DESIGN AND IMPLEMENTATION OF HARDWARE ARCHITECTURES FOR HIGH-SPEED IP ADDRESS LOOKUP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NİZAM AYYILDIZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2013

Approval of the thesis:

DESIGN AND IMPLEMENTATION OF HARDWARE ARCHITECTURES FOR HIGH-SPEED IP ADDRESS LOOKUP

submitted by **NİZAM AYYILDIZ** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ________________

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** ________________

Prof. Dr. Hasan Cengiz Güran
Supervisor, **Electrical and Electronics Engineering Dept., METU** ________________

Assoc. Prof. Dr. Şenan Ece Güran Schmidt
Co-supervisor, **Electrical and Electronics Engineering Dept., METU** ________________

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU ________________

Prof. Dr. Hasan Cengiz Güran
Electrical and Electronics Engineering Dept., METU ________________

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Dept., METU ________________

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU ________________

Assoc. Prof. Dr. Nail Akar
Electrical and Electronics Engineering Dept., Bilkent University ________________

Date: **28.08.2013**

ABSTRACT

DESIGN AND IMPLEMENTATION OF HARDWARE ARCHITECTURES FOR HIGH-SPEED IP ADDRESS LOOKUP

Ayyıldız, Nizam

Ph.D., Department of Electrical and Electronics Engineering

Supervisor : Prof. Dr. Hasan Cengiz Güran

Co-Supervisor : Assoc. Prof. Dr. Şenan Ece Güran Schmidt

August 2013, 83 pages

IP address lookup modules for backbone routers should store 100Ks of entries, find the longest prefix match (LPM) for each incoming packet at 10s of Gbps line speed and support thousands of lookup table updates each second. It is desired that these updates are *non-blocking*, that is without disrupting the ongoing lookups. Furthermore, considering the increasing line rates and table sizes, the scalability of the design is very important. The goal of this thesis is developing hardware IP lookup architectures that perform single clock cycle lookups and non-blocking updates that are entirely carried out on hardware. To this end, we propose a custom TCAM architecture for IP lookup that we call *S-DIRECT-Scalable and Dynamically REConfigurable TCAM* and a complete IP lookup solution that utilizes different types of memory that we call *SHIP-Scalable Highspeed IP lookup*. Both S-DIRECT and SHIP feature a modular design that allows seamless scaling to different table sizes. We implement the developed architectures on FPGA with a resource efficient realization and provide the hardware requirements for implementation on other platforms. We demonstrate the viability of our architectures with a full implementation on FPGA that can store contemporary routing tables.

Keywords: TCAM, IP Lookup, FPGA, Dynamic Reconfiguration

ÖZ

YÜKSEK HIZLI IP ADRES ARAMA İÇİN DONANIM MİMARİLERİNİN TASARIMI VE GERÇEKLENMESİ

Ayyıldız, Nizam
Doktora, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Hasan Cengiz Güran
Ortak Tez Yöneticisi : Doç. Dr. Şenan Ece Güran Schmidt

Ağustos 2013 , 83 sayfa

Omurga yönlendiricilerindeki IP adres arama modülleri 100K'larca girdiyi depolayabilmeli, 10'larca Gbps hat hızında gelen her bir paket için en uzun önek eşleşmesini bulabilmeli ve her saniyede gerçekleşen binlerce arama tablosu güncellemesini destekleyebilmelidir. Bu güncellemelerin engelleyici olmamaları, yani süregelen aramaları engellememeleri, arzu edilmektedir. Üstelik, artan hat hızları ve tablo boyutları göz önünde bulundurulduğunda, tasarımın ölçeklenebilirliği çok önemlidir. Bu tezin amacı, tek saat işareti içerisinde arama ve engelleyici olmayan güncellemelerin tamamen donanımda gerçekleştirildiği donanımsal IP arama mimarileri geliştirmektir. Bu amaçla, S-DIRECT-Ölçeklenebilir ve Dinamik Olarak Yeniden Şekillendirilebilir TCAM adını verdiğimiz, IP aramaya özel bir TCAM mimarisi ve SHIP-Ölçeklenebilir Yüksek Hızlı IP Arama adını verdiğimiz farklı hafıza türlerinden faydalanan bütün bir IP arama çözümü sunuyoruz. Hem S-DIRECT hem de SHIP farklı tablo boyutlarına sorunsuz ölçeklenmeyi sağlayan modüler tasarım vasfına sahiptirler. Geliştirilen mimarileri FPGA üzerinde kaynak etkin bir gerçeklemeyle uyguluyoruz ve diğer platformlardaki uygulamalar için donanım gereksinimlerini belirtiyoruz. Mimarilerimizin uygulanabilirliğini çağdaş yönlendirme tablolarını depolayabilen FPGA üzerindeki bütün bir gerçekleştirim ile gösteriyoruz.

Anahtar Kelimeler: TCAM, IP Arama, FPGA, Dinamik Şekillendirme

To my lovely wife Tülin

and

our unique daughter Türkü

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his supervisor Prof. Dr. Hasan Cengiz Güran and co-supervisor Assoc. Prof. Dr. Şenan Ece Güran Schmidt for their guidance, advice, criticism, encouragements and insight throughout the research.

I would also like to express my special thanks and gratitude to Prof. Dr. Semih Bilgen, Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı, Assoc. Prof. Dr. Halit Oğuztüzün, and Assoc. Prof. Dr. Nail Akar for showing keen interest to the subject matter and accepting to read and review the thesis.

I would like to thank ASELSAN Inc. for letting me involve in this thesis study and for the facilities provided for the completion of this thesis.

A very special gratitude goes to my family for their love and support throughout all my life.

Finally, the most special thanks goes to my wife, Tülin Erçelebi Ayyıldız, for her great encouragement, support, patience and love at every moment of my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
ATB	Augmented TCAM Block
BRAM	Block Random Access Memory
BST	Binary Search Tree
BTB	Basic TCAM Block
CAM	Content Addressable Memory
CIDR	Classless Inter Domain Routing
CLB	Configurable Logic Block
DEMUX	Demultiplexer
DMA	Direct Memory Access
DNHIR	Default Next Hop Index Register
EDA	Electronic Design Automation
EPM	Exact Prefix Match
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HLS	High Level Synthesis
IP	Internet Protocol
LPM	Longest Prefix Match
LUT	Look up Table
MLPS	Million Lookups Per Second
MUX	Multiplexer
NHIM	Next Hop Index Memory
NP	Network Processor
PE	Priority Encoder
PLB	Preprocessing Logic Block
PMR	Prefix/Mask Register
RAM	Random Access Memory
SRAM	Static Random Access Memory
TCAM	Ternary Content Addressable Memory
TCP	Transport Communication Protocol
TBD	To Be Determined
XST	Xilinx Synthesis Technology

CHAPTER 1

INTRODUCTION

The number of Internet users is increasing continually. Not only the number of users increases, but also the versatility of the applications, and data traffic on the Internet increases as well. In order to meet the requirements of Internet communication, higher speed data links are put in place, and the number of connection points are increased on the networks.

An IP router has to look up the destination IP address of each incoming packet to forward the packet to its next-hop router. Consequently, *IP address lookup* is a dataplane router function and has to be implemented in hardware.

IP address lookup is carried out on a table that consists of *IP prefixes* of different lengths and their corresponding next-hop outgoing interfaces. The routing rules for IP networks require the *longest prefix match* (LPM) where the selected next-hop corresponds to the longest prefix that matches the destination address of the IP packet.

The current IP backbone routers run at 10s of Gbps and the routing tables contain 100Ks [7] prefixes. The IP lookup has to be carried out at line speed which requires a look up rate of 125 million lookups /sec at the line speed of 40 Gbps for the smallest size IP packets of 40 Bytes. It is important to note that a large fraction of IP packets in the network are of this smallest size because of the TCP acknowledgments [8].

The routing tables are updated with rates up to thousands of updates/sec [9] because of network congestion state, provided Quality of Service (QoS), addition or removal of links and faults. Furthermore, [10] states that "there is a growing interest in virtual IP routers where more frequent updates happen". To this end, there are two requirements regarding these updates. First, they should be carried out in hardware without the aid of external software such that these high update rates can be achieved. Second, they should be *non-blocking* which means that IP lookup requests are processed without any interruption during the content change of the routing table. Hence, it is a challenging task to perform line rate LPM IP address lookups and fast non-blocking routing table updates to meet these requirements. Furthermore, IP lookup architectures should *scale* with both the continuously increasing sizes of the routing tables and the operating speeds.

There is a wide variety of solutions proposed for IP lookup, implemented both on software and hardware. However, the software solutions are away from satisfying lookups at line rate for high speed data links such as OC-768, 40 Gbps [1][11].

Section 2.4 gives detailed information about the hardware-based IP lookup solutions in the literature. Here, we give an introductory information about those studies.

Hardware based IP lookup solutions are generally categorized into two classes; SRAM-based, and TCAM-based solutions. SRAM-based solutions are basically the hardware counterpart of the software-based solutions. The trie/tree based data structures, which are constructed from the routing table prefixes and their corresponding next hop information, are directly implemented on the SRAM hardware. Since tree traversal algorithms require several memory accesses until the leaf node of the tree is reached, SRAM-based solutions may suffer from high latencies, which are not desired for high-speed IP lookup. Therefore, the studies targeting SRAM platform for the IP lookup solution, generally tend to decrease the depth of the trees to be constructed, by applying some optimization algorithms on the routing table prefix set, or to increase throughput by employing parallel and/or pipelined operation. However, parallel or pipelined architectures require several SRAM memories and the memory sizes of the employed SRAMs may not be of equal size. Especially for pipelined architectures, each level of the tree is assigned to a separate SRAM, and the required size for the SRAMs doubles at each level, which makes this approach infeasible for commercial SRAM utilization. Therefore, parallel and/or pipelined SRAM-based architectures are mostly implemented on FPGA platform, which allows users to design several different-sized SRAM blocks on the same chip.

TCAMs (Ternary Content Addressable Memories) are special hardware that allows the search of the whole content simultaneously for a given key data. For IP lookup, the prefixes are sorted in ascending order with respect to their length, and the longest matching prefix is selected as the matching entry with the highest address via a priority encoder. Conventional TCAMs are larger in size compared their SRAM counterpart of the equal storage capacity, since they contain two SRAM cells for each entry, one for prefix and one for the mask, and additional comparator circuitry. Thus, TCAMs are more expensive and consume more power compared to SRAMs. Therefore researchers, who favor TCAMs for IP lookup, either try to decrease the power consumption, or to increase the storage efficiency. Power consumption techniques are similar to each other, partitioning the match line into segments, a pipelined search is carried, and search is continued only if there is a match in the previous stage. This approach results in the loss of the single cycle search capability of the TCAMs. Storage efficiency is increased by algorithmic transformations of the prefixes, and these type of approaches make the table management hard, especially for the updates. As in the case of SRAM-based solutions, there are also studies utilizing FPGAs for TCAM implementations as well.

Here, we would like to note that, none of the hardware architectures proposed for IP lookup can perform the updates in hardware as well. An auxiliary software maintains a data structure to keep the memory location information of the prefixes. This software performs preprocessing of this data structure to compute the necessary changes on the memory organization. Hence, the high speed operation can only be achieved for the lookups, but when there is an update request, the software overhead degrades the speed.

To this end, the goal of this thesis is developing hardware architectures for IP lookup with the following properties:

1. High throughput in packets/sec: Each IP lookup is carried out in a single clock cycle.
2. Efficient use of hardware resources: Utilization of different types of memory resources on hardware to store contemporary IPv4 routing tables.
3. Table update capability in hardware: Not only the IP lookup operations, but also the update operations should fully be processed and realized on hardware.
4. Non-blocking updates: The update operations should not cause any disruption in the ongoing IP lookup operations.
5. Design scalability: The proposed architecture should be scalable to larger routing table sizes for future demands.

In this thesis, we propose an IP lookup architecture, SHIP (Scalable High-speed IP lookup) that partitions a given IP lookup table over two different types of memory, TCAM and RAM, with their corresponding specific IP prefix data structures as a hybrid solution to achieve the goals listed above. We would like to emphasize that an IP lookup module that can perform single cycle lookups can be constructed using either of these memory types. Our proposal specifically targets the hardware platforms with different types of memory and computing resources such as FPGA to enable utilizing these resources altogether.

We suggest a partitioning for contemporary IP routing table over the resources a state-of-the-art FPGA (Xilinx Virtex-7 1140T) [12] making use of the prefix distribution statistics [7]. Accordingly, we implement SHIP on this platform as a single cycle hardware IP lookup solution for demonstrating its viability and a solid discussion of its features.

SHIP features a custom designed TCAM for IP lookup that we call S-DIRECT (Scalable and Dynamically REConfigurable TCAM) that is developed in the scope of this thesis.

We chose FPGA as the hardware implementation platform for our architectures. On the one hand we provide efficient implementations for S-DIRECT and SHIP on FPGA. On the other hand, we identify the required hardware capabilities for our architectures for implementation on any hardware platform.

The contributions of this thesis are as follows:

1. Comprehensive literature survey on hardware and software IP lookup architecture with comparative evaluations of TCAM implementations and IP lookup solutions that are implemented on FPGA.

2. S-DIRECT: A custom TCAM architecture for IP lookup with the following features:

   (a) An S-DIRECT TCAM of any size is constructed from Basic TCAM Blocks (BTB) each which has an individual PE logic. Hierarchically arranging the PE logic of the BTBs results in inherent PE for the resulting TCAM. The single cycle search of the TCAM includes priority encoding process. Hence, we achieve *design scalability* eliminating the requirement of custom built PE circuitry that depends on the size of the TCAM.

   (b) S-DIRECT has *non-blocking update capability*. Writing an S-DIRECT entry does not interrupt the search on the remaining entries, simultaneous update and search operations can be performed.

   (c) The update requests are processed and subsequently the updates are carried out entirely in hardware without additional software support. The update requests are served in constant time.

   (d) S-DIRECT is not limited to a specific hardware platform or a specific TCAM cell implementation provided that the hardware requirements that we define are satisfied.

3. SHIP: A hybrid SRAM and TCAM architecture for IP lookup with the following features:

   (a) S-DIRECT architecture is used as the TCAM, so that the outperforming properties of S-DIRECT are inherited

   (b) SHIP has the single cycle search capability, as the SRAM part is used as a DMA (Direct Memory Access) engine for prefix-next hop correspondence

   (c) The update requests are processed and subsequently the updates are carried out entirely in hardware without additional software support.

   (d) Non-blocking update capability is achieved if the SRAM is dual-port RAM.

4. A discussion on desired FPGA features based on the issues that we encountered for our implementations to facilitate IP lookup implementation.

The remainder of the thesis is organized as follows. Chapter 2 gives background information about the IP lookup principles and the solution space. Chapter 3 and 4 describe our TCAM architecture for IP lookup on FPGA, and our overall IP lookup solution approach for FPGA platform, respectively. In Chapter 5 FPGA architecture and FPGA-based design flow are explained in correspondence to the IP lookup solution implementation. Finally in Chapter 6 we conclude our work.

CHAPTER 2

IP LOOKUP: PRINCIPLES AND IMPLEMENTATIONS

The main goal of a communication system is transporting the data from a source point (transmitter) to a destination point (receiver). If there is a point-to-point connection between the transmitter and the receiver, data transport from source to destination is simply a transmission of the electrical signal through the established data path. However, if there are more than one transmitters and receivers that constitute a communication network, it is necessary to determine the path that the transmitted data should follow . The process of selecting paths in a network along which to send network traffic is called by a general term *"'routing"'*. For shared medium networks, the routing is performed by addressing the endpoints, and embedding the source and destination addresses into the data packets to be transported. The Internet, consisting of billions of endpoints, is actually a network of networks (Figure 2.1). Local networks (LANs, MANs) connect to other networks through *"'routers"'*, which are the network devices, where the routing process is performed (Figure 2.2). The router determines the egress port through which the incoming packet should be forwarded to its next hop, according to the destination IP (Internet Protocol) address of the packet. On the router, the relationship between the destination IP addresses and the next hops is kept in a data table, which is known as *"'routing table"'*. The router looks up (searches for) the IP address in the routing table to determine the corresponding next hop. In the literature this process is simply called as *"'IP lookup"'*.

2.1 IP Addressing Hierarchy and the Longest Prefix Match (LPM) Lookup

The construction of routing tables, and the methods that are used for IP lookup is directly related with the addressing scheme. Since the primary objective of the Internet Protocol is the interconnection of networks, a routing protocol based on network domains is a natural choice [1]. There is a two-level hierarchical addressing scheme to determine the IP addresses. At the top level of hierarchy, the network is addressed, and at lower level, hosts belonging to a network are further addressed. In IPv4 an 32-bit address is used, and the more significant bits correspond to the network address, while the remaining bits are used for host addresses. The more significant bits corresponding to the network part is called *"'address prefix"'* [1].

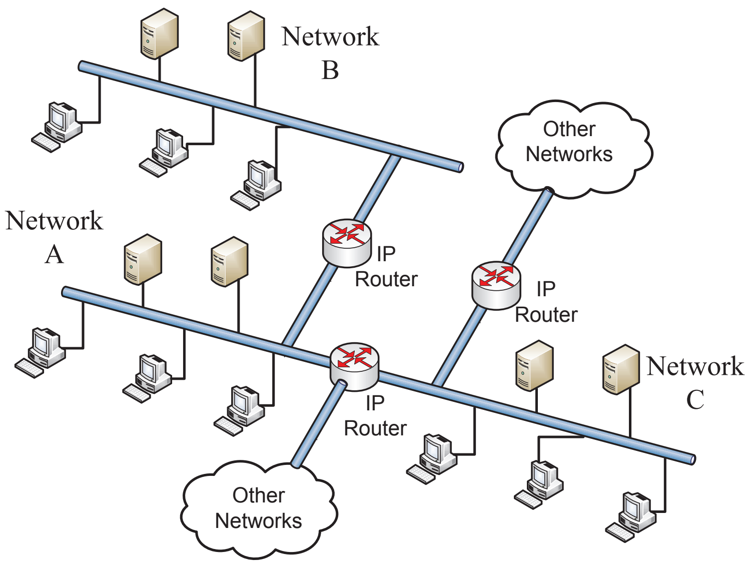


Figure 2.1: Internet illustration

Table2.1: An example routing table

Destination Address Prefix	**Next Hop**	**Outgoing Interface**
24.40.32/20	192.41.177.148	2
130.86/16	192.41.177.181	6
208.12.16/20	192.41.177.241	4
208.12.21/24	192.41.177.196	1
167.24.103/24	192.41.177.3	5

In IPv4, the prefixes are written as bit strings up to 32 bits followed by a *. For example, the prefix 1000001001010110* represents all the 2^{16} addresses that begin with the bit pattern 1000001001010110. Alternatively, prefixes can be indicated using the dotted-decimal notation, so the same prefix can be written as 130.86/16, where the number after the slash indicates the length of the prefix. IP routers forward the packets based on the network domain, until packets reach the destination network. Hence, a routing table contains the prefixes, representing the networks, and the interface address of the next router (next hop) on the way to the final destination. The next hop association can also be the outgoing or exit interface to the final destination. An example routing table can be seen in Table 2.1. The routing table is constructed and continuously updated in time according to certain objectives to achieve optimal paths (the paths with the minimum number of hops, with the minimum cost links) as well as policy decisions.

Sanchez et.al., present a comprehensive survey of state-of-the-art IP address lookup algorithms and compare their performance in terms of lookup speed, scalability, and update overhead [1]. This study was carried out in 2001, and it gives very useful information about the

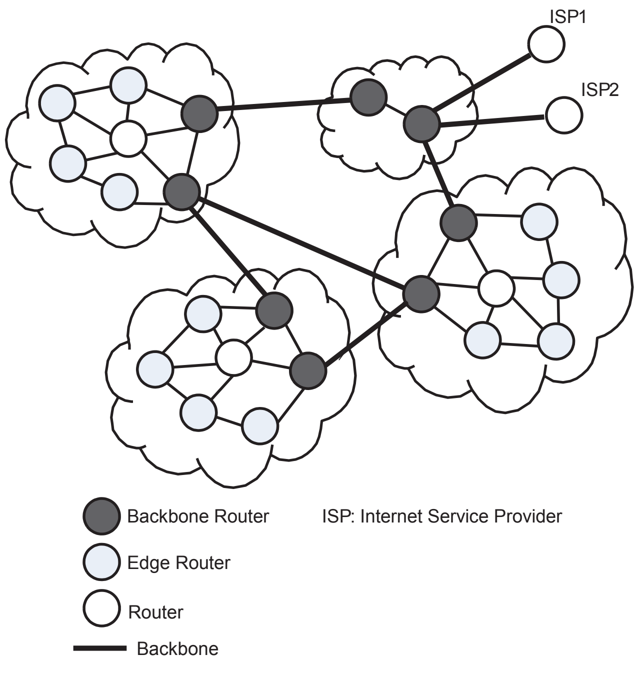


Figure 2.2: Router types

evolution of the Internet addressing scheme, and how the addressing scheme affected the construction of routing tables and the protocols used for IP lookup. While, *"‘classfull addressing scheme"’*, which was used for the initial Internet addressing, proposed a very simple addressing architecture, defining just three different network sizes (Class A, Class B, and Class C) resulted in inefficient use of the address space with the continuous growth of the number of hosts and networks [1]. It has been seen that IP address space was getting exhausted very rapidly, although only a small fraction of the addresses allocated were actually in use. Therefore, classfull addressing scheme is replaced by *"‘classless interdomain routing (CIDR)"’* to allow more efficient use of the Internet address space. However, on the other side, a new problem appeared in prefix-based routing; the address space of a network would be within the address space of another network, although both networks are unrelated with each other. This situation is clearly represented in Figure 2.3. Network A represented by 208.12.21/24 does not reside within network B which is represented by 208.12.16/20, and the IP packet, whose destination address belongs to network A, should not be forwarded to network B. The solution of this problem is finding the *"‘longest prefix match (LPM)"’* within a routing table, since the network represented by a longer prefix corresponds to a small portion of the address space that should be excluded from the network represented by a shorter prefix. In other words, the longest prefix match has the highest priority for determination of the next hop, if there are more than one matching prefixes.

The survey study of Sanchez et.al. covers the state-of-the-art IP lookup algorithms, developed until 2001 [1]. Lim and Lee publish another survey study on binary search IP lookup algo-

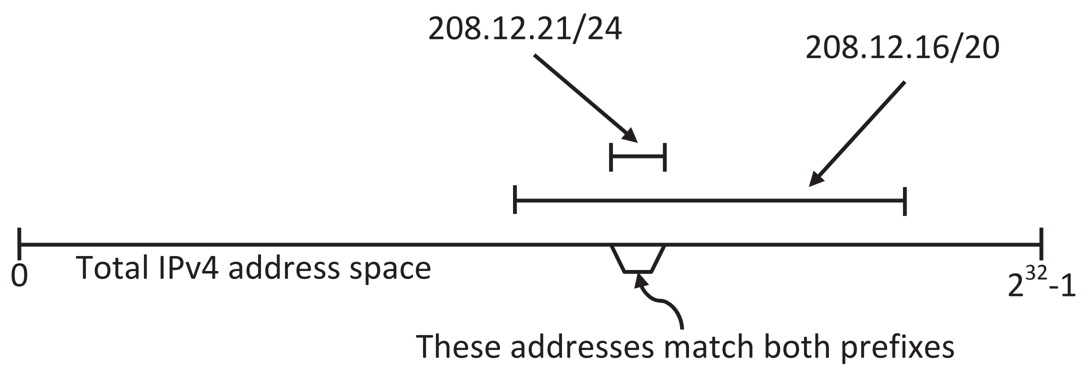


Figure 2.3: Prefix matching problem (adapted from [1])

rithms in 2012, noticing that the only and previous survey study was performed in 2001, and there have been several important binary search IP lookup algorithm proposals from 2001 to 2012 [8]. Lim and Lee state the IP lookup performance metrics as follows:

- Lookup speed (in packets/sec): It should be possible to process the incoming packets at line-rate. The line rate criterion is defined as the rate at which the minimum-sized data packets are entering the router at full capacity of the data link.
- Resource efficiency: The required amount of memory to store the routing table should be as low as possible. The data structure chosen for the routing table storage is the key factor in the resource efficiency. However, the data structure also affects the search algorithm, and hence the lookup speed as well.
- Update capability: It should be possible to update the routing table in compliance with the changes on the network. The changes may include, entering or removal of some nodes, or the change of routing policy. Furthermore, the possible routing table update rate in updates/sec should match the required update rate.
- Scalability to larger routing tables: It should be possible to construct larger routing tables and perform IP lookups with the same data structure and lookup algorithm.

