM needs 100x128 bits of extra storage to reserve only *one* empty entry for only 100 different IPv6 prefixes. This area is equal to the storage area of the 5th stage (5 bit entries) for a 5K entry TCAM with the proposed HLPM. Since the Length Column requires only 5 SRAM bit per entry SRAM rather than 32 CAM bit per entry, the total area of our TCAM is approximately 20% less than comparable designs [4].

4.3 Summary

A novel Hardware-based Longest Prefix Matching (HLPM) scheme for TCAM-based lookup tables is proposed in this Chapter. This technique is applied to pipelined TCAMs and aims at further decreasing power consumption compared to previously reported pipelined TCAM designs, by saving power for matching short prefixes. The HLPM is a two level search. The first level resolves the LPM of prefixes ending in different stages by searching for a *don't*

care in the last bit of each stage. A very simple cell modification is presented in this chapter to perform the first level search. The second level resolves the LPM of multiple prefixes ending in one common stage of the pipeline by finding the *max* value of the coded lengths of prefixes in the last stage of the pipeline. In the next Chapter, a novel routing cache with an HLPM-based LUT are described in detail.

Chapter 5

IP Forwarding Architecture: The Cache and the Look Up Table

5.1 Overview

A routing cache stores recent routing results for reuse. The higher locality in the traffic results in lower cache miss rates. This means if packets with identical destination addresses arrive at the router, they are forwarded to their next hop by referencing the cache only. Since performing a lookup in the main lookup table is much slower than referencing a cache, the miss penalty is rather large. A missing address must be carried to the LUT, a full lookup must be performed and the lookup result must be sent back to the cache and at the end, the cache must be updated. The packets whose next hop information is not in the cache, must be buffered until the lookup is done. Thus the miss penalty delay directly delays the forwarding of packets to their destinations. As a result, a routing cache with very small miss rates can improve the forwarding mechanism efficiency. The choice of what to store in the cache is an important part of a routing cache design. This is explained in detail in Section 5.2. When an IP address misses the cache, the cache must be updated. The main lookup table is accessed to resolve what the cache must be updated with. An entry in the cache is replaced with the new update result. The entry is selected based on the replacement

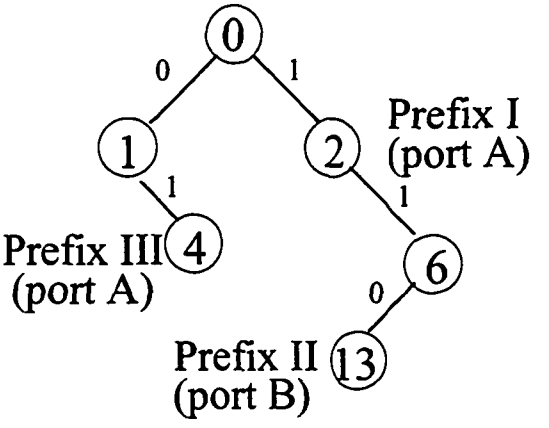
policy of the cache. Later in this Chapter a routing cache is designed with the main goal of reducing both the cache miss rate as well as the effective cache miss penalty. A software and a hardware based LUT are also proposed in this Chapter. The novel software solution requires no table expansion and the novel hardware solution employs HLPM to resolve the LPM and to save power.

5.2 What to Store in a Routing Cache?

A routing cache stores IP addresses and their forwarding information (output port ID). If an IP address exists in the cache, the packet can be easily forwarded to the output port. However, many IP addresses covered with a common prefix in the LUT might be forwarded to the same output port. Thus, if the prefix is cached instead of individual IP addresses covered by it, more IP addresses hit the cache. Thus caching prefixes reduces the cache miss rate because a stronger locality exists among prefixes.

However, routers are required to provide Longest Matching Prefix routing. If multiple prefixes match an address, a situation may arise where the longest matching prefix is not present in the cache, but a shorter matching prefix is in the cache. This short matching prefix will produce a cache hit, leading to an incorrect routing decision. In our previous example given in Figure 2.4 (repeated in Figure 5.1), if Prefix I is cached, it will match with IP addresses whose longest matching prefix in the trie is Prefix II (*e.g.* address 1101). An IP address matching Prefix II could then be incorrectly matched by Prefix I and forwarded to port A. The prefix in node 2 (prefix I) is said to encompasses the prefix in node 13 (prefix II), because node 2 is on the path from the root to node 13. Encompassing prefixes are non-cacheable. Thus prefix caching is not as simple as caching full IP addresses and correct cache results must be ensured somehow. One simple solution is to cache the full IP addresses when the lookup results are non-cacheable prefixes. In this example, the prefix at node 4 (prefix III) is cacheable because it does not encompass any other prefix. As a result, the choice of what to update the cache with, becomes a critical issue. This is explained in the next section.

Prefix	Port ID
Prefix I (1xxx)	Port A
Prefix II (110x)	Port B
Prefix III (01xx)	Port A



(a) (b)

Fig. 5.1. An Example of Longest Matching Prefix

5.2.1 What to Update a Routing Cache With?

A cache miss eventually results in a cache update. In a full address cache, where full IP addresses (*e.g.*32 binary bits for IPv4) and their corresponding output ports are stored, the cache must be updated with the full missing address. The missing output port must be resolved through main table access. The longest matching prefix with the missing IP address is found in the main table and the corresponding output port is read from the table. Then an entry in the cache is replaced with the full missing IP address and the output port. As mentioned before, the replacement policy of the cache decides which entry is selected to be replaced.

On the other hand, a prefix cache can be updated with a prefix or an IP address. If the Longest Matching Prefix of a missing IP address is a cacheable prefix, the cache is updated with that prefix. If the Longest Matching Prefix of the IP address is not cacheable, then the full IP address (*e.g.*32 bits for IPv4) is stored in the cache. However, if non-cacheable prefixes are common lookup results, then a prefix cache degrades into a full-address cache. A prefix cache requires a TCAM, while a full IP cache is implemented with a binary CAM. Thus the prefix cache full of 32 bit IP addresses is not efficient in terms of area and power consumption. Note that for the main lookup table to return correct results to the cache (the

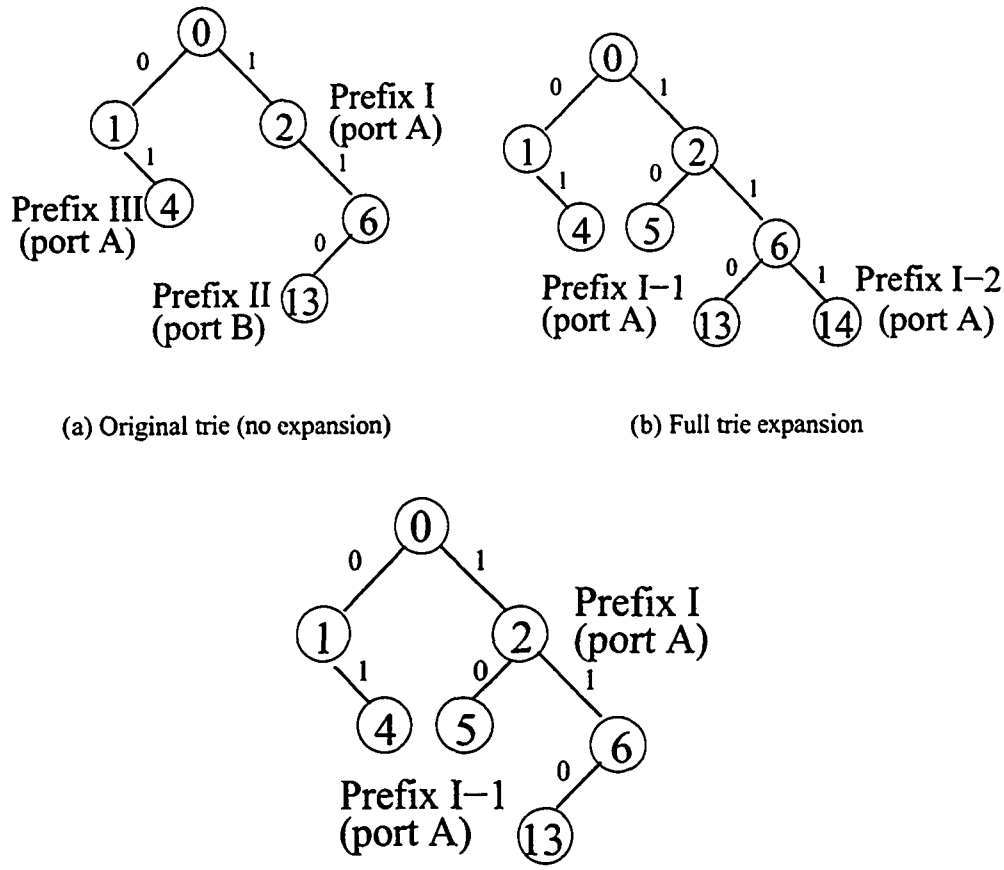
prefix or the full address), the lookup scheme must decide if a prefix is cacheable or not. This complicates the table look ups and table updates and requires more storage area. Note that a cacheable prefix might be no longer cacheable after the table is updated with a new prefix which is covered with the old cacheable prefix.

