UNIVERSITÉ DE MONTRÉAL

A SCALABLE HIGH-PERFORMANCE MEMORY-LESS IP ADDRESS LOOKUP ENGINE SUITABLE FOR FPGA IMPLEMENTATION

IDEH SARBISHEI

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

NOVEMBRE 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

A SCALABLE HIGH-PERFORMANCE MEMORY-LESS IP ADDRESS LOOKUP ENGINE SUITABLE FOR FPGA IMPLEMENTATION

présenté par : SARBISHEI Ideh

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

M. GIOVANNI Beltrame, Ph. D., président

M. LANGLOIS J.M. Pierre, Ph. D., membre et directeur de recherche

M. SAVARIA Yvon, Ph. D., membre et codirecteur de recherche

Mme NICOLESCU Gabriela, Doctorat, membre

DEDICATION

To my parents, for their everlasting love and support.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Prof. Pierre Langlois for his continuous support, guidance and immense knowledge during the development of this work. I have been extremely lucky to have an advisor whose office door was always open whenever I ran into a problem or had a question. I could not have imagined a better advisor and mentor for my Master study. I would like to express my gratitude to my thesis co-advisor Prof. Yvon Savaria for his insight and useful comments that were very beneficial in my thesis completion. I would also like to thank Shervin Vakili who was directly involved with many aspects of Chapter 4 and helped me in my supervisor's absence. I thank my fellow lab mates in LASNEP group, for their stimulating discussions and encouragements. I must express my eternal gratitude to my parents and my brother, who are my source of inspiration. Whatever I have in my life is because of their support and unconditional love. Finally, to my boyfriend, Ali, I wish to offer my deepest thanks. He was always there for me in many moments of crisis. He supported me from thousands of kilometers distance and made this work much easier for me.

RÉSUMÉ

La recherche d'adresse IP est une opération très importante pour les routeurs Internet modernes. De nombreuses approches dans la littérature ont été proposées pour réaliser des moteurs de recherche d'adresse IP (*Address Lookup Engine* – ALE), à haute performance. Les ALE existants peuvent être classés dans l'une ou l'autre de trois catégories basées sur: les mémoires ternaires adressables par le contenu (TCAM), les Trie et les émulations de TCAM. Les approches qui se basent sur des TCAM sont coûteuses et elles consomment beaucoup d'énergie. Les techniques qui exploitent les Trie ont une latence non déterministe qui nécessitent généralement des accès à une mémoire externe. Les techniques qui exploitent des émulations de TCAM combinent généralement des TCAM avec des circuits à faible coût. Dans ce mémoire, l'objectif principal est de proposer une architecture d'ALE qui permet la recherche rapide d'adresses IP et qui apporte une solution aux principales lacunes des techniques basées sur des TCAM et sur des Trie.

Atteindre une vitesse de traitement suffisante dans l'ALE est un aspect important. Des accélérateurs matériels ont été adoptés pour obtenir une le résultat de recherche à haute vitesse. Le FPGA permettent la mise en œuvre d'accélérateurs matériels reconfigurables spécialisés. Cinq architectures d'ALE de type émulation de TCAM sont proposés dans ce mémoire : une sérielle, une parallèle, une architecture dite IP-Split, une variante appelée IP-Split-Bucket et une version de l'IP-Split-Bucket qui supporte les mises à jours. Chaque architecture est construite à partir de l'architecture précédente de manière progressive dans le but d'en améliorer les performances.

L'architecture sérielle utilise des mémoires pour stocker la table d'adresses de transmission et un comparateur pour effectuer une recherche sérielle sur les entrées. L'architecture parallèle stocke les entrées de la table dans les ressources logiques d'un FPGA, et elle emploie une recherche parallèle en utilisant N comparateurs pour une table avec N entrées. L'architecture IP-Split emploie un niveau de décodeurs pour éviter des comparaisons répétitives dans les entrées équivalentes de la table. L'architecture IP-Split-Bucket est une version améliorée de l'architecture précédente qui utilise une méthode de partitionnement visant à optimiser l'architecture IP-Split. L'IP-Split-Bucket qui supporte les mises à jour est la dernière architecture proposée. Elle soutient la mise à jour et la recherche à haute vitesse d'adresses IP. Les résultats d'implémentations montrent que l'architecture d'ALE qui offre les meilleures performances est l'IP-Split-Bucket,

qui n’a pas recours à une ou plusieurs mémoires. Pour une table d’adresses de transmission IPv4 réelle comportant 524 k préfixes, l'architecture IP-Split-Bucket atteint un débit de 103,4 M paquets par seconde et elle consomme respectivement 23% et 22% des tables de conversion (LUTs) et des bascules (FFs) sur une puce Xilinx XC7V2000T.

ABSTRACT

High-performance IP address lookup is highly demanded for modern Internet routers. Many approaches in the literature describe a special purpose Address Lookup Engines (ALE), for IP address lookup. The existing ALEs can be categorised into the following techniques: Ternary Content Addressable Memories-based (TCAM-based), trie-based and TCAM-emulation. TCAM-based techniques are expensive and consume a lot of power, since they employ TCAMs in their architecture. Trie-based techniques have nondeterministic latency and external memory accesses, since they store the Forwarding Information Base (FIB) in the memory using a trie data structure. TCAM-emulation techniques commonly combine TCAMs with lower-cost circuits that handle less time-critical activities. In this thesis, the main objective is to propose an ALE architecture with fast search that addresses the main shortcomings of TCAM-based and trie-based techniques.

Achieving an admissible throughput in the proposed ALE is its fundamental requirement due to the recent improvements of network systems and growth of Internet of Things (IoTs). For that matter, hardware accelerators have been adopted to achieve a high speed search. In this work, Field Programmable Gate Arrays (FPGAs) are specialized reconfigurable hardware accelerators chosen as the target platform for the ALE architecture. Five TCAM-emulation ALE architectures are proposed in this thesis: the Full-Serial, the Full-Parallel, the IP-Split, the IP-Split-Bucket and the Update-enabled IP-Split-Bucket architectures. Each architecture builds on the previous one with progressive improvements.

The Full-Serial architecture employs memories to store the FIB and one comparator to perform a serial search on the FIB entries. The Full-Parallel architecture stores the FIB entries into the logical resources of the FPGA and employs a parallel search using one comparator for each FIB entry. The IP-Split architecture employs a level of decoders to avoid repetitive comparisons in the equivalent entries of the FIB. The IP-Split-Bucket architecture is an upgraded version of the previous architecture using a partitioning scheme aiming to optimize the IP-Split architecture. Finally, the Update-enabled IP-Split-Bucket supports high-update rate IP address lookup. The most efficient proposed architecture is the IP-Split-Bucket, which is a novel high-performance memory-less ALE. For a real-world FIB with 524 k IPv4 prefixes, IP-Split-Bucket achieves a throughput of 103.4M packets per second and consumes respectively 23% and 22% of the Look Up Tables (LUTs) and Flip-Flops (FFs) of a Xilinx XC7V2000T chip.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ALE	Address Lookup Engine
ALUT	Address Lookup Table
APT	Address Position Table
APTA	Address Position Table Address
APTAG	Address Position Table Address Generator
BCAM	Binary Content Addressable Memory
BGP	Border Gateway Protocol
BPT	Bit Position Table
CAM	Content Addressable Memory
CB	Comparator Block
CLIPS	Combined Length-Infix Pipelined Search
DB	Decoder Block
DuPI	Dual linear Pipelined architecture for IP lookup
FF	Flip-Flop
FIB	Forwarding Information Base
FPGA	Field Programmable Gate Array
IoT	Internet of Things
IP	Internet Protocol
LL	Lookup Latency
LPM	Longest Prefix Match
LSB	Least Significant Bit
LUT	Lookup Table
LT	Lookup Throughput
MB	Match Block
ML	Match Line

MSB	Most Significant Bit
NHI	Next Hop Information
NHIB	Next Hop Information Block
OSPF	Open Shortest Path First
PEB	Priority Encoder Block
RCAM	Reconfigurable Content Addressable Memory
RIP	Routing Information Protocol
RIS	Routing Information Services
ROM	Read Only Memory
SDN	Software Defined Networking
SRAM	Static Random Access Memory
SR-TCAM	SRAM based TCAM
TCAM	Ternary Content Addressable Memory
TTL	Time To Live
UL	Update Latency
ULA	Update Latency of Addition
ULD	Update Latency of Deletion
ULM	Update Latency of Modification
UT	Update Throughput
UTA	Update Throughput of Addition
UTD	Update Throughput of Deletion
UTM	Update Throughput of Modification

CHAPTER 1 INTRODUCTION

1.1 Context

A router is a network device that is responsible for routing data packets from their source host to their destination host. As shown in Figure 1.1, a router consists of two functional components: the control and data planes. The control plane deals with the system configuration and the update information [1]. It uses the routing table information of different protocols such as Routing Information Protocol (RIP), Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) and removes non-essential routes to build a Forwarding Information Base (FIB) [2]. A FIB is a table stored in the router's memory that lists the routing information: destination IP address, prefix size and Next Hop Information (NHI) [2]. Every update in the protocol routing tables leads to an update of the FIB. The data plane uses the information extracted from the FIB table to forward a data packet to its proper next node. A router has an Address Lookup Engine (ALE) in its data plane, which is a special purpose engine that performs IP address lookup. IP address lookup is a process that determines the next node to which a packet must be sent in order to reach its destination. The ALE performs the Longest Prefix Match (LPM) algorithm on the FIB. The LPM algorithm receives the destination IP address of the incoming packet as an input. This algorithm finds a match with a FIB entry that has the largest prefix size. Next, the ALE returns the NHI of the match that defines the output port number.

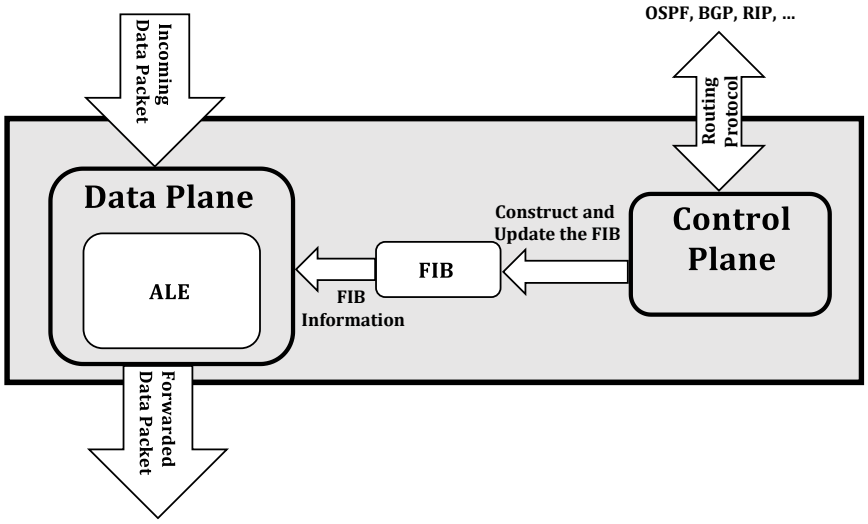

Figure 1.1: Router functional components [3]

An example FIB is shown in Table 1.1, consisting of three columns. The first column contains the destination IP addresses known by the router, the second column defines the prefix size of each entry and the third column is the NHI that determines the output port number of the router corresponding to each entry. There is a special entry in every FIB that determines the default route. The default route is used when the destination IP address of the incoming packet is unknown to the router. The prefix size of the default route is equal to zero and its NHI is the default port number. Suppose an IP address lookup is performed for an incoming packet with destination IP address of 200.103.124.180 and the FIB shown in Table 1.1. Two matches are found: one match with the entry #2 with prefix size of 24 and another match with the entry #4 of the FIB with prefix size of 8. Therefore, since the prefix size of the entry #2 is larger than the entry #4, it is reported as the LPM output. Thus, the router sends the packet on port #4.

Table 1.1: A sample FIB

Entry	Destination IP addresses	Prefix Size	NHI
#1	132.207.153.197	32	5
#2	200.103.124.1	24	4
#3	132.207.123.67	16	1
#4	200.156.46.200	8	3
#5	0.0.0.0	0	Out of range output port

1.2 Motivation

The ever-increasing speed of digital networks demands a high performance realization of LPM in network switches and routers. 5G is the fifth generation for mobile networks that meet new demands of throughput and latency for the next generation technology. Upcoming network devices, including ALEs, should be able to keep up with the performance of the 5G technology. The data rate of 5G technology is up to 10 Gbps for hundreds of active users at once [4], while its latency is in the order of the millisecond [5] [6].

To respect the constraints of the 5G technology, the performance of the ALE is evaluated using two categories. The first category concerns the IP address lookup and the second category

concerns updating the FIB. In the first category, there are two metrics for evaluating the performance of the IP address lookup: Lookup Latency (LL) and Lookup Throughput (LT). LL is the delay between the arrival of one packet to the ALE until the corresponding information of the LPM is available. LT is the number of incoming IP addresses handled in every time unit for one ALE. In the second category, there are two metrics for evaluating the performance of the FIB update: Update Latency (UL) and Update Throughput (UT). UL is the update delay, which measures the amount of time it takes from placing an update request to its actual occurrence. UT is the number of updates handled in a time unit for an ALE.

The control plane of a router can receive three types of update information: Addition (A), Modification (M) and Delete (D). For an addition, a new IP address along with its NHI and prefix size information is added to the FIB. In a modification, only the NHI corresponding to the match is revised. In a delete, the entry corresponding to that address is removed from the FIB completely. The latency and throughput can be measured for each type of update: ULA, UTA, ULM, UTM, ULD, UTD. ULA and UTA are the update latency and update throughput of the addition, respectively. ULM is the update latency of the modification while UTM corresponds to the update throughput of the modification. ULD and UTD are dedicated to the update latency of delete and update throughput of delete, respectively. Future ALEs should meet the requirements of 5G technology in terms of LL and LT, and their update mechanism should respect the constraints of ULA, UTA, ULM, UTM, ULD and UTD.

Apart from performance, another requirement for ALEs designed for 5G technology is to support the growth of the Internet of Things (IoTs). The IoT is the network and communication of the internet-enabled devices, systems, vehicles and other items that are connected to the internet. Due to the ever-increasing growth of the IoT, there will be a great demand for more IP addresses and thus larger FIBs in the routers [7]. Currently the FIB size for IPv4 addresses is near 500 k entries while for IPv6 it is around 26 k entries [8]. In the near future, the number of IPv4 addresses is expected to increase up to 2 M entries causing the depletion of IPv4 addresses [9]. This will likely cause the number of IPv6 entries to increase rapidly instead [10]. In all cases, future ALEs will have to support FIBs with a very large number of entries.