2.2 Overview of IP Lookup Implementations

IP lookup is not a simple operation due to the LPM requirement. LPM requirement dictates the search algorithms for IP lookup to perform a two dimensional search; length and value (prefix). Assuming that the routing table is given as a list of prefixes which are not sorted according to any rule, searching for LPM in this list for a given IP address would require checking all prefixes. Such a search is away from satisfying the lookup speed requirement for line rates of 10s of gigabits per second. Therefore, specialized data structures have been developed to organize the routing table data, such that efficient search algorithms could be implemented. [1] and [8] are comprehensive survey studies on the data structures used for IP lookup. These studies show that there are some primary data structures proposed for LPM IP lookup, and the others are modified versions of them. In the following section we give basic information about just these primary data structures.

2.2.1 Data Structures for IP Lookup

2.2.1.1 Binary Trie

The classical data structure for LPM IP lookup is the *"‘trie"’* structure. A trie is a tree-based data structure, where the nodes’ position in the trie corresponds to a binary string associated with that node. The root node of a trie is an empty node corresponding to *null* string, and the descendants are determined on a digital basis. The left child of an ancestor node corresponds to the string which is ’0’ appended to end of the string that corresponds to the ancestor node. Vice versa a ’1’ is appended for the right child node. An example trie data structure for a given prefix set is shown in Figure 2.4. The nodes that correspond to a prefix in the prefix set, are marked with the letter associated with that prefix.

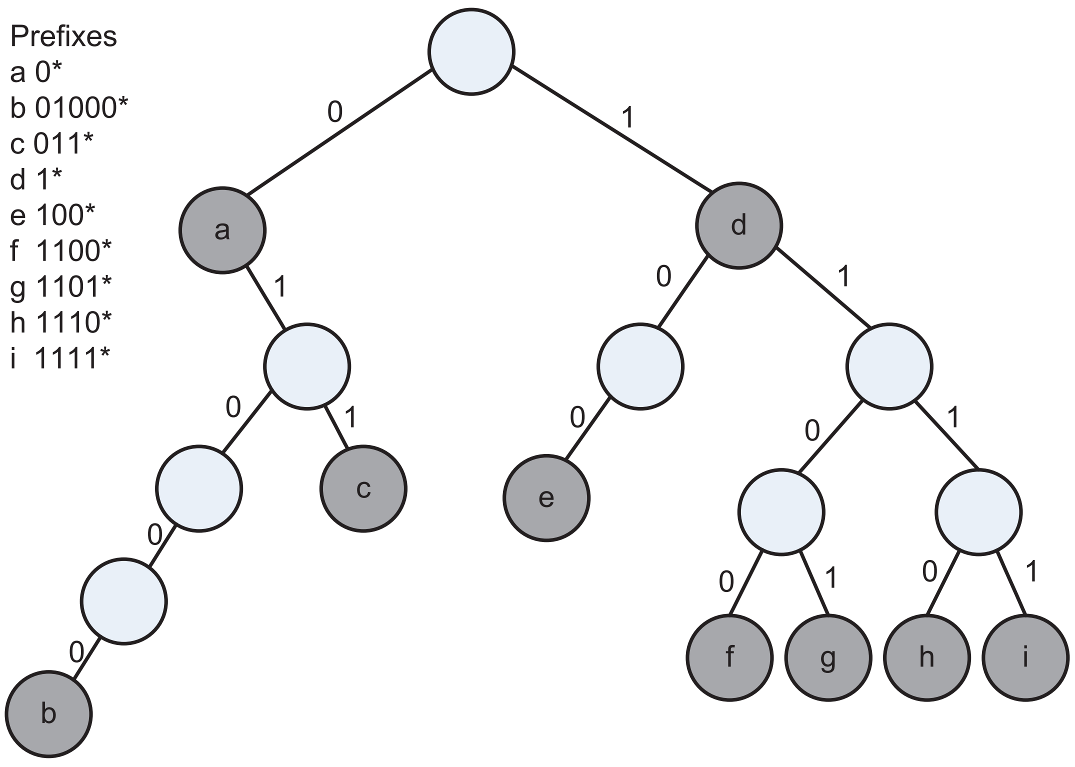


Figure 2.4: Example trie data structure (adapted from [1])

In tries, the longest prefix that matches a given destination address is found in a straightforward way. The search is guided by the destination IP address, starting from the most significant bit at the root, and sequentially moving to next bits at each level traverse. Through the traversal, if a node is encountered that is associated with a prefix is saved as matching prefix, and it is replaced if a new node associated with another prefix is reached. The search continues until a leaf node is reached, and the last matching prefix node is the output of the search, giving the LPM, since the nodes at lower levels corresponds to longer strings. The binary tries have some disadvantages; due to the possible long sequences of one-child nodes, the search time may be longer than necessary, and also one-child nodes that are not associated with any prefix consume additional memory space. Therefore a technique, called as path compression, is applied to improve space and timing efficiency on trie structures. Path compression is collapsing nodes that are not associated with any prefix, to the cost of adding some extra information in the nodes in order the correct operation can be performed. The notable path compression techniques, and resulting trie-based data structures are explained detailed in the survey study of Sanchez et.al. [1].

2.2.1.2 Multibit Trie

For IPv4, in binary tries a LPM search may take 32 memory accesses in the worst case. An approach to reduce the number of memory accesses is inspecting several bits simultaneously instead of one bit a time. For example, inspecting 4 bits at a time would decrease the worst case memory access count to 8 for IPv4. The number of bits to be inspected in one step is called a *"‘stride"’*, and the trie structure that allows inspection of strides of several bits simultaneously is called *"‘multibit trie"’* [1]. In multibit tries, the length of the prefixes should be multiples of strides. For example, in a 4-bit trie, the prefix lengths are constrained to 4,8,12... and so on. The prefixes, that do not obey to this rule are expanded to next multiple of stride. For instance a 3-bit 001* prefix is expanded to 0010* and 0011*. In prefix expansion process, care should be taken for already existing prefixes. If 0011* already exists in the routing table, the expansion of 001* should exclude 0011*, since it is a longer prefix that already exists and has a higher precedence in the IP lookup process. Choosing the stride in a multibit trie is a tradeoff between search speed and memory consumption, and the optimization techniques used to choose the stride are investigated in [1]. In the extreme case a 32-bit trie would give result with just one memory access to the cost of storing 2^{32} entries in the memory.

2.2.1.3 Range Search Tree

Another data structure used for LPM IP lookup is *"‘range search tree"’*. In this structure the idea is based on the fact that a prefix represent a range of IP addresses. For example in a 4-bit domain, 01* represents $[1,7]$. Storing just the boundary points of the ranges, and finding the range in which the given IP address can be positioned is the solution for IP lookup. In this approach, a range may cover another range, and the smaller range has a higher precedence as it corresponds to a longer prefix. Therefore, the boundary conditions for ranges are previously sorted, and a binary decision tree is constructed based on these boundary points.

2.2.1.4 Binary Search Tree (BST)

In BST algorithm, the prefixes are sorted according to a restriction and then a binary decision tree is constructed based on the sorting order of the prefixes. To describe the restriction, some definitions are necessary [8]:

"A prefix is defined as an enclosure, if there exists at least one other prefix that has the prefix as its sub-string. A prefix that has other prefixes as its sub-string is defined as an enclosed prefix. A disjoint prefix is neither an enclosure nor an enclosed prefix. The restriction is to locate the enclosure at a higher level than the enclosed prefix. The list in composing a binary search tree primarily consists of disjoint prefixes and the first level enclosure prefix, and they are sorted according to the provided definition. The medium prefix in the sorted order is selected as the root of current level; if the selected prefix is an enclosure prefix, its enclosed prefixes can be inserted to the list in the sorted order. A binary search tree is constructed by repeating this process [8]."

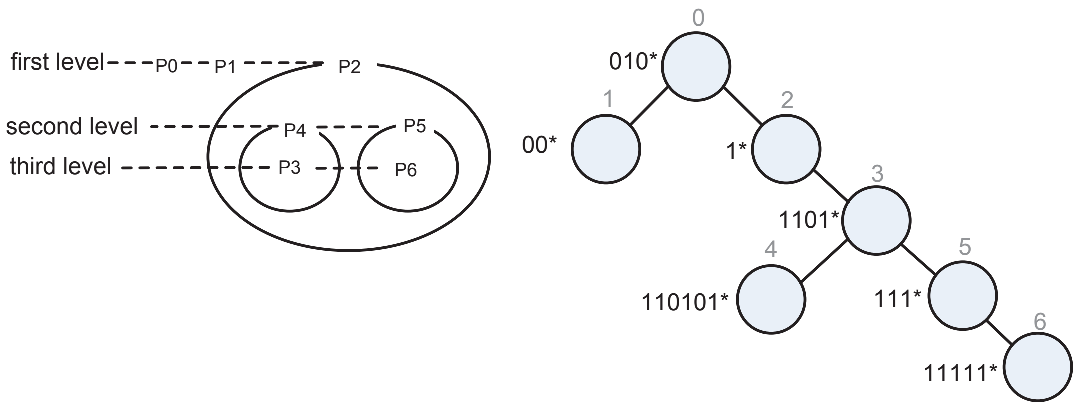


Figure 2.5: BST Construction

BST is highly unbalanced and several algorithms proposed to build balanced BSTs in order to decrease the tree depth are given in [8].

2.2.1.5 Binary Trees on Prefix Length

These type of algorithms perform binary search on prefix lengths and hashing on prefix values. While Waldvogels Binary Search on Length (W-BSL) separates the binary trie according

Table2.2: Routing table for binary search tree

	Prefix	Length	Left Pointer	Right Pointer	Output Port
0	010*	3	1	2	P1
1	00*	2	-	-	P0
2	1*	1	-	3	P2
3	1101*	4	4	5	P4
4	110101*	6	-	-	P3
5	111*	3	-	6	P5
6	11111*	5	-	-	P6

to the level of the trie [13], logW-Elevator Algorithm (logW-E) [14][15] performs a similar process on multibit tries. Binary Search on Length in a Leaf-Pushed Trie (L-BSL) [16] uses leaf-pushing [17] to make every prefix disjoint, and Binary Search on Lengths in Multiple Tries (BSL-MT) [8] proposes to separate the prefix set into multiple disjoint groups and perform binary search on prefix lengths in each group.

2.3 IP Lookup Implementations in Software: Network Processors

Network processors (NPs) are specialized integrated circuits, which are developed to meet the high performance and flexibility requirements of the networking functions while maintaining the line rates. High performance and flexibility requirements come from the fact that the bandwidth of the network grows exponentially, and the diversity and complexity of the networking applications increases rapidly as time passes. The baseline architecture of the NP is different than general purpose processor (GPP), which is still flexible, but yet unable to satisfy processing performance at high speed line rates [18]. NP differs from traditional GPP in three ways [19]:

- The instruction set of many NPs are based on existing RISC processor instruction sets.
- The NP's instruction set contain special instructions, for example, bit manipulation CRC calculation, and search and lookup operations
- Special hardware function blocks are present to accelerate specific packet processing tasks

NP can also be defined as application specific instruction processor (ASIP) for the networking domain, a software programmable device with architectural support for packet processing [20]. Generally, a NP is composed of a programmable processor core, with a number of high speed memory interfaces, and co-processors, that are optimized for packet processing. Packet processing tasks are the main determinants of the requirements and configuration of NPs. Packet processing tasks, also NP functionalities, can be categorized as follows [21][22][23]:

1. Packet framing: Framing is applied to both incoming and outgoing frames to assure that the correct and full packets or datagrams are extracted from or shaped into these frames. This means that incoming frames should undergo correctness tests, correcting attempts if required, and integrity checks. Outgoing frames should be fragmented or segmented as required and "‘framed"’ correctly, i.e., adequate headers should be attached or altered, proper terminators (trailers) should be appended or modified, and error detection and correction information should be added, when applicable, to enable later correctness tests and correcting attempts.

2. Parsing and classification: Parsing is basically identifying the relevant fields in the incoming packets, according to their place and type, and picking the field’s values for further parsing process, or using these values for classification. Each incoming packet must go through some sort of parsing to examine and understand what it is, what type it is, and then it must be classified or handled according to its type and its required processing. Classification means categorizing packets into "‘flows"’, in which they are processed in a similar way by the network entities. These flows are defined by the rules that the packets obey, and the collection of these rules are called classifiers [11].

3. Search, lookup, and forwarding: Search is one of the most important, complex, and common operations done in packet processing. Search is not a phase itself in packet processing, rather is a part of classification phase or forwarding, even any operation that are used for packet processing. Search can simply be defined as obtaining some value from a collection of data, according to a given "‘key"’. The simplest search is performed in plain memories, where the "‘key"’ is the address of the memory and the value returned is the datum stored in that address of the memory. The complexity of the search process depends on the data structures of the stored values. Some data structure examples are lists, tables, trees, and tries. Forwarding is defined as determining the output path for incoming packets, and it is usually done by IP lookups. IP lookups are done not just for forwarding purposes, also for classifications or other applications, for example, billing, access lists, and so on.

4. Modification: Packets are mostly modified as a result of the processing tasks. Modification includes, changing the content of the packet (both header, and payload), deletion some of the contents (e.g. de-encapsulation), adding of additional information (e.g. encapsulation, authentication), simply removing the entire packet from the system (e.g. wrong addressing, exceeded traffic), and duplication of the packet (e.g. multicasting, port-copy operations).

5. Queuing and traffic management: Traffic management includes queuing, buffer management, and scheduling/shaping. The previous stages at packet processing, defines the output port, priority, some handling parameters, and the flow of the packet. According to these parameters, traffic management handles the queues and flows based on the various classes of services (CoSs). Traffic management process also meters the packets, and shapes the transmission pattern to a desired rate and burstiness.

Processing functions are separate tasks, each following the other. The process starts with the packet entering the network processor and immediately goes through framing. The second phase is to parse and classify the packet. Usually for this classification function, searching is required. Searching might also be required for other functions that the application dictates. The last function that the network processor carries is the required modification of the packet. Finally, transmitting the packet usually involves an extra function of queuing, prioritization, and traffic management of the packet to make sure that the receiver can receive the transmitted packet at traffic patterns that it expects. Queuing and traffic management sometimes happens inside the network processors and sometimes happens outside the network processors. Optionally, compression and encryption tasks are utilities that packets sometimes undergo and usually they are done outside of the network processor, although there are some network processors that contain an embedded security functional unit.

2.3.1 NP Architecture and Design Approaches

A general NP architecture is shown in Figure 2.6. The constituent elements of NPs are described shortly in the following [19]:

- The Processing Engine: Many NPs are multi-processors, i.e., they are not built on a big RISC processor. The basic programmable unit in the NP is a processing engine (PE). Different NPs use different architectures for their PEs. PEs can be grouped into functional blocks or independent.
- Co-processor: Co-processor is a programmable hardware accelerator, which is a finite state machine that operates independently.
- Dedicated hardware: These are special hardware dedicated to perform common networking tasks, such as CRC calculation, queue management, forwarding engine and lookup engine.
- Memory architecture: There are mainly three types of memories in NPs, which are instruction memory, packet memory, and route table memory. These memories are generally shared by processing engines and processors in NP, and memory sharing is managed by memory manager in the system.
- Network Interface: This is the point where packets enter and exit the NP. This architecture is generally a switch fabric.

The common NP design approaches are as follows [19][20]:

- ASIC (Application Specific Integrated Circuit): Hardwired design, which is not programmable and flexible.

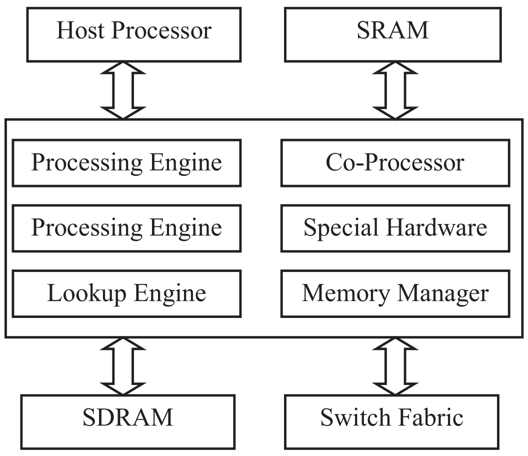


Figure 2.6: A general NP architecture model

- ASIP (Application Specific Instruction Processor): An instruction set processor specialized for a particular application domain.
- Co-processor: A hardwired, possibly configurable with a limited programming interface.
- FPGA (Field Programmable Gate Array): A devices that can be programmed at the gate level.
- GPP (General Purpose Processors): A programmable processor for general purpose computing.

ASICs are the most hardwired (least flexible), but provide the highest performance. GPPs are the most general (flexible) at the cost of the lowest performance. FPGAs provide an interesting value proposition in the absence of ASIPs or co-processors as they show higher performance than GPPs with more flexibility than ASICs.

Parallel processing is an important issue in NP design. Parallel processing is exploited highly in NPs to achieve high performance of computing. Basically there are three types of parallelism [19][24][25]; instruction level parallelism (ILP), thread level parallelism (TLP), and packet level parallelism (PLP). In ILP, the compiler or hardware instruction scheduler determines simultaneous execution of program instructions. In TLP, different threads are executed to avoid idle time in memory references and processing engines, i.e, if a thread waits for the memory it is stalled and then another thread is started. Since memory latency is a great potential concern in network applications, multithreading is a good solution to hide memory latency while maintaining a high processing rate. In PLP, if each incoming packet is independent of the other packets being processed, incoming packets are concurrently processed in the several independent processing units of the NP. During the processing of each packet, if the tasks to process the packet are independent, they can be executed in parallel. This kind of parallelism is called intra-packet parallelism [25].

In the architectural view there are two approaches to achieve parallelism [26]; pipelining and pooling. In pipelining, processing elements are arranged in a pipeline, where each processing element performs a single pipeline stage of packet processing. In the second approach, the processing elements are arranged in a pool, where each processing element performs a similar functionality.

Parallelism in network applications can be efficiently exploited by applying SMT (Simultaneous MultiThreading) and CMP (Chip MultiProcessor) models [25]. SMT can issue multiple instructions from multiple threads in a single cycle and schedule them dynamically so as to concurrently exploit TLP and ILP. CMP are actually multiprocessors which have a full set of architectural resources on the same die. A CMP consists of a single thread processor cores which are relatively simpler than general purpose processors, where multiple threads are executed in parallel across multiple single-thread processor cores [27].

2.3.2 NP in the Literature

The first NP in the literature is encountered in the study of Menzilcioglu and Schlick [28]. They implemented a high-speed NP in 1991 for the high-speed fiber optic network, which was developed at Carnegie Mellon University and called Nectar. The NP was called CAB as the abbreviation for "‘Communication Accelerator Board"’. Nectar CAB, which has three major parts, processing unit, network interface and host interface, has a flexible architecture, where almost all interactions between the network hardware and the host are fully programmable.

As mentioned before, GPPs can also be used for network processing applications. Ghasemi et.al. augmented a modeled GPP for network processing, by adding a special instruction set, which are designed for different processing required in IP networks [29]. The added instructions are parsing and prefix instructions, which accelerate the packet processing, in particular lookup operation.

Parallel processing and multithreading are other important research areas in NPs. Yi et.al. evaluated the applicability and efficiency of SMT as a network processor [30]. They demonstrated that, when executed as independent threads, applications chosen from different network layers show an improved IPC (inter-process communication) and cache behavior when compared with the situation where the program executed comes from a single network application. Wu et.al. studied parallel architecture and multithread mechanism in NPs [31]. They discussed the reasons of thread stalls caused by resource competition, such as co-processor, general purpose register, instruction memory, and data bus.

Reconfigurable architectures are especially new trends in NP design. DynaCORE is a dynamically adaptable coprocessor based on dynamically and partially reconfigurable logic, which is proposed to overcome the restrictions on flexibility in NPs [32]. DynaCORE is a multipurpose hardware accelerator implemented by an FPGA. Its dynamic behavior comes from the abilities of the new generation FPGAs, which are dynamic partial reconfiguration (run-time configurability), and self-reconfiguration. The reconfiguration plane of DynaCORE is formed by the reconfiguration manager and the observation points. The observation points collect data on utilization and other operational characteristics of hardware assists, and collected data is optionally reprocessed and then passed on to the reconfiguration manager. The reconfiguration manager is implemented on software on an embedded processor core of the FPGA, because it utilizes complex algorithms in order to make a schedule of which functionality is to be assigned to the individual hardware assists, which are container for arbitrary processing logics, a processor based subsystem or an off-the-shelf IP core.

FlexPath is another NP implementation based on FPGA, which provides a platform with flexible, reconfigurable processing paths for packet processing [33]. The path decision is made in hardware based on a packet's network application. The path decision is based on a rule that may be reconfigured during run-time.

DynaNP is a coarse-grain dataflow NP architecture, with another dynamic configurable processing path mechanism [34]. DynaNP consists of multiple processing engines (PEs), which are connected by the high-speed communication network. The processing path is determined by a token passing algorithm. The processing path is determined in two stages. The first stage implemented as part of task's software code is to find out which task should be activated for the token according to the characteristic of workload application and current result of processing. The second stage of work implemented in TPM (Token Processing Module) is to look up a hardware mapping table to determine which target PE should process the task.

Kachris et.al. implemented a coarse-grain dynamically reconfigurable FPGA platform for multi-service edge and access network devices [35]. The platform consists of two MicroBlaze processors and a number of hardware co-processors used for the processing of packet payloads. The functionality of the co-processors is dynamically reconfigured to meet the requirements of the network workload.

There are also some studies, which focus on the packet ordering in PLP based environments, since when packets are processed in multiple processors, the order of outgoing packets may be different than the order of the incoming packets due to different timings the packets undergo. Two examples of these types of studies are [36] and [37]. In [36] an alternative packet sorting scheme is proposed, which guarantees complete packet ordering while achieving a throughput of 2.5 Gbps. In [37], another packet scheduling algorithm, named as DBCS (Dynamic Batch Co-Scheduling), to ensure a good load balancing among the processors, with the assumption that the workload is perfectly divisible.

2.4 Classification and Overview of Hardware IP Lookup Implementations

For the minimum TCP packet size of 40 bytes, one lookup must be completed in 8ns for a line speed of 40Gbps. This is equivalent to a throughput of 125 MLPS (Million Lookups Per Second). In [8], it is stated that about 50% of the Internet traffic is TCP acknowledgment packets and TCP acknowledgment packets are minimum-sized. Therefore, minimum-sized packets do not dictate the operating speed just for the worst case scenario, but also for the average case as well. High link bandwidth, and excessive amount of data transfer on the Internet, implies the IP lookup algorithms to be implemented on the hardware rather than the software.

Hardware based IP lookup solutions are mainly categorized into two classes with respect to the memory types that build the base for the implementation. These memory types are RAM (Random Access Memory) and CAM (Content Addressable Memory). In both of types memories the data are stored in an addressed sequence, but while the RAM gets address as the input and returns the content at that address, CAM gets the content as the input and returns the address of that content. In the following sections we will focus on the RAM-based and CAM-based IP lookup solutions.

2.4.1 Tree-based Data Structure Implementations on Hardware

The implementation techniques of notable binary search tree based algorithms on hardware platform are investigated and compared in [38]. The comparison is performed with respect to required memory, the total number of prefixes used, memory access count per lookup, and update procedure. McLaughlin and Sezer conclude that hardware-based solutions offer a way satisfying to minimize the time needed to complete a lookup by minimizing the number of memory accesses in the search. Hardware has the key advantages over software that it can deliver faster processing of algorithms and optimized control over memory accesses and memory management [38].