It is important to see how often the non-cacheable prefixes are the results of the lookups. Our simulation results, given in Chapter 6, show that the LPM of a large portion of IP addresses (up to 46%) correspond to non-cacheable prefixes. Thus it is essential to increase the number of cacheable prefixes, if a prefix cache is to be used.

5.2.2 Prefix Caching and Table Expansion

To increase the locality of data stored in a forwarding cache, full prefix caching is preferred where all the prefixes are cacheable (leaves of the trie). This requires a full LUT expansion. Figure 5.2(b) shows a complete expansion of the trie for our example in Figure 5.2(a). Prefix I is expanded by appending *0* to form prefix I-1 in node 5, and by appending *11* to form prefix I-2 in node 14 and the forwarding information (port A) is copied in both nodes 5 and 14. Thus full table expansion means replacing any non-cacheable prefix with a set of new cacheable prefixes (with the same output port ID). The new prefixes ensure that all IP addresses covered with the old non-cacheable prefix, are covered with the set of these new prefixes. Thus all addresses whose longest matching prefix was the old non-cacheable prefix, are now matched with a cacheable prefix and are forwarded to the correct output port.

However, note that the number of valid prefixes increases and the lookup table gets larger. Liu reports up to an 118% increase in table size [14]. Routing table expansion is unfavorable due to memory area limitations, power consumption and cost. On the other hand, table expansion pushes prefixes lower in the trie, increasing the search time for Software searches. Also, updates in a fully expanded table are challenging, since all prefixes remaining in the table after an update, must be cacheable. Note that an expanded prefix no longer exists and the update must find the prefixes created by expansion.



(a) Original trie (no expansion)

(b) Full trie expansion

(c) Partial trie expansion in [14]

Fig. 5.2. Trie presentation of a small lookup table.

A partial LUT expansion can be adopted to increase the locality as well as to keep the LUT small. Figure 5.2(c) depicts the partial expansion in [14] where non-cacheable prefixes are expanded only to their first level of expansion. This partial expansion means for every non-cacheable prefix, only one new cacheable prefix is added to the table. The new prefix is formed by adding either a 0 or a 1 to the non-cacheable prefix. Note that the non-cacheable prefix is not removed from the table and the new prefix is added only if it is cacheable. The goal of this expansion method is to increase the chance that longest matching prefix of more IP addresses, is cacheable. If the new prefix (either by adding a 0 or a 1) is cacheable, half of the IP addresses covered with the non-cacheable prefix are covered with this new prefix. The longest matching prefix of these addresses are now

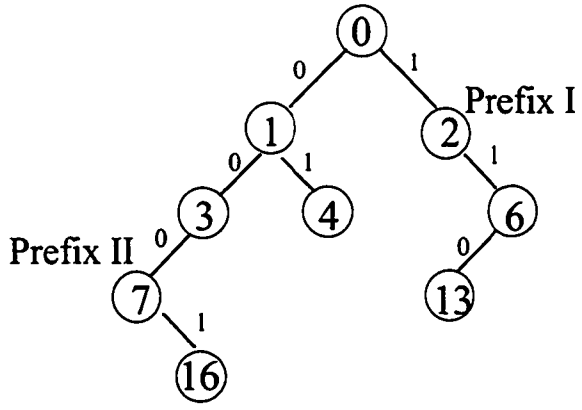
this new cacheable prefix. In the previous example, Prefix I-1 is added to the trie at node 5 with the same forwarding information as prefix I. In the original trie, the LPM of all the IP addresses covered by node 5 and 14, is prefix I (node 2). Since prefix I is not cacheable, those IP addresses must be cached in full. After addition of Prefix I-1 to the trie (See Figure 5.2(c)), the LPM of all IP addresses covered by node 5, is prefix I-1 which is cacheable. Caching Prefix I-1 instead of IP addresses increases the locality of the cache and results in lower miss rates. Note that the table size after this partial expansion is less than the fully expanded table.

However, the level one expansion method is not useful for some non-cacheable prefixes which remain non-cacheable by adding either a 0 or a 1. This is especially important because the short prefixes on top of the trie, which cover most of the IP addresses, encompass many other prefixes and level one expansion is most likely to be useless. Moreover, the lookup scheme must still decide if a prefix is *cacheable* or not.

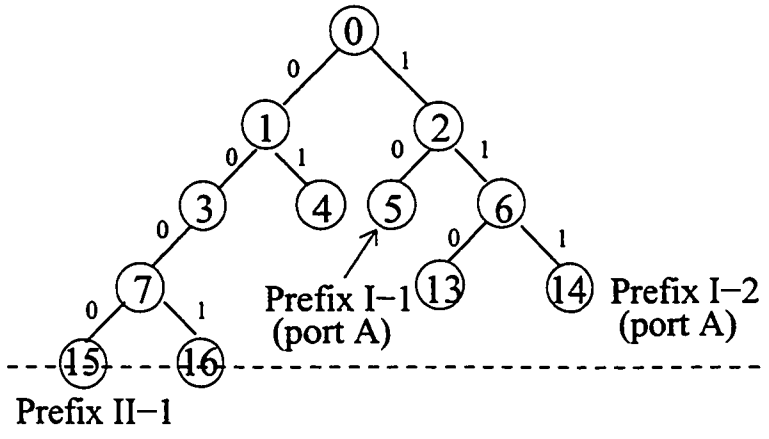
5.2.3 Short Prefix Expansion (SPE)

Short prefixes cover more IP addresses than longer prefixes. Thus higher locality exists in short prefixes. Although there are fewer short prefixes in a routing table, short prefixes might be referenced more often [41,42]. Short Prefix Expansion (SPE) fully expands the trie for the prefixes less than 17 bits long (short prefixes). SPE ensures that any prefix less than or equal to 16 bits is fully expanded and thus, cacheable. A short prefix covers a large number of addresses. If short prefixes are cached, better cache hit rates are expected. On the other hand, most prefixes stored in lookup tables are between 16 to 24 bits long and there are many cacheable 16 bit prefixes as well [15]. Thus, SPE performs the table expansion for fewer number of prefixes compared to a full table expansion and provides the most coverage of IP addresses with cacheable prefixes.

An example of SPE for a 5 bit addressing scheme is given in Figure 5.3 which expands prefixes of size 3 or less. The original trie depicted in Figure 5.3(a) is similar to our previous example bus has one more non-cacheable prefix of size 3 (Prefix II at node 7). As depicted



(a) Original Trie



(b) Trie with SPE

Fig. 5.3. SPE: The Proposed Partial Table Expansion.

in Figure 5.3(b), SPE fully expands short prefixes (whose sizes are less than or equal to 3) by pushing them down the trie until they either become leaf nodes (cacheable), or become 4 bits long. In this example, Prefix I is completely expanded by Prefix I-1 and Prefix I-2. But Prefix II is only pushed down to node 15 and is replaced with prefix II-1. In a 32 bit addressing scheme applicable to IPv4, SPE pushes the short prefixes down the trie until they either become leaf nodes (cacheable), or become 17 bits long. Thus all short prefixes are cacheable. This table transformation (SPE) has the following advantages:

1. Routing table expansion is limited to those prefixes that provide the greatest coverage

of the IP address space. This is the important difference between SPE and the full expansion. The increase in the table size is much less than a fully expanded table (See Chapter 6) while the non-cacheable prefixes with the most coverage are replaced with cacheable prefixes.

2. All short prefixes are cacheable and a length check is sufficient to determine if a prefix is cacheable or non-cacheable.

3. Table updates are simpler compared to a fully expanded trie. In a fully expanded trie, if a new prefix matches with an other prefix (either shorter or longer), several extra new prefixes must be added to the table, no matter what size these two prefixes are. In table with SPE, if the new prefix and old matching prefix are both longer than 16 bits, no table modification is required and the new prefix can be simply added to the table.

If the LUT is implemented in a two stage pipelined TCAM, our HLPM technique can be employed to easily find the longest cacheable prefix on a cache miss. This hardware-based LUT implementation is described in detail in Section 5.3. Section 5.4 describes a Multizone Pipelined Cache (MPC) based on the SPE table expansion.

5.2.4 Expansion-Free (EF) Software Lookups

Lookups may be implemented in hardware or software. A software lookup walks down the trie to find the longest matching prefix for an IP address [7]. The search ends when there are no more branches to take. The longest matching prefix is the last prefix encountered in the trie. Some of these longest matching prefixes might not be cacheable in a non-expanded table. We propose a new *Expansion-Free* (EF) method to generate cacheable prefixes using a simple and inexpensive mechanism during a software lookup. EF forwards the generated cacheable prefixes to the cache but does not store them in the lookup table, thus eliminating problems associated with table expansion. Figure 5.4 illustrates the EF method on the example of Figure 5.2. Let n be the last node visited during a traversal of

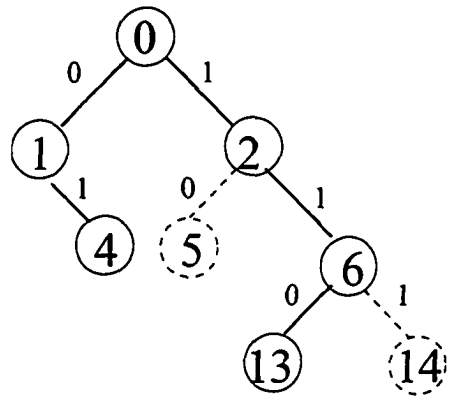


Fig. 5.4. Expansion Free Transformation.

the trie for an IP address, and let p be the last node visited containing a prefix during the traversal. EF has three rules.