1.3 Objectives

The existing approaches for IP lookup can be categorized into three types: Content Addressable Memory (CAM)-based [11], [12], [13], and trie-based search techniques [14], [15], [16], [17] and CAM-emulation [18], [19], [20], [21]. CAM is a special type of high-speed memory that has the ability to search its entire contents in one clock cycle. CAM-based techniques suffer from high power consumption and high cost of CAMs. Trie-based techniques employ the trie data structure in their design which causes nondeterministic latency and external memory accesses. CAM-emulation techniques emulate the CAM functionality to avoid the shortcomings of CAM-based techniques.

In this thesis, we propose multiple high-performance ALE architectures that support large FIB tables. We suggest a novel architecture using CAM-emulation techniques to avoid the limitations of CAM-based and trie-based techniques. The proposed architectures for ALEs should meet the LL and LT requirements of the 5G technology and constraints of a potential next generation designs in terms of FIB sizes. Our design should support a FIB with a size of around 500 k entries for IPv4 addresses. In terms of LT, IP addresses normally arrive at a router at a rate of between 150 M to 250 M packets per second. The maximum acceptable LL is application dependent and varies between applications.

In summary, the main objectives of this thesis are:

- Proposing high performance architectures for IP address lookup supporting existing large FIBs in accordance with networks constraints in terms of LT and LL.
- Avoid using expensive and power hungry CAMs in the design and producing comparable results to CAM-based approaches in terms of logical resource usage and CAM cost.
- The proposed architectures must eliminate the drawbacks of trie-based techniques in terms of using external memory usage and having nondeterministic latency.
- Simulating and synthesizing the proposed architectures and comparing their results together and the existing work on IP address lookup.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the existing work on IP address lookup and provides a comparison on the conventional approaches. Chapter 3 describes the proposed architectures for IP address lookup in four sections: the Full-Serial (section 3.1), the Full-Parallel (section 3.2), the IP-Split-Bucket (section 3.3), and the IP-Split-Bucket (section 3.4). Chapter 4 presents the experimental results of the proposed architectures and comparison of the most efficient proposed architecture with CAM-based, trie-based and CAM-emulation techniques. Chapter 5 concludes the thesis and highlights possible directions for future work.

CHAPTER 2 RELATED WORK

This chapter introduces the framework of the IP address lookup process in the ALE. It provides an overview of the existing work on the IP lookup categorized into three types: CAM-based, trie-based and CAM-emulation techniques. Section 2.1 presents an overall description of the ALE and its structure. The remaining sections are dedicated to describing and comparing the existing work.

2.1 Address Lookup Engine

The ALE performs the IP lookup on all entries of the FIB table. As shown in Figure 2.1, the ALE consists of three main blocks: a Match Block (MB), a Priority Encoder Block (PEB) and a NHI Block (NHIB) [12], [18]. A MB searches the FIB entries for all possible matches with the incoming IP address. Match Lines (MLs), shown in Figure 2.1, specify whether there is a match in the FIB or not. For example, in case of a match in address i, $ML(i)$ would be 1 otherwise it would be 0. A PEB receives all the MLs and selects the match with the highest priority. In a LPM algorithm, the prefix size determines the priority. An NHIB contains a memory storing the output port numbers dedicated for each entry of the FIB. The NHIB uses the address of the selected match and gives the next hop number as its output.

The ALE architecture regardless of its NHIB is equivalent to a string matching engine. A string matching engine searches for all occurrences of an input string inside a predetermined set of target strings. When a match is found, the corresponding location is given as the output result.

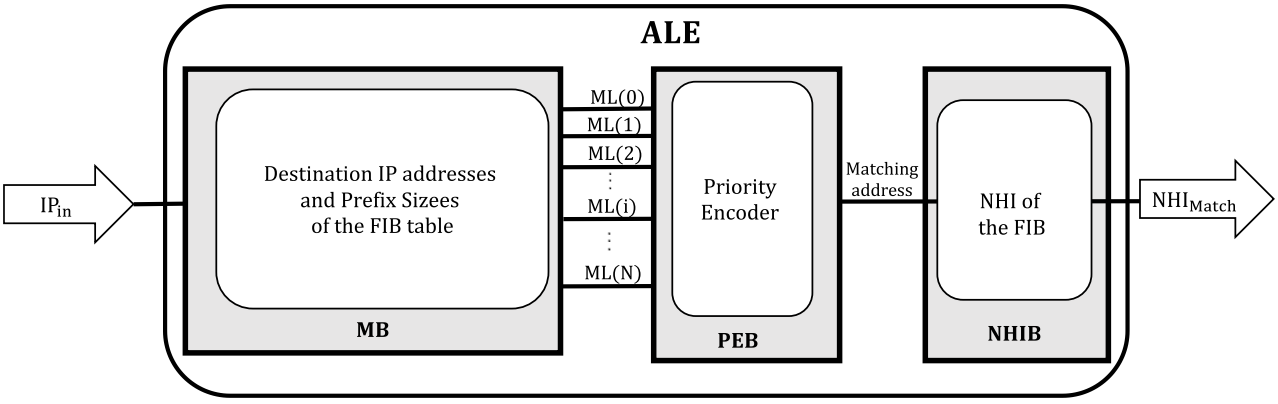

Figure 2.1: Address lookup engine [12]

Each string is an ordered vector of symbols of a given alphabet. Latin, Binary and DNA are examples of existing alphabets. High-performance string matching is required in several applications, particularly in real-time systems such as IP address lookup. A string matching engine with a binary alphabet and input string size of 32 (IPv4) or 64 (IPv6), can be applied for the MB and the PEB of the ALE.

Several approaches have been proposed to implement the MB and PEB in prior work. CAM-based techniques employ a CAM to perform the process of the MB and the PEB of the ALE [12]. Researchers using the trie-based techniques implement the MB using a trie data structure [14]. In CAM-emulation techniques, researchers emulate the CAM functionality to implement the MB and PEB to avoid the shortcomings of the CAM-based [22]. We review the previous approaches on the IP address lookup in the following sections. Section 2.2 describes the CAM-based existing work on IP address lookup. Section 2.3 and section 2.4 present the existing work using trie-based and CAM-emulation techniques, respectively.

2.2 CAM-Based Techniques

A CAM is a memory that acts as the opposite of a standard RAM. A RAM receives an input address and gives the data stored in the corresponding address. On the other hand, a CAM receives a data as an input and passes the address of the data item, if it is present. In a CAM search for an entry, there are three possible cases for a result:

1. If a match is found, the address of the match is the output. For some implementations, a CAM gives the incoming data along with its match address as the outputs.
2. If no match is found, a default out-of-range address is generated as the output address or a special error signal is activated.
3. If more than one match is found, the match with the highest priority is selected using a priority encoder. The priority of the contents is design dependent. For instance, in the LPM algorithm the match corresponding to the largest prefix size has the highest priority.

CAM-based techniques thus use CAMs to perform an instant search in the entire FIB. Despite the high-speed search they offer, CAM-based techniques have three drawbacks: their throughput is limited by the CAM access speed, they consume high power, and they are expensive. Aiming to mitigate these drawbacks, some researchers have proposed techniques and algorithms to

minimize the size of required CAMs. These techniques commonly combine CAMs with low-cost circuits that handle less time-critical activities. The existing work using CAM-based techniques can be classified based on the CAM type. There are two types of CAMs: Binary CAMs (BCAM) and Ternary CAMs (TCAM). Section 2.2.1 presents the TCAM-based approaches to solve the LPM problem. Section 2.2.2 reviews an approach using a hybrid architecture of TCAMs and BCAMs for IP address lookup.

2.2.1 TCAM-Based Techniques

A TCAM is a type of CAM that provides a flexible search in the memory. Bits in a TCAM have three states: 0, 1 and ternary state. The ternary state is defined with 'X' and is a 'don't care'. It provides the possibility of matching a '1' and a '0' at the same time. For instance, the data word '11X' is a match with either '110' or '111'. Thus, there is a possibility of finding multiple matches in a TCAM. Therefore, a TCAM requires a priority encoder in order to find the entry with the highest priority.

There are two types of priority mechanisms in a TCAM: explicit and inherent. In an explicit priority mechanism, an extra field of priority is stored in the TCAM along with its contents. Therefore, the match with the highest priority is selected based on its priority field. This mechanism is easy to update but consumes more resources since an additional priority field is added to every entry of the table. In an inherent priority mechanism, the contents are sorted based on their priority. Therefore, the match with the lowest address in a TCAM is the match with the highest priority. This mechanism is costly to update since it requires resorting all the contents of the table when a new IP address is added to the table [22].

Existing TCAM-based approaches to implement the MB and PEB of an ALE will now be presented in the order of their publication years.

Zheng *et al.* (2004) proposed a TCAM-based ALE with high throughput and low power consumption [11]. They applied multiple parallel partitioned TCAM chips in their architecture to achieve high throughput. They divided the FIB into 16 evenly distributed groups of IP addresses. Each group is stored inside one of the partitions of the TCAMs. A "load-balance-based" algorithm is proposed to balance the lookup traffic of the 16 groups between the partitioned

TCAMs. In Zheng *et al.*'s approach one partition of the TCAM is enabled at a time to minimise the power consumption. For every lookup, the incoming IP address defines the enabled partition.

The architecture is shown in Figure 2.2. It consists of four sections: index logic, priority selector, lookup unit and ordering logic. Index logic starts the lookup process by determining the enabled partition using the incoming IP address. Due to the possibility of having multiple TCAMs with enabled partitions, a priority selector is required to choose the least busy TCAM to perform the lookup. Since each lookup unit contains one TCAM chip, the problem size is decreased into a partition in only one of the TCAMs. The last process is the ordering logic that puts all the NHIs of incoming IP addresses in the same order of the incoming traffic.

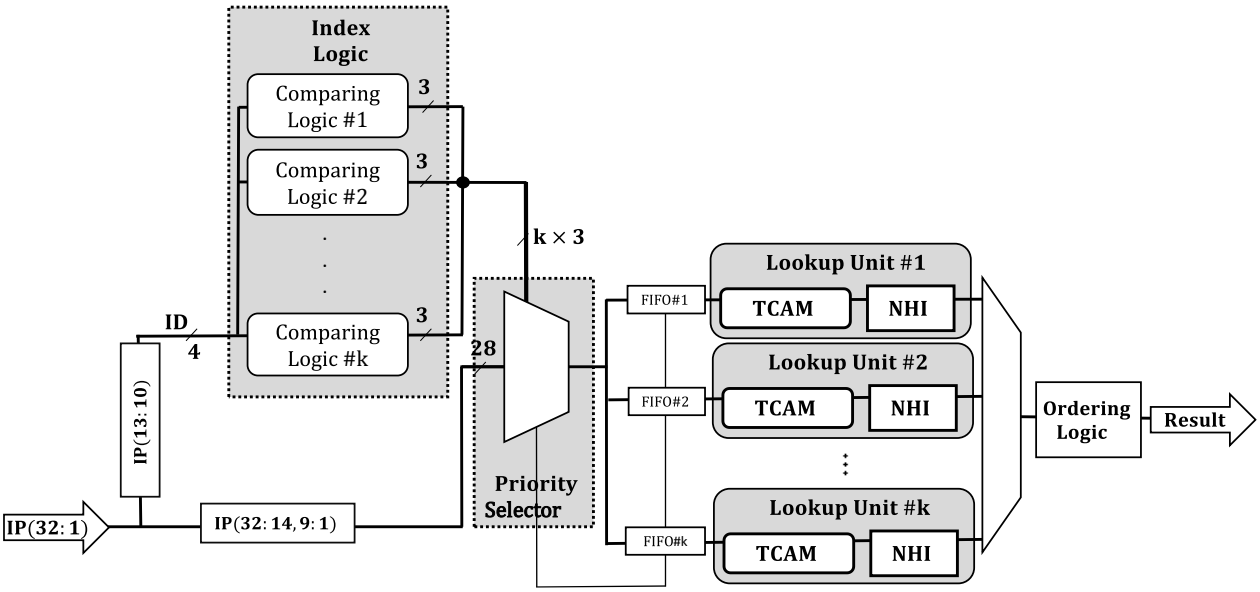

Figure 2.2: Architecture of multi-chip structure and chip partitioning technique [11]

For implementation results, Zheng *et al.* have proposed an example with 4 TCAMs of size 256 $k \times 36$ b which contain 8 partitions. Supporting a FIB with maximum size of 819.2 k, it achieved a 133 MHz clock frequency. Maximum power consumption of the implementation was 4 watts and the maximum lookup throughput was 533 Mpps with an average processing latency of 75 ns.

In 2005, Pao presented a new organization for commercial TCAMs of the SiberCAM family [23] for the purpose of IPv6 Address Lookup [12]. Available commercial TCAM chips have a fixed word length size of 36, 72, 144 or 288 bits. Therefore, for a 128-bit IPv6 address lookup technique, a commercial TCAM with word length of 144 bits is required for all prefixes

regardless of their actual size. Pao proposed a novel two level organization that exploits the prefix length distribution of real-life FIBs. Most IPv6 prefixes are less than 64 bits long. The largest width of the available TCAMs in the 64-bit range is 72-bit TCAMs. Therefore, a 72-bit TCAM is applied for the IP addresses with prefix size of 72 or less. For prefix sizes of more than 72 bits, the IPv6 address lookup process is split into two steps handled by 72-bit TCAMs. Pao's approach leads to efficient TCAM utilization. It improves space utilization by 50% and reduces lookup time by 30% to 45% compared to the conventional method.

Figure 2.3 illustrates a sample FIB organized inside a TCAM and a SRAM following Pao's approach. The MB is divided into two partitions P_s and P_l. The P_s partition contains table entries with prefix sizes less than 72 bits (not a marker) and the first 72 bits of prefix sizes more than 72 bits (marker). The P_l partition contains the entries with prefix sizes more than 72 bits (marker). When a match is found in the P_s, it is checked whether the match is a marker or not. If the match is not a marker the NHI of that address is valid. Otherwise, we need to calculate the address of NHI based on the marker and the remaining 56 bits of the IPv6 address.