SRAMs are the usual hardware platform for implementation of tree-based data structures. A memory location is dedicated to a node, where the memory content is the data associated with the corresponding node, and the addresses of the nodes that are pointed by that node. The search process includes reading the memory content, comparing the associated data with the key value used in search, and determining the next memory location to be read by selecting one of the addresses according to the linking criteria. If a single memory is used for the whole routing table, a single search may take several memory accesses, depending on the depth of the final leaf node reached at the end of the search. Therefore single memory implementations suffer from long latencies, and low lookup throughput. To overcome the latency and throughput bottlenecks, two common applied approaches are pipelining and/or paralleling to the cost of using multiple memories. In the paralleling approach, the whole tree is tiled into separate pieces, with respect to the prefix length, and each part is implemented on separate memory blocks. The search is performed simultaneously on the memory blocks. The tree depths stored in the memory blocks may not be of equal length, so the final result is not obtained until the search for the deepest tree is finished. If there are more than one matches, the match that corresponds to the longest prefix is selected as the LPM. In the pipelining, a memory block is assigned to each level of the tree. Each memory block (each level) is accessed only once in a single complete search, it is possible to build a pipeline between the memory blocks to allow successive search operations. In this approach the LPM is determined at the end of the pipeline, as in the case of conventional tree search. As we mentioned previously, paralleled and pipelined tree architectures require multiple memories. However, the data partitions (separate tree blocks in paralleling, and levels of a tree in pipelining) are not of comparable size (the metric is number of nodes). If commercial memories are used for implementation, selecting a fixed size memory for all data partitions would not be a cost effective solution, since the selected memory should be capable of storing largest data partition, and there will be lots of empty spaces on memories that are assigned to smaller sized partitions. Especially, in pipelining approach, the required amount of memory is doubled at each level. Therefore, assigning a fixed size of memory to each level is non-sense. Also, getting different sized memories for each level is not a feasible solution, since the types of memories is getting increased and there are no memories with just several entries as low as 2 or 4. FPGAs (Field Programmable Gate Arrays) allow the users to design any size of memory blocks and more than one memory blocks can be implemented on a single chip. Therefore, FPGAs give the possibility of implementing paralleled or pipelined architectures on a single chip. There are several studies implementing paralleled or pipelined tree-based IP lookup architectures on FPGA.

Bemmann discusses algorithms for IP lookup on FPGA platforms, focusing on comparability and tradeoffs [39]. A hashing-based algorithm and a tree-based algorithm as examples for the two most common classes of IP lookup approaches are investigated with regard to their implementation in programmable logic, their resource use and their performance, measured in lookup time.

Le et. al. implemented first trie-based design that uses the on-chip FPGA resources only to support a fairly large routing table [40].This architecture is also claimed to be the first architecture that employs IP caching on FPGA resources without using TCAM to exploit Internet traffic locality. The challenges they tried to achieve were; high size routing table, high throughput, in-order packet throughput and low power consumption. Since the search time in a trie structure depends on the depth of the tree, a pipelined approach is preferred. And also in order to make all packets be processed in the same amount of cycles, a balanced memory structure is targeted, the trie is built as "‘leaf-pushed uni-bit trie"’. A trie is called a uni-bit trie if only one bit is used for making branching decision at a time. Leaf-pushed uni-bit trie is obtained by applying the optimization algorithms called as "‘leaf pushing"’ [41]. The design maintains packet input order, and supports in-place updates (one node replaced with a new value, trie structure is not affected) as well. However, major table updates requires reloading of the FPGA.

Le et.al. implemented another SRAM-based dual linear pipeline architecture, named DuPI, for IP lookup on FPGAs [42].The architecture was based on balanced binary search tree (BST) this time. Since the search time in BST depends on the depth of the tree, a pipelined approach is preferred as well. Each table is one level of a binary tree structure, so the memory size doubles at each stage (1, 2, 4, ...). To ensure that every IP lookup results in the same number of operations or cycles, the IP address continues with all the comparisons even though a match may have already been found. The design also maintains packet input order, and supports in-place non-blocking updates. However, when the route table needs to be completely changed, then FPGA must be fully reconfigured with the new tree structure.

Fadishei et. al. dealt also with IP lookup as some previous studies [43]. They proposed an architecture which finds solutions to implementing large routing tables into FPGAs and updating them, taking advantages of both traditional hashing scheme and reconfigurable hardware. The problem of large hardwired data size is addressed by only implementing the collided entries on reconfigurable hardware. Other prefixes can be searched with a single memory access to the main table. They also address the problem of high route update cost by proposing a novel reconfigurable search tree architecture for collision resolution. The binary search tree constructed in this study is a fixed skeleton complete tree, in which hardwired data elements are stored in the reconfigurable hardware resources of each node. Unlike the conventional binary search tree, a new prefix can be inserted in any node of the tree, making partial reconfiguration possible for update operations. The reconfigurable search tree has a limited number of nodes. When all nodes of the tree are in use, new insertion requests are rejected. In such a situation, which is due to physical limitations, more nodes must be allocated to the tree using a full hardware reconfiguration.

A state-of-the-art SRAM-based architecture is implemented by Erdem and Bazlamacci in 2012 [44]. They propose a two-dimesional SRAM-based multiple pipeline structure for high-speed IP lookup. Different than the conventional parallel pipelined architectures which employ n parallel pipelines each with m processing engines, the proposed two-dimensional pipeline consists of $\sqrt{nm} \times \sqrt{nm}$ processing engines building a square-grid architecture. The pipeline is constructed as a circular pipeline, so that the necessity of building equal length pipelines is eliminated, and therefore it is more resource efficient. Erdem and Bazlamacci further applies an algorithm, what they called *compact clustering*, on leaf-pushed binary trie prefix sets, in order to further reduce the number of nodes and hence the depth of the tries to be employed in the overall data structure. Finally, employing the dual-port SRAMs in the processing engines they also increase the throughput by two-fold compared to single-port SRAM usage as well. They simulated their sample architecture of 16×16 processing engines for twenty different real-life packet traces, each having 1.2M packets, and observed that they achieve about 29 lookups per cycle on the average. It is stated that, by optimizing the number of selector units and the cache size in the design, this average number can be as high as 40 lookups per cycle. Simulating their architecture for 350 nm process technology, they calculate the critical path delay for their design as 3.38 ns. As a result, they can achieve a IP lookup throughput of 11.8 billion packets per second, which corresponds to a 3.8 Tbps line rate operation for the smallest 40-Byte packet sizes. With the assumption of all delays except the memory access tme to be zero, the upper bound for the line rate is stated as 7.9 Tbps. In this work the updates are carried with the help of software by employing an auxilary unibit trie which is called as the soft shadow of the routing trie. The changes to be performed are determined on this software platform, and then the necessary chenges are perfomed on the processing engines. However, there is no numerical result about the required number of cycles to perform an update request, and also how these updates affect the ongoing IP lookup processes.

2.4.2 TCAM: Custom Hardware for IP Lookup

Considering the sequential processing nature of the SRAMs for search-and-compare type problems directed the researchers to find a memory architecture, Content Addressable Memories (CAM), which allows parallel search of entire content [45]. The operating principle of the CAM is comparing all the stored words with the given search word, and producing the encoded address location of the word that matches the comparison. The CAM architecture concept is shown in Figure 2.7. In the basic CAM architecture, the comparison is performed in binary, i.e., all bits of the search word and the stored word should be equal one-to-one in order to produce a match result. One-to-one matching condition is called as exact match. For the memory arrangement, where the stored words are expected to be disjoint, i.e., there may be only a single word that matches the given search word, a basic address encoder logic is accompanied to produce the address of the matching entry.

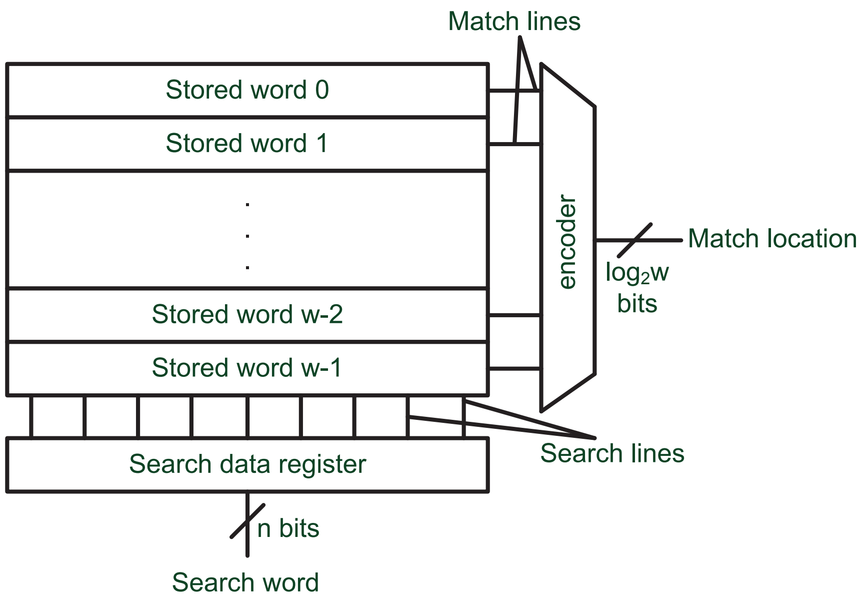


Figure 2.7: TCAM Conceptual Architecture (adapted from [2])

Ternary CAM (TCAM) is an extension of CAM, where wildcard search is also made possible.

In TCAMs, it is possible to store don't care (X) bits as well as standard binary bits (0 and 1). This is achieved by accompanying each word with a mask register, that shows the binary and don't care bit positions. While binary bits are masked by logic '1', don't care bits are masked by '0'. An illustration of the TCAM architecture is shown in Figure 8.

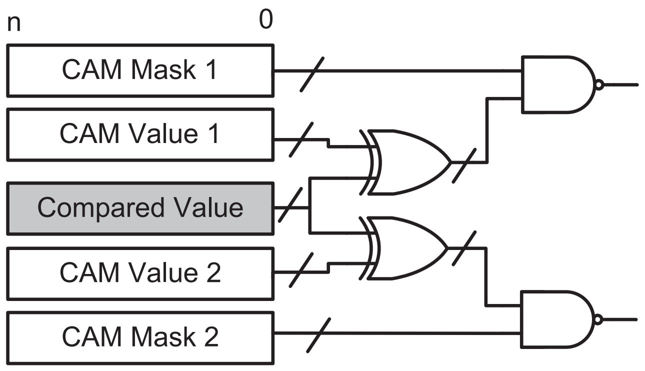


Figure 2.8: Standard TCAM architecture

In TCAMs, the address encoder logic is replaced by a priority encoder (PE) logic since there may be more than one matching entries. The matching entries are sorted with respect to their precedence given according to a rule, and the address of the matching entry with the highest precedence (priority) is encoded at the output of the PE. The precedence rule depends on the application and some of the notable applications, that utilize TCAMs, are given in [2]. IP lookup is the most common application targeting TCAMs. TCAMs are very suitable hardware platforms for IP lookup because of two reasons:

1. The prefixes are composed of binary string followed by a don't care string
2. The entire prefix list can be searched within a single cycle, which is desired for high speed lookup operation.

The precedence rule for IP lookup is actually the LPM rule. Therefore the PE selects the prefix according to the length information. At first sight, it is thought to accompany length information to the prefixes for PE design. However, such an approach would make both the TCAM and the PE design complex. Instead, the prefixes are sorted in the TCAM with respect to their length in ascending order, and the PE just selects the prefix with the highest address. There is no restriction for the sorting of the prefixes of the same length, since they are disjoint. TCAMs are actually large in size, since they hold two SRAM cells for one entry; one for prefix, and one for the mask. Additionally, there is a comparator circuitry for each entry, which makes the power consumption of the TCAMs high. Therefore, the IP lookup solutions targeting TCAM platform mainly focus on either increasing the storage efficiency or decreasing the power consumption by proposing some modifications on the TCAM architecture.

2.4.2.1 TCAM Implementations on FPGA

TCAM-based implementations for FPGA platform is also a popular research area as FPGAs are widely used in logic applications. Song and Lockwood implement BV-TCAM (Bit-Vector TCAM) architecture on a Xilinx FPGA [46]. They used a tree bitmap implementation of the Bit Vector (BV) algorithm for source and outgoing port lookup, while a TCAM performs the lookup for other header fields, which can be represented as a prefix or exact value. TCAM is implemented using embedded logic cells and block RAMs of the FPGA.

Jedhe et.al. implement a firewall using Extended TCAM (ETCAM) [47] architecture, which is implemented on the FPGA [48]. This is the first reconfigurable hardware implementation of ETCAM, which is used for range matching. The proposed solution is a mixed software and hardware solution, supporting 1 Gbps interface for 128 rules. When there is an update requirement in the rule table, the memory is re-initialized in software.

A comprehensive study on CAM implementation on FPGA is performed by McLaughlin et.al. [49]. They implement CAMs on FPGAs with three different approaches. In the register approach each word is made up of a register and a comparison circuitry. The register-based approach depletes the FPGA resources very quickly. In the RAM based approach, BRAM (Block RAM) modules are used as address pins replaced by data pins, and the data content as CAM address output. BRAM-based TCAM implementation is not suitable for IP lookup, since one location can be reached at a time in the BRAM architecture, and this makes the priority encoding for the longest prefix match hard to implement. The third approach is a LUT (lookup table) based architecture where they utilized the LUTs as function tables giving '1' output for a given input and '0' otherwise.

McEwan and Saul implemented a CAM architecture on the FPGA, utilizing ROM blocks of the FPGA, based on bit-serial comparators [50]. In this architecture the write and compare times are proportional to data widths, which is inconsistent with the single cycle search capability of the CAMs. Another drawback of this solution is that, the content of the TCAM cells cannot be changed on the run-time, since they are composed of ROM blocks, hence, although the proposed architecture is parameterizable, it is not dynamically reconfigurable.

Guccione et.al. propose a reconfigurable CAM (RCAM) architecture on Xilinx FPGA platform [51] . They utilized lookup tables (LUTs) for the basic TCAM cell array. They also state that using LUTs on FPGA, larger and even faster CAMs can be implemented. They also implement a software called as JBits [52], which can be used to change the LUT contents on run-time. JBits was developed for Xilinx XC4000 and Virtex families, and cannot be used with the concurrent FPGAs.

Gokhale et.al. propose CAM implementation on Xilinx Virtex 1000 FPGAs, in their software-hardware based solution for network security problem [53]. They implemented CAMs using shift registers (SRL16Es) and fast carry chains (MUXCY). With this approach they achieved single cycle read latency while 16 cycles are needed for write operations . They implemented

120 bits by 32 deep CAMs, operating at 66 MHz. In their implementation, there are no details about the priority encoder circuitry, and how the CAM entry is selected for writing operations.

A recent study published by Ionescu et. al. describes the CAM implementation on FPGAs using Hopfield neural networks [54]. In this study, each logic cell is constructed as neurons. Each neuron is connected to other neurons by weight circuits. A cell is composed of multi-operand adder circuit and comparison circuit. They implemented maximum 40 neurons consuming 5200 slices. The neural network has a 700 ns settling time, which makes this implementation infeasible for networking applications.

Most recent study about TCAM implementation on FPGA platform is published by Ullah et.al. in 2012 [55]. In this study, Ullah et.al. tiles conventional TCAM architecture into horizontal and vertical partitions, and implement these partitions on SRAM-based units. The main goal of this study is minimizing the total power consumption of the TCAMs with the proposed hybrid partitioned SRAM-based architecture. The architecture is implemented to perform a flow chart actions for search operation. The total latency of a search operation takes 5 clocks, which is a contradictory behavior for conventional TCAMs, since main advantage of conventional TCAMs is the single clock search latency. However, the pipelined architecture makes consecutive search operations possible, which eventually considered as single clock search capability in this study. The update in this TCAM architecture requires external evaluation of the update location in conventional TCAM, and there is no explicit information about how this is performed.

Table 2.3 summarizes and compares the TCAM architectures implemented on FPGA platform including those given in in terms of architecture, priority encoder (PE) implementation, scalability, update capability and maximum size and operating speed of the TCAM implementation for the selected FPGA.

2.4.2.2 Previous Work on TCAMs and IP Lookup with TCAM

A large number of studies in the literature focus on decreasing the power consumption and increasing the storage efficiency of TCAMs.

Selective pre-charging/enabling, pipelined/hierarchical search and match line partitioning techniques [56, 57, 58, 59, 60, 61] aim to reduce average power consumption. Such approaches divide the TCAM match line into segments and then sequentially enable each segment for search. The search stops at the first no match hence the remaining segments are not enabled saving power. This approach is inherently suitable for IP prefix look up and explicitly suggested for this purpose in some of the works. In [62], Mohan et.al. propose two match line sense amplifiers (MLSA) that use positive feedback that do not consume any static power and with fewer transistors compared to previous MLSA architectures. [61] further exploits the vertically continuous ‘don’t care’ bits. Two very recent studies [63, 64] are on power gating to reduce the leakage power dissipation. [64] proposes a parity-bit based architecture

Table2.3: TCAM implementations on FPGA.

Ref., Year	**Approach**	**Priority Encoder (PE), Scalability**	**Update Capability**	**FPGA, Max. Size@Operating Speed**
[51], 2000	LUT-based	PE is not implemented	An auxiliary software (JBits)[52] is required for update. JBits is valid only for Xilinx XC4000 and Virtex families	Virtex 1000, 3Kx32, 1Kx64 No information about speed
[50], 2001	ROM-based bit-serial comparators	PE is not implemented	Fixed at compile time, requires full reconfiguration	Xilinx XC40150XV, 1Kx32@20 MHz
[53], 2002	Shift Register-based	PE is not implemented, Fixed size implementation	New TCAM contents can be shifted serially into shift registers, but TCAM cell to be selected for update is evaluated externally	Xilinx XCV1000E, 32x160@66 MHz
[46], 2005	Logic Cells and BRAM-based	Bit-Vector (BV) Implementation for PE, BV encoding does not scale	Requires re-synthesis	Xilinx XCV2000E, 33x72@100 MHz
[49], 2006	Register-based, BRAM-based, LUT-based	No information for PE implementation	Register and RAM-based solutions can be changed on run-time, LUT -based solution requires re-synthesis	Altera APEX, 256x128@42.5 MHz, 256x64@49.1 MHz, 512x16@58.1 MHz
[48], 2008	LUT-based, TCAM outputs are not encoded	A separate PE is implemented	Re-initialized in software, requires re-synthesis	Xilinx XC2VP30, 128x32@50 MHz
[54], 2011	Hopfield Neuron implementation	PE is not implemented, Scales in terms of neuron numbers	Neuron states can change during the operation	Xilinx Spartan 3 XC3S400, 700 ns settling time for 40 neurons
[55], 2012	Pipelined SRAM-based partitions, PE gives lowest matching address in 5 cycles	Scales in terms of TCAM size	Requires external evaluation at conventional TCAM	Xilinx Virtex-5, 512x36@100 MHz

that with lower peak current, and average power consumption, and faster search, with a cost of 11% area overhead.

Storage efficiency is most commonly achieved by algorithmic transforms of the prefixes. [65] introduces EaseCAM that is compatible with the prefix aggregation and expansion. [66] encodes the explicit mask bits of the TCAM cells with 9 SRAM cells for each 8-bit block and a more complicated match line structure. Despite reducing the mask bit storage, masking each individual entry for a number of bits is still required to find the longest match. [67, 68] propose a multi-chip TCAM with an adaptive load balancing mechanism for TCAM table construction, resulting in a higher throughput and lower power consumption. A very recent study [69] proposes a TCAM coprocessor, which allows simultaneous operation of LPM, policy filtering (PF), and content filtering (CF) tasks on the same TCAM to increase storage efficiency. In [70], a methodology is proposed for compacting the routing table and then storing the prefixes in TCAM, leading to reduction in TCAM power and space consumption.

Considering the specific requirements of the IP address lookup we identify achieving non-blocking table updates entirely on hardware and modular construction of the TCAM architecture with inherent priority encoding and address decoding circuitry as important issues.

In all previous works, the PE implementation directly depends on the TCAM cell array size, and PE is a separate component. Changes in the TCAM size require redesigning the PE, hence, decrease the *design scalability*. Furthermore the PE operation is carried out sequentially following the match production of TCAM consuming additional time. [71] is a survey study that investigates TCAM design techniques and proposes a conceptual PE design as a separate unit.

Conventional TCAMs share the same *bitlines* for both read and write operations [2]. Therefore, concurrent read and write cycles are not possible on conventional TCAMs, which also prohibits fulfilling the non-blocking update requirement for IP-lookup problem. To the best of our knowledge, [70] is the only work that considers non-blocking TCAM updates. However, they achieve this by duplicating the entire TCAM table and conducting the updates and searches on different copies. [72] proposes storing the prefixes ordered in their length where the LPM returns the match in the lowest address. The prefix blocks have flexible sizes and can be moved if required to maintain the ordered prefix length. They propose two algorithms to reduce the memory movements during updates. The algorithms use a trie data structure to keep track of the prefixes stored in TCAM, in order to support the update operations. It is important to note that these algorithms are realized by an external software.

2.5 IP Lookup Implementations on FPGA

In this thesis we focus on a hardware-based solution, and selected FPGA (Field Programmable Gate Array) as the targeting platform, since FPGAs provide plenty of logic and memory resources with high speed data interfaces on a single chip, which makes them very suitable for networking applications. FPGAs are programmable which lead to significantly shorter development times compared to ASIC design and concurrent FPGAs provide close performance to ASICs in terms of operating speed. Furthermore, it is possible to convert the generated FPGA design files for ASIC implementation [73][74]. The high speed transceivers enable employing the design in a real network environment. It is good evidence that FPGAs are valuable platforms for networking applications, that Cisco, one of the major network processor shareholders, also uses FPGAs in their network processors [75].

In Table 2.4, the previous studies are compared with respect to the major performance metrics. In [76] Taylor et.al. implemented a lookup engine using logic resources of the FPGA that implements the Tree Bitmap search algorithm on an external SRAM interfaced to the FPGA. Fadishei et.al. implemented a parallel hashing scheme utilizing the BRAM (Block RAM) resources of the FPGA [43]. They utilized partial reconfiguration capability of the FPGAs for updating the routing table content. They consider the partial reconfiguration time (in the order of microseconds) for the update rate, but they do not consider the time required for the generation of the configuration file for the update, which can take minutes, and sometimes hours. The remaining studies given in 2.4 ([40][77][42][78][79]) propose tree-based architectures on FPGA. In those studies real-life routing tables are inserted into some type of tree architecture (trie, BST, 2-3 tree). Each level of the tree is implemented on a separate RAM block. RAMs are constructed using the BRAM resources of the FPGA. The BRAM capacity of the FPGA determines the supported number of prefixes also. The search algorithm depends on the pipelining through the levels of the tree. On each level independent search is possible. In those studies the BRAMs are implemented as Dual-Port RAMs (DPRAMs). DPRAMs have two ports sharing the same storage area, and it is possible to make simultaneous read operations at both ports. The throughput of these structure is calculated as the operating frequency given by synthesis result (pipelining) multiplied by 2 (dual-port reading). This throughput is only achievable if the line is fully utilized, otherwise in the worst case the throughput is divided by the depth of the largest tree in the architecture. In [40][77][42] only in-place updates are possible, i.e., the update is possible if the node to be updated does not necessitate the tree reconstruction. In [78] and [79], the tree structure can be changed incrementally, and hence there is no restriction for the update. The major drawback of tree-based architectures is that the latency depends on the depth of the tree. For a 250K prefix table, the tree depth is 18, which means that the latency of the search is the operating frequency divided by 18.

2.5.1 Summary

In this section, we want to give a summary about the SRAM-based and TCAM-based IP lookup solutions. SRAM-based solutions utilize tree or trie based data structures, and therefore suffer from latency due to the successive memory accesses for the tree search. These kinds of solutions utilize pipelining and/or paralleling to increase lookup throughput. Since pipelining and paralleling require multiple memories, FPGAs are selected as the hardware platform for the implementation. A major drawback with tree-based solutions is that tree structures completely depend on the prefix set, and any change on the routing table may result in a complete reconstruction of the tree. Moreover, if additional operations are performed for tree balancing or range-based structures, more complex calculations are required for an update. The tree structure and the nodes' positions in the tree are kept on auxiliary software, and all the calculations for update are performed on the software to perform the necessary reconstruction of the tree structure. The studies based o tree/trie data structures present the

Table2.4: FPGA-Based IP Lookup Comparison

Ref., Year	**FPGA, Architecture**	**Prefix Count**	**Throughput**	**Latency**	**Scalability**	**Update Performance**
[76], 2002	Virtex 1000E + External SRAM, Tree Bitmap Algorithm	16,564	≈10 MLPS (8 engine parallel operation)	440-660 ns	Yes	10,000 updates per second (Software support)
[43], 2005	Virtex-II Pro 100, Hash-based parallel lookup (one for each prefix length (8-32), total 25 engines)	89,979	263 MLPS	≈50 ns (13 clock cycle with 263 MHz clock frequency)	No information	Requires partial / full reconfiguration
[40], 2008	Virtex-4 FX140, Trie Architecture	Mae-West rrc08 (≈80K prefixes)	325 MLPS	No information (Depends on the trie depth, 8 cycles with 162,5 MHz ≅50 ns)	Yes	In place updates possible (when trie structure is not affected)
[77], 2008	Virtex-4 FX140,BST Architecture	228K prefixes	324 MLPS	Depends on the trie depth, 17 cycles with 162 MHz ≅100 ns)	Yes	In place updates possible
[42], 2009	Virtex-4 FX140, Hybrid Architecture (DMA + BST)	257K prefixes	340 MLPS	Depends on the trie depth, 18 cycles with 170 MHz ≅105 ns)	Yes	In place updates possible
[78], 2010	Virtex-5 FX200T, 2-3 tree	200K prefixes	348 MLPS	No information	Yes	Update in 1 cycle (deletion / insertion) possible (requires precomputation)
[79], 2010	Virtex-5 LX330, Pipelined 2-3 tree	96K 32-bit keys	242 MLPS	15-level tree with 135 MHz clock (≈111 ns)	No information	3.97 Million DELETEs or INSERTs per second

update performance they achieve with respect to the time spent for the operations performed on hardware, but they omit the time spent on software for pre-processing the update.