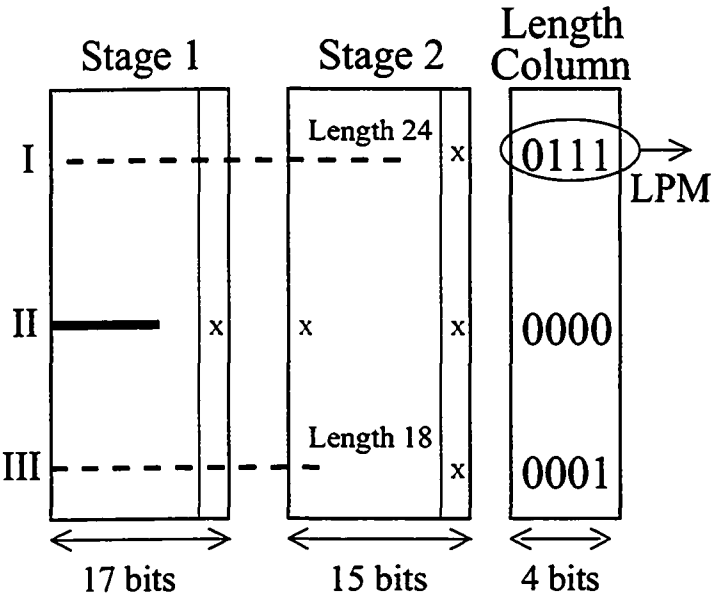
1. If $p = n$ is a leaf node in the trie, then the prefix in p is cacheable and can be forwarded to the prefix cache. For example, for IP addresses covered by node 4, $p = n = 4$. Thus the prefix in node 4 is cacheable and is forwarded to the cache as the lookup result.

2. If $p = n$ is not a leaf node in the trie, p is not cacheable. In Figure 5.4, assume an IP address matches node 5, $n = p = 2$. A cacheable prefix can be produced by adding the next bit in the address (in this case a 0) to the path followed to encounter p. This generated cacheable prefix is forwarded to the cache.

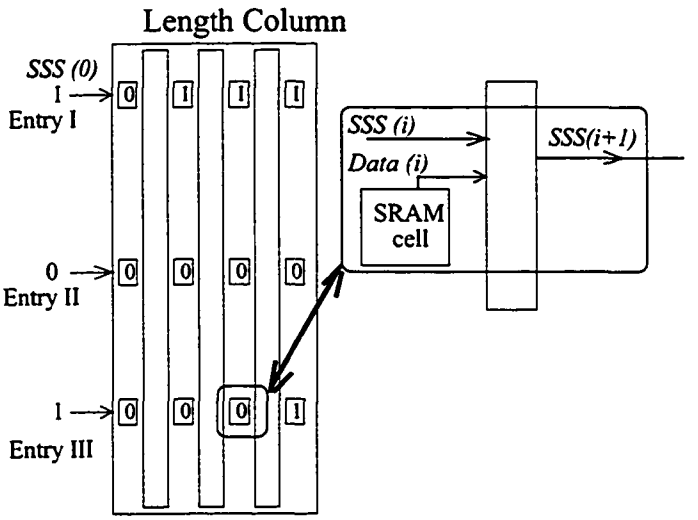
3. If $p \neq n$, the prefix in p is not cacheable. In Figure 5.4, assume an IP address matches node 14 in the trie. For this IP address, $p = 2$ and $n = 6$. A cacheable prefix is produced by adding the next bit in the address (in this case a 1) to the path traversed to find n. This generated cacheable prefix is forwarded to the cache.

5.3 HLPM-based LUT for SPE Implementation

In this section we employ an HLPM-based pipelined TCAM to implement the LUT. The general description of HLPM is given in Chapter 4. We scale and modify HLPM to be



(a) Pipelined TCAM



Data (i): Data at stage i
SSS (i): Second Search Signal at stage i

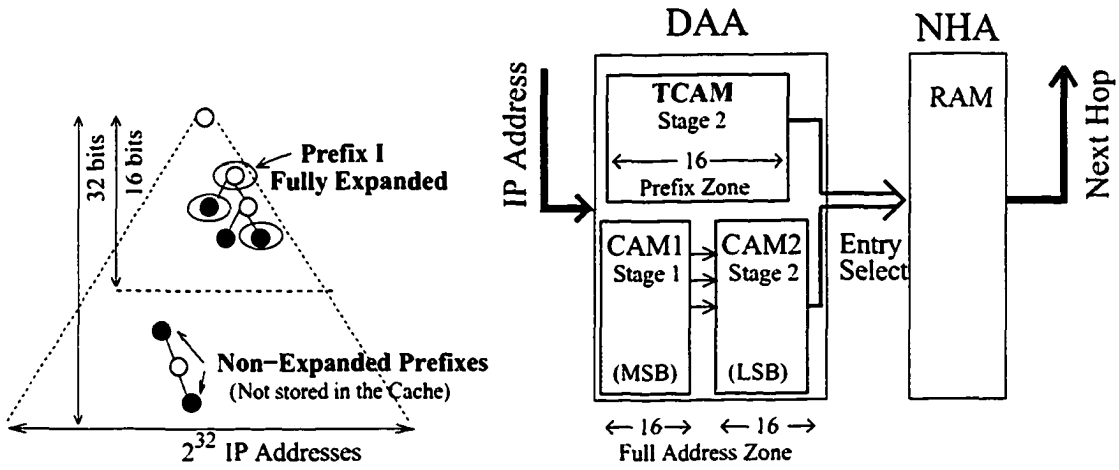
(b) Length Column

Fig. 5.5. Two-stage Pipelined TCAM with HLPM.

applicable to our cache in our forwarding system. Note that the design presented in this section is a demonstration of the fact that HLPM is easily scalable to any application. Figure 5.5(a) depicts the two stage pipelined TCAM which stores 32 bit IPv4 prefixes. The first stage stores the 17 Most Significant Bits (MSB) of prefixes and the second stage stores the 15 Least Significant Bits (LSB). The *Length Column* records for each entry, as a 4-bit binary coded value, the number of non-don't-care bits stored in the second stage. For example *0000* is stored in the corresponding *Length Column* entry of a prefix with size 17 (or less), because there is no non-don't-care bit in the second stage before the don't care bits. *0111* and *0001* are stored for prefixes of length 24 and 18, respectively.

When an IP address misses the cache, it is searched in the LUT. This search is done in two steps in our two-stage pipelined TCAM. In step one, the MSBs (bits 31 to 15) of the IP address are applied to the first half of the TCAM (all entries are searched in parallel). If the IP address matches with any entry that has a *don't care* value in its last cell (the 17^{th} bit), it means that the IP address matches with a prefix of length less than 17. SPE ensures that this prefix is cacheable and is the LPM (See Section 5.2.3). Thus the search ends and the second stage of the pipeline is a No-Operation task for that IP address. For example, in Figure 5.5(a), if an IP address matches with prefix II, this prefix and corresponding output port can be forwarded to the cache. The required TCAM entry modifications are depicted in Figure 4.2 in Chapter 4.

However, if the search in the first stage results in a matching entry with no *don't care* in the last cell, it means the LPM is more than 16 bits long and a second step search is required. For example, assume an IP address matches with prefix I in Figure 5.5(a). In the second stage, the LSB of the IP address (bits 14 to 0) are applied to the second part of the matching entry in the first stage (entry I). If a prefix matches with the IP address in the second stage as well, the matching prefix is found. The full IP address and the corresponding output port can be forwarded to the cache. However, multiple long prefixes might match with an IP address after the second stage. For example, an IP might match with both Prefix I and III as shown in Figure 5.5(a). In this case, the data in the Length Column resolves the LPM as explained in the general description of HLPM presented in Section 4.2.3. The modification



(a) SPE Table expansion

(b) MPC Memory Allocation

Fig. 5.6. MPC With SPE.

required for the Length Column is depicted in Figure 5.5(b).

5.4 Multizone Pipelined Cache (MPC) Architecture

This Section describes the Multizone Pipelined Cache (MPC) [43]. Figure 5.6 presents a structural description of MPC. The DAA is divided into two parts horizontally. The two parts form the two zones of the cache, and have independent sizes. The upper *Prefix Zone* stores *short* IP prefixes, which are 16 or fewer bits long. The lower *Full Address Zone* stores full 32-bit IPv4 destination addresses. The *Full Address Zone* is further divided vertically, with each entry split in half. The most significant 16 bits of the address are stored in `CAM1`, while the least significant 16 bits are stored in `CAM2`. A TCAM is used for the Prefix Zone implementations. A binary CAM is used to implement the Full Address Zone of MPC.

MPC searches all entries of the DAA in parallel for an IP address. If the MPC finds the address in the DAA, a cache hit occurs, and the corresponding next hop is read from the NHA and returned to the processor. If no entry in the DAA matches the IP address, a cache miss occurs. In this case, a lookup in the full routing table is performed and the cache is updated with the new destination address/next hop pair.