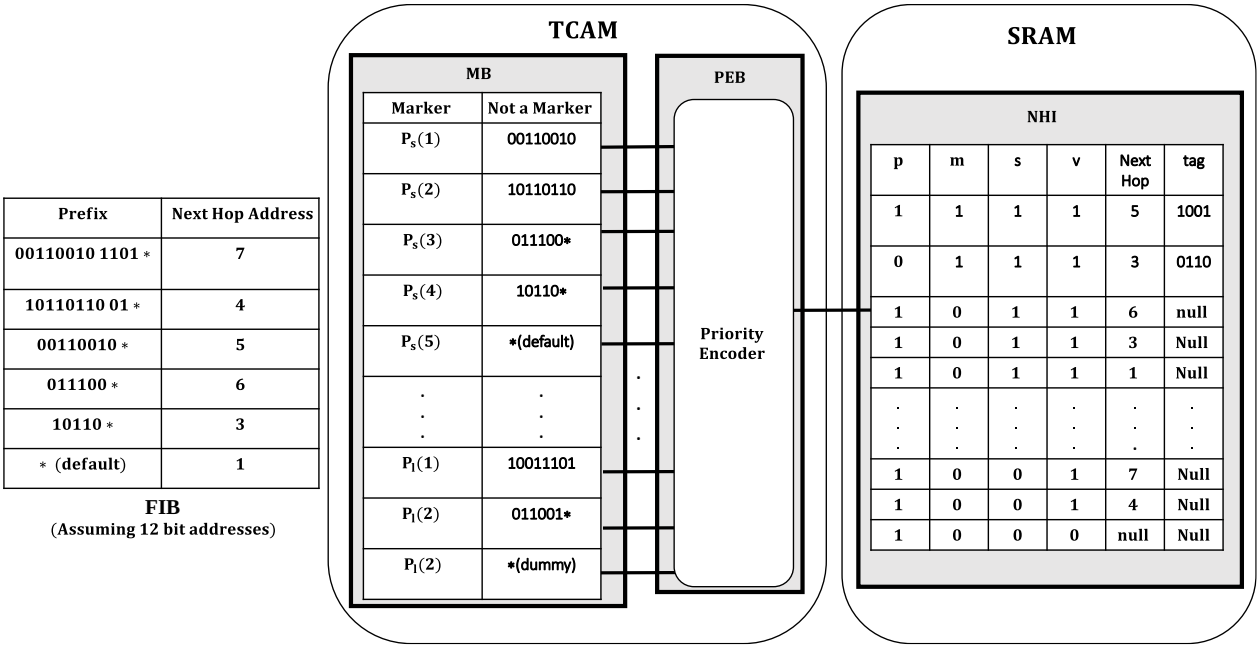

Figure 2.3: Two-level organization in TCAM [12]

Figure 2.4 shows the architecture of the overall system. First, a network processor sends all the incoming 128-bit IPv6 addresses to the input FIFO buffer. Second, the level-selection module

sends a 1-level command for the prefix sizes smaller than 72 and a 2-level command for prefix sizes more than 72. Moreover, it sends the search key to the TCAM for the second cycle operation. The pipelined TCAM and the SRAM are the third and fourth modules defining the NHI. The fifth module consists of two registers: B1 and B2. For a longest match with prefix sizes of more than 72-bits, five fields of the B2 register specify the second cycle operation. However, for a longest match with a prefix size less than 72 bits, the results are sent directly to the output FIFO buffers regardless of the operations in B2.

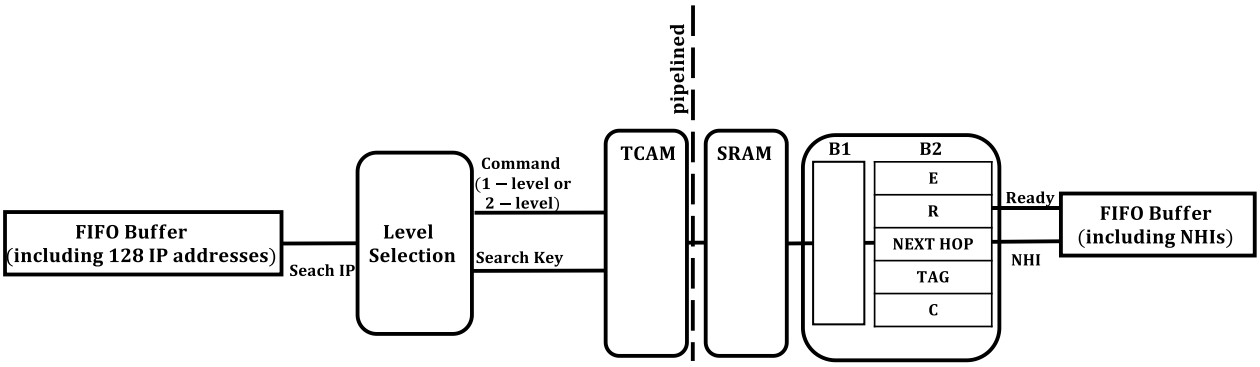

Figure 2.4: Hardware interface for TCAM co-processor [12]

2.2.2 Hybrid TCAM-BCAM Technique

A BCAM is the simplest type of CAM with the two states 0 and 1. Applications that require an exact match utilize BCAMs. It is not possible to find multiple matches simultaneously in a BCAM. Therefore, BCAMs have a simpler comparison circuit than TCAMs. They require fewer transistors and have lower power consumption than TCAMs [24]. Some of the existing works exploit the advantages of BCAMs by replacing TCAMs with BCAMs when possible.

In 2010, Sun and Kim proposed a hybrid TCAM-BCAM-based ALE that stored the fixed FIB parts in BCAMs instead of TCAMs [24]. They suggested a power efficient architecture with low area consumption compared to traditional TCAMs. Sun and Kim studied the distribution of IP address prefixes of real-world FIBs of the years 1997 to 2009. Using the characteristics of prefix distributions, they divided the FIB into seven groups ($P_1, P_2, P_3, P_4, P_5, P_6, P_7$) based on their prefix sizes. The groups P_1 to P_7 contain IP addresses with prefix sizes of 8, 9-15, 16, 17-23, 24, 25-31, 32, respectively. All the IP addresses of each group are divided into two parts: fixed and

unfixed. The fixed part contains only the binary values (where the state of ‘X’ does not exist) that are handled by BCAMs. The unfixed part contains the remainder of the IP addresses that are implemented in TCAMs. The purpose of this division is to minimise the area and power consumption of the design by replacing TCAMs by BCAMs whenever possible.

The approach presented in [24] splits the LPM process into a three-stage operation as shown in Figure 2.5. The first two stages are equivalent to the MB of the ALE. The first stage consists of parallel TCAMs and BCAMs giving the comparison results of the unfixed and fixed parts, respectively. The second stage is the parallel AND operations calculating the match results. The last stage is equivalent to the PEB that consists of two levels of priority encoders. The PEB consists of multiple levels of small priority encoders instead of one large priority encoder in order to reduce the complexity of the design.

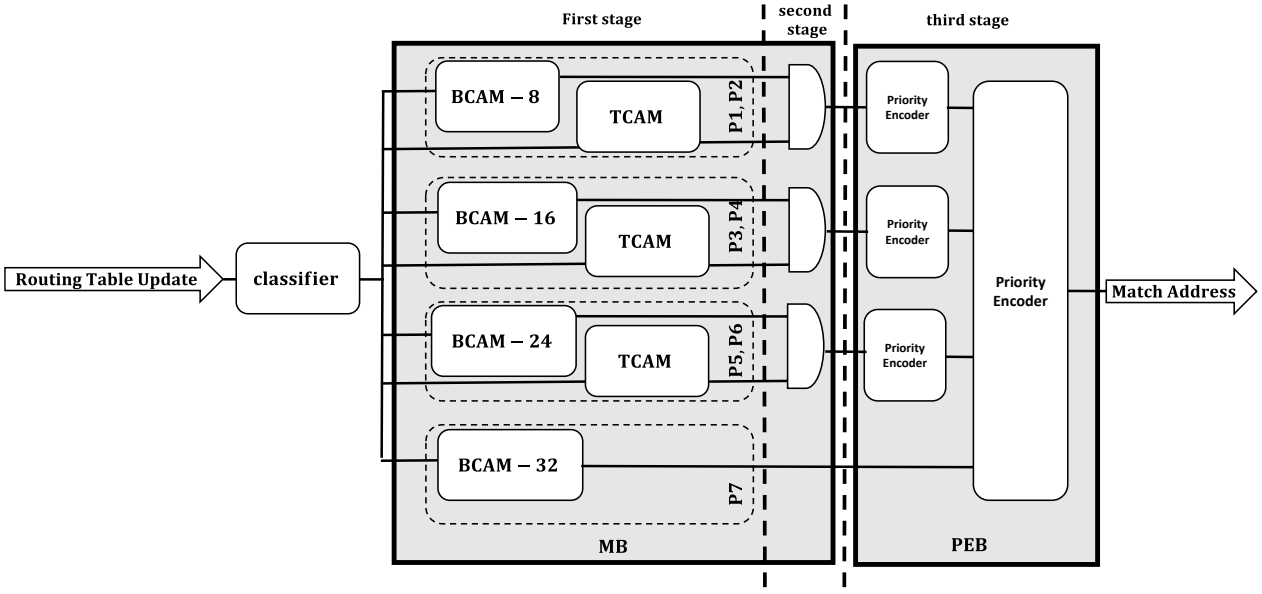

Figure 2.5: Hybrid architecture of TCAM and BCAM [24]

By exploiting the characteristics of BCAMs and TCAMs, this hybrid architecture results in an improvement in power consumption, area usage and throughput. In terms of update, a traditional TCAM sorts a large FIB based on the prefix sizes. However, an update of Sun and Kim’s approach requires multiple small groups of IP addresses to be sorted based on their prefix sizes in parallel. Therefore, it reduces the number of clock cycles for an update process to 50% compared

to traditional TCAMs [24]. Moreover, this parallel update leads to an increase in the throughput of IP packet processing.

2.3 Trie-Based Techniques

Static RAM (SRAM) memories are more efficient than TCAMs in terms of hardware complexity and performance. SRAMs support higher speed, more density and lower power consumption compared to TCAMs, as shown in Table 2.1. Since commercially available TCAMs have fixed word length size with bounded depth, they are less versatile to new protocols and addressing techniques compared to SRAMs. Moreover, TCAMs are more expensive than SRAMs. Some approaches [14], [15], [16], [17], [20], [25] exploit the advantages of using SRAMs instead of traditional TCAMs such as trie-based techniques.

Table 2.1: TCAM and SRAM comparison [14]

Type of Memory	**Maximum Clock Rate (MHz)**	**Cell Size (Number of Transistors)**	**Power Consumption (Watts)**
TCAM (18 M bits Chip)	200	16	12~15
SRAM (18 M bits Chip)	400	6	≈ 0.19

Trie-based techniques create search trees from the FIB and store the tree information in SRAMs using a trie data structure [14], [15], [16], [17]. Trie-based techniques require traversing the search tree from the root node to leaves serially. Therefore, these techniques are inherently slower than TCAM-based ones. As the size of the FIB increases, the memory space required to store the trie information grows rapidly. For large LPM problems, trie information cannot be stored in on-chip memories. Hence, the utilization of large external memories is inevitable [15]. External memory access is the main performance bottleneck for trie-based techniques. Several approaches attempt to minimize either the number of external memory accesses for a search or their latency [16], [17]. However, accessing external memory remains the main performance-limiting factor in large trie-based designs.

In 2006, Baker and Prasanna [17] proposed an efficient string matching approach using a trie-based technique. They suggested a tool that provides automatic synthesis of highly efficient NIDS on an FPGA. This tool generates two architectures. The first architecture is a pre-decoded

shift and compare block. A high-level graph-based partitioning of strings is applied in the first architecture to reduce the size of decoders and share the shift registers. With partitioning, the goal is to maximize the similarity of patterns inside each partition. The second architecture is a tree-based prefix sharing block that is responsible for reducing the redundant comparisons. Hence, the tool they proposed improves the area consumption and performance while reducing redundant comparisons.

Hoang Le *et al.* (2008) exploited the advantages of SRAMs over TCAMs by implementing a SRAM-based IP-lookup architecture with a binary-tree-based design on FPGA. It supports a FIB of 228 k entries with a high throughput of 324 MLPS (multi lookup per second) while using an external SRAM to support a larger FIBs [14]. Figure 2.6 shows the global view of the proposed DUal linear Pipeline architecture for IP lookup (DuPI). The FIB is converted into a binary search tree according to their prefixes sizes. The tree information is stored inside a SRAM. Using two parallel-pipelined levels and a dual Read/Write SRAM, this architecture supports two packets at a time. The maximum number of pipelined stages is determined by two factors: the size of the tree and the maximum number of operations required to traverse the tree.

Figure 2.7 illustrates the top-level architecture of the DuPI supporting updates. It handles two types of updates: in-place update and new-route update. In-place update modifies, removes or adds every incoming IP to the binary search tree individually after the tree is constructed. A new-route update changes the binary search tree completely and requires a rebuild of the tree.

In 2011, Yang, Erdem and Prasanna proposed an FPGA-based architecture for IP lookup using trie-based technique [15]. They suggested a Combined Length-Infix Pipelined Search (CLIPS) that performs a LPM in $log\ (l - c)$ phases, supposing that l is the size of the IP address and c is a design constant. Each phase has an individual local infix table mapped to an FPGA using on-chip BRAMs and off-chip SRAMs. Using external memories, CLIPS supports very large FIBs with high throughput. According to the simulation results for a 9.5 M - entry IPv4 FIB, the CLIPS architecture supports 312 MLPS throughput while using 4 external SRAM memories with 28 Mb of BRAM on chip memory.