TCAM-based solutions have the superior advantage of single-cycle search capability. However, most of the researchers consider TCAMs unfavorable due to their high price and high power consumption. Therefore the studies are conducted around to compact the data to be stored or to decrease the power consumption with some modifications on the TCAM architecture. Similar to SRAM-based solutions, the position of the prefixes on the TCAM is known via a software that keep tracks of the prefixes, and any update is first pre-processed in software, and then applied on the hardware.

Although both SRAM-based and TCAM-based solutions are hardware solutions, they still need auxiliary software for update operations. A software compromises the high speed operation performance of the hardware, and therefore we think that the software should be eliminated also for the update.

Another issue that we want to emphasize is about PE design. PE is an important part of the TCAM chips. Former TCAMs employ processing elements for resolving multiple matches and encoding the best match. As a result the processing element is usually designed in two stages: 1. Multiple Match Resolver (MMR), and 2. Match Address Encoder (MAE) [71]. Fung and Sachdev propose a priority encoder circuitry, called as "Direct Encode", which performs MMR and MAE operations in a single stage [80]. Even the priority encoder is implemented in two stages or in single stage; the implementation directly depends on the TCAM cell array size, and are implemented separately. If the TCAM size changes, the priority encoder should also change accordingly. Therefore, a separate priority encoder implemented after TCAM cell array decreases the scalability.

CHAPTER 3

S-DIRECT : SCALABLE AND DYNAMICALLY RECONFIGURABLE TCAM ARCHITECTURE FOR HIGH-SPEED IP LOOKUP

Ternary Content Addressable Memories (TCAMs) are the most often used components to build hardware IP lookup engines for backbone routers. TCAMs are special type of memories which search the entire content of the memory in a single cycle for a given input and then return the address of the matching entry if there is a match in the stored content of the TCAM. A *priority encoder* (PE) has to accompany the TCAM, when there is more than one match for the searched input data. Therefore, the design of the PE logic is very significant to enable LPM when TCAMs are used to build the IP lookup tables in routers.

One of the main contributions of this thesis is the is the *S-DIRECT-Scalable and Dynamically REConfigurable TCAM* architecture that is custom designed for IP address look-up. The novel features of this architecture are:

1. An S-DIRECT TCAM of any size is constructed from Basic TCAM Blocks (BTB) each which has an individual PE logic. Hierarchically arranging the PE logic of the BTBs results in inherent PE for the resulting TCAM. The single cycle search of the TCAM includes priority encoding process. Hence, we achieve *design scalability* eliminating the requirement of custom built PE circuitry that depends on the size of the TCAM.
2. S-DIRECT has *non-blocking update capability*. Writing an S-DIRECT entry does not interrupt the search on the remaining entries, simultaneous update and search operations can be performed.
3. The update requests are processed and subsequently the updates are carried out entirely in hardware without additional software support. The update requests are served in constant time.
4. S-DIRECT is not limited to a specific hardware platform or a specific TCAM cell implementation provided that the hardware requirements that we define are satisfied.

We demonstrate the viability of S-DIRECT by implementing it on FPGA as the selected hardware platform. FPGA fulfills the hardware requirements that we define for the update capabilities. The conventional way to implement a TCAM architecture is using a prefix register and a mask register for tri-state marking that we call prefix/mask register (PMR)-based implementation. A comparator circuitry is used for each entry. To this end, we implement S-DIRECT TCAM entries with both this legacy PMR approach and with a different approach which utilizes the Look Up Table (LUT) resources to validate that it is a general architecture. Combined with our integrated priority encoder and address decoder circuitry, we implemented a 16Kx32 TCAM with our LUT-based approach and tested it. We achieved a 153 MHz operating frequency that corresponds to a line rate close to 50 Gbps.

The remainder of this chapter is organized as follows. Section 3.1 describes the S-DIRECT architecture in detail together with the port settings and executions of LPM lookups and prefix updates. Section 3.2 describes our implementation of the S-DIRECT architecture that is constructed with the LUT resources on FPGA. We then provide resource utilization, clock frequency and power figures for different lookup table sizes for both our implementation and a traditional PMR based TCAM implementation on FPGA. Finally, we conclude this chapter with the remarkable contributions of S-DIRECT architecture.

3.1 S-DIRECT Architecture and Operation

S-DIRECT is a custom TCAM architecture that is designed to perform two main functions for IP lookup. The first one is finding the LPM for a given IP address with the corresponding next hop index value for the matching prefix. The second one is updating the entries in the IP lookup table.

The overview of the S-DIRECT architecture is depicted in Fig. 3.1. The main components of S-DIRECT are the *Augmented TCAM Block* (ATB), the *Preprocessing Logic Block* (PLB), and the *Next Hop Index Memory* (NHIM). Each TCAM entry in ATB stores an IP prefix together with its length. Accordingly, we partition an ATB of D entries into up to 32 blocks where D is the desired size of the IP look up table. Each block b_k has m_k entries of prefix length k where $1 \leq k \leq 32$. The locations of the blocks in ATB are sorted in the increasing order of the prefix lengths and there is no presumed order within a block. We assume that, m_k are known for all k at the design stage similar to the assumptions in [40][42][77][78].

PLB is responsible for appropriately formatting the inputs of S-DIRECT to be applied to ATB and NHIM and for controlling the write cycles during the TCAM update operations. NHIM is an auxiliary memory of D lines and stores the IP router outgoing port index (next hop index) corresponding to the matched IP prefix which is represented by M bits. Here, M is determined according to the number of outgoing ports of the IP router. The next hop index for the IP prefix that is stored in address a in ATB is also stored in address a in NHIM. The *Default Next Hop Index Register* (DNHIR) holds the default next hop index that is assigned for IP addresses that are not matched in ATB. The output of S-DIRECT is `next_hop_index` which is the next hop index for the searched IP address if there is a match or the default next

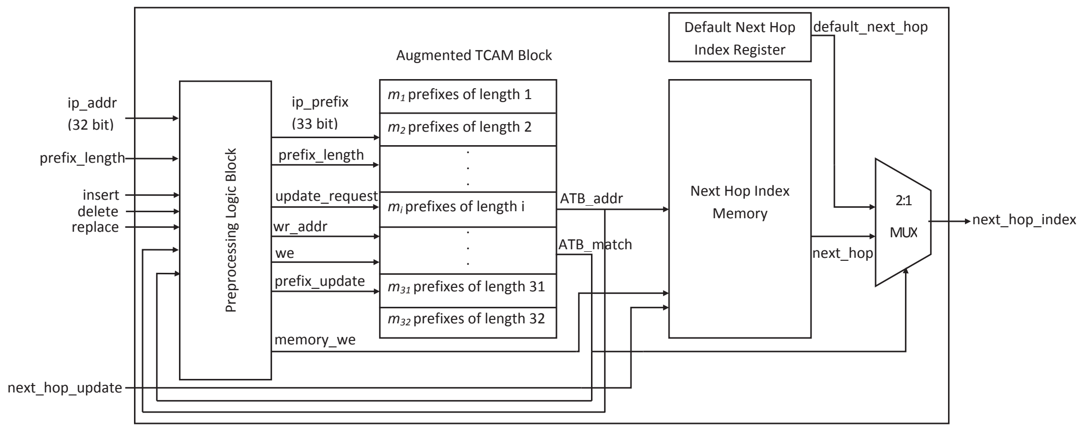


Figure 3.1: S-DIRECT Complete Architecture.

hop value otherwise.

Each TCAM entry in the ATB consists of a *prefix cell* augmented with an additional *length cell* as seen in Fig. 3.2. The prefix cell implements a logic function where `ip_prefix` and `prefix_match` are the input and output of that logic function, respectively. Furthermore, `prefix_update` is the input that changes the truth table of the prefix cell logic function when `we=1`. S-DIRECT is custom designed for LPM IP lookup. Hence, the prefix cell logic function is defined such that `prefix_match=1` if the p most significant bits of `ip_prefix` input are the same as a given p bit prefix. For the rest of this thesis we call this given p bit prefix as the *stored* content of the prefix cell.

Similar to the prefix cell, the length cell implements a logic function with the respective input and output ports of `prefix_length` and `length_match`. The length cell logic function is defined such that, for a TCAM entry that resides in block b_k in the ATB, `length_match=1` if `prefix_length=`$k-1$. Similar to the prefix cell, we call this $k-1$ value as the stored content of the length cell. Here, k is fixed once before the operation and does not change during the operation.

The updates of the IP look up table require an exact prefix match (EPM) different than LPM as we define in Section 3.1.2. To this end, if there is an update request indicated by `update_request=1`, then `match=1` requires `length_match=1` and `prefix_match=1` at the same time. Otherwise `match=prefix_match` for LPM.

We define two types of entries which are the *valid entries* and *empty entries* in the ATB. A valid entry that resides in block b_k has a prefix cell that stores a $k+1$ bit prefix `'0'&px` where `px` is the IP prefix and & indicates the concatenation operation. Accordingly, the `ip_prefix` is an input that is 33 bits wide. The values of the least significant $32-k$ bits of the prefix are ignored in the matching. We represent these contiguous *don't care bits* with a single * for

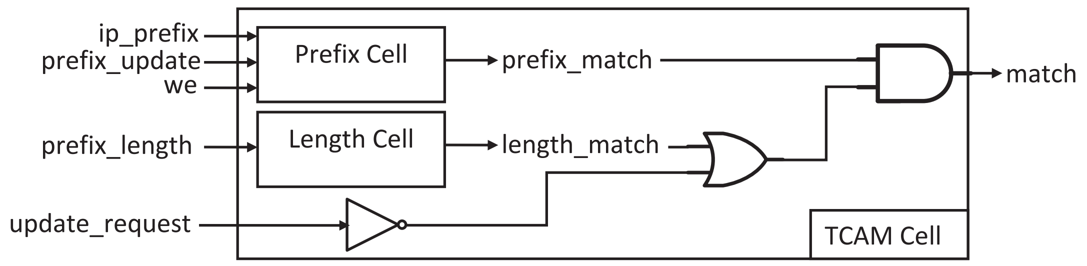


Figure 3.2: Augmented TCAM cell.

the rest of this thesis complying with the standard IP prefix representation. Hence, the stored prefix is represented as `'0'&px*`. An empty entry at a given time does not store any prefix. It has a prefix cell that stores a 1 bit prefix `'1'*` that is differentiated from the valid entries by its most significant bit. The empty entries may be allocated at the initialization of the TCAM for further prefix insertions, or they may emerge after deletion operations.

An example lookup table construction for S-DIRECT is shown in Table 3.1. *M* is selected to be 3 in this example.

Table3.1: Example ATB and NHIM Construction.

Original Prefix	**ATB Address**	**Prefix cell content**	**Length cell content**	**Next Hop Value in NHIM**
10*	000	010*	00001	001
01*	001	001*	00001	011
empty	010	1*	00001	-
011*	011	0011*	00010	111
101*	100	0101*	00010	101
010*	101	0010*	00010	100
empty	110	1*	00010	-
empty	111	1*	00010	-

We next present the inputs and outputs of S-DIRECT and ATB together with their functional definitions in Tables 3.2 and 3.3 respectively. Note that `prefix_length` is an S-DIRECT input that is directly forwarded to each entry in the ATB and the width of the `prefix_update` is a hardware platform dependent parameter to be determined (TBD).

The S-DIRECT output `next_hop_index` is the output of the 2:1 multiplexer as seen in Fig. 3.1. One input of the multiplexer is the NHIM output `next_hop` which is the content of NHIM block at `ATB_addr`. The other input is `default_next_hop`. The `ATB_match` is the multiplexer selector. NHIM inputs are `next_hop_update` which is the write port for changes in the next hop indices and the respective write enable port `memory_we`.

Table3.2: S-DIRECT Inputs and Outputs

Port	Width	Function
`ip_addr`	32	IP address searched by LPM for destination IP address lookups Update operations: Prefix to update specified with its length
`prefix_length`	5	Length of the prefix in bits (valid for update operations)
`insert`	1	'1' : for insert requests, '0' : otherwise
`delete`	1	'1' : for delete requests, '0' : otherwise
`replace`	1	'1' : for replace requests, '0' : otherwise
`next_hop_update`	M	Next hop value for the update request (valid for insertion, and replace requests)
`next_hop_index`	M	Next hop index value for the incoming packet (valid for non-update operations)

3.1.1 Longest Prefix Match (LPM) Lookup with Inherent Output Encoding

The 32 bit destination IP address is applied to `ip_addr`, `insert`, `delete`, `replace` are all reset. Accordingly, `update_request = 0` and `prefix_length` does not affect the match.

PLB generates 33 bit ATB input `ip_prefix='0'&ip_addr` such that valid entries are searched for LPM.

The elementary unit for building an ATB is the BTB (Basic TCAM Block) with two TCAM entries as seen in Fig. 3.3. Accordingly, if there is a single match, `addr_out` returns the address of the matching entry. If there is no match, `addr_out` is '0'. If both entries match, then `addr_out` is '1' to implement LPM.

ATB of size $D_l = 2^l$ in Fig. 3.1 is recursively constructed by two TCAM Blocks of size $D_{l-1} = 2^{(l-1)}$ where, first D_{l-1} entries and the second D_{l-1} entries constitute the Low Nibble and the High Nibble respectively as shown in Fig. 3.4 (a).

The match signal and the corresponding address if there is a match are generated with a logic that is constructed similar to a *Binary Tree* of depth l. The match signal at stage s is generated with an OR of the match results from Low Nibble and High Nibble at stage $s-1$. An ATB of size $D = 2^l$ requires $2^l - 1$ OR gates at l levels. An address encoding logic generates the

Table3.3: Inputs and Outputs of the Augmented TCAM Block (ATB)

Port	Width	Function
`ip_prefix`	33	IP lookup: Destination IP address, preceded by a '0'. INSERT operations: k-bit prefix, preceded by a '1' DELETE and REPLACE operations: k-bit prefix, preceded by a '0'
`update_request`	1	`= insert OR delete OR replace`
`wr_addr`	$log_2(D)$	Address of ATB entry to write
`we`	1	write enable for ATB
`prefix_update`	TBD	new prefix value to be matched
`ATB_match`	1	LPM or EPM match result
`ATB_addr`	$log_2(D)$	`match = true`: Address of the TCAM entry that matches the input query (LPM or EPM) `match = false`: Ignored

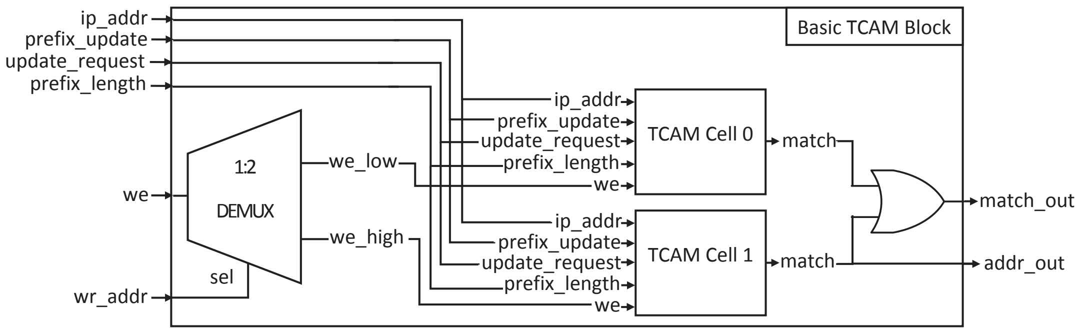
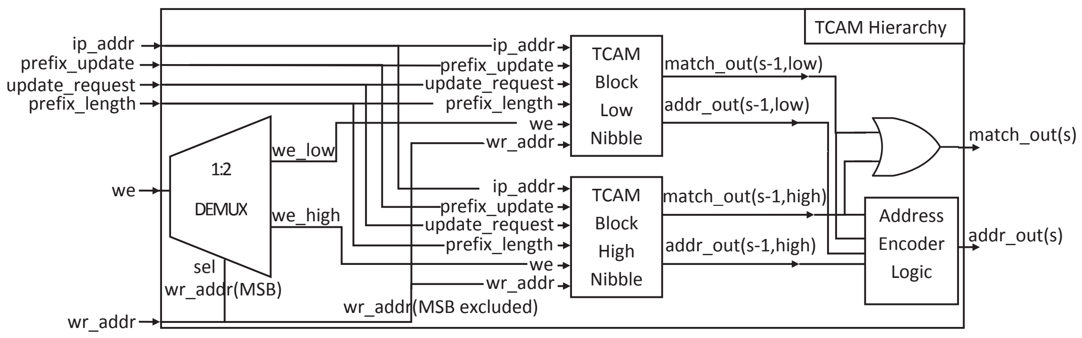


Figure 3.3: Basic TCAM Block (BTB).

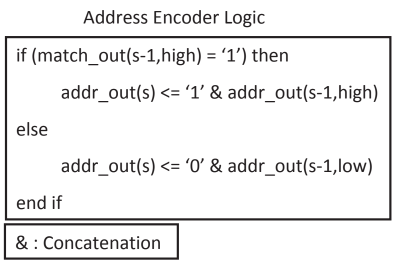
match address at stage s for TCAM blocks with a number of entries equal to or larger than 4 from the match addresses of stage $s-1$ as shown in Fig. 3.4 (b). Consequently, a total of $2^{(l-1)}-1$ address encoder logic blocks are required at $l-1$ levels.

Note that `ATB_match` and `ATB_addr` as defined in Table 3.3 and Fig. 3.1 are the `match_out` and `addr_out` of the TCAM block at the last stage as shown in Fig. 3.4.

We demonstrate our encoding logic with an example: We perform an LPM search for 010* in



(a)



Address Encoder Logic

```
if (match_out(s-1,high) = '1') then
    addr_out(s) <= '1' & addr_out(s-1,high)
else
    addr_out(s) <= '0' & addr_out(s-1,low)
end if
```

& : Concatenation

(b)

Figure 3.4: (a)TCAM Hierarchy (b) Address encoder logic behavioral description.

a TCAM of size 8 with the entries shown in Table 3.1. '*' denotes a padding of 29 bits. We assume that $M = 3$ and the content of DNHIR is `000`. Accordingly;

The S-DIRECT inputs: `ip_addr=010*`, `prefix_length=*` (ignored for LPM match), `insert=0`, `delete=0`, `replace=0`, `next_hop_update=*` (ignored).

The PLB outputs and ATB inputs: `ip_prefix=0010*`, `prefix_length=*` (ignored), `update_request=0`, `we=0`, `wr_addr=*` (ignored), `prefix_update=*` (ignored), `memory_we=0`.

ATB outputs and NHIM inputs: `ATB_match=1`, `ATB_addr=101` (See Fig. 3.5)

NHIM output, 2:1 MUX inputs: `ATB_match=1` (MUX select), `next_hop=100`, `default_next_hop=000` (not selected) .

S-DIRECT output: `next_hop_index=100`

Note that prefix `01*` also matches to `ip_addr=010*` but the longer matching prefix is selected.

Now assume that the PLB outputs and ATB inputs are as follows: `ip_prefix=0010*`, `prefix_length=00010` `update_request=1`, `we=0`, `wr_addr=*` (ignored), `prefix_update=*` (ignored), `memory_we=0`.

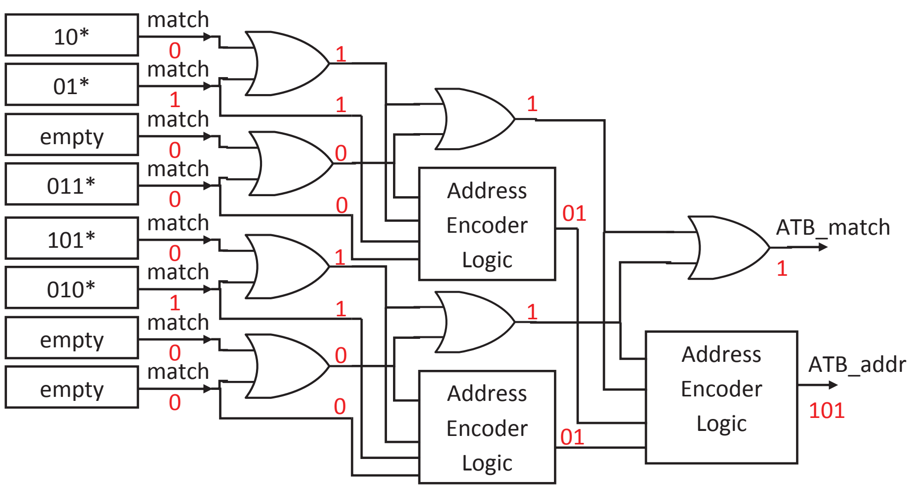


Figure 3.5: Hierarchical operation of the address logic for the Encoding Example.

`update_request=1` results in an EPM that takes the values stored in the length cells into account as shown in Fig. 3.2.

Then;

ATB outputs and NHIM inputs: `ATB_match=1`, `ATB_addr=001`

NHIM output, 2:1 MUX inputs: `select=1`, `next_hop=011`, `default_next_hop=000` (not selected) .

S-DIRECT output: `next_hop_index=011`

S-DIRECT completes both LPM and EPM in single clock cycle each.

3.1.2 Updates and Writing the Selected TCAM Entry

S-DIRECT can perform the following update operations which write on the content of the prefix cells;

- DELETE: deleting a prefix from the ATB.
- REPLACE: replacing the next hop index of a prefix by a new value in NHIM.
- INSERT: inserting a new prefix into the ATB.

We assume that in each clock cycle only one request (LPM IP lookup, INSERT, DELETE or REPLACE) can be issued.

In each BTB there is a simple 1:2 demultiplexer (DEMUX) unit that forwards the write enable input `we` to TCAM Cell 0 or TCAM Cell 1 as shown in Fig. 3.3 according to the one bit address `wr_addr` applied to the select input.

For an ATB of $D = 2^l$ entries, a total of $2^l - 1$ demultiplexers are organized in a Binary Tree of depth l similar to the logic that is described in Section 3.1.1. Let the l-bit TCAM entry address be shown as $b_1 b_2 \ldots b_l$ where b_1 is the most significant bit. The root of the DEMUX tree is at level 1 and the leaves which are in each BTB are at level l. Accordingly, for all levels $1 \ldots l$, b_n is the select for the demultiplexer(s) at level n.

Consequently, only the entry to be updated is enabled for writing a new value, and the remaining entries are not affected by the update process. The search process can continue during updating the selected entry achieving *non-blocking update* capability.

The LPM search can return the wrong address for the TCAM entry to be updated for these operations. Assume that the prefix to be updated is "01*" as shown in Table 3.1. The search returns the address of the "010*". Hence, it is required to find the EPM that is achieved if and only if both the prefix cell and length cell match.

In the remainder of this Section, we present the details of the update operations.

DELETE/REPLACE:

Suppose a k bit prefix is to be deleted or its next hop index is to be replaced by another value. A 32 bit input to `ip_addr` is applied padding the less significant $32 - k$ bits with any bit values, `insert=0` and `prefix_length = k-1`.

`delete=1` and `replace=1` for DELETE and REPLACE operations respectively setting `update_request=1` in both cases. The S-DIRECT input `next_hop_update` is set to the new next hop index value for the prefix to be replaced.

DELETE/REPLACE are both two step operations. The first step for DELETE/REPLACE operations is the EPM search for the prefix of `ip_addr`. To this end, the PLB applies `ip_prefix='0'&ip_addr` with `prefix_length=k-1`. If ATB output `ATB_match=0` then none of the entries are deleted/replaced. In the second step, if `ATB_match=1` and `delete=1`, then the PLB provides the content to write the prefix cell of the selected TCAM entry, applies the match address `ATB_addr` to `wr_addr` and sets `we=1` starting the write of the TCAM entry. If `ATB_match=1` and `replace=1`, then the PLB sets `memory_we=1` starting the write of the `ATB_addr` location of the NHIM with the value of `next_hop_update`.