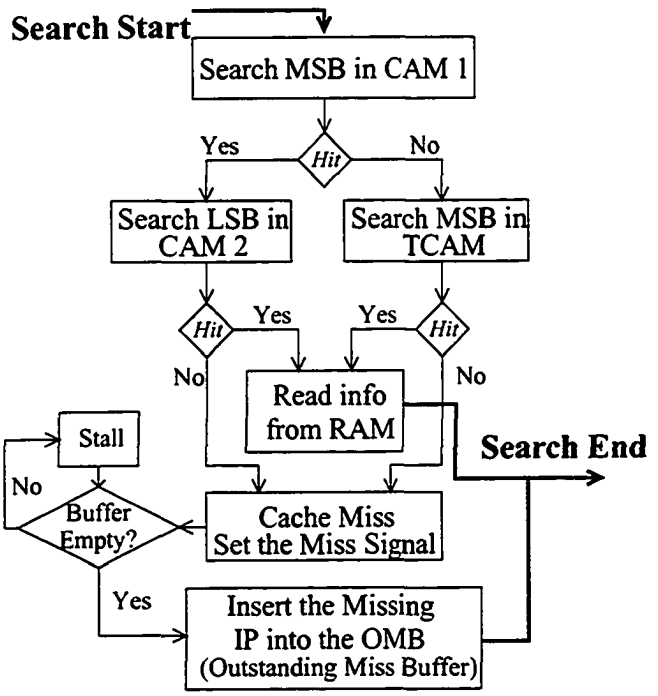
5.4.1 Cache Functionality

Breaking the DAA into three pieces allows cache lookups to be pipelined. The pipeline has three stages: (1) a lookup in CAM1; (2) a lookup in either CAM2 or the *Prefix Zone*, as required by the results of the first stage; (3) an access to the *NHA RAM* (on a hit) to return the lookup result (forwarding information or cache miss indication). In stage 1 the most significant 16 bits of the address are applied to CAM1. If there are any matches in CAM1, in stage 2 the corresponding entries of CAM2 are searched with the 16 least significant bits of the address to complete the full-IP match in the *Full Address Zone*. Otherwise, stage 2 applies the 16 most significant bits of the address to the prefixes cached in the *Prefix Zone*. If there is a match in either the *Full Address Zone* or the *Prefix Zone*, stage 3 accesses the RAM location corresponding to the matching entry, and returns the next hop data as the lookup result. SPE ensures that an IP address either hits the Full Address Zone *or* the Prefix Zone, but not both (See Section 5.2.3).

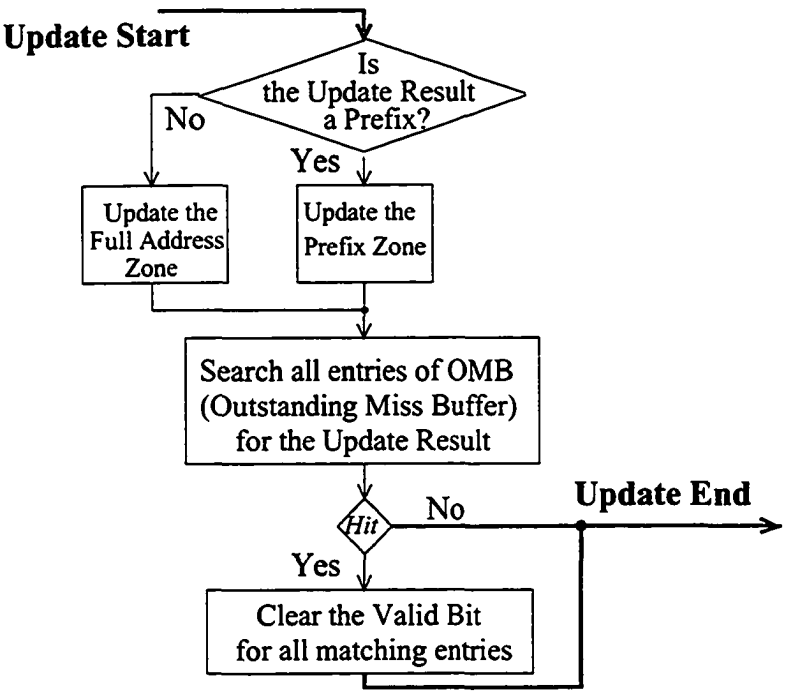
When there is no match either in the Full Address Zone or in the Prefix Zone, a cache miss is reported. A routing table search returns the routing information that is then stored in the MPC. The time required to complete this search and store the value in the cache is known as *miss penalty*. Servicing a miss is time consuming because of the slow main memory accesses to the routing table. MPC stores recent misses in an *Outstanding Miss Buffer* (OMB) until the processor returns the lookup results (see Section 5.4.2). Figure 5.7(a) depicts the pipeline flow diagram for a cache search. The diagram for an update is in Figure 5.7(b). The update process is later discussed in detail in Section 5.4.3.

5.4.2 Outstanding Miss Buffer

MPC uses the OMB to store recent misses until the processor returns their lookup results. Without OMB, MPC would need to stall while each cache miss is serviced. Blocking hinders cache throughput because further cache searches cannot proceed until the lookup is performed and the cache updated, even if pending requests would hit the cache. When a miss occurs in the non-blocking MPC, the address is stored in the OMB and the cache



(a) Cache Search



(b) Cache Update

Fig. 5.7. Flow Diagram of the Cache Performance.

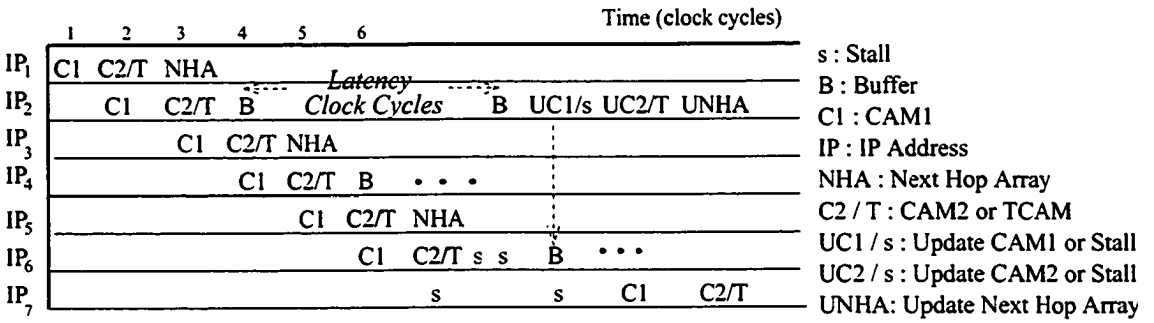


Fig. 5.8. Pipeline Diagram of the Cache.

continues performing lookups. If subsequent IP addresses hit the cache while a miss is being serviced, a *hit under miss* occurs. A *miss under miss* (secondary miss) occurs when a subsequent IP address also misses the cache. Secondary misses are stored in the OMB until the buffer is full, at which point MPC blocks and the processor stalls until misses are serviced and removed from OMB. An example of MPC functionality with a two-entry OMB is given in Figure 5.8. In this example, IP2, IP4 and IP6 are cache misses. The MPC is able to search for IP3 and IP5 and forward their corresponding information to the processor while the main table lookup for IP2 is in progress. The MPC stalls after searching for IP6 because OMB is full. No new IP can be searched until IP2 is serviced and removed from OMB to make room for IP6.

5.4.3 Cache Update

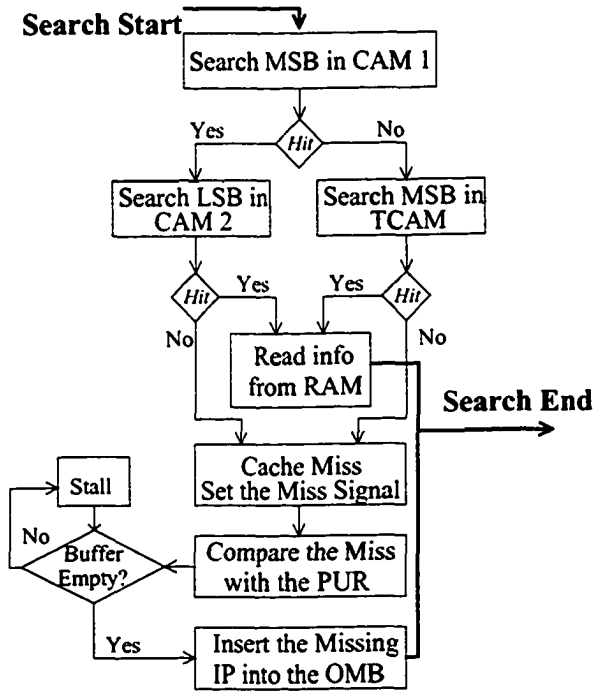
When the lookup result of a pending IP address in the OMB comes back from the main memory lookup, the MPC updates either the prefix zone or the full address zone with the corresponding information according to the MPC replacement policy. An expansion of the lookup table ensures that the result is either a short prefix that updates the prefix zone, or a full address with 32 bits that updates the full address zone (See Section 5.2.3).