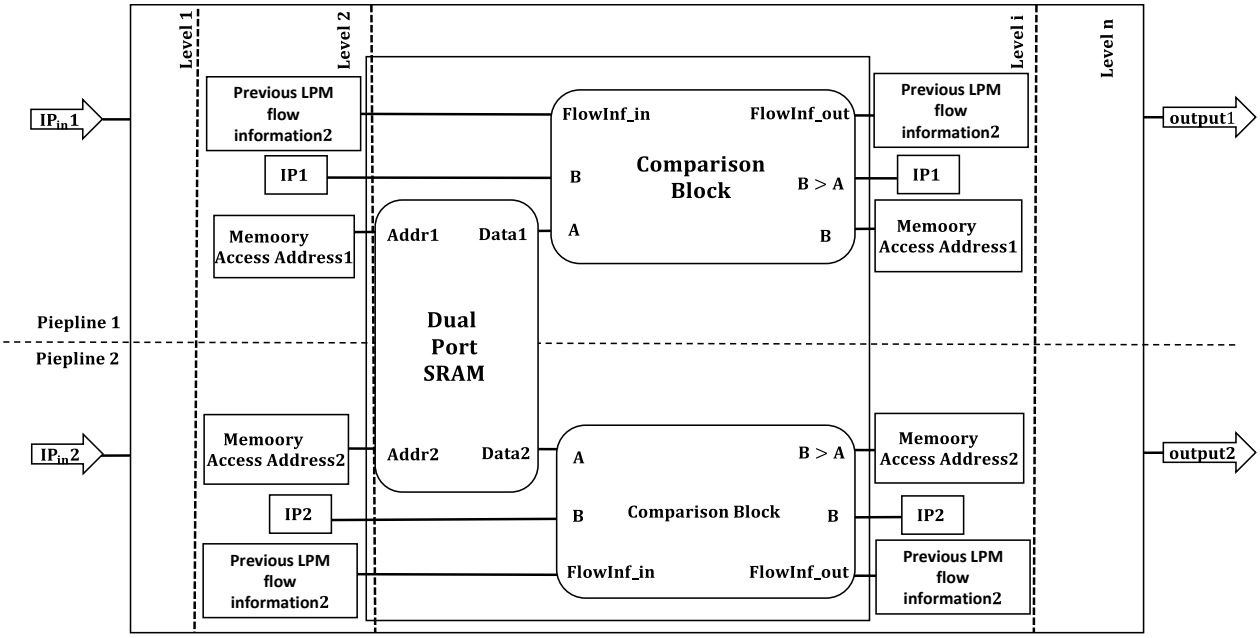

Figure 2.6: DuPI architecture [14]

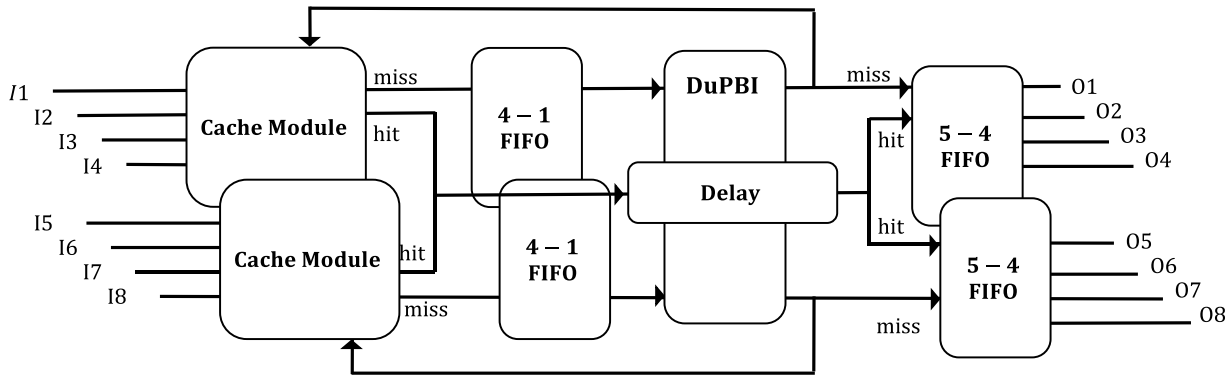

Figure 2.7: Global DuPI architecture supporting updates [14]

In 2013, Matoušek, et al. [16] proposed a trie-based approach introducing memory efficient dedicated hardware for IP address lookup. They suggested a new representation of IP prefix set in memory applying novel types of nodes and an algorithm to map the nodes to the tree. The generated tree consumes less memory compared to existing trie-based approaches. The proposed architecture consists of several pipelined processing elements as shown in Figure 2.8. Each processing element is responsible for one step of a LPM algorithm. Using dual port memories and two levels of pipeline, the performance is improved by a factor of two.

In 2016, Mun and Lim presented a trie-based IP address lookup using a Bloom filter [26]. To improve the efficiency, they reduced the number of false positive results of the bloom filter significantly using the characteristics for the trie-based techniques. Consequently, the number of off-chip trie accesses of non-existing nodes is reduced. They performed an IP address lookup using a reasonable amount of on-chip Bloom filter and off-chip trie accesses.

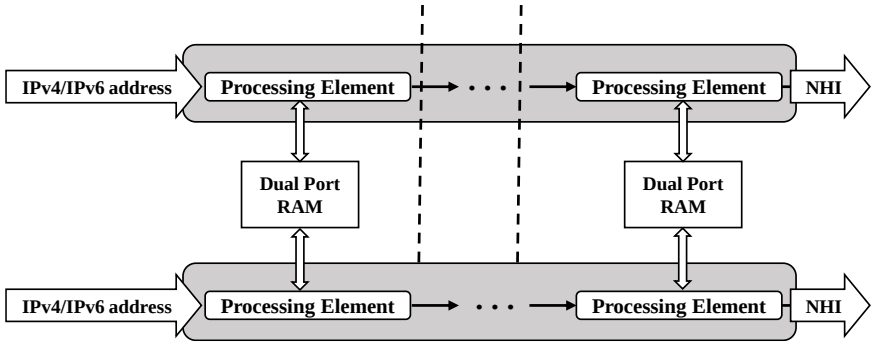

Figure 2.8: Double level of pipelined processing elements [16]

2.4 CAM-Emulation Techniques

Some researchers proposed approaches to emulate the functionality of commercially available CAMs while reducing their cost and power consumption. These approaches are known as CAM-emulation techniques [18], [19], [20], [21]. Taking advantage of FPGA configurability, implementing ALEs on FPGAs offers flexible and scalable IP address lookup process. The objective of CAM-emulation technique is to provide a fast and parallel search that addresses the main shortcomings of CAM-based approaches. It avoids the high hardware cost and high power consumption of CAMs, while providing comparable performance. In the following, some of the existing work on CAM-emulation technique are categorized into two groups based on the type of the emulated CAM: BCAM-emulation and TCAM-emulation techniques.

2.4.1 BCAM-Emulation Techniques

In a BCAM-based FIB, only one match is allowed at a time. Therefore, no priority encoder is required for a BCAM. To convert a FIB into a BCAM-based FIB, all the entries of the FIB with prefix size less than 32 bits should be expanded to 32-bit size. After the 32-bit expansion, it is possible to have multiple equivalent entries. For such a case, all entries except the one with the

largest prefix size are removed from the FIB. As a result, the FIB is adapted to the rules of the LPM for IP address lookup. In this section, we discuss an existing approach that emulates the BCAM functionality for the LPM problem.

In 2000, Guccione *et al.* in [27] proposed a run-time reconfigurable high-speed implementation of BCAM on FPGA called Reconfigurable CAM (RCAM). RCAM produces a faster, smaller and more adjustable BCAM compared to traditional ones. Figure 2.9 illustrates the MB of the RCAM used for IPv4 address lookup. The incoming IPv4 address is divided into 8 groups of 4 bits. The comparison of each group is handled by one Look Up Table (LUT). There are two intermediate AND gates handling the results of every four LUT. As shown in Figure 2.9, the final match result is the output of a final AND gate applied on the results of the intermediate AND gates. In this approach, the data in the FPGA are stored in LUTs instead of Flip Flops (FFs). This leads to a reduction in the CAM's size and an increase in its throughput [27].

The JBits tool is applied to reconfigure the BCAM and modify different parts of the FPGA at run-time, such as LUTs. Therefore, it is possible to resize dynamically the RCAM at run-time. This results in having a possibility of allocating and reallocating the BCAM resources at run-time. In [27], the authors does not present implementation results for their approach except one test case. The maximum supported table-sizes for the RCAM implementation on the Virtex V1000 FPGA are 3 k and 1 k for IPv4 and IPv6 addresses, respectively [27]. However, the size of real-world FIBs are larger as mentioned in section 1.2. For example, the Mae-West FIB [14] is 27 times larger than the supported table size.

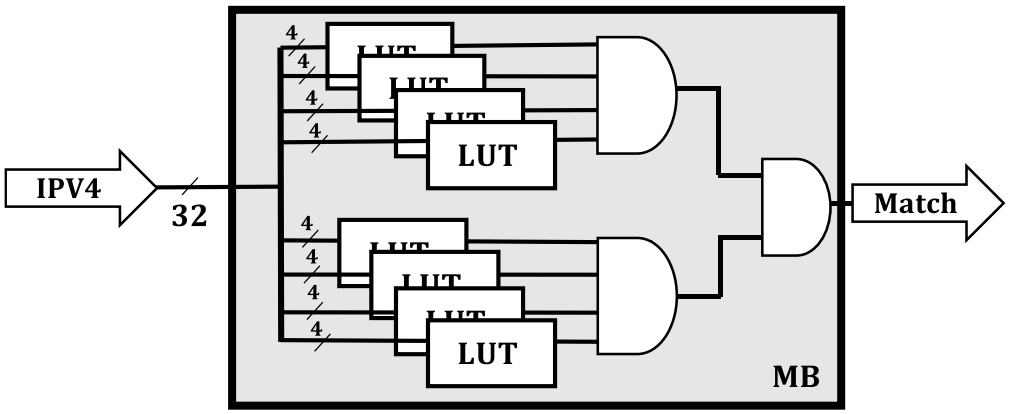

Figure 2.9: RCAM matching for IPv4 [27]

In 2004, Clark and Schimmel proposed a BCAM-emulation approach for string matching [21]. The authors suggested an FPGA implementation of a scalable string matching design supporting network constraints. Their approach allows adjusting the trade-off between capacity and throughput according to application requirements. They applied multi-character decoders to their design in order to improve performance and eliminate redundant comparisons in string matching. The multi-character decoder shown in Figure 2.10 processes multiple input strings at once and provides all possible comparison results at once using character decoders.

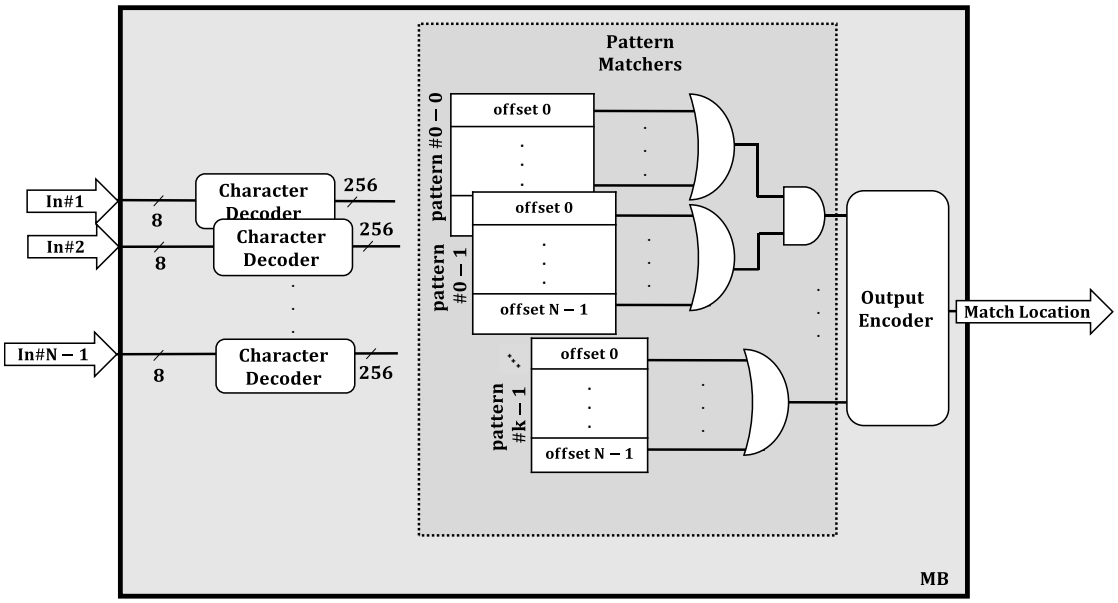

Figure 2.10: String matching with multi-character decoder [21]

2.4.2 TCAM-Emulation Techniques

TCAM-emulation techniques have been proposed for many TCAM applications such as IP address lookup, to replace expensive TCAMs with low cost circuits.

In 2002, Ditmar *et al.* suggested a dynamically reconfigurable FPGA-based TCAM-emulation design for IP characterizations [22]. IP characterization is the procedure of classifying the packets based on the information in their header. For an IPv6 characterization, the TCAM should be 315 bits wide in order to contain all the information of the IPv6 header such as source address, destination address, incoming link, outgoing link, etc. Thus IP characterization cannot be easily supported by commercially available TCAMs, since a TCAM's word length size is not usually

equal to 315 bits. Ditmar *et al.* described a TCAM-emulation design for IPv6 characterization with the following abilities:

1. Supporting variable size TCAM search words
2. Dynamic update of the FPGA using the JBits tool
3. A hybrid explicit-inherent priority mechanism for a more efficient update

The authors propose dividing each search word into 5 pipelined stages of blocks of 64-bits and to connect the blocks to each other with shift registers. Since ternary states are not required in the comparison in a TCAM, the blocks containing the ternary states are not stored. The reduction in the size of the TCAM word is shown from 'a' to 'c' in Figure 2.11. If no block is removed in the design, a match result is given every 5 cycle. In order to respect the 5 cycles of the match, deletion of a ternary block leads to two consecutive levels of pipelined stages. Consequently, there is a possibility of storing variable-size words. Moreover, there is no waste of space in the TCAM. Consequently, it is possible to store more entries.