INSERT: The IP prefix to be inserted of length k is applied to `ip_addr` padding the less significant $32 - k$ bits with any bit values, `prefix_length = k-1`. `insert=1`, `delete=0`, `replace=0` hence, `update_request=1`.

Subsequently, the INSERT operation is carried out in three steps. The first step is the search for an exact match for the prefix to be inserted to prevent multiple existences for the same prefix. To this end, the PLB applies `ip_prefix='0'&ip_addr` with `prefix_length=k-1`. If `ATB_match=1` then the INSERT operation terminates.

If `ATB_match=0`, in the second step, PLB sets `ip_prefix='1'&ip_addr` with `prefix_length=k-1` which returns the address of the empty entry at the highest address among all the empty entries in block b_k. If there is no empty location in b_k, then `ATB_match=0` and the INSERT operation terminates.

If `ATB_match=1`, then in the third step, the PLB provides the content to write by `prefix_update='0'&ip_addr`, applies the match address `ATB_addr` to `wr_addr` and sets `we=1` starting the write of the TCAM entry.

Both of the first and second step are completed in one clock cycle. The number of clock cycles for the writing process in the third step depends on the implementation platform.

3.1.3 Architecture Evaluation and Implementation Platform Requirements

S-DIRECT is a hardware architecture for TCAM-based IP lookup table implementations with the following platform independent architectural features:

- Each TCAM entry consists of a prefix cell and a length cell (Fig. 3.2).
- BTB consists of two TCAM entries (Fig. 3.3), a match output for the two entries, inherent encoding of the higher match address, 1:2 DEMUX that selects the entry to be updated by forwarding the write enable signal.
- An address encoder logic for TCAM tables with 4 and more number of entries which returns the higher match address.
- Binary tree organization of the match output, the address encoder logic, the write enable DEMUX'es.

The run-time reconfigurability, non-blocking update, and update on the hardware require certain write capabilities in the prefix and length cells which are listed below:

1. The *bitlines* for read and write operations should not be shared. There should be a "data write" port for each prefix cell for the runtime change of IP prefixes where any write through this port should not affect the lookup. This port corresponds to `prefix_update` of the prefix cell. The lookup is carried out with input provided by another port which corresponds to `ip_prefix` of the prefix cell.
2. Each TCAM cell should have its individual "write enable" ports so that any selection does not block the lookup on other cells which are not selected.

It is important to note that S-DIRECT can be built by integrating its encoder and match circuitry with any TCAM hardware. The TCAM cells can be built with any approach including the legacy PMR based cells provided that they have the above listed write features. In that case writing to a prefix cell through `prefix_update` port corresponds to changing the prefix and mask registers.

The capabilities of S-DIRECT and the respective timings can be summarized as follows:

- LPM and EPM lookup: Completed in a single clock cycle hence latency of these operations is one clock period.
- INSERT a prefix: Completed in two clock cycles (existence check and the search for the empty location to write the prefix) plus the time to subsequently write the TCAM entry (depends on the hardware platform)
- DELETE a prefix and REPLACE the NHIM: Both of these operations are completed in one clock cycle (search for the corresponding prefix) plus the time to subsequently write the TCAM entry (depends on the hardware platform)

Assuming that one LPM or update request is issued in each clock cycle, S-DIRECT can achieve an IP lookup rate of 1 lookup/clock cycle if there are no update requests. Provided that the hardware requirements above are satisfied, DELETE/REPLACE do not affect this lookup rate as the first step of these operations is an exact match search followed by write operations which can be concurrently executed with the subsequent lookup request in the next clock cycle. Different than DELETE/REPLACE, the INSERT operation starts with two lookups. Hence, if another LPM lookup or DELETE/REPLACE request is issued during the second step of the INSERT operation, either the INSERT has to terminate or the new request has to be dropped.

Consider an S-DIRECT implementation that runs at a frequency of F clock cycles/sec and with a maximum rate of INSERT operations of K prefixes/second. Over an interval of time T seconds, $K \cdot T$ clock cycles out of $F \cdot T$ are used for the second search of the INSERT operation resulting in the maximum packet process rate (throughput) of $\frac{F \cdot T - K \cdot T}{T} = F - K$ packets/sec (including all updates and LPM lookups).

The operating frequency F is defined by the critical path which is determined by the combinational circuit with the longest delay between two clocked elements. Note that all inputs/outputs are clocked at the ingress/egress ports of S-DIRECT. In S-DIRECT with $D = 2^l$ entries there are two major combinational circuits. The first one is the priority encoder circuit that selects the matching entry at the highest address that is implemented as a Binary Tree of MUXes with $l - 1$ levels (See Section 3.1.1). The second one is the address decoder circuit, used for selecting the TCAM entry to be updated which is also arranged in a Binary Tree of DEMUXes with l levels (See Section 3.1.2). The address decoder circuit has one extra level, however, the priority encoder has an additional comparison circuit to determine whether the matching entry is in the high or low nibble at each stage. Furthermore, the priority encoder circuit gets the `match` outputs of all TCAM cells as inputs. Note that the match outputs of the ATB are also generated with a combinational circuit. Hence, the entire ATB is also part of the priority encoder circuit making it the critical path in our design. The source of the critical path is `ip_addr` input and the end is `ATB_addr` output of the ATB. It is interesting to note that [58] also states that the critical path of priority encoding prohibits achieving a high searching throughput.

The critical path of the design is conceptually the same for all TCAM sizes, but the the critical path delay increases as the TCAM size increases as well. In our LUT-based FPGA implementation (Section 3.2.2) the clock frequency that is determined by this path is 338 MHz, and 153 MHz for 16x32 TCAM and 16Kx32 TCAM respectively.

Finally, the match signal is generated according to the prefix match rule of IP lookup as defined for the TCAM cell previously in this Section. For a different definition of the match signal, S-DIRECT can be used as a TCAM that returns the matching entry at the highest address. Furthermore the length cell can be used as an additional match parameter by simply assigning `update_request=1`.

Here we would like to compare S-DIRECT in terms of architectural properties to a relevant recent work [70] for reference. Similar to our work, they propose a TCAM architecture for IP forwarding tables. To the best of our knowledge this is the first work that considers the non-blocking TCAM updates. However, they achieve this property by employing two copies of the tables stored in two TCAMs, one for IP look-up and one for table updates doubling the hardware resource requirement. Furthermore, the processing for the update is carried out in software hence, they do not feature update in hardware. The design of the PE is not mentioned in this work.

3.2 Implementation and Evaluation of S-DIRECT on FPGA

Any hardware platform that satisfies the requirements in Section 3.1.3 can be used for S-DIRECT implementation. We choose Xilinx XC7VX1140T (Virtex-7) FPGA [12] as our implementation platform in this thesis. The S-DIRECT architecture is implemented on ISE 14.1 [81] software and the total on-chip power consumption is estimated by Xilinx's Power Estimator tool [82]. This total power includes both the device dependent *static power* which is dissipated when there is no signal activity and the additional *dynamic power* that is dissipated during the operation.

The conventional way to implement a TCAM architecture is using a prefix register and a mask register for tri-state marking that we call prefix/mask register (PMR)-based implementation. A comparator circuitry is used for each entry. PMR-based implementation depletes the FPGA resources very quickly as also indicated in [49]. We implement S-DIRECT with the ATBs of different sizes with PMR-based TCAM entries on Virtex-7 and present the resource and power consumption results in Table 3.4 to demonstrate this problem.

Table3.4: Resource Utilization for Different Size PMR-Based TCAM Implementations on Xilinx XC7VX1140T FPGA.

Size (Number of ATB entries)	**Number of Slice LUT6s (Available:712000)**		**Number of Slice Registers (Available:1424000)**		**On-chip Power (mW)**		**Max. Freq. (MHz)**	**Corr. line rate for IP lookup (Gbps)**
	Used	Utilization (%)	Used	Utilization (%)	Dynamic	Total		
16	378	0.05	1100	0.07	18	463	284	90.88
64	2195	0.31	4293	0.30	43	488	224	71.68
256	9182	1.29	17008	1.19	143	589	195	62.40
1024	36108	5.07	67735	4.76	497	947	173	55.36
4096	140573	19.74	270828	19.02	1955	2423	160	51.20
16384	553679	77.76	1102287	77.41	8003	8554	153	48.96

We implement the prefix and mask registers with D-type flip flop arrays. A comparator circuitry, that compares the `ip_prefix` with the prefix register according to the mask register content, accompanies each prefix/mask register pair. The contents of the prefix and mask registers can be changed in parallel in one clock cycle. The priority encoder and the address decoder circuitry are built as described in Sections 3.1.1 and 3.1.2 respectively.

In this thesis, we propose a more resource-efficient way for TCAM implementation on FPGA, which utilizes the LUT resources of the FPGA. The implementation details are given in the following sections.

3.2.1 S-DIRECT LUT-based FPGA Implementation

We make use of the large amount of LUT resources that are available on contemporary FPGAs in our implementation. An n-input LUT can be configured to implement any function with n inputs by storing and returning the function output for all 2^n input combinations. To this end, an n-bit TCAM entry can be implemented with an n-bit LUT by storing the match result for all possible 2^n n-bit inputs. For example a 3-input LUT can be configured to give '1' output for a match of "101" or to give '1' output for all inputs starting "10", i.e. match for "10*" as shown in Table 3.5.

There are two types of LUTs in Xilinx Virtex FPGAs [83]. One is the standard 6-input LUT (LUT6), which requires the reconfiguration of the FPGA if its content is to be updated. The other type is *Dynamically reconfigurable* 5-input LUT (CFGLUT5), which can dynamically be reconfigured during runtime [83] through a separate "write" port serially in 32 cycles. When a collection of CFGLUT5 is built for TCAM implementation, the TCAM entry to be updated can be selected individually for "write" operation, while the remaining entries are not affected for performing the standard LUT function. According to the hardware requirements listed in Section 3.1.3, we implement the prefix cells with CFGLUT5s and length cells with

Table3.5: LUT content examples.

Input	**LUT Content for 101 match**	**LUT Content for 10* match**
000	0	0
001	0	0
010	0	0
011	0	0
100	0	1
101	1	1
110	0	0
111	0	0

the standard LUT6s.

CFGLUT5 stores a total of $2^5 = 32$ bit VAL value for all combinations of its 5 inputs where VAL[i] is returned for input combination i. CFGLUT5 can return a match value for 5-bits. Hence, each ATB entry that stores a 33 bit value is implemented with 7 CFGLUT5s as shown in Fig. 3.6.

Let prefix "01011001101001100111010*" be stored in a TCAM entry constructed as shown in Fig. 3.6. The prefix is partitioned into 7 CFGLUT5s as shown in Table 3.6 with each VAL value for the corresponding CFGLUT5s.

Table3.6: Prefix partition into 7 CFGLUT5s and corresponding VAL values for "01011001101001100111010*"

CFGLUT5 Number	**Corresponding Prefix partition**	**Corresponding 32 bit-VAL (Hexadecimal)**
1	01011	00000800
2	00110	00000040
3	10011	00080000
4	00111	00000080
5	010*	00000F00
6	*	FFFFFFFF
7	*	FFFFFFFF

LUT6s can be configured to implement functions up to 6-inputs. So, we use one LUT6 per TCAM entry for 5-bit comparator logic for the length matching.

We implement a TCAM of size $D\times33$-bit by instantiating the single TCAM entry that consists of 7 CFGLUT5 and 1 LUT6 D times.

We store IP prefixes in the form of 33 bit IP addresses in the TCAM entries of S-DIRECT. This requires changing the contents of all 7 CFGLUT5s with parallel run-time reconfiguration. The reconfiguration writes the LUT contents with '1's and '0's according to the prefix that we want to store in them. We develop an update logic for this purpose which can change the

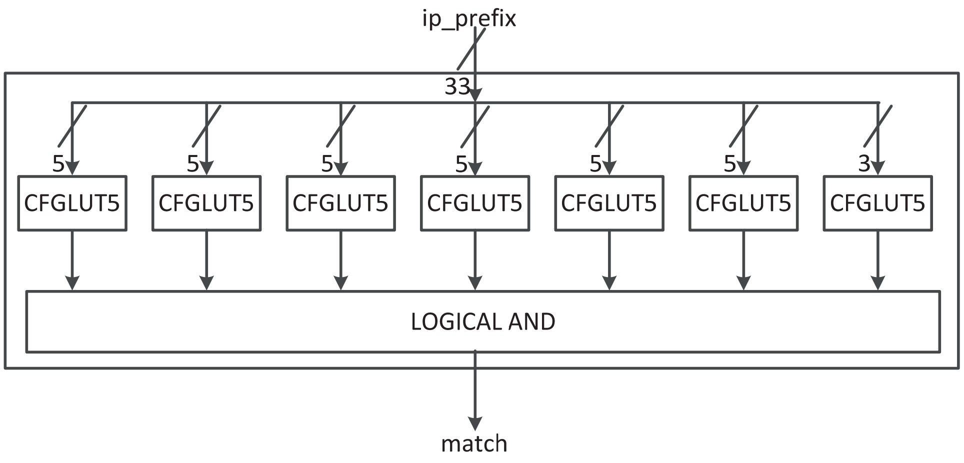


Figure 3.6: 33-bit Prefix LUT implementation.

LUT content automatically in hardware and perform write operations on LUTs.

The content of a CFGLUT5 is reconfigured serially by shifting a new VAL value in most significant bit (MSB) first mode. The VAL to be written in a CFGLUT5 of a TCAM entry is produced by a logic unit implemented on the FPGA that is designed as shown in Fig. 3.7.

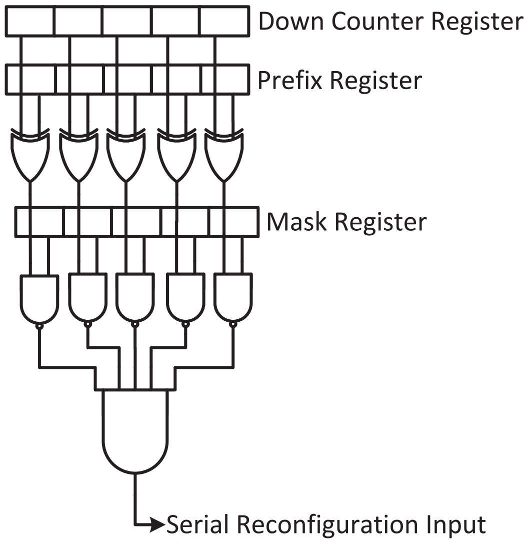


Figure 3.7: CFGLUT5 update logic.

The new prefix partition to store in a CFGLUT5 is first stored into the "Prefix Register", and the "Mask Register" is set according to the prefix partition length. Note that these registers are used only for writing the LUT content. The most significant *k* bits of the Mask Register are set to '1', and others to '0', where *k* is the prefix length. The Down Counter Register is initially set to all '1's. When the update request comes, the counter is enabled for 32 cycles, and counts down until again the register is set to all '1's. During the down-counting period, the Prefix Register is compared with the Down Counter Register content bit by bit (XOR operation). The outputs of the XOR gates are then NANDed by the Mask Register bits. The Mask Register content is '0' and the output of NAND is '1' regardless of the XOR output for the "don't care" bits. The Mask Register content is '1' and the output of NAND is only '1' if the XOR output is '0', i.e., the Prefix Register and Down Counter Register bits are equal for the "cared bits". The outputs of NAND gates are fed into a 5-input AND gate finally, to produce the serial input for the CFGLUT5.

The `we` inputs to TCAM Cell 0 and TCAM Cell 1 in BTB as shown in Fig. 3.3 are connected to the reconfiguration clock enable (`ce`) inputs of the CFGLUT5 that implement the respective prefix cells. The `we` is set only for the TCAM entry selected by `wr_addr` as described in Section 3.1.2 and stays set during the 32 cycles to write all VAL[i] in parallel for the 7 CFGLUT5 that construct the prefix cell. The TCAM entry selection is performed at the lowest level, so the other entries are not affected while an entry is being updated. Hence, the update operations in TCAM do not block the IP-lookup operations. Note that an entry is available for lookup 32 cycles after the update request is issued.

3.2.2 S-DIRECT Implementation Evaluation

The resource utilization summaries for 16-entry, 64-entry, 256-entry, 1024-entry, 4096-entry and 16384-entry LUT-based TCAMs are shown in Table 3.7. The results show that, the number of CFGLUT5s used is directly proportional to the size of the TCAMs; e.g, 16Kx32 TCAM utilizes as many CFGLUT5s as 1K=1024 times of the 16x32 TCAM. The size of the TCAM, that can be implemented on a FPGA directly depends on the number of CFGLUT5s available on that FPGA. The NHIM implementation is not a major consideration in our study. Hence, we implement a Block RAM (BRAM) of data width $M = 8$ for the sake of completeness of the IP lookup solution. The power results are reported with the assumption of maximum operating frequency.

The Number of Slice LUTs show the logic resources used for length cells and the circuitry for address decoding/encoding. We observe that this value increases almost linearly with respect to the number of ATB entries similar to the CFGLUT5. This shows that the length cells dominate this resource and despite the exponential growth of the address decoding and priority encoding logic with the depth of the binary tree (See Sections 3.1.1 and 3.1.2) the consumed resource for this peripheral logic is small. Hence, the design scalability is paralleled with the resource use.

The operating speeds according to the synthesis results are 338 MHz, and 153 MHz for 16x32 TCAM and 16Kx32 TCAM respectively. As the size of the TCAM increases, the number of logic levels in priority encoding logic also increases, and since the logic in the priority encoder is combinatorial, the data path gets longer, resulting in a lower clock frequency operation. 153

Table3.7: Resource Utilization and Operating Speed Table for Different Size LUT-Based TCAM Implementations.

Size (Number of ATB entries)	**Number of Slice LUT6s (Available:712000)**		**Number of Slice CFGLUT5s (Available:283200)**		**On-chip Power (mW)**		**Max. Freq. (MHz)**	**Corr. line rate for IP lookup (Gbps)**
	Used	Utilization (%)	Used	Utilization (%)	Dynamic	Total		
16	94	0.01	112	0.04	8	453	338	108.16
64	365	0.05	448	0.16	14	459	261	83.52
256	1634	0.23	1792	0.63	41	486	214	68.48
1024	6520	0.92	7168	2.53	90	536	191	61.12
4096	26556	3.73	28672	10.12	323	771	168	53.76
16384	106347	14.94	114688	40.50	1121	1578	153	48.96

MHz clock rate for 16Kx32 implementation is equivalent to 153 Million lookups per second (MLPS) without any update requests. The BGP records at [9] show a peak prefix update rate of 1442 updates/sec as of May 2013. Hence, the slowdown due to the one additional clock cycle that is required for INSERT as discussed in Section 3.1.3 is negligible. Considering the smallest IP packets of 40 Bytes, 153 Million packets/sec correspond to a data rate of 48.96 Gbps, which is over OC-768 40 Gbps link data rate. At this operating frequency we achieve approximately 6.5 ns latency for IP lookup (single cycle search), a 215.7 ns latency for DELETE/REPLACE operations (33 cycles; 1 cycle for the search, and 32 cycles for the update) and 221 ns latency for INSERT (34 cycles; 2 search cycles and 32 cycles for the update).

When we compare the PMR-based implementation to LUT-based implementation, we see that the PMR-based implementation consumes more power than the same size LUT-based implementation. This is due to the extra comparator circuitry for each TCAM entry, and more resource usage of the PMR based implementation.

In addition, the size of the logic for both implementations affects the operating frequency and the total power consumption. To this end, we investigate the *energy per operation* in joules/search which is a normalized metric. Note that we compute this metric according to the dynamic power values as it is dissipated during the operation and the static power is device dependent.

As an example computation, for 16K LUT-based implementation, the dissipated dynamic power is 1121mW = $1121 \cdot 10^{-3}$J/sec. As we perform one search operation per clock cycle, each search takes $1/153 \cdot 10^{6}$sec at 153 MHz operating frequency. Then, the consumed energy is $(1121 \cdot 10^{-3}) \cdot (1/153 \cdot 10^{6}) = 7.32$nJ/search.

We observe that, the energy per search grows linearly with the TCAM sizes for both LUT-based and PMR-based implementations of S-DIRECT as shown in Fig.3.8. These results provide evidence for the power scalability of our architecture.

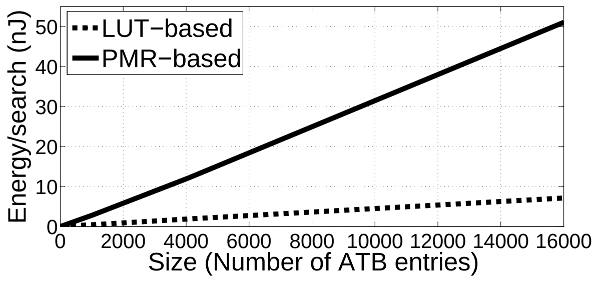


Figure 3.8: Energy/search.

The S-DIRECT architecture is simulated in MODELSIM 10.1a [84]. The following functionalities are tested and verified.

- The TCAM entries are written according to the routing table entries (dynamic configuration, updating)
- Single cycle search (Expected TCAM operation)
- Search during updating (Non-blocking update)
- LPM guarantee (Priority encoder)

We would like to note that multiple FPGAs can be used for higher-sized TCAM implementations. In that case, TCAM nibbles shown in Fig. 3.4 are the possible largest TCAM blocks, that can be implemented on a FPGA, and each nibble reside in different FPGAs. The same hierarchical architecture is implemented using multiple FPGAs, while the connections between the TCAM blocks are now inter-FPGA connections.

Here we would like to provide a brief discussion of previous TCAM implementations on FPGA in comparison to our implementation.

A large number of previous IP lookup table implementations on FPGA are utilizing only the BRAM (Block RAM) resources of the FPGA [40, 77, 42, 78, 79]. In these studies trie/tree data structures are constructed by BRAM modules and LPM is carried out with a pipelined search in multiple cycles. In the BRAM based TCAM design by [49], BRAM modules' address pins are used as data pins and data content as TCAM address output. One location can be reached at a time which makes the encoding for the LPM hard to implement.

LUT based implementations exist in [49] and [48]. These works do not utilize the reconfigurable LUT blocks hence they require re-synthesis for updates. [55] presents pipelined SRAM-based partitions and a separate priority encoder implementation that returns the matching address in 5 cycles. External evaluation is needed for updates.

Table3.8: Comparison of Embedded PE with Separate PE

	Number of Slice LUTs	**Max. Combinational Path Delay (ns)**
TCAM without PE	2048	1.180
PE only	992	6.131
TCAM + Separate PE	2645	7.129
TCAM + Embedded PE	2645	7.635

The achievable TCAM operating frequency in our FPGA implementation of S-DIRECT is less than the contemporary commercial TCAM products such as in [85] with 360 Million look ups/sec. Commercial TCAMs are specially designed ASIC circuits, and any FPGA logic implementation would be slower compared to its ASIC implementation counterpart. In addition, the non-blocking update, and update in the hardware features of our architecture and implementation are not available on commercial TCAMs. Hence, the commercial TCAMs do not suffer from the size increment due to additional circuitry.

The embedded PE of S-DIRECT provides design scalability by eliminating the need for designing a separate PE. The PE design affects two parameters which are the number of slice LUTs utilized, and the maximum combinational path delay for different configurations. Here, we investigate the effect of embedding the PE into basic TCAM block on these parameters by comparing our design with embedded PE to a separate PE circuit. To this end, we designed a PE for 1024-to-10 bits exactly the same way as it is designed as embedded to our BTB. We hierarchically constructed a MUX chain in a binary tree complemented with our Address Encoder logic at each level, and constructed our TCAM architecture without PE for 1024 entries. Then we combined our TCAM and the separate PE in a higher top level, and compared the synthesis results with S-DIRECT architecture. In Table 3.8 we compare the number of slice LUTs utilized, and the maximum combinational path delay for different configurations.

First of all, note that, according to the results obtained, the major fraction of the critical path delay is due to the PE logic. The single cycle search in the previous TCAM implementations and conventional TCAM architectures only include the generation of match signals and do not consider the time spent for PE, as the PE logic is a separate part on the overall architecture. However, in S-DIRECT we embed the PE logic in the TCAM architecture, and the single cycle search includes both TCAM search and priority encoding as well. Thus, PE logic adds extra delay to our architecture with respect to the previous studies and conventional TCAM implementation.