MPC requires a pipeline stall to update the data stage by stage. After an update is complete, the missing IP is removed from OMB. However, other pending IP addresses in the OMB might be identical to the recently updated IP address. Additionally, an update result might be a prefix covering multiple pending IP addresses in the OMB. To ensure

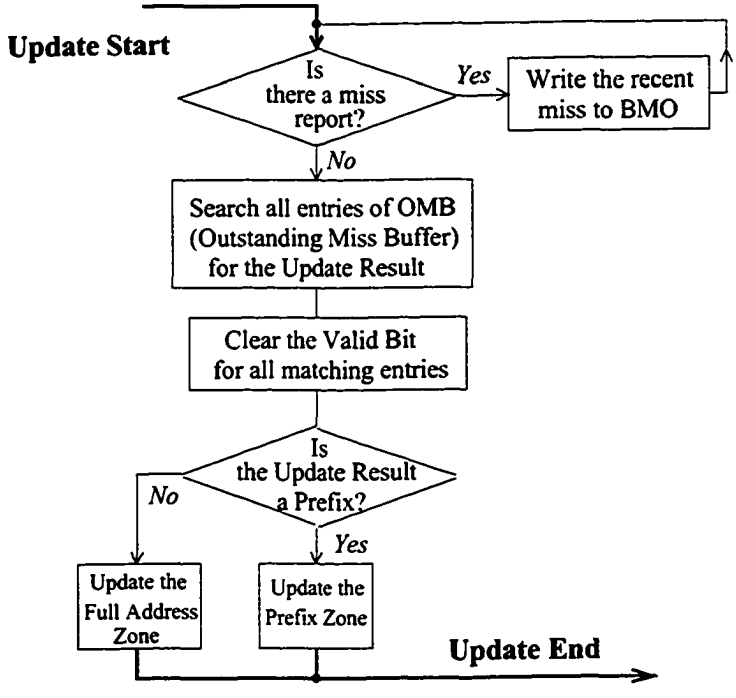
that the same lookup result is not written into the cache multiple times, we implement the OMB as a 33-bit CAM. Each OMB entry stores a 32 bit address and a *valid bit*. Only valid entries require a software lookup and cache update. After each update, an associative search of OMB identifies all matching entries. If the lookup result is a prefix, its *don't care* bits are externally masked to ensure that they match with the data in OMB. The valid bits of all matching entries are cleared. A second search for those matching OMB entries will now hit the cache, and provide the processor with the next hop information. The flow diagram for an MPC update is shown in Figure 5.7(b).

This simple update scheme allows the cache to issue update requests one by one. After one update is complete the next update request can be issued. However, a more complicated update scheme can be employed to allow the lookup algorithm and the corresponding interface to decide when pending requests are serviced, how many requests are serviced at a time, and the order in which the requests are serviced with no restrictions. We call the update algorithm *Out-Of-Order Cache Update*, or OCU, and present it in Figure 5.9(b). OCU lets the interface to the lookup and the lookup algorithm benefit from out-of-order processing and batch requests to move data more efficiently. The general functionality of this algorithm is similar to the simple update algorithm, but there are three issues requiring special consideration, namely update/lookup interlacing, batch updates, and contention for the OMB.

Cache lookups and cache updates are both pipelined operations that move through the same pipeline stages. Therefore, both lookups and updates can be fed arbitrarily into the pipeline without introducing stalls. If an update precedes a lookup, the lookup will see the updated data at each stage. Alternatively, if the lookup generates a miss and writes to the OMB before an update starts, the update will clear that entry from the OMB. However, a lookup that occurs shortly before an update could produce a miss on the same data that the update is writing into the cache. If the miss were written into the OMB, a second main table lookup would return the same result, and the second update would write a second, identical entry into the cache. We wish to avoid this situation. Therefore, before an update starts, the main table lookup result is stored in the *Pending Update Register* (PUR). Every



(a) Cache Search



(b) Cache Update

Fig. 5.9. Search / Update Diagram with Free Interface.

cache miss checks the missed value against the address or prefix in the PUR before writing to the OMB. If there is a match, the next hop information in the PUR can be forwarded as a cache hit. Otherwise, the miss is written to the OMB just as in the simple scheme. After an update is finished, the PUR is replaced with a zero value, which cannot match any address.

The lookup system may batch update requests, and subsequently batch the lookup results, in order to improve data movement efficiency. A batch request can easily be formed by taking multiple valid entries from the OMB. However, batch updates conflict with the serialized cache update mechanism. Furthermore, batch and out-of-order lookups introduce the problem that two requests may generate the same lookup result, which should only be written to the cache once. To rectify this situation, lookup results are placed in a FIFO, and sequentially moved into the PUR. Therefore, each lookup result generates a separate cache update, and these updates are completely serialized. A second identical update result will not match any valid entries in the OMB, and will be discarded.

With OCU, the OMB becomes a point of contention in the system. Cache misses write to the OMB, while cache updates read from and write to the OMB. Only one of these actions can be performed in a given clock cycle. The write from a miss occurs at the end of the lookup pipeline. The accesses for an update occur at the beginning of the update, before it enters the pipeline, and multiple updates are serialized. Therefore, we prioritize miss writes over update accesses. In the worst case, a long sequence of misses may cause an update to be delayed until the OMB fills and blocks further cache lookups. However, the OMB is small, and any cache hit provides an opportunity for an update to proceed.

An example of all three cases are depicted in detail in Figure 5.10. In this example an update request comes back to the cache after two clock cycles. Thus the update result for IP_2 is written to PUR at the end of cycle 5. Since there is a miss report for IP_4 at the end of cycle 5, the miss must be compared with the PUR and written to the OMB at cycle 6. IP_4 is not written in the OMB if IP_4 matches with the PUR. Otherwise the valid bit of IP_4 in the OMB is set.

At cycle 7, again there is another missing IP waiting to be compared with the PUR and be written in the buffer. In cycle 8, the OMB starts the search operation before the update.

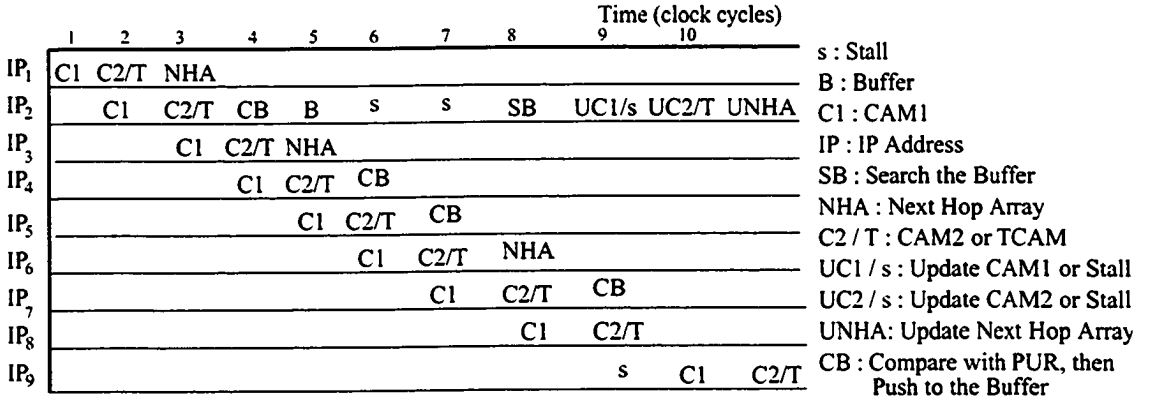


Fig. 5.10. Update Complications.

During cycles 9, 10 and 11, the cache is updated with the update result of IP_2. However if the OMB was full at the end of cycle 5, IP_4 could not be written to OMB in cycle 6. The OMB would search for update results of IP_2 at cycles 6 and write IP_4 in cycle 7 (if empty).

5.5 Summary

In this Chapter we have proposed MPC, a non-blocking, multizone, half-prefix half-full address cache that dedicates different zones to different lookup prefix lengths. The *Prefix Zone* is able to store and search prefixes with 16-bits or less. The *Full Address Zone* stores and searches for full IP addresses whose lookup prefixes are more than 16 bits long. Prefix caching increases the cache coverage while a relatively small table expansion is required. The EF method, proposed in this Thesis, completely eliminates table expansion for software lookups. Also MPC potentially can achieve higher throughput and low power consumption due to pipelining. The effective miss penalty is also reduced by using the non-blocking buffer to let the cache search for new IPs while waiting for the lookup results of cache misses. The next Chapter presents the simulations results and performance evaluations of MPC and HLPM.

Chapter 6

Performance Analysis and Simulation

6.1 Overview

This Chapter presents a simulation-based performance evaluation of the new forwarding mechanism described in this thesis. This performance study used a high level architectural simulator run with real IP traces and lookup tables of three distributing, neither core nor edge, routers. Table 6.1 contains the characteristics of these traces.

Our forwarding architecture is comprised of the MPC and the Look Up Table as described in detail in Sections 5.4 and 5.3 respectively. The IP addresses, existing in our real traces, are referenced to the cache (MPC) one by one. If an IP address misses the cache, after some clock cycles, the lookup result of the missing IP is returned to the cache. We do not specify the LUT at this point and we simulated the performance of MPC independent of the type of the LUT employed in the system. The LUT can be either implemented in software or hardware. Note that the LUT implementation has a direct impact on the number of clock cycles it takes the cache to receive the lookup results for a missing IP address. But this delay not only depends on the LUT speed, but also on the whole router system to carry the missing IP from the cache to the LUT, perform the look up and send the results back to the cache. For example when a single LUT handles multiple caches on multiple Input / Output interfaces, the contention between lookup requests from differ-

TABLE 6.1

Trace Characteristics

	ISP1	ISP2	ISP3
Trace Length (Packets)	99117	98948	98142
Routing Table Size (Prefixes)	10219	10219	6355

ent caches, might increase the lookup delay as well. This delay relates to the Cache Miss Penalty in general and is presented as a parameter named *Latency* in our simulations. Also note that the cache miss rate is totally independent of the LUT implementations. Thus in our simulations, the cache (MPC) performance is evaluated independent of the LUT implementations. Later in this Chapter we present the simulation results of the HLPM-based LUT for our forwarding mechanism as well.