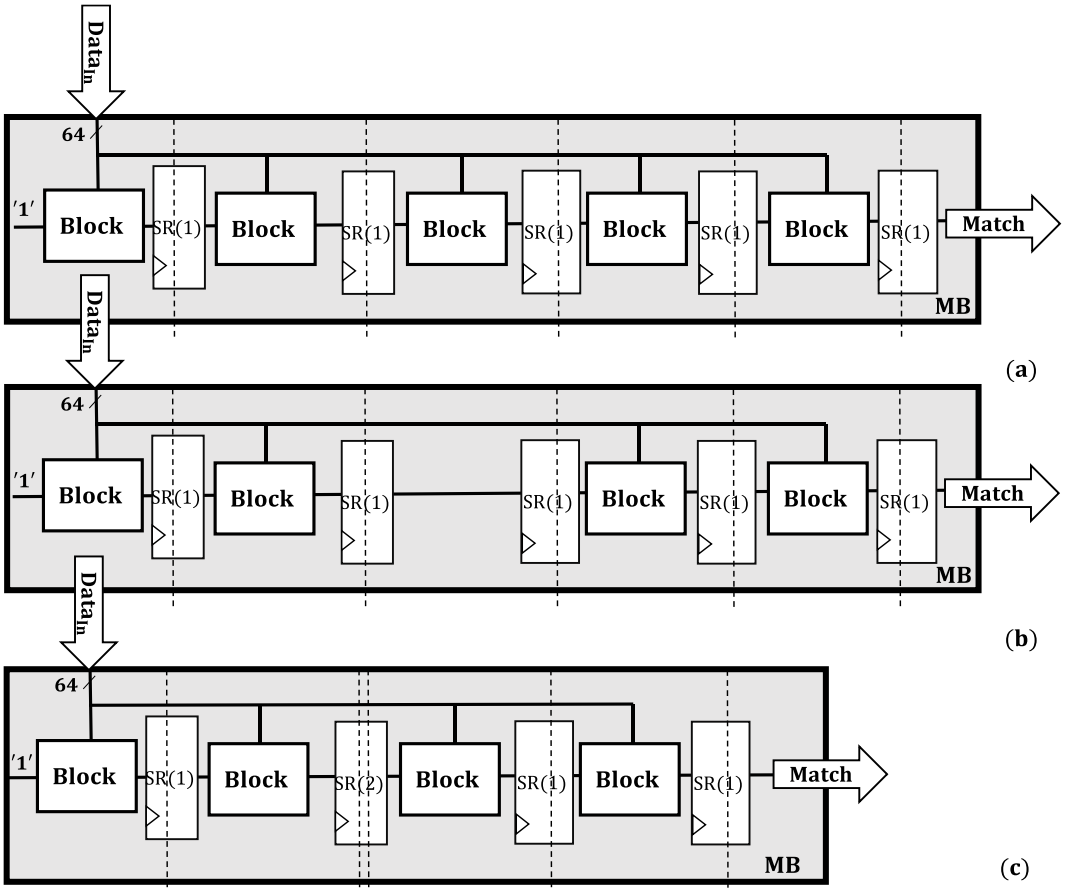

Figure 2.11: TCAM design supporting variable word size [22]

The general architecture of this approach is presented in Figure 2.12. The MB implemented by the structure shown in Figure 2.11, finds multiple matches with the incoming data ($Data_{in}$). Therefore, the MB is followed by the PEB to find the match with the highest priority among all possible matches. In order to exploit the advantages of both priority mechanisms, the authors employ a hybrid explicit-inherent priority encoder. As shown in Figure 2.12, beside a typical priority encoder they have added a switch box configured by JBits tools. The switch box is responsible for routing the possible matches to the priority encoder. There are 8 possible priority values for the entries. When there are multiple matches with the same priority value, inherent priority is applied.

According to Ditmar *et al.*, the most efficient TCAM implementation for IPv6 characterisation could fit a maximum of 256 320-bit words where an inherent/explicit priority mechanism is applied. For this case, there is a 47% utilization of the Xilinx Virtex XCV1000 FPGA while achieving a frequency of 17.2 MHz and throughput of 3.4 M searches per second.

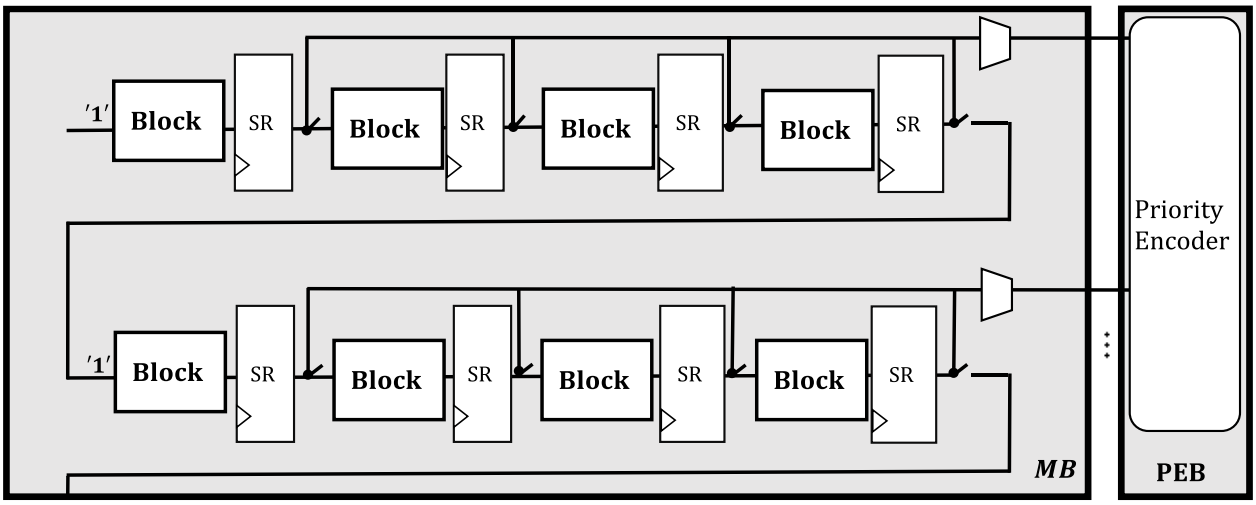

Figure 2.12: Physical structure of dynamic reconfigurable FPGA-based CAM [22]

In 2012, Ullah *et al.* proposed a SRAM-based TCAM-emulation design i.e. SR-TCAM [20]. SR-TCAM is a generic architecture, applicable to any TCAM application. As shown in Figure 2.13, this architecture is specified into the MB and the PEB. The entries are stored in SRAMs instead of TCAMs in this architecture. Therefore, the conventional TCAM is divided vertically into n groups. The groups are stored in corresponding SRAMs known as Bit Position Tables (BPTs) and Address Position Table (APT).

To perform the search, the incoming W-bit search-word is partitioned into n sub-words of w bits; where $W = n \times w$. Each sub-word is sent to its corresponding BPT to check for its availability. Every BPT accepts w bits as an input and contains all possible combinations of w bits (2^w bits). For example, the i^{th} sub-word of the incoming search-word is checked inside the i^{th} BPT for its availability. If the value of the sub-word is M, then the M^{th}bit among all the 2^w bits of the corresponding BPT is checked. The BPT has 1-bit output of 0 or 1. When the output value is '1', a sub-word match is occurred. Otherwise, the search process is stopped.

If all BPTs have outputs with a value of '1', then the Address Position Table Address Generator (APTAG) will be activated. The APTAG generates a value that corresponds to a row inside the APT. The APT contains 2^w rows of K bits. Each row is dedicated to one possible sub-word value that contains all the K addresses of the TCAM words. Therefore, every APT accepts an Address Position Table Address (APTA) generated by APTAG to check inside its APT. If there is a '1' in the j^{th} position of the corresponding row, the value of j indicates the location of the sub-word inside the TCAM.

The last step is checking all the APT outputs with value of '1' to find those with common positions. In other words, if all the matched sub-words are dedicated to the same address position, there is a match with the input word on that position. For that matter, a K-bit AND gate is applied to find all potential matches of the incoming search-word. Next, the output of the K-bit AND gate is sent to the PEB to find the match with the highest priority.

Ullah *et al.* implemented the SR-TCAM for a table size of 512×36 on a Virtex-5 xc5vlx220 FPGA. It consumes 1966 LUTs and 1975 FFs. For that design, the SR-TCAM achieves a clock period of 48.7 ns and power consumption of 2.16 mW.

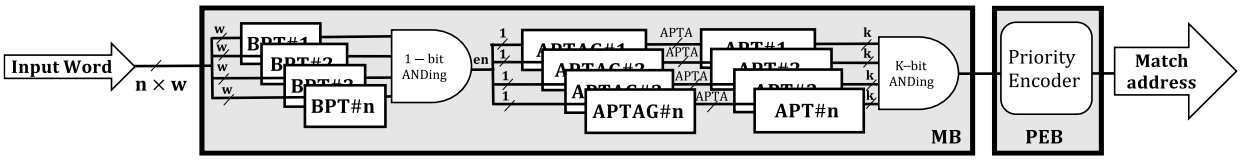

Figure 2.13: Architecture of SR-TCAM [20]

In 2013, Rasmussen *et al.* proposed a TCAM-emulation implementation of the LPM on FPGA [18]. The implementation of the MB is equivalent to ALE. On the other hand, the PEB is

implemented by two approaches: CN-LPM and G-LPM. In the CN-LPM architecture proposed by Rasmussen *et al.*, the PEB is implemented by a pipelined multi-level address encoder as shown in Figure 2.14. The PEB has $N-2$ inputs for a FIB of N entries. The inputs of the PEB are identified with ML_m. The ML_m indicates the match result and its prefix length of the m^{th} entry of the FIB. There are several comparator blocks in every level of the PEB. The $C_{i,j}$ block compares two of the ML_ms to find the match with largest prefix. Therefore, the output of every comparator consists of: the prefix length and Least Significant Bit (LSB) of the address of the match. Consequently, at each level one bit is added to the previous match address, which provides the final address of the match with longest prefix size. The final match address is used by the NHI to find the output port number.

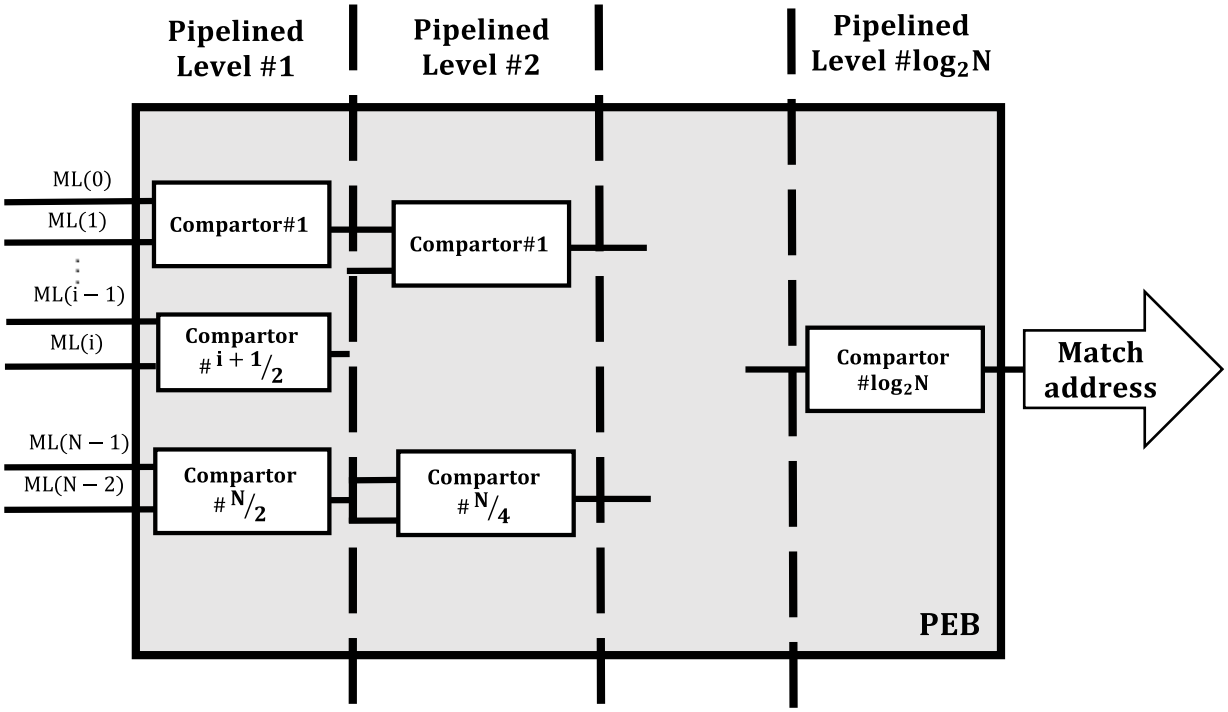

Figure 2.14: PEB of TCAM-emulation LPM [18]

As shown in Figure 2.15, Rasmussen *et al.* implemented the G-LPM architecture for local LPM process that is proposed by Gamache, *et al.* [28]. The MB consists of N parallel comparator blocks (TC) containing the N entries of the FIB. The sequential pipelined levels of the PEB are shown in Figure 2.15. They leave out all the matches except the match with the longest prefix

size in a step by step operation. For IPv6 address lookup, with prefix sizes in the range from 0 to 128, it requires 7 levels of pipelined levels. Each pipelined level is responsible for one bit of the prefixes. In the first level, the validity of all the most significant bits are checked. Only the survivors of the comparison with a validity are sent to the next level. This process continues until there is a local winner (one survivor) which is the LPM between of the N inputs.

The FIB is divided into several sections, with each section handled by a local LPM block. As shown in Figure 2.16, all the local LPM blocks are processed in parallel to find the global LPM. Therefore, to find the global winner, all the local winners go into the same process as the local LPM, called *global LPM*. Finally, the address of the global winner is used to give the NHI as shown in Figure 2.16.

This architecture supports a fast incremental update of the FIB. Moreover, there is no restriction in the order of storing the entries in the FIB. To compare with traditional TCAM, a FIB of 1024 entries is implemented on the CN-LPM and G-LPM. The implementation results show an increase of 771% and 789% in the number of Adaptive LUTs (ALUT) of the CN-LPM and G-LPM, respectively.