Our results also show that, number of slice LUTs utilized for both separate PE and embedded PE implementations are the same, implying that the same architecture is generated for both configurations. However, for both implementations the total number of slice LUTs is less than the sum of slice LUTs used for TCAM without PE and PE itself. This shows that synthesis tool performs some optimization when combining PE logic to TCAM architecture.

Considering the maximum combinational path delays, TCAM+PE path delay is close to sum of TCAM path delay and PE path delay. However, TCAM+Separate PE path delay is less than TCAM+Embedded PE, implying that synthesis tool performs better optimization for separate modules. We deduce that the combinational logic that should be synthesized for TCAM+embedded PE is somehow more complex to be optimized although the same architecture is achieved. Actually, the behavior of the synthesis tool is not deterministic, and can be different for the same design described in different ways.

As a result, we can safely say that, timing and resource utilization do not differ drastically for embedded PE and separate PE designs, but embedded PE has the advantage when scaling the TCAM to larger sizes in terms of the design complexity.

3.3 Concluding Remarks

In this thesis, we propose S-DIRECT, a new TCAM architecture that is custom designed for hardware IP lookup. S-DIRECT is hierarchically built from BTBs each with an inherent priority encoder logic. Hence, the priority encoder does not need to be redesigned when the routing table size is changed and scales with the changing routing table size. S-DIRECT is capable of performing single-cycle LPM lookups and constant time non-blocking routing table lookups in hardware provided that the TCAM entries have certain write capabilities.

S-DIRECT is not specific to the hardware platform or the TCAM cell implementation. We demonstrate the viability of S-DIRECT by implementing it with PMR based and LUT based TCAM cells on FPGA. Our LUT-based implementation achieves an operation frequency of 153 MHz that corresponds to a packet processing rate (throughput) of 153 million packets/second including lookups and table update requests. Assuming the worst case of smallest size IP packets, this packet rate is close to a bit rate of 50 Gbps.

We employ S-DIRECT in a single cycle full IP lookup solution, for the contemporary real routing tables in the next stage of our research.

CHAPTER 4

SHIP : SCALABLE HIGH SPEED IP LOOKUP ENGINE

Different than the TCAM-based IP lookup, there is also another hardware solution that gives IP lookup result in single cycle, which is the DMA (Direct Memory Access). In DMA, the IP address field of the incoming packet is used as the address of the RAM, which stores the destination port numbers corresponding to that IP address. Since the next hop values are directly associated to the IP addresses, the next hop value corresponding to a prefix is written to all possible IP address space spanned by that prefix. However, as we discussed in Chapter 2, in CIDR addressing scheme the address space of a network would be within the address space of another network, although both networks are unrelated with each other, and this situation results in the LPM problem. Therefore, next hop values should be carefully inserted into the target memory, considering the relation between prefixes. The process of determining the address spaces for the prefixes in a routing table is called as prefix expansion, and is performed as follows:

Prefix Expansion: Let m be the length of a prefix in the routing table, where $m < 32$. Then, there are 2^{32-m} possible matches for that prefix. Let A denote the set of all possible matching prefixes for the prefix of length m. So, all possible elements of set A should be considered, as if they are also in the routing table and are equivalently handled as the prefix of length m. However, there may be another prefix of length l, where $m < l <= 32$, and the set of all possible matches for this prefix, let say set B, may be a subset of A. In this case the elements of B should be removed from A, and the elements of B should be handled as the prefix of $l - bit$.

It may seem that, prefix expansion algorithm is run just at the initial address space determination phase, but actually this algorithm should be run whenever there is an update request as well. Once the routing table is mapped to the target memory, there is no more information about which address is a spanning of which prefix. Therefore, especially for update management, the conventional way to keep the track of prefix-address space relation is constructing a data structure on an auxiliary software. This software runs prefix expansion algorithm when there is an update request, and determines the addresses where the changes to be performed. However, employing software for routing table management on hardware platforms, contradicts with the high speed performance provided by the hardware, and therefore we believe that a hardware solution should also be capable of handling update request totally in hardware. To the best of our knowledge, there is no study that aims the update on the hardware for DMA-based IP lookup platforms, as in the case of other hardware-based solutions we discussed in this thesis.

So, we propose a hardware architecture for DMA-based IP lookup solution, that fully handles updates on the hardware, and additionally we propose hardware requirements for the updates being non-blocking as well.

4.1 Data Structure and Update Management

The update request for a given prefix, requires the management of the address space of the memory spanned by that prefix. A straightforward approach, that updates the whole address space spanned by the prefix for which the request is performed, would result in a loss of information for those prefixes, whose address span is within the address space on which the update operation is performed. Therefore, the update algorithm should be capable of differentiating the address spaces belonging to longer prefixes within the address space of the requested prefix. This can be achieved by inserting the length information of the prefixes to the memory locations together with the corresponding next hop values. So, the update algorithm can decide that a non-intended address space is encountered by comparing the length value read from the memory content with the length value of the prefix for which the request is performed. The length information can be encoded by 5-bits for IPv4. "‘00000"’ corresponds to length 1, while "‘11111"’ corresponds to length 32. However, there may be empty spaces, which are not spanned by any prefix in the routing table, and they should also be differentiated from occupied spaces. Normally, an additional empty-sign bit would be associated for this purpose, but there is no prefix of length less than 8 bits in real-life routing table [7], and hence any length value less than 8 could be used for indication of empty space. For convenience, "‘00000"’ is used for the sign of emptiness. Accordingly, the following update operations are performed on DMA-based IP lookup architectures as described:

- **INSERT:** A prefix and its corresponding next hop value are inserted into the routing table. There may be two conditions:

  1. Non-existing prefix: The length information of the prefix to be inserted is compared with the length field of the space covered by that prefix, and if the read length field is “00000” or smaller than the requested prefix length, new length value and corresponding next hop value is written to that memory location. If the read length field is larger than requested prefix length, that space is skipped.
  2. Existing prefix: The length information of the prefix to be inserted is compared with the length field of the space covered by that prefix, and if the read length field is equal to the requested prefix length value, nothing is done. Note that, the spaces covered by two different prefixes of the same length are disjoint and do not

have any intersecting space. Therefore, the insertion process can be exited at the first comparison if the read length field is equal to the requested prefix length.

- **DELETE:** A prefix and its corresponding next hop value are removed from the routing table. There may be two conditions:

  1. Non-existing prefix: The length information of the prefix to be inserted is compared with the length field of the space covered by that prefix, and if the read length field is not equal to the requested prefix length, nothing is done. The operation is exited after first comparison.
  2. Existing prefix: The length information of the prefix to be inserted is compared with the length field of the space covered by that prefix, and if the read length field is equal to the requested prefix length, the length field is written as "00000", and if the read length field is larger than the requested length, no operation is performed, and that space with longer prefix length is skipped.

- **REPLACE:** The next hop value corresponding to an already existing prefix is replaced with a new value. REPLACE is similar to DELETE, with a single change; the length field is not changed, but the next hop value is replaced with the new value on the memory.

As we described, there may be address spaces that are skipped during the update operation. The length information for the address space to be skipped carries also the information about the number of address locations that should be skipped. For example, a 24-bit prefix length corresponds to a $2^{32-24} = 2^8 = 256$ locations. So, when the algorithm encounters a memory location corresponds to a longer prefix, the next address location to be checked can be calculated by simply adding the 2^l to the current address, where l is the read length.

4.2 Hardware Requirements for Non-blocking Update

The main principle for non-blocking update is that the write operations should not block the read operations. Similar to the S-DIRECT hardware requirements, the memory contents should be capable of being accessed by separate read and write ports. The dual-ported RAMs (DPRAM) are actually capable of this property, having two different ports, both access the shared storage area. It is possible to perform simultaneous read and write operations on DPRAMs.

There is another memory architecture called as QDR (Quad Data Rate) memory, which operates at double data rate (DDR) clock frequency for read and write operations, and allows consecutive read and write operations within the same clock cycle. Although, QDR memories are not exactly having two separate ports, allowing read and write operations in the same cycle makes them also candidate for non-blocking IP lookup operations, since lookup operations can continue at rising edges of the clock, while write operations are performed at the falling edges of the same clock.

4.3 SHIP Architecture and Implementation

For a complete DMA-based IP lookup solution, a RAM of $2^{32} = 4G$ address size is needed. However, nor available DPRAMs neither QDR memories can provide such a capacity. Therefore, we decided to combine our two solutions proposed for TCAM and DMA-based hardware architectures, for a complete IP lookup solution supporting a real-life routing table. As we discussed in chapter 1, contemporary real routing tables have certain statistical properties. For IPv4 CIDR 96% of the real-life routing tables [7] are composed of prefixes of length 24 or smaller, and about 50% of the prefixes are of 24-bit length [42].

We exploit this statistical property of the real-life routing tables and propose a solution, that utilizes DMA-based approach for prefixes up to 24-bit length, and TCAM-based approach for prefixes longer than 24-bit. The overall architecture we propose is shown in Figure 4.1. The architecture is designed for parallel operation of BRAM and TCAM parts (S-DIRECT). The output is selected via a priority selector. The TCAM part has precedence, since it holds longer length prefixes. Priority selector is a simple multiplexer circuit, that passes the TCAM output if there is a match in the TCAM part, and DPBRAM output otherwise. There is a special processor unit for update operations that determines the unit on which the update will be processed and controls the write operations both on the BRAM and TCAM parts. Update processor constitutes two finite state machines. The first one is activated if the requested prefix length is less than or equal to 24 bits, and performs the prefix expansion operation and writing to the BRAM. The second one is activated if the requested prefix length is more than 24 bits, and performs the CFGLUT5s reconfiguration, which are the basic building blocks of TCAM.

As the implementation platform we select the state-of-the-art FPGA (Xilinx Virtex-7 1140T) [12], that we selected also for S-DIRECT implementation. However, on this FPGA, it is possible to implement a dual-ported BRAM (Block RAM)(DPBRAM) of $2^{24} = 16M$ addressed locations with 4 bit memory width only, and there is no other available FPGA, that offer more BRAM resources than our selected FPGA. We need at least 5-bit data width for the update on hardware requirement. So, for a complete solution, we have to use two FPGAs to achieve our goals. In this case we implemented a $8Mx8$ DPBRAM on each FPGA, 5-bit of the data width for length encoding, and 3-bit of the data width for next hop value. With this configuration we could also be able to double the achievable S-DIRECT size as well. The overall architecture's maximum operating frequency is the same as the maximum operating frequency limit of S-DIRECT, which is 153 MHz, since DPBRAM part does not have longer combinational path delays than the S-DIRECT. As we implemented DPBRAM part separately, we saw that it can operate at 258 MHz. Since S-DIRECT also gives result in one clock cycle, running parallel search on both DPBRAM and S-DIRECT, we achieve a single clock cycle latency for the IP

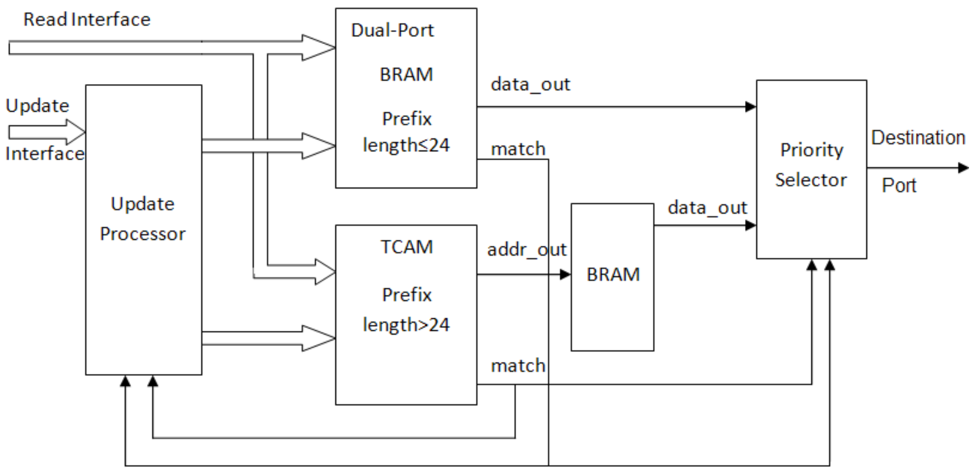


Figure 4.1: SHIP Overall Architecture

lookup process. S-DIRECT output has a higher priority, since it holds the longer prefixes. We use an update processor, and priority selector in order to determine where the update process will take place, and which output to select as the final result of the IP lookup, respectively.

The throughput of the overall architecture is 153 MLPS as in the case of S-DIRECT. As an additional information, the estimated power consumption is increased to 4906 mW, indicating that the DPBRAM adds 3328 mW to the total power consumption.

Here we should note that, we do not give the implementation details and resource utilization or power consumption results for different-sizes of DMA parts, since DMA part is built by instantiating XILINX FPGA's embedded BRAM primitives, and the resource and power consumptions are directly proportional to the BRAM size implemented. Furthermore, we implemented the maximum achievable DPBRAM on the target FPGA, and achieved a higher operating speed compared to the S-DIRECT architecture, which shows that the operating frequency of the SHIP architecture is limited by S-DIRECT. Any smaller size of DPBRAM implementation operates at no less frequency than maximum achievable DPBRAM implementation, what we do in this thesis.

Regarding the update performance of the DMA part of the overall architecture, in the worst case, the highest number of cycles to be passed for one update is 2^{16} = 65536 cycles, since there is no prefix of length less than 8 bits in real-life routing table [7]. 65536 cycles takes 0.43 ms for the clock rate of 153 MHz. With this rate it is possible to perform 2334 updates per second, which is higher than the average requirement of several hundreds of updates per second at concurrent routers [1] [86]. However, in the best case, an update for the 24-bit prefix takes only one cycle, which corresponds to a 153 Millions updates per second. Actually considering that 50% of the total prefixes are of 24-bit length, this update rate can also be considered as the average rate as well.

As a second alternative the same architecture can be constructed by employing Cypress CY7C1625KV18-333BZXC QDR-II SRAM memory [87] for the DMA part. This memory is organized as $16Mx9$ memory array, and can operate at 333 MHz. So, this memory is compliant with our FPGA implementation part performance metrics.

CHAPTER 5

IP LOOKUP IMPLEMENTATION ON FPGA

A general FPGA design flow is depicted as shown in Figure 5.1. If we discard the verification and bitstream generation phases, FPGA design flow can be investigated in three phases:

1. Entering the design and selecting the hierarchy
2. Synthesizing the design
3. Placing and Routing the design (Physical synthesis)

In the design entry phase, the designer defines the logic to be implemented in an abstraction level. The most common and conventional abstraction level is *"‘register-transfer level (RTL)"’*. RTL defines the digital circuit in terms of the flow of digital signals between the registers and the logical operations performed on those signals. RTL description of the design is performed using hardware description languages (HDLs), and the two most common HDLs are Verilog and VHDL. A higher level of abstraction, *"‘algorithmic level"’*, starts with an algorithmic description and uses a high-level language such as SystemC and Ansi C/C++. Algorithmic level is used mainly for more complex logical functions such as digital signal processing and fixed or floating point arithmetic operations.

The RTL description of the design is translated to an equivalent *"‘gate-level"’* hardware via an EDA (Electronic Design Automation) tool, and this process is called as *"‘logic synthesis"’*. If the tool uses algorithmic level of abstraction as input, then the hardware translation is process is called as *"‘high level synthesis (HLS)"’* [88]. HLS tools transform the functional code into RTL, and then the created RTL implementations are directly used in a conventional logic synthesis flow to create a gate-level implementation [89]. A very detailed and comprehensive survey on FPGA design automation is given in [90], which covers all major steps in FPGA design flow which includes: routing and placement, circuit clustering, technology mapping and architecture-specific optimization, physical synthesis, RTL and algorithmic-level synthesis, and power optimization. Another comprehensive survey study [91], mainly focus on high level synthesis for FPGAs, discussing why early HLS tools could not be successful and what kind of recent advancements are achieved in order the HLS become popular again.

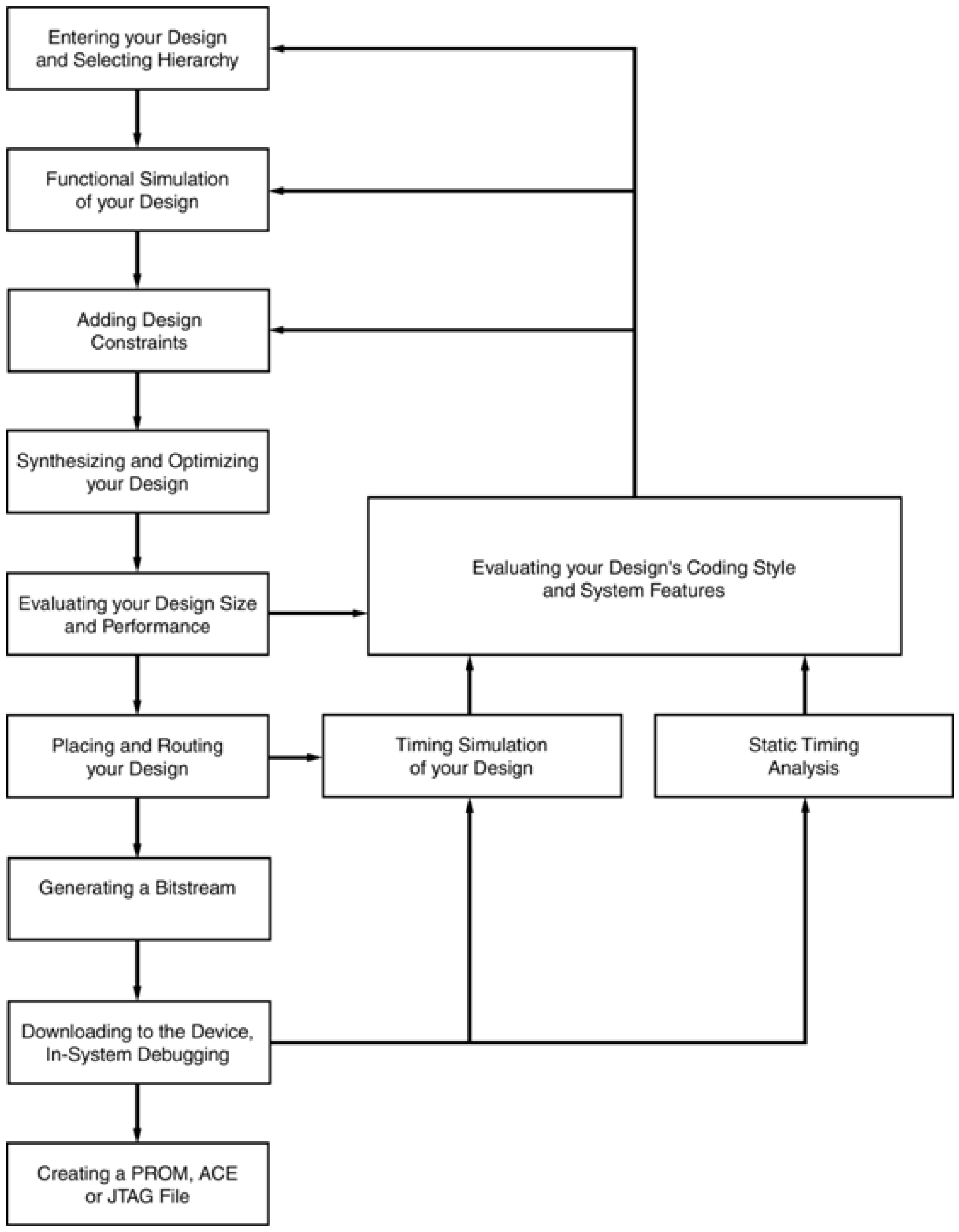

Entering your Design and Selecting Hierarchy
Functional Simulation of your Design
Adding Design Constraints
Synthesizing and Optimizing your Design
Evaluating your Design Size and Performance
Placing and Routing your Design
Generating a Bitstream
Downloading to the Device, In-System Debugging
Creating a PROM, ACE or JTAG File
Evaluating your Design's Coding Style and System Features
Timing Simulation of your Design
Static Timing Analysis

Figure 5.1: FPGA Design Flow [3]

The ultimate goal of a FPGA design is to map the logic onto the limited resources while achieving the desired operating speed. The operating speed is determined actually by the longest combinatorial path in the design, which is also named as *"‘critical path"’* of the design. The critical path dictates the maximum clock frequency at which the logic circuit can operate. The synthesis tool also performs some optimizations during RTL-to-gate level translation, in order to achieve these kind of area and delay constraints as well. The synthesis tool aims to minimize i) area occupied by the logic gates and interconnects, ii) critical path delay of the longest path through logic, and to maximize i) placeability, ii) routability.

The synthesis optimization has two phases; architecture-independent optimization, and architecture-specific optimization. In the architecture-independent optimization phase, the synthesis tool elaborates the RTL design, and identifies logical descriptions of general logic blocks, such as registers, latches, multiplexers, decoders, and finite state machines (FSMs). Combinational logic optimization, FSM encoding/minimization, removal of redundant logic, and determination of sharable logic blocks are performed also in this phase. In the architecture-specific phase, the logical blocks in the design are mapped to the FPGA-specific logic components, such as on-chip multipliers, embedded block memories, clock buffers etc. Although embedded logic components are pre-optimized hardwired blocks, and exploiting these resources in the design results in a more area and delay efficient implementations generally, fixed location of these resources may sometimes limit the placeability and routability. Therefore, the synthesis tool, also applies further optimization and re-mapping algorithms, in the final FPGA mapping and routing phase, to meet the area and timing constraints.

In this thesis, we use Xilinx Virtex-7 FPGA family as the targeting platform, and accordingly use Xilinx's synthesis tool (XST) for logic synthesis. Figures 5.2 to 5.6 show the intermediate steps of a FPGA implementation for an example arbitrary design.

5.1 XILINX FPGA Architecture

Xilinx FPGAs are basically composed of configurable logic blocks (CLBs). CLBs are main logic resources for implementing sequential and as well as combinational circuits [4]. Each CLB is connected to a switch matrix for access to the general routing matrix, as shown in Figure 5.7 [4]. The CLBs are surrounded with interconnects, and the connections between the CLBs are built through the programmable connection boxes (Figure 5.8). This placement is known as the island-style FPGA architecture in the literature [5].

Xilinx CLBs contain two slices, where each slice contains four 6-input LUTs, eight flip-flops as well as multiplexers and arithmetic carry logic. LUTs are main resources for the logical function implementations. 6-input LUT is actually a 64x1 SRAM, where the address of the SRAM is the input literals of the Boolean function to be implemented, and the SRAM content is the truth table corresponding to that Boolean function. 6-input LUTs can implement any Boolean function of up to 6 variables. There are two types of slices on the Xilinx Virtex-7

CLBs, which are SLICEL and SLICEM. SLICEL and SLICEM differ from each other with respect to the LUT structures they contain. While the LUTs in the SLICEM can be reconfigured on the run-time, the LUTs in the SLICEL are configured at the start-up and remain static during the operation. About 1/3 of the total slices on a Virtex-7 FPGA are SLICEM, and they are spread out throughout the FPGA [4]. Note that, the reconfigurable CFGLUT5s, which build the basic infrastructure of our S-DIRECT TCAM architecture, are the LUTs that reside in the SLICEM slices. CLBs are not the only logic resources in the Virtex-7 FPGAs. There are also embedded block memories, digital signal processing elements, clock management modules, clock buffers and high-speed transceivers as well, which make the Xilinx Virtex-7 FPGAs very suitable design targets for system-on-chip (SOC) implementations.

5.2 XILINX Synthesis Technology (XST) Flow

XST (Xilinx Synthesis Technology), which synthesizes VHDL, Verilog, or mixed language designs to create Xilinx-specific netlist files known as NGC files, which contain both logical design data and constraints. While Figure 5.9 shows a general XST flow, Figure 5.10 shows the details of the synthesis flow in three main steps.