Before we evaluate the MPC performance in detail, we consider the question raised in Section 5.2: To what extent do non-cacheable prefixes impact cache performance? If the LPM of most IP addresses are cacheable prefixes, there is little benefit from table expansion. In that situation, the cache would store full IP addresses whose LPM are non cacheable prefixes, and there is no need to generate cacheable prefixes. Thus, we simulated a cacheless architecture where all IP addresses reference the LUT directly. The results in Table 6.2 indicate that the LPM of up to 47% of all IPs are actually non-cacheable prefixes. Thus the spatial locality of the cache decreases dramatically if IP addresses are cached in full due to non-cacheable prefixes. Table 6.3 compares the miss rates of different cache types. Based on these simulation results, if non-cacheable prefixes are cached in full (in the *Simple Prefix Cache*), the miss rates are close to the full address cache, while Ternary CAM is required for implementations of the prefix cache. Thus the non-cacheable prefixes, make a Simple Prefix Cache degrade to a Full Address Cache.

However in the ISP3 trace very few non-cacheable prefixes are referenced. A careful investigation of the ISP3 trace indicates that all the IP addresses of this trace are covered with 7% of the prefixes in the LUT, which indicates that there is very strong locality in this traffic. Although this amount of locality is not commonly observed in routers closer to the

TABLE 6.2

Non-cacheable prefixes in a LUT.

	ISP1	ISP2	ISP3
Referenced Non-cachebale Prefixes %	46.8	37.8	2

backbone, we completed our simulations for this trace as well as our other two traces.

6.2 MPC Performance Evaluation

To evaluate the performance improvements achieved by MPC, several cache types are simulated and compared. Table 6.3 compares the miss rates of the MPC with a Full Address Cache, a Simple Prefix Cache (as explained above) and a Full Prefix Cache (fully expanded). Note that a cache hit in MPC is either in the full address zone (a hit in both CAM1 and CAM2) or in the prefix zone (a miss in CAM1 and a hit in the TCAM). Also, MPC miss rates, presented in Table 6.3, represent miss rates in the full address zone and the Prefix zone in total. A cache miss in the full address zone is the result of a hit in CAM1 but a miss in CAM2. A miss in the TCAM, is the result of a miss in both CAM1 and the TCAM. For a fair comparison, the Full Address cache is simulated as a two-zone two-stage pipelined cache with equal sized zones. This architecture caches full IP addresses and is implemented in a 32 bit binary CAM. The Simple Prefix Cache and the Full Prefix Cache are 32-bit Ternary-CAMs that store prefixes. The former stores cacheable prefixes and stores full addresses if the lookup results are non-cacheable prefixes. The later uses a fully expanded version of the real lookup table (LUT) in which all prefixes are cacheable.

The Full Prefix Cache clearly outperforms all other caches. However, the prefix caches must be implemented in a 32-bit TCAM. Since the area required for a TCAM cell is almost twice the area of a CAM cell, MPC and the IP Cache use half the area of a prefix cache with the same number of entries. To compare the performance of caches with the same storage area, MPC and the Full Address cache should be compared to a prefix cache with

half as many entries. The simulation results indicate that for equal cache size (storage area), the performance of MPC is almost as good as the full prefix cache. Moreover, the full prefix cache requires full LUT expansion while MPC can be implemented with SPE (See Section 5.2. Table 6.4 compares the total number of prefixes in the LUT after the expansion for a prefix cache and MPC.

Lookup tables contain redundant information. A prefix p_i is redundant if the lookup table still returns correct results when p_i is removed from the table. Figure 6.1 gives two examples of redundancy in a LUT. In this example, p_1 at node 1, encompasses p_2 at node 8, and that they both forwarded to the same port. Since there is no other prefix on the path from p_1 to p_2 on the lookup trie, p_2 merely duplicates information from p_1 lower in the trie, and can be removed with no change in the lookup results. Figure 6.1(b) depicts the new trie in which p_2 is removed. On the other hand, p_3 at node 5 and p_4 at node 6, of the same length, differ only in their last bit, are both forward to the same port. Those two prefixes can be replaced with one new prefix, p_5 at node 2, which is identical to p_3 and p_4, but has a "don't care" in the last bit. Figure 6.1(c) depicts the trie with no redundancy. Some software based lookup tables compress the total trie by removing all the redundant prefixes [7]. Although the initial table size is significantly reduced in such a compressed table, it is difficult to perform some table updates, and to keep updates from generating redundant prefixes. Thus, the table usually has some redundancy in it.

TABLE 6.3

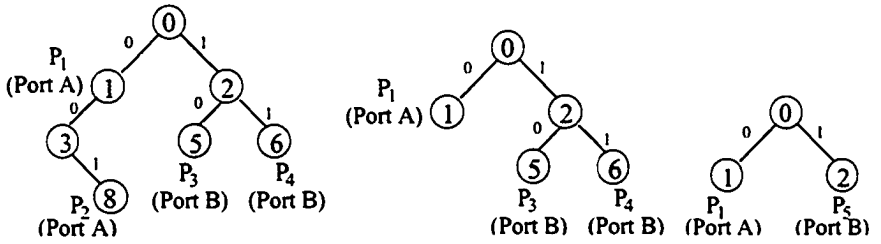
Miss Rates (%) vs. Cache Sizes (No. of Entries) for Three Traces

	ISP1			**ISP2**			**ISP3**		
Entries	**512**	**1024**	**2048**	**512**	**1024**	**2048**	**512**	**1024**	**2048**
Full Address Cache	22.7	15.4	10.5	10.8	7.2	4.9	3.6	2.2	1.9
Simple Prefix Cache	17.2	11.2	6.5	7.1	4.5	3.1	0.6	0.6	0.6
MPC	15.5	7.9	3.7	6.2	3.3	2.0	3.0	2.0	1.6
Full Prefix Cache	7.4	2.5	1.4	2.9	1.3	1.2	0.5	0.5	0.5

TABLE 6.4

Number of Prefixes after Table Expansion

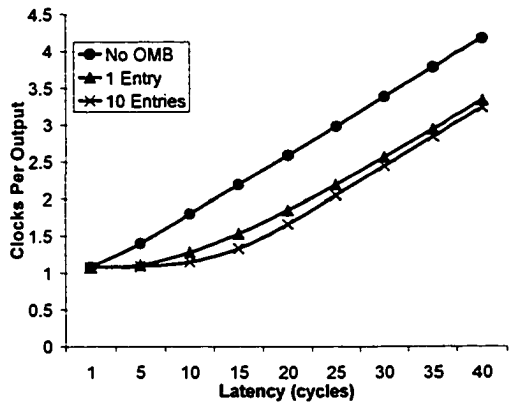
	ISP1		ISP2		ISP3	
	Entries	% Larger	Entries	% Larger	Entries	% Larger
Original Table	10219	–	10219	–	6355	–
Full Expansion	30620	199	30620	199	7313	15
SPE	17485	71	17485	71	6469	2



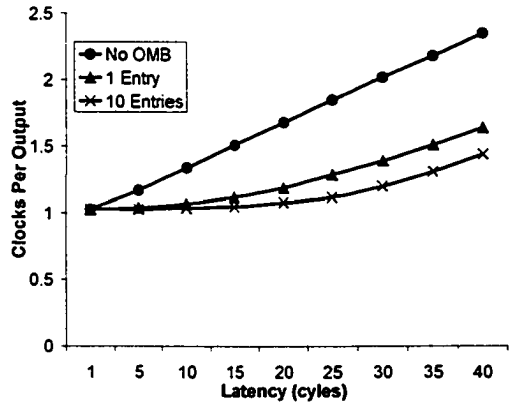
(a) (b) (c)

Fig. 6.1. Examples of Existing Redundancy in a LUT.

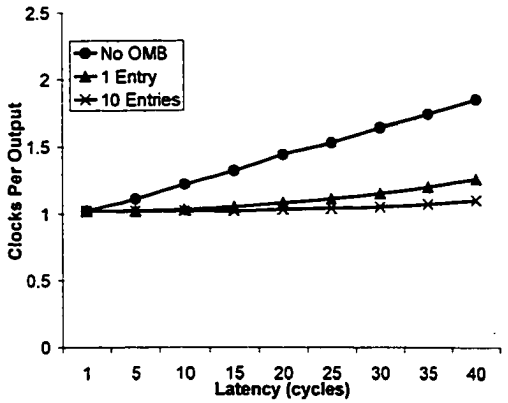
If redundancy is removed from a table, there are fewer prefixes to cache, and the effectiveness of caching should improve. Furthermore, redundant prefixes often give rise to situations where a short prefix unnecessarily encompasses an redundant prefix. In the ISP1 and the ISP2 tables 27.0% of the prefixes are redundant while 28.6% of the prefixes in the ISP3 table are also redundant. To investigate the impact of the table redundancy on the cache performance, we removed the redundant information from our real tables and then transformed the tables to ensure correct cache results (See Section 5.2). As shown in Table 6.5, cache miss rates are equal or better when non-redundant tables are used. Section 6.2.3 presents the impact of redundancy on the Lookup scheme.