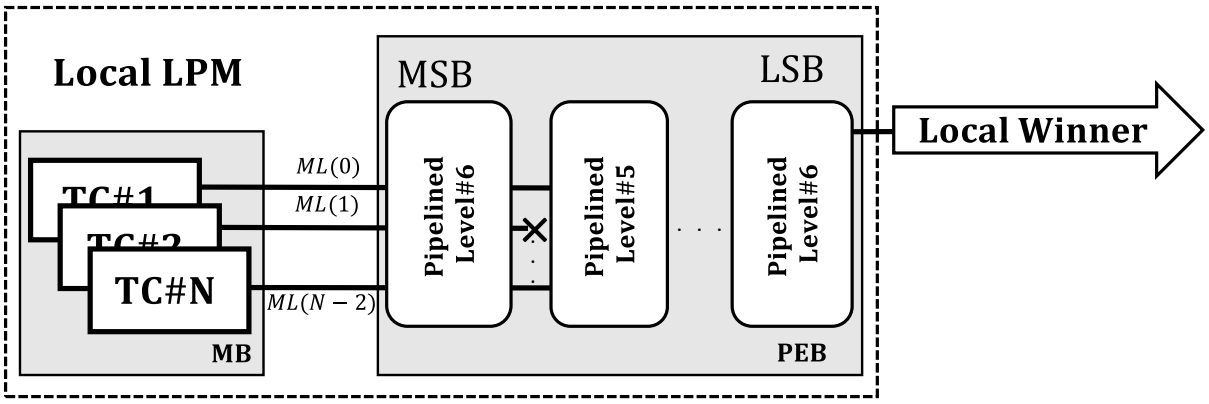

Figure 2.15: Local LPM TCAM-emulation consists of MB and PEB [18]

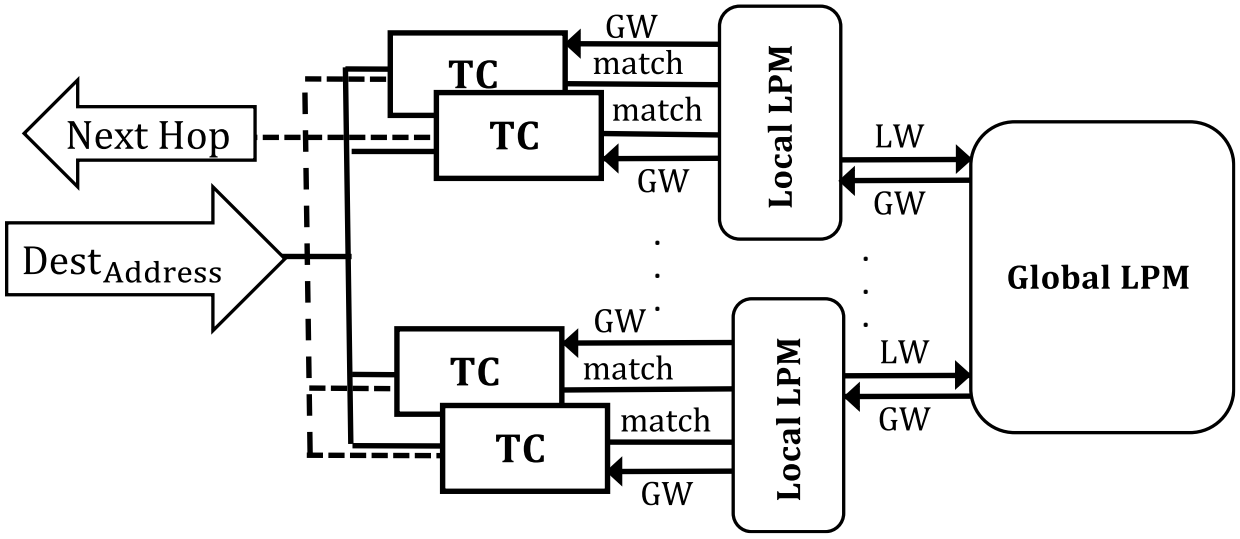

Figure 2.16: Global LPM TCAM-emulation [18]

In 2013, Jiang proposed a power efficient FPGA-based TCAM-emulation architecture [25]. Similar to Ullah *et, al.*'s approach [20], Jiang employed small units of SRAM-based TCAMs. Figure 2.17 shows the architecture without its update logic. All the FIB is partitioned into p parallel RAMs. The incoming search *key* to the MB has the size of w that is divided into p small words of size w_i sent to RAM_i. The size of RAM_i is equal to $2^{w_i} \times N$. RAM_i checks for a match on a small portion (w_i) of the input world (w); where the width of the word inside i^{th} RAM partition is equal to w_i. In case of a match in a partitioned RAM, there is a bit value of 1 in N-bit output in the location of the match. Afterwards, the results of all partitions are sent to an AND gate. If a match has been found on i^{th} word of the TCAM, then the i^{th} bit must remain 1 in the final result of the AND gate. The output of the AND gate illustrates all possible matches therefore it is equivalent to the output of the MB. Subsequently, all the results are sent to the PEB to find the match with the highest priority. In this architecture, the inherent priority mechanism is applied. The output port named *match* has the value of 1 in case of a match and the output port named *ID* defines the position of the match.

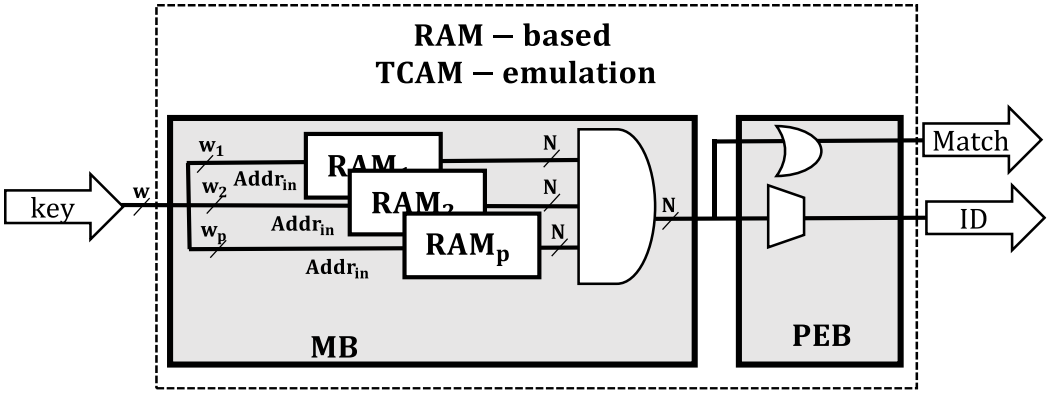

Figure 2.17: Underlying architecture scalable RAM-based TCAM [25]

Figure 2.18 illustrates the top level architecture of Jiang's approach that consists of several units. Each unit is equivalent to the architecture shown in Figure 2.19. Each unit contains a $U \times W$ RAM-based TCAM-emulation block identical to the block shown in Figure 2.17.

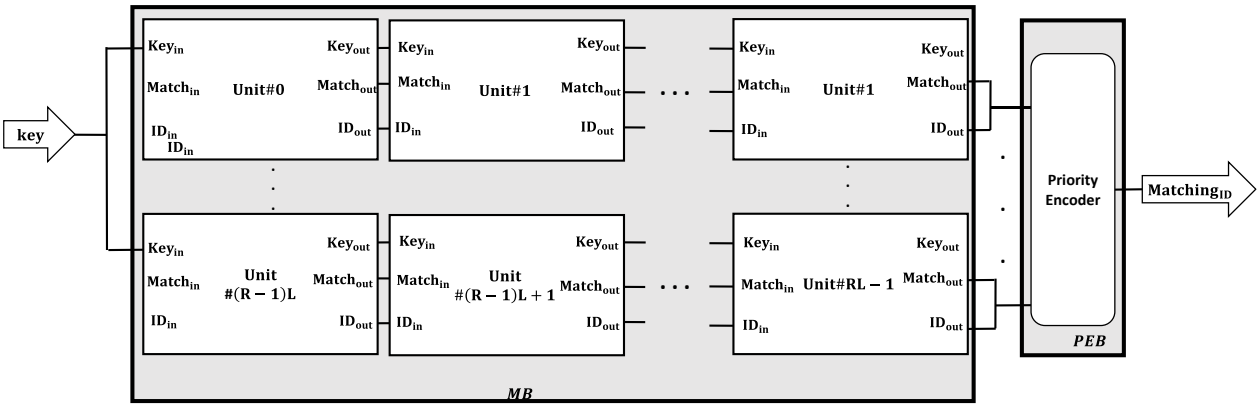

Figure 2.18: Global view architecture [25]

As shown in Figure 2.19, the variable $Match_{in}$of each unit specifies the existence of a match in the previous units. In case of a match in the previous units, the multiplexer chooses the previous match since it has the higher priority. Otherwise, ID_{out} is chosen based on the match that is found in the existing unit. The proposed architecture in [25] presented a beneficial implementation of TCAMs on FPGA compared to the aforementioned approaches in the literature. Jiang aimed for an FPGA-based design of a TCAM with efficient power consumption while supporting large TCAM sizes.

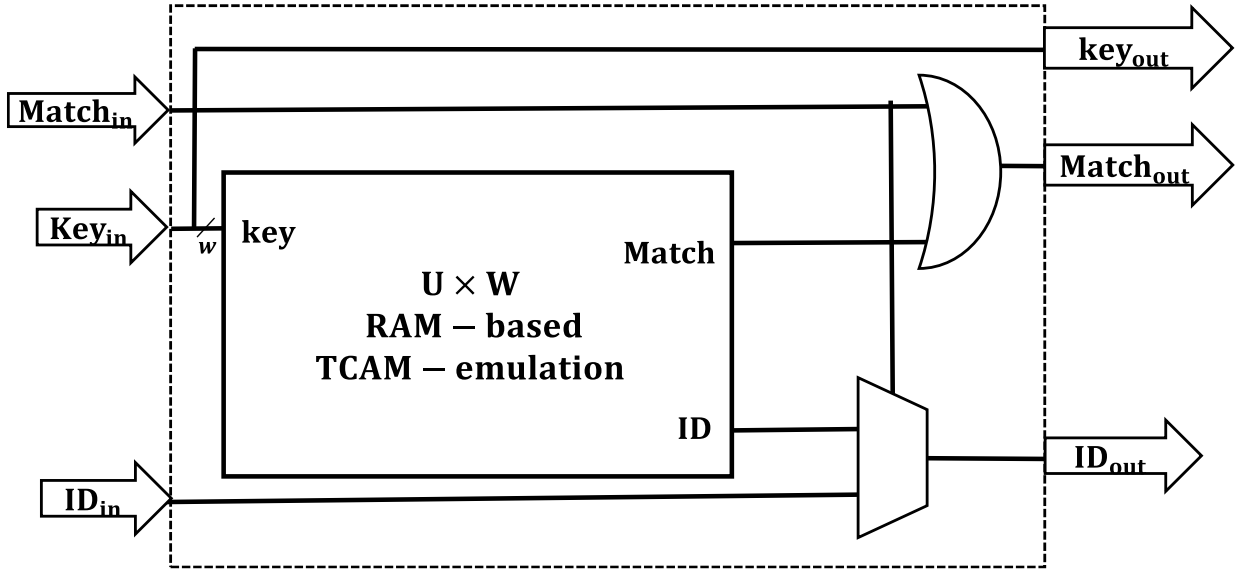

Figure 2.19: Unit architecture [25]

2.5 Comparison of the Existing Work

In this section, we present a comparison of the existing approaches in the literature. Each column of Table 2.2 shows one property of the applied technique. The first column specifies each approach by their reference number. The second column specifies the type of technique. The third one defines the applied method to implement the MB; whether it uses logical resources of FPGAs, memories, TCAM or BCAMs. The fourth column defines the type of the priority algorithm used for each technique (Inherent or Explicit). The fifth column defines whether the proposed approach supports an update mechanism to reconfigure the FPGA or not. The sixth column defines the FPGA family used in each approach. The sixth column determines the size of the tested design in each approach. The last column specifies the hardware resource usage of the FPGA implementation for each design.

Table 2.2: Comparison of existing CAM-based, trie-based and CAM-emulation techniques

Approach	Technique	MB	PEB	Update Mechanism	FPGA family	Table Size		LUTs	FFs
[27] (1990)	BCAM-emulation	Cells	N/A	JBit tool (Configurability at run-time)	Virtex XC2V1000	3 k× 32	1 k× 64	-	-
[22] (2002)	TCAM-emulation	Cell	Explicit and Inherent	JBit tool (Configurability at run-time)	Virtex XCV1000	256× 320		54 k	216 k
[28] (2003)	TCAM-emulation	N/A	Explicit	Fast Incremental Update	Stratix IV	1024× 64		15.8 k	11.3 k
[11] (2004)	TCAM-based	Commercial TCAMs	Commercial TCAMs	N/A	-	-		-	-
[21] (2004)	BCAM-emulation	Cells	N/A	N/A	Virtex-2 8000	17,537× 32		55 k	55 k
[17] (2006)	Trie-based	Cells	Cells	N/A	Virtex-2 XCVP100	602 × 20		6 k	6 k
[24] (2010)	Hybrid TCAM-BCAM-based	Commercial TCAMs/BCAMs	Inherent	Parallel Update	-	-		-	-
[15] (2011)	Trie-based	Cells, SRAM, External RAM	Cells, SRAM, External RAM	Incremental and Dynamical Update	Virtex SX475T	9.5 M× 32		7 k	22 k
[20] (2012)	TCAM-emulation	SRAM	Inherent	N/A	Virtex XC5VLX220	512×36		1.9 k	1.9 k
[18] (2013)	TCAM-emulation	N/A	Explicit	Fast Incremental Update	Stratix IV	1024× 64		15.5 k	10 k
[25] (2013)	TCAM-emulation	RAM	Inherent	N/A	Virtex XC7V2000T	4096×150		182 k	182 k
[16] (2013)	Trie-based	Cells	Cells	N/A	Virtex XC6VSX475T	442,748× 32		88 k	44 k

CHAPTER 3 PROPOSED ADDRESS LOOKUP ENGINE ARCHITECTURES

In this Chapter, we propose and describe in detail four different ALE architectures: Full-Serial, Full-Parallel, IP-Split and IP-Split-Bucket. The advantages and drawbacks of each architecture are explained as well. Section 3.1 presents the Full-Serial architecture. Section 3.2 describes the second proposed architecture called Full-Parallel. The third architecture is the IP-Split architecture described in section 3.3. The last architecture is IP-Split-Bucket architecture that is described in section 4.4.

3.1 Full-Serial Architecture

The Full-Serial architecture is a memory-based TCAM-emulation architecture for IP address lookup. It consists of two components: a memory and a comparator. The first component is a memory that stores all the entries of the FIB sorted based on their prefix size. The second component is a single comparator that performs a serial comparison of the incoming IP address with the memory entries. The first match to occur in the serial search determines LPM output. The address of the match is determined using a counter (cnt) that determines the position of the search and is incremented at every cycle. The first match can occur in the first cycle (first entry) or in the N^{th} cycle (N^{th} entry) for a FIB of N entries. Therefore, the search time is equal to $N/2$ in average.

Figure 3.1 shows the Full-Serial architecture applied for an IPv4 FIB with N entries. The memory consists of N entries of $40 + P$ bits. Each entry contains 32 bit of IPv4 address, P bits of prefix and 8 bits of NHI. Since the egress port IDs of every router has 8 bits, the NHI is stored with 8 bits in the memory. This architecture was implemented on an FPGA using two options. In option A, the prefix size is stored using 32 bits ($p = 32$), while in option B it is stored with 6 bits ($p = 6$). The size of P determines the format for storing the FIB entries and the comparator architecture. The following subsections describe the detailed comparator architecture of both proposed options for IPv4 ALE.