During HDL synthesis, XST analyzes the HDL code and attempts to infer specific design building blocks or macros (such as MUXes, RAMs, adders, and subtractors) for which it can create efficient technology implementations. To reduce the amount of inferred macros, XST performs a resource sharing check. This usually leads to a reduction of the area as well as an increase in the clock frequency. During low level optimization, XST transforms inferred macros and general glue logic into a technology-specific implementation [6]. The inferred macros can be stated as follows:

- Carry logic (MUXCY, XORCY, MULT_AND)
- RAM (block or distributed)
- Shift Register LUTs (SRL16, SRL32)
- Clock Buffers (IBUFG, BUFG, BUFGP, BUFR)
- Multiplexers (MUXF5, MUXF6, MUXF7, MUXF8)
- Arithmetic Functions (DSP48, MULT18x18)

In order the XST achieves a high performance, Xilinx recommends two major design practices:

1. Using HDLs as much as possible to describe the logic rather than instantiating specific primitives

2. Selecting the correct design hierarchy

Using HDLs in the description of logic results in a device-independent logic design, and the XST can infer the logical components more suitably than the designer. And also XST can apply optimization algorithms the HDL. Xilinx also provides some pre-optimized, sometimes pre-synthesized IP (Intellectual Property) cores through its proprietary Core Generator (CoreGEN) tool. CoreGEN IPs are generally the logic blocks, that are widely used, such as bock memories, FIFOs, clock managers, and also some sophisticated logic blocks such as Ethernet MAC module, FIR filter, FFT core etc, which are hard to implement by the designer. Selecting the correct design hierarchy is important especially in large-scaled designs. The hierarchical blocks are optimized within themselves and combined at higher level if the hierarchical boundaries are carefully described. The main criterion is the registering the inputs and outputs (IOs) of the hierarchical blocks (flip-flops at the boundaries for all IOs), such that if the clock period constraint is satisfied within a sub-block, its placement and routing can be kept when combining them at higher levels. If the IOs of the hierarchical sub-blocks are not registered, then the design is flattened as if there is a single logic description. A flattened design results in a more complex connectivity graph to be optimized, and the synthesis run time also increases.

5.3 Suggestions for Custom FPGA Features for IP Lookup Implementations

The implementation of S-DIRECT is mainly based on the CFGLUT5 primitives and the primary MUX and DEMUX chains for the priority encoder and input addressing logic respectively. The logical units are well-defined structurally in terms of basic primitives, and there is little HDL description for the peripheral logic. To say in different words, the overall S-DIRECT design is so defined that, there is no optimization (removal of logic or resource sharing) that can be done in the RTL. In the point of hierarchical design view, the S-DIRECT is fully constructed in a hierarchical manner. However, the boundaries of the sub-blocks are not registered, since the full hierarchy constitutes the primary encoder or address decoder, which should be a combinational circuit at the final stage. This is a necessity for the single cycle search capability. The registers are only at the top-level. As the size of the S-DIRECT increases, the depth of the hierarchy also increases, and hence the critical path delay to be optimized increases as well. Since there are no registers at the hierarchical sub-block boundaries, the whole design is evaluated as if it is single flattened design, which causes the XST to handle a larger connection graph as the design size gets larger. Another major issue that hardens the XST synthesis is that the SLICEM slices, which contains the CFGLUT5 primitives are spread around the FPGA. As the number of CFGLUT5s in the design increases, the connection matrix must cover a larger FPGA area, and consumes more interconnection resources. As the design size increases, XST tool spends more time on placing and routing the design. Finally, the last issue is related with high fanouts. For example, the 32-bit long IP address bus should be connected to all TCAM entries, which makes hard routing process even harder. The inconsistencies that make the large-scale S-DIRECT implementation a hard task for the XST can be summarized as follows:

- Based on structural primitive description, rather than HDL description
- CFGLUT5s are spread around the FPGA
- Deep hierarchical structure, where boundaries are not registers
- Long combinational paths
- High fanouts

As a result, although not impossible, large-scale S-DIRECT implementation does not suit the Xilinx FPGA architecture and XST expectations at all. We therefore suggest the following improvements on both FPGA architecture, and the synthesis tool, so that TCAM implementations on the FPGAs are not a burden.

a) Suggestions on FPGA Architecture

1. The main limitation on the FPGA architecture is that the CFGLUT5s are on the SLICEM type slices, and only 1/3 of the total slices are SLICEM type. So, if all slices are of the SLICEM type, the mapping and routing would not be that hard, and also, larger sized TCAM could be implemented.
2. CLBs with SLICEM slices should be gathered close to each other for better placement and routing
3. Due to switch matrix capacity limitation, a further resource may be preferred for routing, although there is a closer resource, but cannot be reached because of the congestion around it. This situation is more likely to occur in our design, which requires routing between a high number of SLICEM resources, which are located far away from each other. Therefore it is suggested that the switch matrix capacity should be increased to avoid congestion.
4. Although the MUXes in the PE design are very simple logic blocks, one slice is consumed for a single 2-to-1 MUX, and the nets cross across CLBs (Figures 5.11 to 5.13). Therefore, the path delays in PE design are higher than expected, which result in a slower speed implementation, since the critical path delay occurs on the PE logic. So, if there would be pre-built embedded PE logic blocks on the FPGA, both the synthesis effort would be alleviated, and a higher operating speed would be achieved.

b) Suggestions on logic synthesis

1. Synthesis tools are manly tailored for synchronous design optimization, and long hierarchical path definitions are not expected. According to the suggested design criteria, combinational paths should be within a sub-block, and there should not be a combinational path, which crosses two hierarchical blocks. The synthesis tools should be gained also optimization algorithms for combinational paths across hierarchical boundaries also. If the boundary register necessity for the incremental design is eliminated, and the combinational logic optimization is performed within the sub-blocks and further across the hierarchical stages, the overall synthesis effort would be decreased. Since the combinational path length (in terms of gate level) is not affected whether there are registers at the boundaries or not, the overall optimization for the combinational path can be done as if there are registers on the boundaries.

2. Resource duplication is a preferred methodology especially for high fanout nets, so that the routing task is tiled into pieces. However, unconscious duplication would result in overuse of interconnection resources, which may also worsen the routing. Therefore, resource duplication should not be left to the designer, and the synthesis tool should duplicate the nets without preventing the interconnection resource sharing.

```
-----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity module_example is
    Port ( clk : in  STD_LOGIC;
           a : in  STD_LOGIC;
                    b : in STD_LOGIC;
                    c : in STD_LOGIC;
                    d : in STD_LOGIC;
                    e : in STD_LOGIC;
                    y : out  STD_LOGIC;
           z : out  STD_LOGIC);
end module_example;

architecture Behavioral of module_example is

signal int1, int2, int3, int4 : std_logic;

begin

int1 <= a AND b;
int2 <= c AND d;
int3 <= int1 OR int2;
int4 <= (NOT int1) AND (NOT int2);

process(clk)
begin
      if rising_edge(clk) then
            if (e='1') then
                  y <= int1 and int2;
                  z <= int2 and int4;
            else
                  y <= int1 or int2;
                  z <= int2 or int4;
            end if;
      end if;
end process;

end Behavioral;
```

Figure 5.2: VHDL description of an example digital logic design

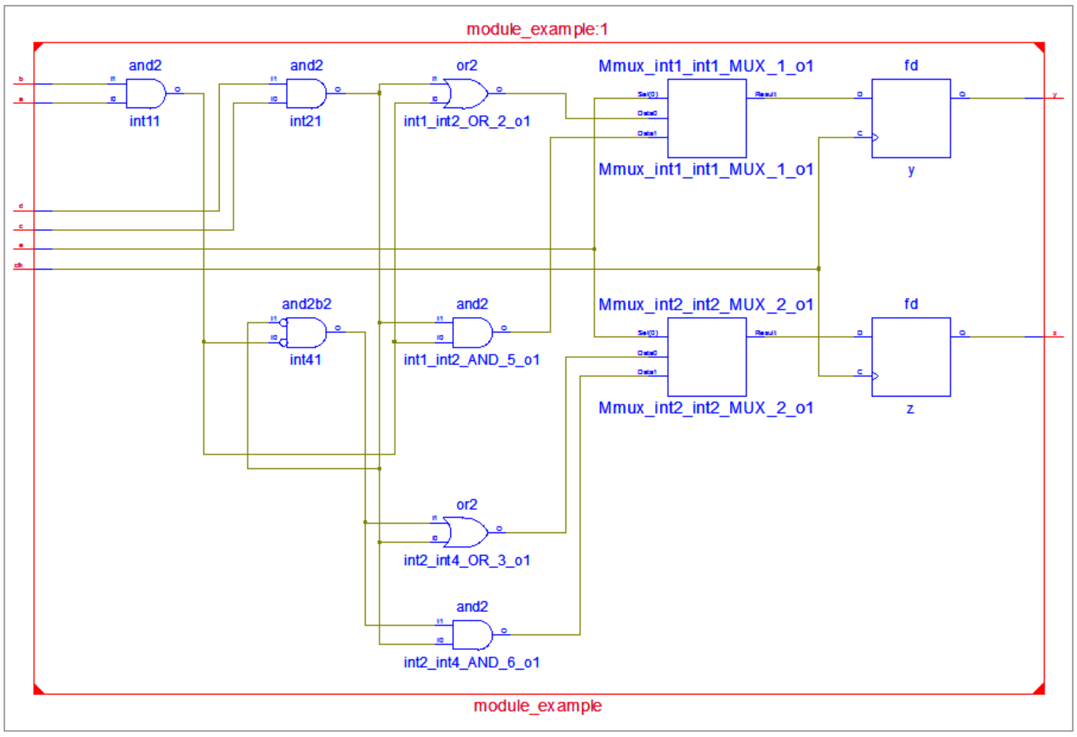


Figure 5.3: RTL synthesis of the example design

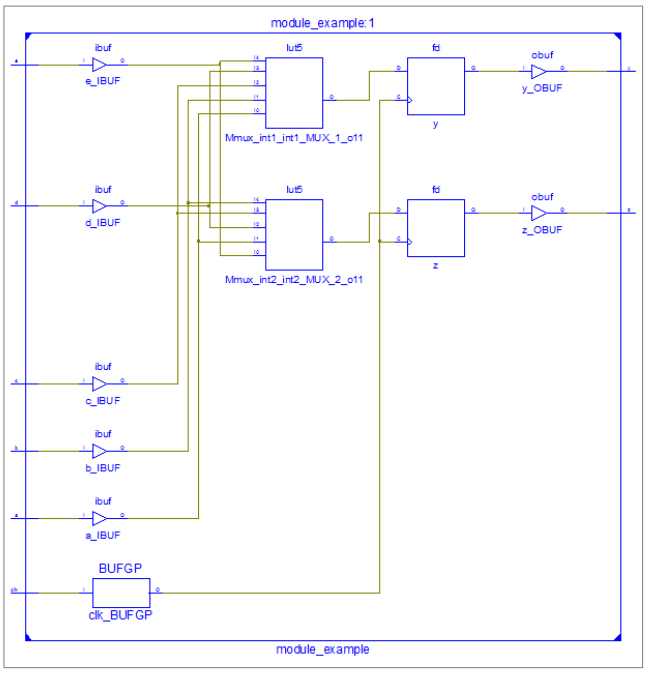


Figure 5.4: Technology mapped (physical) synthesis of the example design

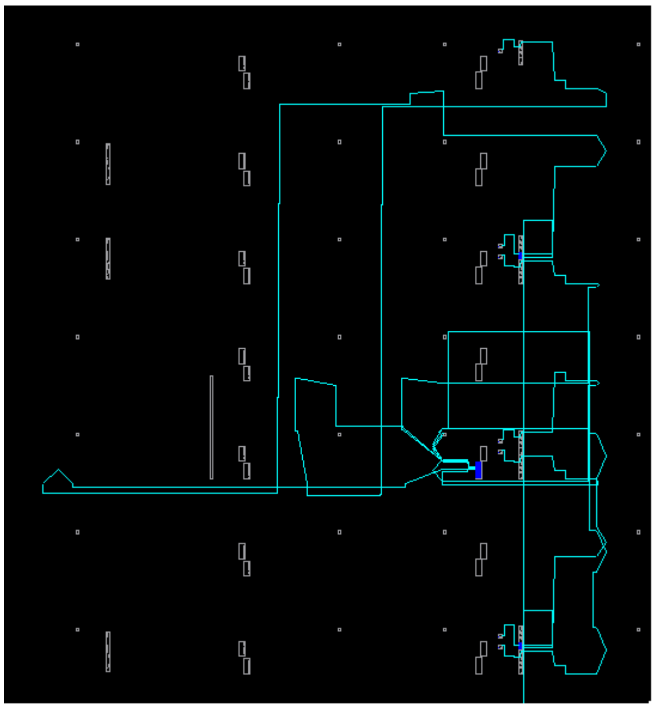


Figure 5.5: Placed and routed design

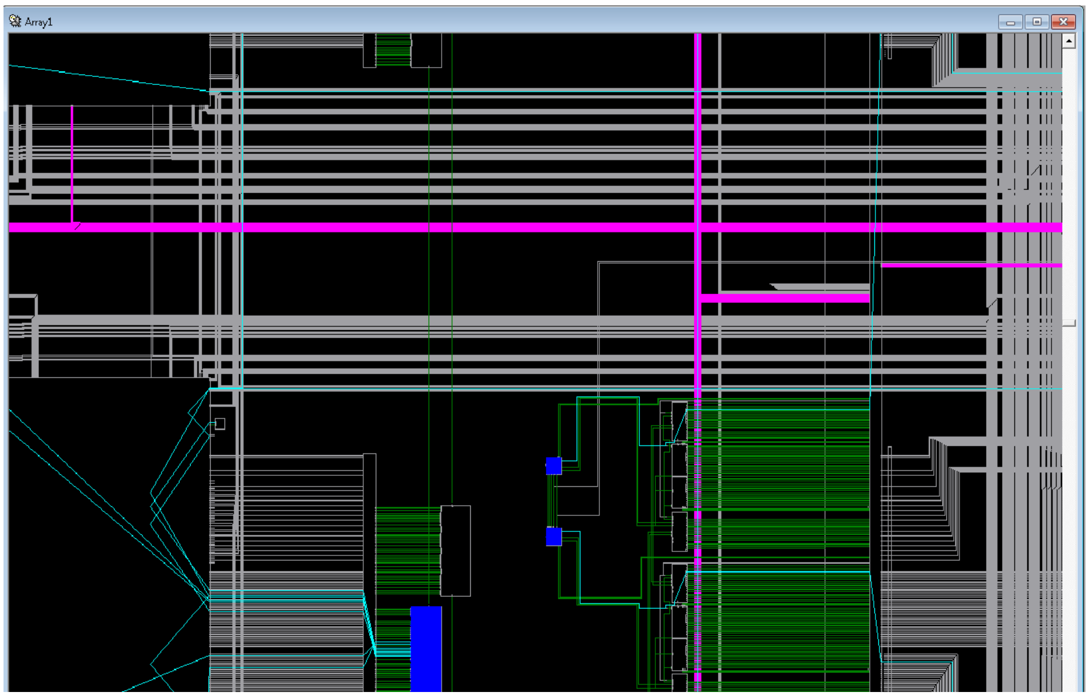


Figure 5.6: Placed and routed design (zoomed view)

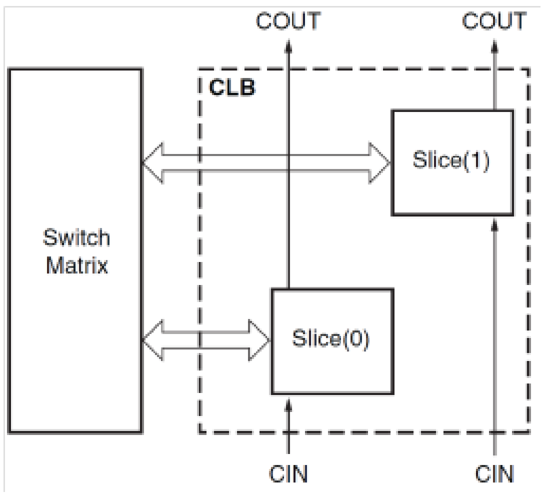


Figure 5.7: Xilinx CLB [4]

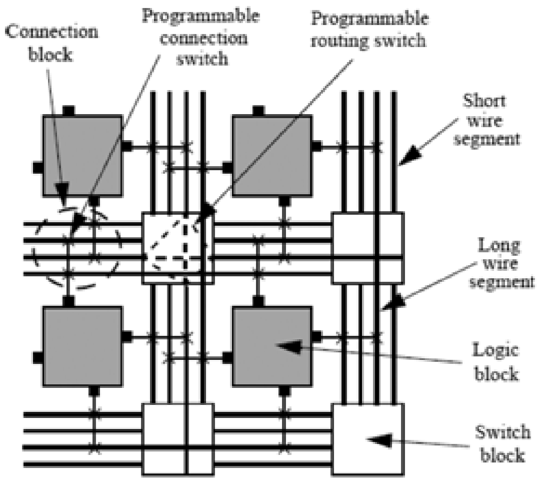


Figure 5.8: Island-style FPGA architecture [5]

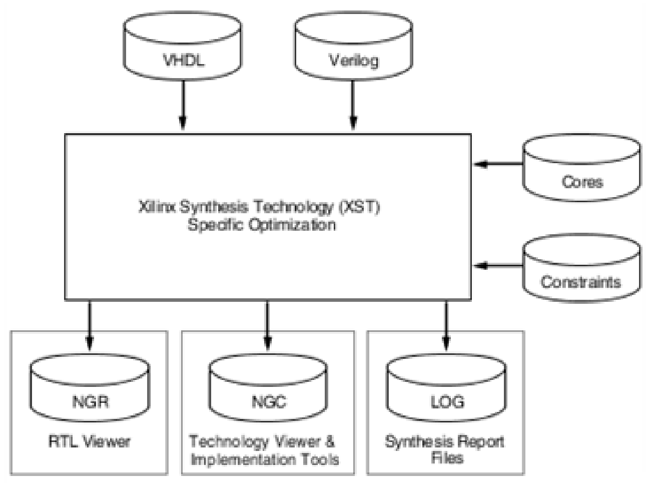


Figure 5.9: General XST Flow [6]

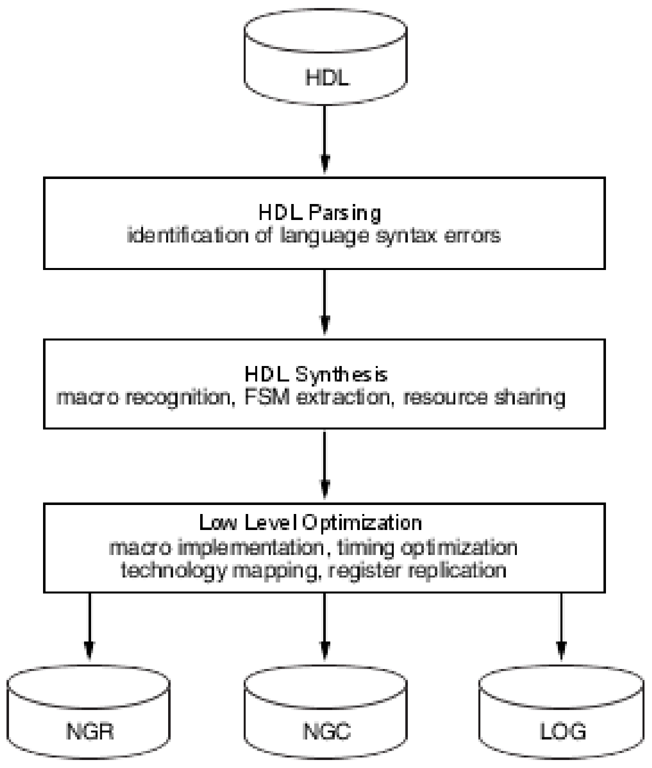

HDL
HDL Parsing
identification of language syntax errors
HDL Synthesis
macro recognition, FSM extraction, resource sharing
Low Level Optimization
macro implementation, timing optimization
technology mapping, register replication
NGR
NGC
LOG

Figure 5.10: XST Synthesis Steps [6]

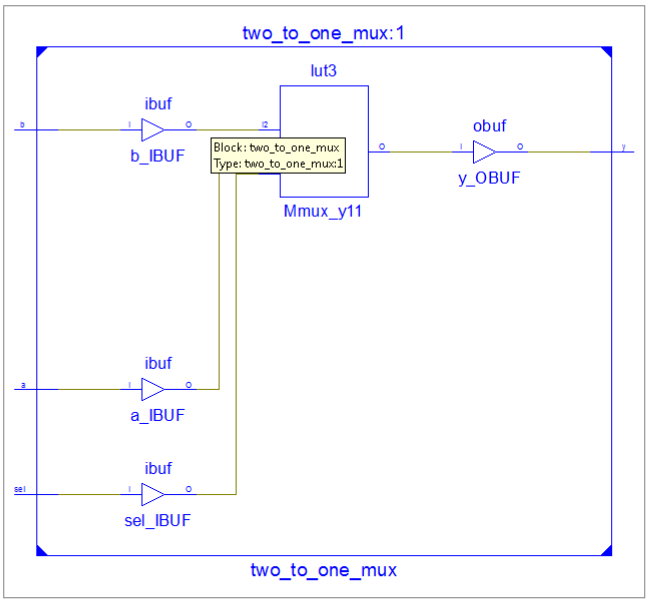


Figure 5.11: Technology mapping of a 2-to-1 MUX on Xilinx FPGA

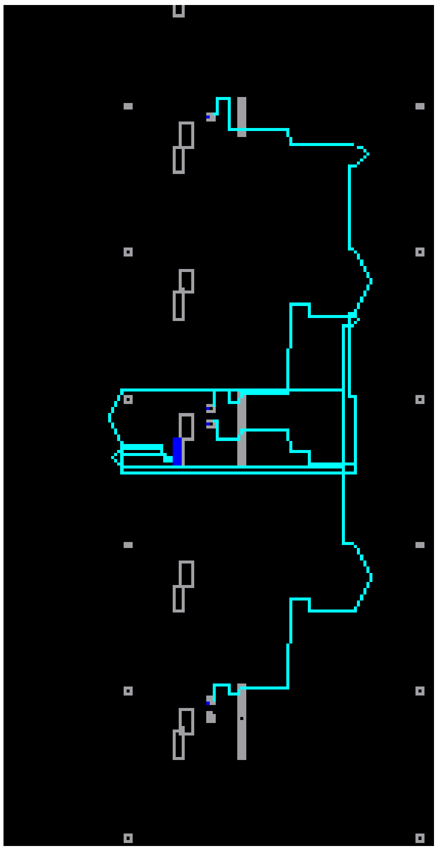


Figure 5.12: Placed and routed 2-to-1 MUX

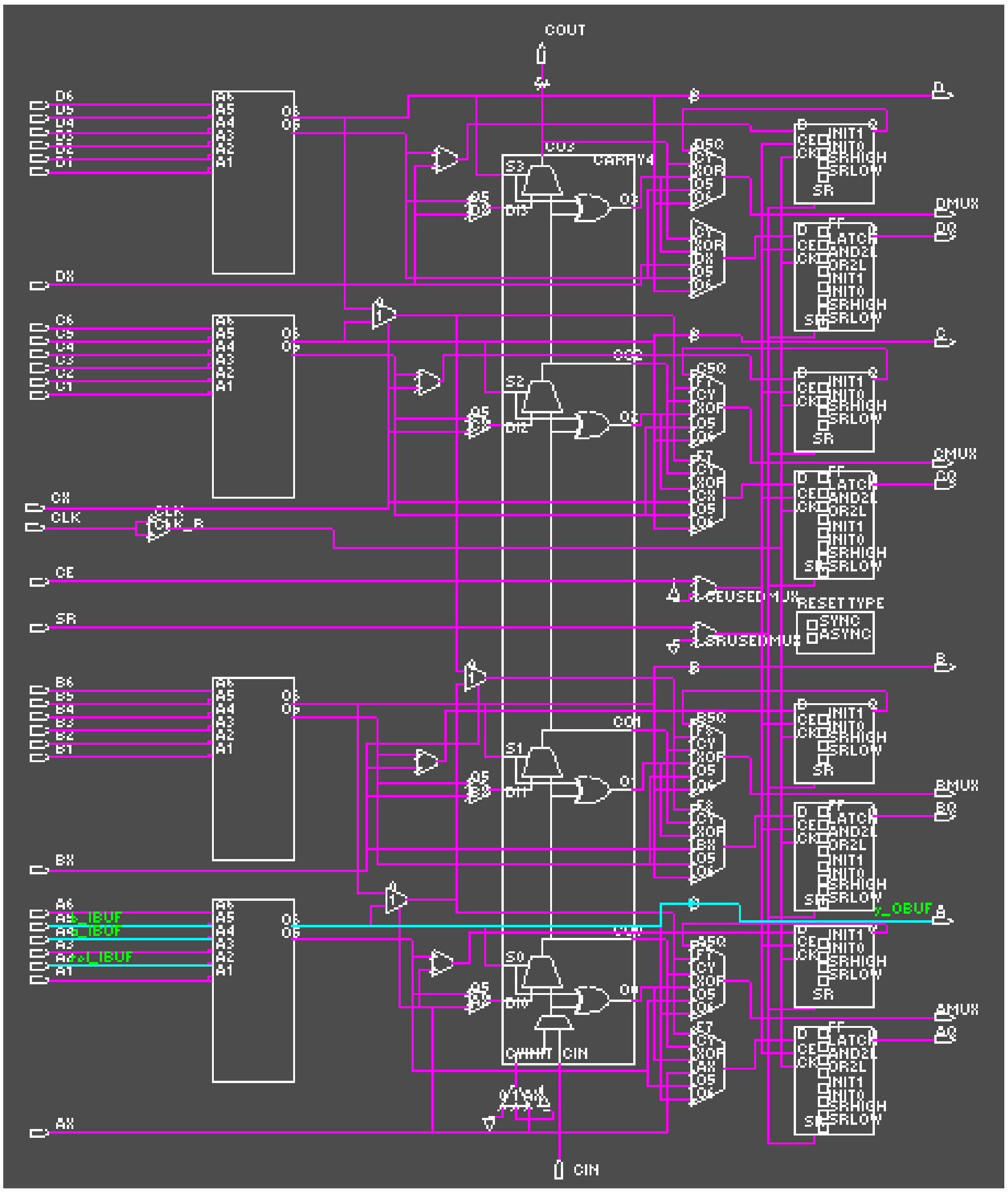


Figure 5.13: The inside view of the CLB slice where the 2-to-1 MUX mapped

CHAPTER 6

CONCLUSIONS

The objective of this thesis is the design and implementation of IP look up architectures with design scalability and non-blocking update capability where the updates are carried out entirely on hardware. SRAM-based and TCAM-based architectures are the two main IP lookup approaches that are implemented on hardware. The researchers, who utilize SRAM-based architectures in their solutions, basically criticize the high power consumption of TCAMs to justify their solutions. SRAM-based algorithms utilize tree-based data structures and algorithms for routing table management. A tree search requires several memory accesses to get LPM, hence, pipelined tree structures are used to increase the throughput. Pipelined architectures require several SRAMs of different sizes, and therefore implementing pipelined structures is only feasible on FPGA platform. There are studies promoting tree-based pipelined IP lookup algorithms claiming low power designs on FPGA [40][77][42][78][79]. These studies state that their motivation for the the pipelined tree-based SRAM architectures is that the TCAMs are power hungry.