(a) ISP1



(b) ISP2



(c) ISP3

Fig. 6.2. CPO vs. Latency for 1K-Entry (equally sized zones) MPC.

TABLE 6.5

Miss Rates (%) vs. Cache Sizes (No. of Entries) for Three Traces with No Redundancy in the LUTs

	ISP1			ISP2			ISP3		
Entries	**512**	**1024**	**2048**	**512**	**1024**	**2048**	**512**	**1024**	**2048**
Full Address Cache	22.7	15.4	10.5	10.8	7.2	4.9	3.3	2.0	1.9
MPC	15.5	7.9	3.7	6.0	3.2	2.0	3.0	2.0	1.6
Full Prefix Cache	7.4	2.5	1.4	2.5	1.1	1.1	0.3	0.3	0.3

6.2.1 OMB Performance Evaluation

MPC uses a small buffer (OMB) to hide the miss penalty. As mentioned before, the simulator uses a *latency* parameter to model miss penalty. A cache that has no buffer to store recent misses has to stall at each miss and wait until the update result is returned to the cache. To evaluate the impact of the miss penalty we measure a metric called CPO (Clock Per Output) that reports the average number of clock cycles necessary to provide the Next Hop Information for an IP address. Figure 6.2 depicts CPO versus Latency for MPC with no OMB, OMB with a single entry, and OMB with 10 entries. As expected, CPO increases linearly with latency for a cache with no buffer. For small latencies, in an MPC with a single entry OMB, the CPO is almost independent of the latency. For larger values of latency, CPO again increases linearly, but remains less than without the OMB. The OMB becomes more important in systems where a single LUT handles the misses from multiple caches, due to contention between the caches for service from the LUT. This performance study does not simulate LUT contention directly. Effects of contention are longer and non-uniform latencies. Therefore, the overall effects of contention can be roughly estimated by observing cache throughput under longer-latency conditions.

6.2.2 Power Savings

In a CAM-based device, power consumption is an important constraint that is addressed by many designs [30, 33]. The power consumption in a CAM-based device can be separated into three components: Evaluation Power (Search power), Input Power and Clocking Power [24]. All these sources are linearly dependent on the number of entries searched. If 50% of the time, only half of the entries of the cache are searched, the effective number of entries during each search operation is reduced to 75% of the physical number of entries. Thus 25% power is saved.

MPC divides the cache entries into two zones in the DAA: the full address zone (the 32 bit CAM) and the prefix zone (the 16 bit TCAM) as described in detail in Section 5.4. We assumed these two zones are equally sized (equal number of entries). During the pipelined search operation of the cache, half of the cache entries in the full address zone (in CAM1) are always searched in the first stage of the pipeline (the 16 MSB of the IP address are searched in all entries of CAM1, See Section 5.4). The second half of the cache entries in the prefix zone are searched only if the IP address misses the full address zone during the first stage. If the IP address hits the full address zone (CAM1) in the first stage, in the second stage, the 16 LSB of the IP address are searched in CAM2. The prefix zone is searched only if the address misses CAM1 of the full address zone.

The vertical pipelining feature of MPC leads to the fact that not all cache entries are always searched for an IP address. Thus the effective number of cache entries is actually less than the physical number of entries. This results in power savings compared to the caches that search all entries. Our simulation results, presented in Table 6.6, indicate that almost 60% of the IP addresses hit CAM1, eliminating the need to search the TCAM. Since half of the total cache entries are dedicated to each zone, this results in a 30% reduction in the effective number of entries searched in the cache, and a corresponding 30% power savings in the cache search operation compared to caches that search all their entries during a search.

TABLE 6.6

CAM1 Hit Rates

# Entries	ISP1 %	ISP2 %	ISP3 %
512	62	64	74
1024	63	65	74
2048	63	65	74

6.2.3 The HLPM-based LUT Performance Evaluation

In the forwarding architecture, a cache miss results in a reference to the LUT. The LUT stores the routing prefixes after SPE expansion. We simulated the HLPM performance considering one MPC with 1K entries. Table 6.7 shows the simulation results for the real lookup tables and for the ones with redundant information removed. We found that approximately 35% of measured expanded prefixes stored in the LUT are short (less than or equal to 16 bits). Almost 28% of the missing IP addresses match these short prefixes (cacheable prefixes) for ISP1 and ISP2.

Cacheable prefixes of the ISP3 table are referenced infrequently because most IPs in this trace match with a very small set of cacheable prefixes, In other words, the ISP3 trace has very high spatial locality. When those prefixes are cached, most IP addresses hit the cache. This result is also observed in MPC simulation results given in Tables 6.2 and 6.4. In a prefix cache, the miss rates does not improve when the cache size increases. This insensitivity to cache size suggests that only a small number of prefixes need to be cached to hit most addresses in the trace. The miss rates improve very little for larger MPC or a larger full address cache. Naturally, this increase is due to caching full IP addresses.

As shown in Table 6.7, Length Column searches are only required for a portion of the matching prefixes. Almost 40% of the time, the LUT can find the LPM with no Length Column search. The pipelined design saves power by avoiding unnecessary searches in the second stage of the missing IPs that match with cacheable prefixes. When the missing IP matches with a short cacheable prefix, the second stage of the pipeline is a No-Op and no

TABLE 6.7

Simulation Results for HLPM-based LUTs (Tables with redundancy are the real tables).

	ISP1		ISP2		ISP3	
Redundancy	Yes	No	Yes	No	Yes	No
LUT Short Prefixes %	36.6	41.6	36.6	41.6	19.5	23.2
Referenced Short Prefixes %	27.8	27.8	27.2	29.1	<1	<1
Second Level Search %	55.8	55.8	59.2	57.5	70.0	83.0
Power Savings %	14.0	14.0	13.6	14.5	<1	<1

search is performed, as described in detail in Section 5.3. Thus fewer bits are searched. This leads to potential power savings. The power savings estimates reported in Table 6.7 are in comparison with a standard full length TCAM processing the same traces. In previously reported pipelined TCAM designs, matching entries are searched to the last stage, but in our pipelined TCAM, matching entries are searched until a *don't care* is observed and not all matching entries are searched in all stages. Since low power TCAM designs use most of the power for matching entries [2], this power reduction directly affects the total TCAM power consumption. The HLPM architecture should be even more power-effective for IPv6 prefixes that have wider prefix length variation.

6.3 Summary

In this Chapter, we presented our simulation results for both the Hardware-based Longest Matching Prefix (HLPM) scheme as well as our Multizone Pipelined Cache (MPC). We simulate our forwarding mechanism comprised of the MPC and a HLPM-based LUT. Our MPC simulation is independent of the LUT implementations. We also simulate a situation where there is no cache and the HLPM-based LUT is referenced with IPs.

Our simulation results demonstrate that our forwarding mechanism meets our goals. MPC has lower miss rates compared to other caches while saves power through smart

search operations. The LUT is able to find the LPM with no table management and requirements as well as saving power for short prefixes.

Chapter 7

Conclusions and Future Work

In this thesis, IP forwarding is discussed and an efficient forwarding mechanism is proposed, designed and simulated. Our forwarding mechanism is comprised of an efficient routing cache and a hardware-based Lookup Table.

In summary, the list of contributions of our design is as follows.

1. (a) A novel Hardware-based Longest Prefix Matching (HLPM) technique for Ternary Content Addressable Memories (TCAM) is proposed in this thesis. We demonstrate a TCAM with HLPM is very suitable for efficient look up table implementations.

   (b) HLPM provides very fast table updates (no worst-case delays) with no table maintenance/management requirements. The prefixes can be stored anywhere in the table independent of their sizes.

   (c) HLPM can be applied to any pipelined TCAMs and is very easily scalable.

   (d) HLPM performs a two level search on entries of a TCAM. The first level resolves the LPM of prefixes ending in different stages by searching for a *don't care* in the last bit of each stage. A very simple cell modification is presented in this thesis to perform the first level search. The second level resolves the LPM of multiple prefixes ending in one common stage of the pipeline by finding the *max* value of the coded lengths of prefixes in the last stage of the pipeline.

(e) HLPM aims at a further decrease in power consumption compared to previously reported pipelined TCAM designs, by saving power for matching short prefixes.

(f) HLPM saves area compared to other fast table update solutions for TCAM based lookup tables.

2. We employ a TCAM with HLPM for the look up table implementations of the forwarding mechanism presented in this thesis. HLPM is applied to a two-stage pipelined TCAM. The first stage is 17 bits long and the second stage is 15 bits, providing 32 bits for IPv4 prefixes.

3. (a) We proposed a novel Multizone Pipelined Cache (MPC) to implement the cache of the forwarding mechanism proposed in this thesis. MPC is a non-blocking, multizone, half-prefix half-full address cache that dedicates different zones to different lookup prefix lengths. The *Prefix Zone* is able to store and search prefixes with 16-bits or less. The *Full Address Zone* stores and searches for full IP addresses whose lookup prefixes are more than 16 bits long.

(b) MPC has lower miss rates compared to previously reported caches.