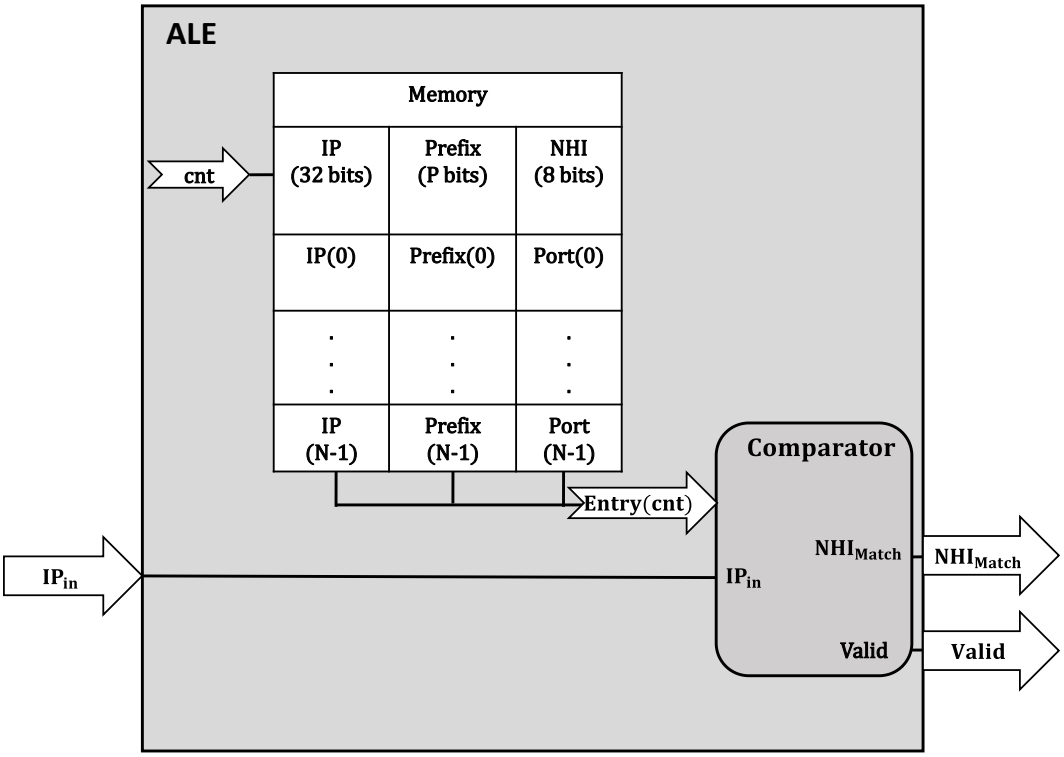

Figure 3.1: Full-Serial Architecture

3.1.1 Option A

In Option A, there is a 32-bit prefix entry with i bits of ones followed by $32 - i$ bits of zeros, where i determines the prefix size. For instance, if the prefix size is equal to 10, the 10 MSB of the prefix are equal to one and the remaining bits are equal to zero. The architecture of the comparator for Option A is shown in Figure 3.2. The comparator performs a comparison of the cnt^{th} entry of the FIB and the incoming IP address using two 2-input AND, one 2-input OR and one 32-input NOR. The first AND logic has the IP address of the cnt^{th} entry and its prefix as its inputs. Another AND logic has the incoming IP address and the prefix size of the cnt^{th} entry as its inputs. The results of the two latter AND operations are tested for equality by an OR and a NOR logic. In case of equality, a match is found ($valid = 1$) and the process is terminated. Otherwise, cnt is incremented and the same process is applied on the next entry of the FIB.

This architecture was implemented for different FIB sizes on various FPGAs. Section 4.1 presents the synthesis results of FPGA implementations of Full-Serial using option A.

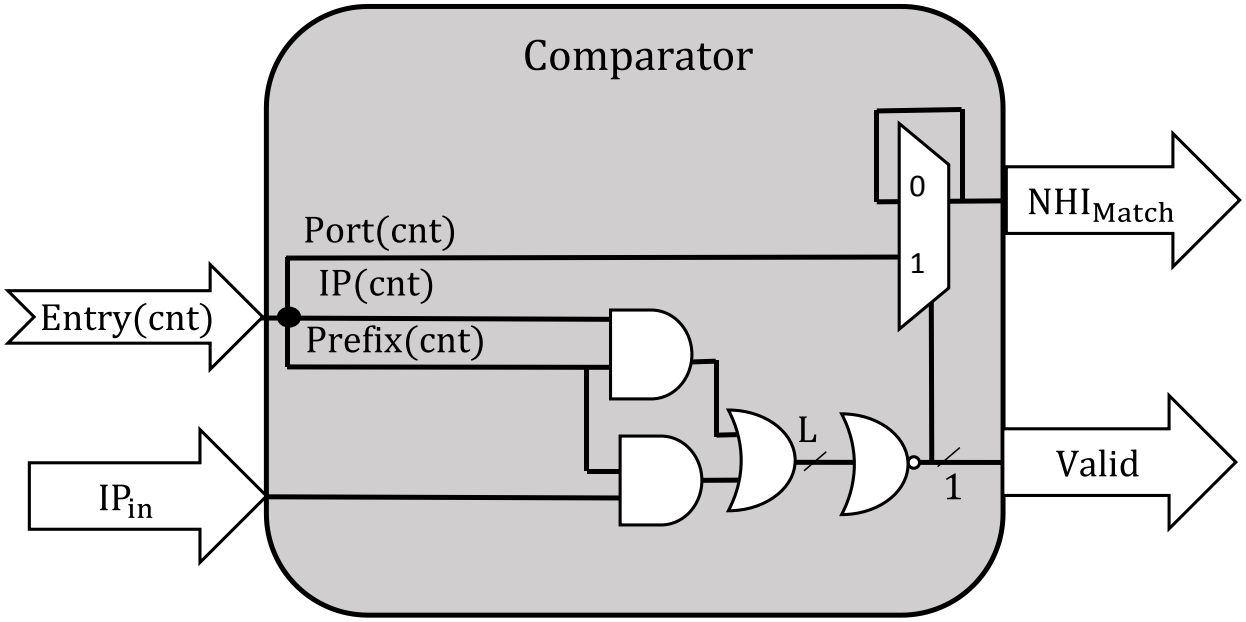

Figure 3.2: Full-Serial, architecture of the comparator using option A

3.1.2 Option B

The maximum prefix size for an IPv4 addresses is 32, therefore, in option B, the prefix has 6 bits ($P = 6$). For instance, a prefix size of 15 is stored as "001111". While option A requires twenty-six more bits to store the same prefix size: "11111111111111100000000000000000".

The detailed architecture of the comparator for option B is illustrated in Figure 3.3. The comparator performs an OR operation on certain portion of the IP address of the cnt^{th} entry and the incoming IP address. Since the comparison is performed on the MSB of the IP addresses, the ending index of each portion is equal to the length of the IP address minus one (31). The prefix size of the cnt^{th}entry determines the portion size of comparison at every cycle shown as $integer(prefix(cnt))$ in Figure 3.3. The comparator architecture and the format of storing the FIB entries in both options shows that option B is much simpler than option A. The Full-Serial architecture is implemented using both options. Section 4.1 presents the experimental results and comparison of option A and B.

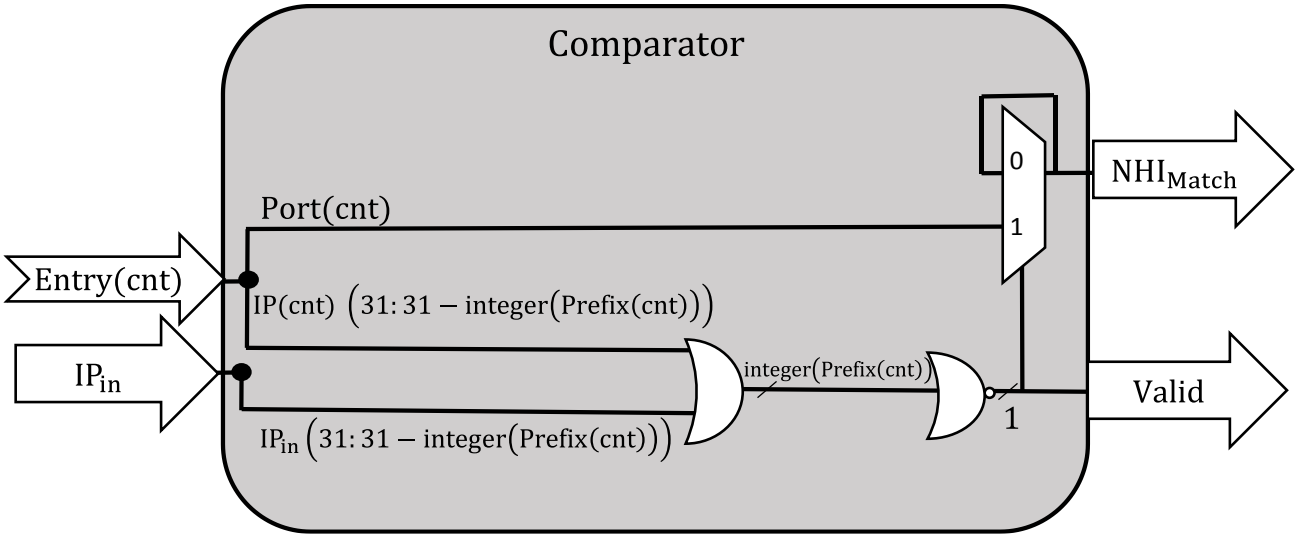

Figure 3.3: Full-Serial, architecture of the comparator using option B

3.2 Full-Parallel Architecture

Full-Parallel is the second proposed architecture for IP address lookup. It applies a parallel search on all the entries of the FIB to perform the LPM algorithm. Figure 3.4 illustrates the Full-Parallel architecture divided into three blocks: MB, PEB and NHIB. Let N be the number of IP addresses in the IPv4 FIB. The MB consists of N sorted parallel cells and N parallel comparators. Every cell contains 46 bits: 32 bits of IP address, 6 bits of prefix and 8 bits of NHI. The cells are sorted based on their prefix size in descending order. Therefore, the PEB finds the match with the lowest address that determines the output of the LPM algorithm ($Addr_{Match}$). Next, the NHIB selects the 8-bit NHI corresponding to the $Addr_{Match}$. It consists of 8 parallel multiplexers of size N: 1 with $Addr_{Match}$as their selector. The i^{th} multiplexer receives i^{th} bit of the N NHIs stored in the N parallel cells. The output of all N parallel multiplexers determine the NHI_{Match} (see Figure 3.4).

The Full-Parallel architecture reduces the latency and increases the throughput of the IP address lookup by performing a parallel search on all the entries. However, it requires high resource utilization in terms of FF consumption, since all the FIB is stored in parallel cells. Section 4.2 describes the synthesis results of the FPGA implementation of the Full-Parallel architecture.

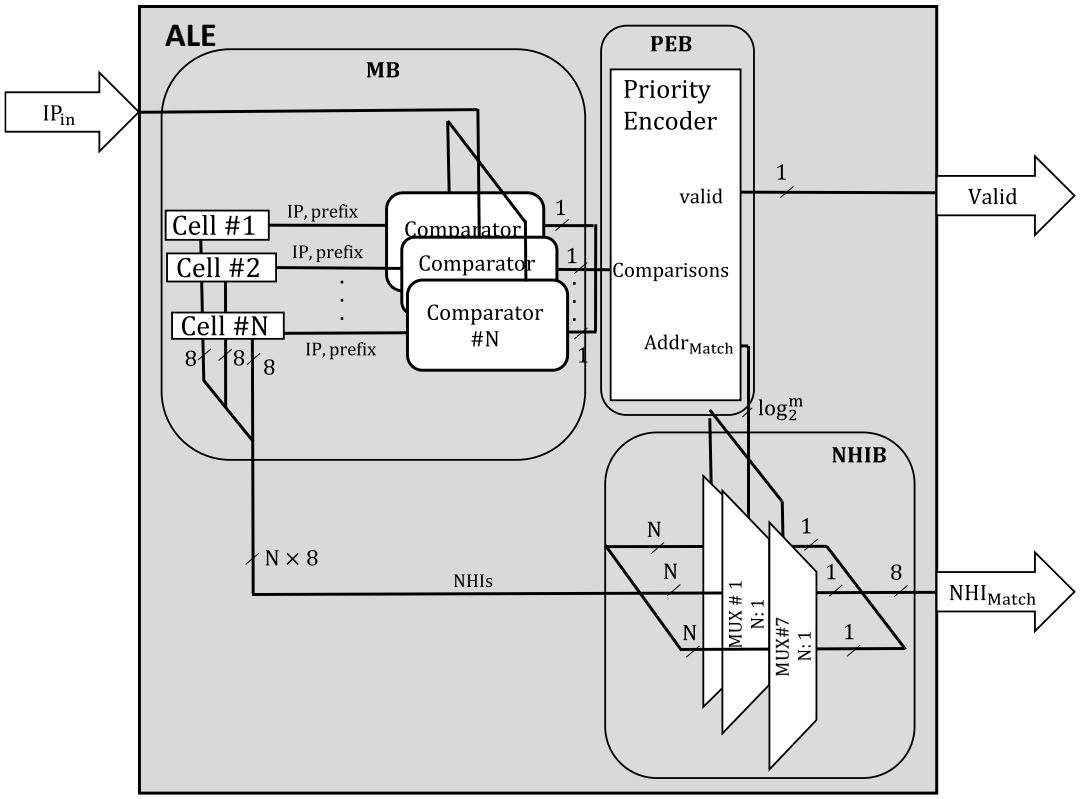

Figure 3.4: Full-Parallel architecture

3.3 IP-Split Architecture

IP-Split is the third proposed ALE architecture. As shown in Figure 3.5, it is divided into three main blocks and its MB consists of two sub-blocks: Decoder Block (DB) and Comparator Block (CB). In the following sub-sections, the behaviour of DB, CB, PEB and NHIB are explained in detail.

3.3.1 Decoder Block

To perform a traditional parallel comparison with the N entries of the FIB, N comparators are required (as shown in Figure 3.4, the Full-Parallel architecture). A large FIB may contain several IP addresses that are identical in some segments. This would waste resources, since multiple equivalent comparisons would be performed. To avoid such repetitive comparisons, the incoming IP address is split into v segments of k bits. Each segment of the incoming IP is expanded to its fully decoded form by an individual binary decoder. As shown in Figure 3.5, the DB consists of v

binary decoders of size k-to-2^k. The decoding method facilitates comparisons on an input segment by simultaneously providing all possible comparisons results for that segment. Some existing approaches on string matching employ a similar decoding method to improve comparison efficiency for large sets [19], [21].