However we should note that the FPGA's power consumption is not mentioned in these studies. In [92] it is stated that an FPGA consumes 14 times more power than its ASIC counterpart for logic-and-memory only implementations.

TCAMs have single cycle search capability. Moreover, updating the forwarding table in TCAM-based schemes is generally simpler than that in trie-based algorithms [67]. We further improve the update capability on the TCAM by proposing a novel architecture for TCAMs. Our architecture adds the non-blocking update capability to the standard TCAMs, and with the added length cell field to the basic TCAM entry, determining the address to update can directly be performed on the hardware without requiring a software support.

We propose a TCAM architecture, which differs from the TCAM architectures in the literature in the following ways:

1. Embedded priority encoder: Single cycle search also includes priority encoding process
2. Non-blocking update: Write process on any entry do not block the search on the remaining entries, simultaneous update and search operations can be performed

3. Update on the hardware: No additional software support is required for processing the update request to determine the address of the TCAM entry where the update will occur

We feature S-DIRECT together with an SRAM block in order to generate a hybrid IP lookup solution for a real-life routing table that we call SHIP. SRAM is used as a DMA engine for prefixes of length smaller than or equal to 24-bit, so that it gives also match result in single cycle, compliant with the TCAM operation. We implemented both S-DIRECT and SRAM block on a high-end FPGA (Xilinx Virtex-7 1140T). SRAM implementation on the FPGA is performed by instantiating the dual-port BRAM components which are already embedded in the FPGA.

Our main contribution with the proposed SHIP architecture is that routing table updates are fully processed and realized on hardware, which is not accomplished by any hardware-based IP lookup solution previously. Furthermore, the updates are performed without blocking the ongoing IP lookup processes. Besides these contributions, our solution also satisfies major IP lookup performance requirements in the following ways:

- The proposed architecture is scalable to larger routing table sizes
- We can support more than 16M prefixes on a simple FPGA-memory combination
- IP lookup is performed in one clock cycle, independent of the number of prefixes in the table, hence it gives the lowest latency among the solutions
- A lookup throughput of 153 MLPS can be achieved on the FPGA, which is more than the requirement for line rate operation of OC-768, 40 Gbps link.

The proposed architectures can be implemented on any hardware platform provided that they satisfy the architectural requirements stated in the thesis. While performing FPGA implementations of our architectures, we not only demonstrated the viability of our proposals, but also provided the efficient way of implementations of our architectures on the FPGA platform.

For the FPGA implementation phase, we investigated the inconsistencies of the FPGA architecture and synthesis tool for our design, and proposed some suggestions for both FPGA architecture and the synthesis tool for better FPGA utilization. We believe that, if the number of run-time reconfigurable logic blocks is increased and/or they are placed close to each other on the FPGA, the path delays would be smaller, resulting in a higher operating frequency, and hence in a higher lookup throughput.

The operating frequency of our design is limited by the long combinational path delays in the PE design of the S-DIRECT. However, this is inevitable when the single cycle search capability is desired. As the number of match lines (TCAM size) increases, the number of logic levels in the PE design also increases, and the achievable clock frequency decreases.

The BRAM resources on the contemporary FPGAs did not allow us to build a single-chip solution with our proposed hybrid DMA and TCAM-based architectures. However, we believe that in the near future this inefficiency will be eliminated, and our architecture can be implemented as a single-chip solution as well.

S-DIRECT architecture can also be easily scaled to IPv6 addressing scheme by employing 128-bit for prefix cells, and 7 bits for length cells, keeping the remaining logic as it is. Actually, the global unicast IP address of the IPv6 assigns most significant 64 bits to network ID, and the remaining 64 bits to Interface ID. Therefore, for the practical application, the required number of bits for the prefix cell and length cell is 64 and 6 respectively. A current study [93] also investigates the prefix distribution of the contemporary IPv6 routing tables, and observes that over 80% of the prefixes are either 32, 48 or 64 bit length, and propose a hash-based algorithm for IPv6 address lookup. Note that, the TCAM architectures are the natural hash tables where no collision occurs, and hence compliant with the [93], the S-DIRECT architecture proposed in this thesis study can be used in IPv6 routers as well.

The DMA-based direct lookup solution that we employed in SHIP architecture can also be used in IPv6 in a similar fashion that we used in IPv4. However, the maximum prefix length that should be looked up in the DMA engine strictly determines the practical feasibility of the DMA engine, and therefore for the time being it is more likely to be used for prefixes of smaller than or equal to 24-bit length. With the improvement of technology this limitation may also be eliminated in the future.

As a future work, our design can be revisited for implementing more hardware efficient algorithms to the cost of more complex update management algorithms on the hardware. And also an ASIC implementation of the proposed S-DIRECT architecture would be another future work in order to investigate timing and area utilization compared to conventional TCAM architectures.

REFERENCES

[1] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of ip address lookup algorithms," *Network, IEEE*, vol. 15, no. 2, pp. 8 –23, mar/apr 2001.

[2] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: a tutorial and survey," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 3, pp. 712 – 727, march 2006.

[3] (2013) Xilinx synthesis and simulation design guide (ug626). [last accessed on 15/08/2013]. [Online]. Available: www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/sim.pdf

[4] (2013) Xilinx ug474 7 series fpgas configurable logic block user guide. [last accessed on 15/08/2013]. [Online]. Available: www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[5] D. Chen, J. Cong, Y. Fan, and L. Wan, "Lopass: A low-power architectural synthesis system for fpgas with interconnect estimation and optimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 4, pp. 564–577, 2010.

[6] (2013) Xst overview. [last accessed on 15/08/2013]. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_using_xst_for_synthesis.htm

[7] (2013) Active bgp entries. [last accessed on 11/04/2013]. [Online]. Available: http://bgp.potaroo.net/as2.0/bgp-active.html

[8] H. Lim and N. Lee, "Survey and proposal on binary search algorithms for longest prefix match," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 3, pp. 681–697, 2012.

[9] (2013) Bgp update reports. [last accessed on 11/04/2013]. [Online]. Available: http://bgp.potaroo.net/index-upd.html

[10] L. Luo, G. Xie, Y. Xie, L. Mathy, and K. Salamatian, "A hybrid ip lookup architecture with fast updates," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 2435–2443.

[11] P. Gupta and N. McKeown, "Algorithms for packet classification," *Network, IEEE*, vol. 15, no. 2, pp. 24 –32, mar/apr 2001.

[12] (2013) Virtex-7 fpga families. [last accessed on 15/08/2013]. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm

[13] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed ip routing lookups," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 4, pp. 25–36, Oct. 1997.

[14] R. Sangireddy, N. Futamura, S. Aluru, and A. Somani, "Scalable, memory efficient, high-speed ip lookup algorithms," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 4, pp. 802–812, 2005.

[15] N. Futamura, R. Sangireddy, S. Aluru, and A. Somani, "Scalable, memory efficient, high-speed lookup and update algorithms for ip routing," in *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, 2003, pp. 257–263.

[16] J. H. Mun, H. Lim, and C. Yim, "Binary search on prefix lengths for ip address lookup," *Communications Letters, IEEE*, vol. 10, no. 6, pp. 492–494, 2006.

[17] K. S. Kim and S. Sahni, "Ip lookup by binary search on prefix length," in *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, 2003, pp. 77–82 vol.1.

[18] J. Huang, "Network processor design," in *ASIC, 2003. Proceedings. 5th International Conference on*, vol. 1, 2003, pp. 26–33 Vol.1.

[19] M. Ahmadi and S. Wong, "Network processors: Challenges and trends," in *In Proceedings of the 17th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2006*, 2006, pp. 223–232.

[20] N. Shah, "Understanding Network Processors," Master's thesis, Berkeley University, USA, 2001.

[21] P. C. Lekkas, *Network Processors Architectures, Protocols, and Platforms*. USA: McGraw-Hill, 2003.

[22] R. Giladi, *Network Processors Architecture, Programming, and Implementation*. USA: The Morgan Kaufman Series in Systems on Silicon, 2008.

[23] M. Gries, "Algorithm-Architecture Trade-offs in Network Processor Design," Ph.D. dissertation, Swiss Federal Institute of Technology, Switzerland, 2001.

[24] F. Khunjush, M. Watheq El-Kharashi, K. Li, and N. Dimopoulos, "Network processor design: issues and challenges," in *Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on*, vol. 1, 2003, pp. 164–168 vol.1.

[25] K. Yi and J.-L. Gaudiot, "Features of future network processor architectures," in *Modern Computing, 2006. JVA '06. IEEE John Vincent Atanasoff 2006 International Symposium on*, 2006, pp. 69–76.

[26] J. Fu and O. Hagsand, "Designing and evaluating network processor applications," in *High Performance Switching and Routing, 2005. HPSR. 2005 Workshop on*, 2005, pp. 142–146.

[27] L. Hammond, B. A. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, 1997.

[28] O. Menzilcioglu and S. Schlick, "Nectar cab: a high-speed network processor," in *Distributed Computing Systems, 1991., 11th International Conference on*, 1991, pp. 508–515.

[29] H. Ghasemi, H. Mohammadi, B. Robatmili, and N. Yazdani, "Augmenting general purpose processors for network processing," in *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 416–419.

[30] K. Yi and J.-L. Gaudiot, “Architectural support for network applications on simultaneous multithreading processors,” in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–10.

[31] C. Wu, X. Shi, X. Yang, and J. Su, “The impact of parallel and multithread mechanism on network processor performance,” in *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference*, 2006, pp. 236–240.

[32] C. Albrecht, J. Foag, R. Koch, and E. Maehle, “Dynacore: A dynamically reconfigurable coprocessor architecture for network processors,” in *Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on*, 2006, pp. 101–108.

[33] M. Meitinger, R. Ohlendorf, T. Wild, and A. Herkersdorf, “Flexpath np - a network processor architecture with flexible processing paths,” in *System-on-Chip, 2008. SOC 2008. International Symposium on*, 2008, pp. 1–6.

[34] T. Li, X. ming Zhang, and Z. gang Sun, “Dynanp - a coarse-grain dataflow network processor architecture with dynamic configurable processing path,” in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, vol. 3, 2007, pp. 182–187.

[35] C. Kachris, S. Wong, and S. Vassiliadis, “Design and performance evaluation of an adaptive fpga for network applications,” *Microelectron. J.*, vol. 40, no. 7, pp. 1103–1110, Jul. 2009.

[36] S. Govind, R. Govindarajan, and J. Kuri, “Packet reordering in network processors,” in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–10.

[37] J. Guo, J. Yao, and L. Bhuyan, “An efficient packet scheduling algorithm in network processors,” in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, 2005, pp. 807–818 vol. 2.

[38] K. McLaughlin and S. Sezer, “High-speed ip address lookups using hardware based tree structures,” in *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, 2007, pp. 625–632.

[39] D. Bemmann, “Ip lookup on a platform fpga: A comparative study,” in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS’05) - Workshop 3 - Volume 04*, ser. IPDPS ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 166.1–.

[40] H. Le, W. Jiang, and V. Prasanna, “A sram-based architecture for trie-based ip lookup using fpga,” in *Field-Programmable Custom Computing Machines, 2008. FCCM ’08. 16th International Symposium on*, april 2008, pp. 33 –42.

[41] V. Srinivasan and G. Varghese, “Fast address lookups using controlled prefix expansion,” *ACM Trans. Comput. Syst.*, vol. 17, no. 1, pp. 1–40, Feb. 1999.

[42] H. Le and V. Prasanna, “Scalable high throughput and power efficient ip-lookup on fpga,” in *Field Programmable Custom Computing Machines, 2009. FCCM ’09. 17th IEEE Symposium on*, april 2009, pp. 167 –174.

[43] H. Fadishei, M. S. Zamani, and M. Sabaei, “A novel reconfigurable hardware architecture for ip address lookup,” in *Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, ser. ANCS ’05. New York, NY, USA: ACM, 2005, pp. 81–90.

[44] O. Erdem and C. F. Bazlamaçci, “High-performance ip lookup engine with compact clustered trie search,” *The Computer Journal*, 2012.

[45] L. Chisvin and R. Duckworth, “Content-addressable and associative memory: alternatives to the ubiquitous ram,” *Computer*, vol. 22, no. 7, pp. 51–64, 1989.

[46] H. Song and J. W. Lockwood, “Efficient packet classification for network intrusion detection using fpga,” in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, ser. FPGA ’05. New York, NY, USA: ACM, 2005, pp. 238–245.

[47] E. Spitznagel, D. Taylor, and J. Turner, “Packet classification using extended tcams,” in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, nov. 2003, pp. 120 – 131.

[48] G. Jedhe, A. Ramamoorthy, and K. Varghese, “A scalable high throughput firewall in fpga,” in *Field-Programmable Custom Computing Machines, 2008. FCCM ’08. 16th International Symposium on*, april 2008, pp. 43 –52.

[49] K. McLaughlin, N. O’Connor, and S. Sezer, “Exploring cam design for network processing using fpga technology,” in *Telecommunications, 2006. AICT-ICIW ’06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, feb. 2006, p. 84.

[50] A. A. McEwan and J. Saul, “A high speed reconfigurable firewall based on parameterizable fpga-based content addressable memories,” *J. Supercomput.*, vol. 19, no. 1, pp. 93–103, may 2001.

[51] S. Guccione, D. Levi, and D. Downs, “A reconfigurable content addressable memory,” in *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, ser. IPDPS ’00. London, UK, UK: Springer-Verlag, 2000, pp. 882–889.

[52] S. Guccione, D. Levi, and P. Sundararajan, “Jbits: Java based interface for reconfigurable computing,” 1999.

[53] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, “Granidt: Towards gigabit rate network intrusion detection technology,” in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, ser. FPL ’02. London, UK, UK: Springer-Verlag, 2002, pp. 404–413.

[54] L. Ionescu, A. Mazare, G. Serban, V. Barbu, and A. Constantin, “Fpga implementation of an associative content addressable memory,” in *Applied Electronics (AE), 2011 International Conference on*, sept. 2011, pp. 1 –4.

[55] Z. Ullah, K. Ilgon, and S. Baeg, “Hybrid partitioned sram-based ternary content addressable memory,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. PP, no. 99, p. 1, december 2012.

[56] V. Ravikumar and R. Mahapatra, "Tcam architecture for ip lookup using prefix properties," *Micro, IEEE*, vol. 24, no. 2, pp. 60 – 69, mar-apr 2004.

[57] K. Pagiamtzis and A. Sheikholeslami, "A low-power content-addressable memory (cam) using pipelined hierarchical search scheme," *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 9, pp. 1512–1519, 2004.

[58] H. Noda, K. Inoue, M. Kuroiwa, F. Igaue, K. Yamamoto, H. Mattausch, T. Koide, A. Amo, A. Hachisuka, S. Soeda, I. Hayashi, F. Morishita, K. Dosaka, K. Arimoto, K. Fujishima, K. Anami, and T. Yoshihara, "A cost-efficient high-performance dynamic tcam with pipelined hierarchical searching and shift redundancy architecture," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 1, pp. 245–253, 2005.

[59] S. Baeg, "Low-power ternary content-addressable memory design using a segmented match line," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 55, no. 6, pp. 1485–1494, 2008.

[60] S. Kasnavi, V. Gaudet, P. Berube, and J. Amaral, "A hardware-based longest prefix matching scheme for tcams," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, may 2005, pp. 3339 – 3342 Vol. 4.

[61] Y.-J. Chang, "A high-performance and energy-efficient tcam design for ip-address lookup," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, no. 6, pp. 479 –483, june 2009.

[62] N. Mohan, W. Fung, D. Wright, and M. Sachdev, "A low-power ternary cam with positive-feedback match-line sense amplifiers," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 3, pp. 566–573, 2009.

[63] Y.-J. Chang, K.-L. Tsai, and H.-J. Tsai, "Low leakage tcam for ip lookup using two-side self-gating," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 6, pp. 1478–1486, 2013.

[64] A.-T. Do, S. Chen, Z.-H. Kong, and K. S. Yeo, "A high speed low power cam with a parity bit and power-gated ml sensing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 151–156, 2013.

[65] V. Ravikumar, R. Mahapatra, and L. N. Bhuyan, "Easecam: an energy and storage efficient tcam-based router architecture for ip lookup," *Computers, IEEE Transactions on*, vol. 54, no. 5, pp. 521 – 533, may 2005.

[66] M. Akhbarizadeh, M. Nourani, D. Vijayasarathi, and P. Balsara, "A nonredundant ternary cam circuit for network search engines," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 3, pp. 268–278, 2006.

[67] K. Zheng, C. Hu, H. Liu, and B. Liu, "An ultra high throughput and power efficient tcam-based ip lookup engine," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3, march 2004, pp. 1984 –1994 vol.3.

[68] K. Zheng, C. Hu, H. Lu, and B. Liu, "A tcam-based distributed parallel ip lookup scheme and performance analysis," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 4, pp. 863 –875, aug. 2006.

[69] Z. Cai, Z. Wang, K. Zheng, and J. Cao, “A distributed tcam coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering,” *Computers, IEEE Transactions on*, vol. 62, no. 3, pp. 417–427, 2013.

[70] T. Mishra and S. Sahni, “Petcam a power efficient tcam architecture for forwarding tables,” *Computers, IEEE Transactions on*, vol. 61, no. 1, pp. 3–17, 2012.

[71] N. Mohan, W. Fung, D. Wright, and M. Sachdev, “Design techniques and test methodology for low-power tcams,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 6, pp. 573 –586, june 2006.

[72] D. Shah and P. Gupta, “Fast updating algorithms for tcam,” *Micro, IEEE*, vol. 21, no. 1, pp. 36–47, 2001.

[73] J. Jaeger, “FPGA-based prototyping grows up,” *Electronic Engineering Times*, no. 518, 2008.

[74] B. Kirk, “FPGA-prototyping and ASIC-conversion considerations,” *EDN*, vol. 52, no. 21, pp. 67 –70, 2007.

[75] (2013) Cisco field programmable device upgrades. [last accessed on 15/08/2013]. [Online]. Available: http://www.cisco.com/en/US/docs/routers/7200/configuration/feature_guides/fpd.html

[76] D. Taylor, J. Lockwood, T. Sproull, J. Turner, and D. Parlour, “Scalable ip lookup for programmable routers,” in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2002, pp. 562 – 571 vol.2.

[77] H. Le, W. Jiang, and V. Prasanna, “Scalable high-throughput sram-based architecture for ip-lookup using fpga,” in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, sept. 2008, pp. 137 –142.

[78] H. Le and V. Prasanna, “High-throughput ip-lookup supporting dynamic routing tables using fpga,” in *Field-Programmable Technology (FPT), 2010 International Conference on*, dec. 2010, pp. 287 –290.

[79] Y.-H. E. Yang and V. K. Prasanna, “High throughput and large capacity pipelined dynamic search tree on fpga,” in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA ’10. New York, NY, USA: ACM, 2010, pp. 83–92.

[80] W. Fung and M. Sachdev, “High performance priority encoder for content addressable memories,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, may 2004.

[81] (2013) Ise design suite. [last accessed on 15/08/2013]. [Online]. Available: http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm

[82] (2013) Xilinx power estimator. [last accessed on 15/08/2013]. [Online]. Available: http://www.xilinx.com/products/design_tools/logic_design/xpe.htm

[83] (2013) Xilinx 7 series fpga libraries guide for hdl designs. [last accessed on 15/08/2013]. [Online]. Available: www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/7series_hdl.pdf

[84] (2013) Modelsim - advanced simulation and debugging. [last accessed on 15/08/2013]. [Online]. Available: http://www.mentor.com/products/fpga/model

[85] (2013) Renesas 20 mbit quad-search content addressable memory. [last accessed on 15/08/2013]. [Online]. Available: http://www.renesas.com/media/products/memory/TCAM/r10pf0001eu0100_tcam.pdf

[86] J. Zhao, X. Zhang, X. Wang, Y. Deng, and X. Fu, "Exploiting graphics processors for high-performance ip lookup in software routers," in *INFOCOM, 2011 Proceedings IEEE*, april 2011, pp. 301 –305.

[87] (2013) Cy7c1625kv18-333bzxc data sheet. [last accessed on 15/08/2013]. [Online]. Available: http://www.cypress.com/?docID=40501

[88] P. Coussy and A. Morawiec, *High-Level Synthesis: from Algorithm to Digital Circuit*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[89] E. Casseau, L. Gal, P. Bomel, C. Jego, S. Huet, and E. Martin, "C-based rapid prototyping for digital signal processing," in *in Proc. of the 13th European Signal Processing Conference (EUSIPCO*, 2005.

[90] D. Chen, J. Cong, and P. Pan, "Fpga design automation: A survey," *Found. Trends Electron. Des. Autom.*, vol. 1, no. 3, pp. 139–169, Jan. 2006.

[91] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 473–491, 2011.

[92] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 203–215, 2007.

[93] Y.-M. Hsiao, Y.-S. Chu, J.-F. Lee, and J.-S. Wang, "A high-throughput and high-capacity ipv6 routing lookup system," *Computer Networks*, vol. 57, no. 3, pp. 782–794, 2013, cited By (since 1996)0.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Ayyıldız, Nizam
Nationality: Turkish (TC)
Date and Place of Birth: 05.05.1980, Sivas
Marital Status: Married
Phone: +905333489098

EDUCATION

Degree	**Institution**	**Year of Graduation**
M.S.	METU/Electrical and Electronics Engineering	2006
B.S.	METU/Electrical and Electronics Engineering	2003
High School	Sivas Selçuk Anadolu Lisesi	1998

PROFESSIONAL EXPERIENCE

Year	**Place**	**Enrollment**
2003 November-	ASELSAN Inc.	Digital Design Engineer
2003 June-2003 November	TUBITAK / SAGE	Researcher

PUBLICATIONS

Ayyildiz, N., Schmidt, E.G., Güran, H., "S-DIRECT : Scalable and Dynamically Reconfigurable TCAM Architecture for High-speed IP Lookup", *Circuits and Systems I: Regular Papers, IEEE Transactions on*, (Submitted in January 2013, currently under third revision)

HOBBIES

Swimming, Cycling, Running, Camping, Music