(c) Also MPC potentially can achieve higher throughput and low power consumption due to vertical and horizontal pipelining.

(d) MPC uses a non-blocking buffer to let the cache search for new IPs while waiting for the lookup results of cache misses. Thus the effective miss penalty is reduced.

(e) MPC saves power through smart search operations where unnecessary search operations are avoided.

4. A novel Short Prefix Expansion (SPE) technique is proposed to allow MPC to store prefixes instead of full IP addresses to increase the locality. The increase in the table size after SPE transformation, is less than other expansion techniques while SPE allows MPC to store the short prefixes with the most coverage.

5. A novel EF method is proposed in this thesis to completely eliminate table expansion for software lookups. This techniques generates cacheable prefixes during each look up process and requires zero increase in table size.

6. A high level simulator was written in C++ and ran with real traces (IP packets and lookup tables) taken from several local distributing routers in Edmonton, Alberta.

Although different parts of the forwarding mechanism are designed and simulated through a high level architectural simulator using real traces, as presented in Chapter 6, there are many potential research directions as the future work of this thesis. The main directions include:

1. The circuit level simulations and exact timing and throughput analysis based on IP arrivals are the primary steps towards the future work of our research. The power consumption evaluations of the whole system (the cache and the lookup table) are required through a circuit level simulation.

2. Although our high level simulation results proved the efficiency of the proposed forwarding mechanism, hardware implementations and testing of both the Lookup table (TCAM with HLPM) and the cache (MPC) are required as a precise evaluation of the forwarding architecture proposed in this thesis.

References

[1] P. Berube, A. Zinyk, J. Amaral, and M. MacGregor, "The bank Nth chance replacement policy for FPGA-based CAMs," in *13th International Conference on Field Programmable Logic and Applications (FPL)*, Lisbon, Portugal, September 2003.

[2] A. Sheikholeslami and I. Arsovski, "A mismatch-dependent power allocation technique for match line sensing in content-addressable memories," *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 1958–1966, Nov. 2003.

[3] S. Y. W. C. A. Zukowski, "Use of selective precharge for low-power content-addressable memories," in *Proceeding of the 1997 International Symposium on Circuits and Systems*, 1997, pp. 1788–1791.

[4] M. Kobayashi, T. Murase, and A. Kuriyama, "A longest prefix match search engine for multi-gigabit ip processing," in *International Conference on Communications (ICC 2000)*, New Orleans, LA, June 2000, vol. 3, pp. 1360–1364.

[5] N. McKeown, "Trends in the design and analysis of Internet Routers," http://www.stanford.edu/nickm/talks.

[6] W. D. Grover, *Mesh-Based Survivable networks*, first ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2004.

[7] M. A. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, pp. 8–23, Mar./ Apr. 2001.

[8] A. S. Tanenbaum, *Computer Networks*, third ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 1996.

[9] D. Feldmeier, "Improving gateway performance with a routing-table cache," *IEEE INFOCOME 88*, pp. 298–307, March 1988.

[10] W. Shyu, C. Wu, and T. Hou, "Efficiency analyses on routing cache replacement algorithms," *IEEE International Conference on Communications*, vol. 4, pp. 2232–2236, 2002.

[11] B. Talbot and B. L. T. Sherwood, "IP caching for terabit speed routers," *GLOBECOM 99*, vol. 22, pp. 1565–1569, 1999.

[12] I. Chvets and M. MacGregor, "Multi-zone caches for accelerating IP routing table lookups," *Merging Optical and IP Technologies Workshop on High Performance Switching and Routing*, pp. 121–126, May 2002.

[13] T. cker Chiueh and P. Pradhan, "High performance IP routing table lookup using CPU caching," in *IEEE INFOCOMM (3)*, 1999, pp. 1421–1428.

[14] H. Liu, "Routing prefix caching in network processor design," in *Tenth International Conference on Computer Communications and Networks*, Scottsdale, AZ, Oct. 2001, pp. 18–23.

[15] W. Shyu, C. Wu, and T. Hou, "Multilevel aligned IP prefix caching based on singleton information," *GLOBECOM 02*, vol. 3, pp. 2345 2349, Nov 2002.

[16] T.-C. Chiueh and P. Pradhan, "Cache memory design for network processors," in *Sixth International Symposium on High-Performance Computer Architecture*, Toulouse, France, January 2000, pp. 409–419.

[17] H. Liu, "Reducing cache miss ratio for routing prefix cache," *GLOBECOM 02*, vol. 3, pp. 2323 2327, Nov 2002.

[18] T. Chen and J. Baer, "Reducing memory latency via non-blocking and prefetching caches," in *5th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Oct 1992, pp. 51–61.

[19] K. I. Farkas and N. P. Jouppi, "Complexity/performance tradeoffs with non-blocking loads," in *21st Int. Symposium on Computer Architecture*, 1994, pp. 211–222.

[20] D. Kroft, "Lookup free instruction fetch/prefetch cache organiztion," in *8th Int. Symposium on Computer Architecture*, May 1981, pp. 81–87.

[21] L. Bhuyan and H. Wang, "Execution-driven simulation of IP router architectures," *IEEE Int. Symposium on Network Computing and Applications*, pp. 145–155, Oct. 2001.

[22] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, A Design Perspective*, second ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2003.

[23] I. Arsovski, T. Chandler, and A. Sheikholeslami, "A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme," *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 155–158, Jan. 2003.

[24] H. Hsiao, D. Wang, and C. Jen, "Power modeling and low-power design of content addressable memories," *IEEE Int. Symposium on Circuits and Systems*, pp. 926–929, May. 2001.

[25] G. Thruhnanam, N. Vijaykrishnan, and M. J. Irwin, "A novel low power CAM design," in *Proceeding of 14th Annual IEEE International ASIC/SOC Conference*, Sept 2001, pp. 198–202.

[26] H. Miyatake, M. Tanaka, and Y. Mori, "A design for high-speed low-power CMOS fully parallel content-addressable memory macros," *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 8–23, June 2001.

[27] t. Juan, T. Lang, and J. Navarro, "Reducing TLB power requirements," in *Proceeding of the 1997 International Symposium on Low Power Electronics and Design*, 1997, pp. 196–201.

[28] S. C. Liu, F. A. Wu, and J. B. Kuo, "A novel low-voltage content-addressable-memory (CAM) cell with a fast tag-compare capability using partially depleted (PD) SOI CMOS dynamic-threshold (DTMOS) techniques," *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 712–716, November 2001.

[29] F. Shafai, K. J. Schultz, G. F. R. Gibson, A. G. Bluschke, and D. E. Somppi, "Fully parallel 30-MHz, 2.5 Mb CAM," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1690–1696, November 1998.

[30] A. Efthymiou and J. Garside, "A CAM with mixed serial-parallel comparison for use in low energy caches," *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 325–329, March 2004.

[31] N. Mohan and M. Sachdev, "Low power dual matchline ternary content addressable memory," in *Proceeding of the 2004 International Symposium on Circuits and Systems*, 2004, pp. II–633–636.

[32] A. Natarajan, D. Jasinski, W. Burleson, and R. Tessier, "A hybrid adiabatic content addressable memory for ultra-low power applications," in *Proceeding of the IEEE/ACM Great Lakes Symposium on VLSI*, 2003, pp. 72–75.

[33] K. Pagiamtzis and A. Sheikholeslami, "Pipelined match-lines and hierarchical search-lines for low-power content addressable memories," in *IEEE Custom Integrated Circuit Conference*, San Jose, CA, Sept. 2003, pp. 383–386.

[34] C. S. Lin, J. C. Chang, and B. D. Liu, "A low-power precomputation-based fully parallel content-addressable memory," *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 654–662, April 2003.

[35] G. Huston, "Routing table status report," http://bgp.potaroo.net.

[36] H. H. andS. Tachibana, M.Minami, and T. Nagano, "A 2-ns, 5-mw, synchronous-powered static-circuit fully associative TLB," in *Symposium on VLSI Circuits, Digest of Technical Papers*, 1995, pp. 21–22.

[37] K. H. Cheng, C. H. Wei, and S. Y. Jiang, "Static divided word matching line for low power content addressable memory design," in *Proceeding of the 2004 International Symposium on Circuits and Systems*, 2004, pp. II–629–632.

[38] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 515–528, Oct. 1999.

[39] D. Shah and P. Gupta, "Fast updating algorithms for TCAMs," *IEEE Micro*, vol. 21, pp. 36–47, Jan./ Feb. 2001.

[40] S. Sharma and R. Panigrahy, "Sorting and searching using ternary CAMs," in *10th Symposium on High Performance Interconnects*, Stanford, CA, Aug. 2002, pp. 101–106.

[41] M. J. Akhbarizadeh and M. Nourani, "Efficient prefix cache for network processors," in *12th Annual IEEE Symposium on High Performance Interconnects*, 2004, pp. 41–46.

[42] S. Kasnavi, P. Berube, V. C. Gaudet, and J. N. Amaral;, "A hardware-based longest prefix matching scheme for TCAMs," *Accepted for Publication in the IEEE Int. Symposium on Circuits and Systems, ISCAS 2005*, 2005.

[43] S. Kasnavi, P. Berube, V. C. Gaudet, and J. N. Amaral;, "A multizone pipelined cache for IP routing," *Submitted to Networking 2005*, 2005.