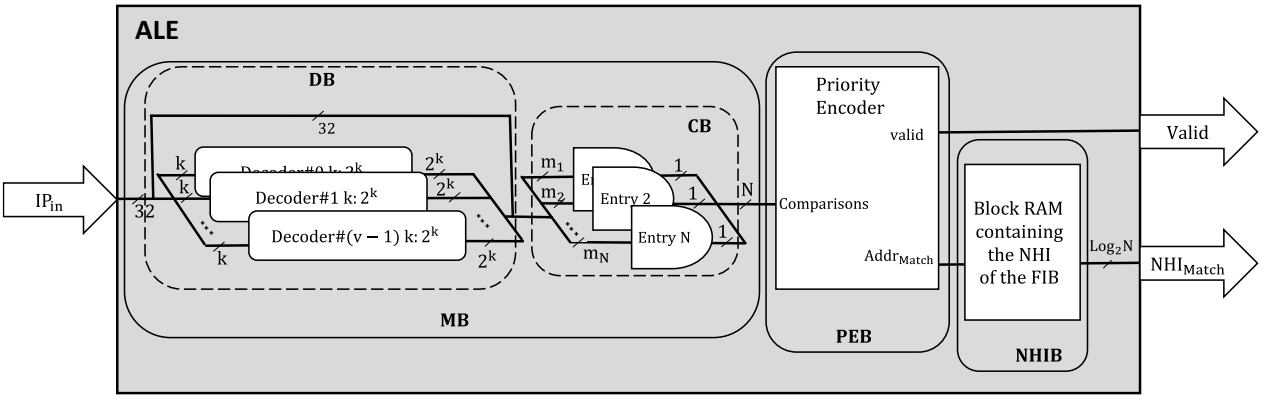

Figure 3.5: IP-Split architecture

3.3.2 Comparator Block

The CB compares the decoded incoming IP address with all IP addresses of the FIB using AND operations. Each IP address of the FIB is hard-coded in a single AND operation. If one new entry is added to the FIB, the corresponding AND operation must be synthesized and appended to CB. Consequently, changing the FIB demands a re-synthesis of CB, while other blocks are fixed. A partial reconfiguration only for the CB is favorable for such cases.

As shown in Figure 3.5, the CB consists of N AND operations of different sizes. The values of m_1 to m_N determine the size of the first to N^{th} AND operations. Each input port of the AND operation is connected to the output port of the decoder for the corresponding segment. For instance, the i^{th} output port of the j^{th} decoder is connected to one of the input ports of the AND operation for an IP address that has the value i on its j^{th} segment. The number of input ports of the l^{th} AND operation ($InPorts_l$) depends on the prefix size of the l^{th} entry of FIB ($prefix_l$). If k divides $prefix_l$, the $InPorts_l$ is equal to $\frac{prefix_l}{k}$. Otherwise, the number of input ports of the l^{th} AND operation for this case is calculated using the following equation:

$$InPorts_l = \left\lfloor \frac{prefix_l}{k} \right\rfloor + prefix_l \bmod k,\ l = 0,1,\dots,N-1, \quad (3.1)$$

The first $\left\lfloor \frac{prefix_l}{k} \right\rfloor$ MSB bits of the IP$(31:32 - prefix_l)$ are handled by the decoders while the $prefix_l \bmod k$ ending bits will not completely fill the k input ports of a decoder. Therefore, they are directly connected to the corresponding AND logic. An example of the DB and CB is illustrated in Figure 3.6.

Suppose that the FIB consists of two entries with the following values: $IP_1 =$ "43.180.0.0", $IP_2 =$ "43.176.0.0", $Prefix_1 = 15$, $Prefix_2 = 12$. Let v and k be 3 and 5, respectively. The prefix of each entry determines the portion of each IP address that requires a comparison. $IP_1(31:17)$ and $IP_2(31:20)$ of the first and second entries are compared with the same portion of the incoming IP address. The AND operations corresponding to the first and second entries are calculated as follows. For the first entry, since k divides $Prefix_1$, the corresponding AND operation has $\frac{15}{5} = 3$ inputs. However, the second entry, which has prefix size of 12, requires a 4-input AND operation. These four inputs are composed of the output ports of the first $\left\lfloor \frac{12}{5} \right\rfloor = 2$ decoders and the $12 \bmod 5 = 2$ ending bits of the $IP_2(31:20)$.

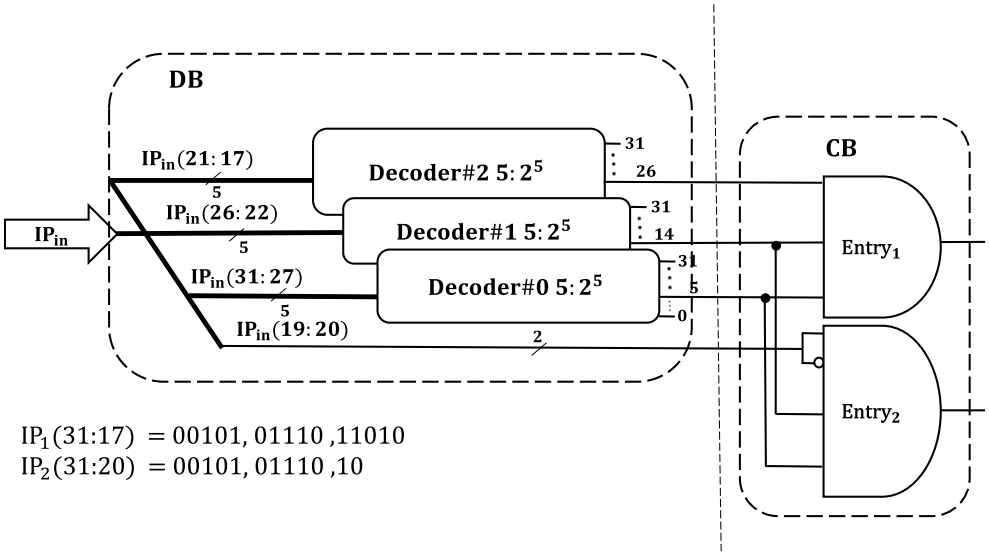

Figure 3.6: Example on AND operations of the comparator block

To perform the LPM algorithm, the AND operations are sorted based on their prefix size in a descending order. The result of l^{th}AND operation is one bit that determines if there is a match of

the incoming IP address with the l^{th} entry of the FIB or not. The 1-bit outputs of the N AND operations are sent to the PEB for further processing.

3.3.3 Priority Encoder Block

The PEB receives the N-bit sorted comparison result of CB based on the prefix size. Therefore, the match with the lowest address determines the output of the LPM algorithm. The PEB receives an N-bit input and finds the address of the first occurred match i.e. $Addr_{Match}$. It also determines if there was at least one match found in the FIB using an output variable called $valid$.

3.3.4 Next Hop Information Block

NHIB is the last block of the IP-Split architecture that determines the output of the ALE. NHIB uses RAM to store the NHI corresponding to every entry of the FIB. The input $Addr_{Match}$ determines the address of the NHI_{Match} inside the RAM.

3.4 IP-Split-Bucket Architecture

The IP-Split-Bucket is the final proposed ALE architecture that is an upgraded version of the IP-Split architecture. It is a novel, scalable, high-performance and memory-less architecture for real-time IP address lookup. The complete IP-Split-Bucket architecture, shown in Figure 3.7, consists of six pipeline stages. It is divided into three main blocks while its MB consists of two sub-blocks: DB and CB. The architecture of DB and NHIB are identical to these in the IP-Split architecture. The functionalities of the remaining blocks are described in the sections 3.4.1 and 3.4.2.

3.4.1 Comparator Block

The IP-Split-Bucket architecture employs a partitioning scheme similar to certain existing work [11]. Accordingly, the IP-Split-Bucket partitions the FIB into a predefined number of buckets using a *bucket identifier*. A bucket identifier is a predetermined n-bit portion of the IP address ($IP(BI_s{:}\,BI_e)$). The BI_s, BI_e indicate the starting and ending indexes of the bucket identifier, respectively. All IP addresses with the same bucket identifier value belong to the same bucket, and the total number of buckets is 2^n. The distribution of the IP addresses among the buckets depends on the FIB and the bucket identifier. An effective bucket identifier should partition the

IP addresses uniformly, leading to a balanced distribution with $\cong \frac{N}{2^n}$ IP addresses in each bucket. As shown in Figure 3.7, the CB consists of 2^n parallel components, called *CBucket*. Each CBucket compares the decoded incoming IP address with the IP addresses of the corresponding bucket using several AND operations. The architecture of every CBucket is identical to the architecture of the CB in the IP-Split architecture.

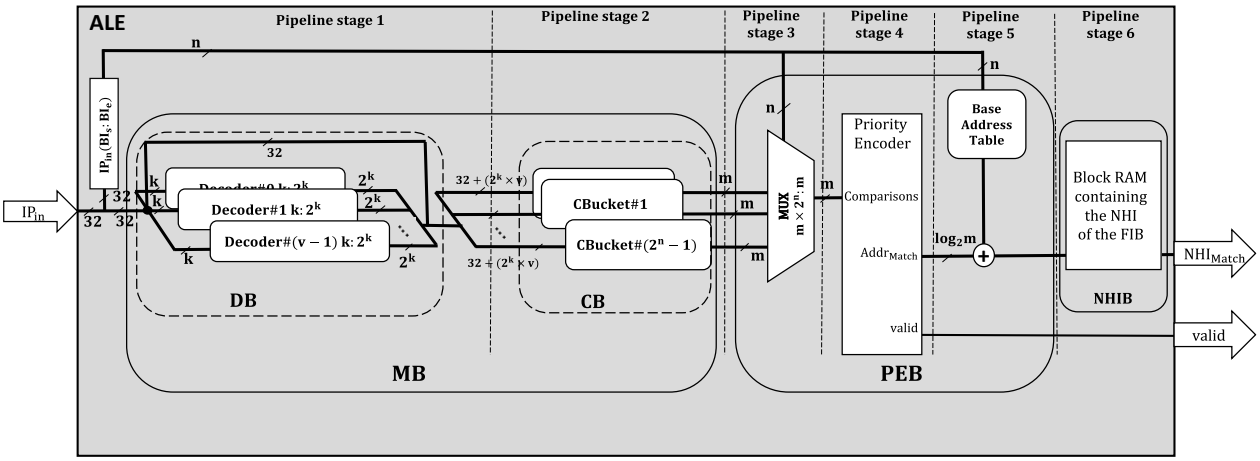

Figure 3.7: IP-Split-Bucket architecture

3.4.2 Priority Encoder Block

The bucket identifier of the incoming IP address ($IP_{in}(BI_s: BI_e)$, in Figure 3.7) defines the CBucket that contains useful comparison results. Therefore, a multiplexer is applied that passes the outputs of the appropriate CBucket based on the value of $IP_{in}(BI_s: BI_e)$. The multiplexer has $m \times 2^n$ inputs, where

$$m = MAX\big(size(CBucket\#b)\big), \qquad b = 0,1, \dots, 2^n - 1, \tag{3.2}$$

and $size(CBucket\#b)$ is the total number of IP addresses that have been allocated to the b^{th} bucket.

Next, a priority encoder receives all the comparison results of the appropriate CBucket. It passes the local address ($Addr_{Local}$) of the match with longest prefix size in the appropriate CBucket. To calculate the location of the match in the FIB, it is required to calculate the global match address ($Addr_{Global}$) using the following equation:

$$Addr_{Global} = Addr_{Local} + BaseAddr_b, \quad b = 0, \dots, 2^n - 1, \tag{3.3}$$

where the match occurred in the $CBucket\#b$. The address calculation in (3.3) corresponds to adding the local address of the match found by the priority encoder with the base address of the appropriate bucket. Since a fixed-number of entries of the FIB are assigned to each bucket, the base address of each bucket is a constant value stored in Base Address Table. The entries of the Base Address Table are found with (3.4):

$$BaseAddr_b = \sum_{i=0}^{b-1} size(CBucket\#b), b = 1,2, \dots, 2^n - 1, \tag{3.4}$$

where $BaseAddr_b$ is the base address for the b^{th} bucket, and $BaseAddr_0 = 0$.

3.5 Update-Enabled IP-Split-Bucket Architecture

In this section, we enhance the IP-Split-Bucket architecture so that is supports FIB updates. This architecture supports all three types of updates, additions (A), modifications (M) and deletions (D), by the three parallel modules shown in Figure 3.8. The update module supports A and the two modified IP-Split-Bucket modules support D and M. The update module is a small ALE with a FIB size of S containing all the new additions (A). The value of S depends on the update rate and the throughput of the system. The update module can be implemented using a TCAM or a Trie data structure. The remaining two parallel modules are a modified version of IP-Split-Bucket shown in Figure 3.9. The modified IP-Split-Bucket architecture consist of the three blocks MB, PEB and NHIB along with an additional block PrefixB. PrefixB stores the prefixes of the FIB in a memory. It passes the $Prefix_{Match}$that is the prefix of the match, to the output. The $Prefix_{Match}$ is required in the global updatable IP-Split-Bucket to find the global match.

When an update type of D is requested for an entry, the prefix size and the NHI of the corresponding entry are modified in the PrefixB and NHIB, respectively. Delete of an entry from the FIB is similar to replacing the prefix and the NHI of the corresponding entry with the prefix and the NHI of the previous longest prefix match found in the FIB. For instance, suppose $IP_{in} =$ "00101011101101011010111010101" and the FIB has the following two entries: $IP_1 =$ "00101011101101000000000000000000", $Prefix_1 = 15$, $NHI_1 = 8$, $IP_2 =$ "00101011101100000000000000000000", $Prefix_2 = 12$, and $NHI_2 = 19$. Two matches are found in the FIB that the first entry determines the LPM output. If a D of $IP_1$is requested in the FIB, the LPM algorithm determines IP_2 as the match with the largest prefix size and returns the

NHI of IP_2 After a D of IP_1, the NHIB replaces the NHI_1 by the NHI_2 to return correct NHI_{Match}. Moreover, the PrefixB replaces $Prefix_1$ by $Prefix_2$ to return correct $Prefix_{Match}$. This change of entries in NHIB and PrefixB is similar to a deletion of the first entry.

When an update type of M is requested for an entry, only the NHI of the corresponding entry is modified in the NHIB. A modification requires only change of the corresponding NHI.

For a new FIB, all the data set dependent blocks require a re-synthesis, which is all blocks except the PEB. However, an update does not require a re-synthesis and can be handled at its request time. Two modified IP-Split-Bucket modules are applied to keep the system active during re-implementation and maintain the high throughput of the architecture, where one is active at a time. The update module determines the active modified IP-Split-Bucket architecture. When its FIB size reaches its maximum value (S), a new sorted FIB is created containing all the recent updates (D, M and A). To perform an IP address lookup as shown in Figure 3.8, the incoming IP address is sent to the update module and the active modified IP-Split-Bucket module in parallel. IP address lookup is performed on both modules in parallel. Next, the arbiter module determines the NHI_{Match} of the whole system. The arbiter module selects the NHI_ias the NHI_{Match}, when $Prefix_i$ is larger than $Prefix_j$.

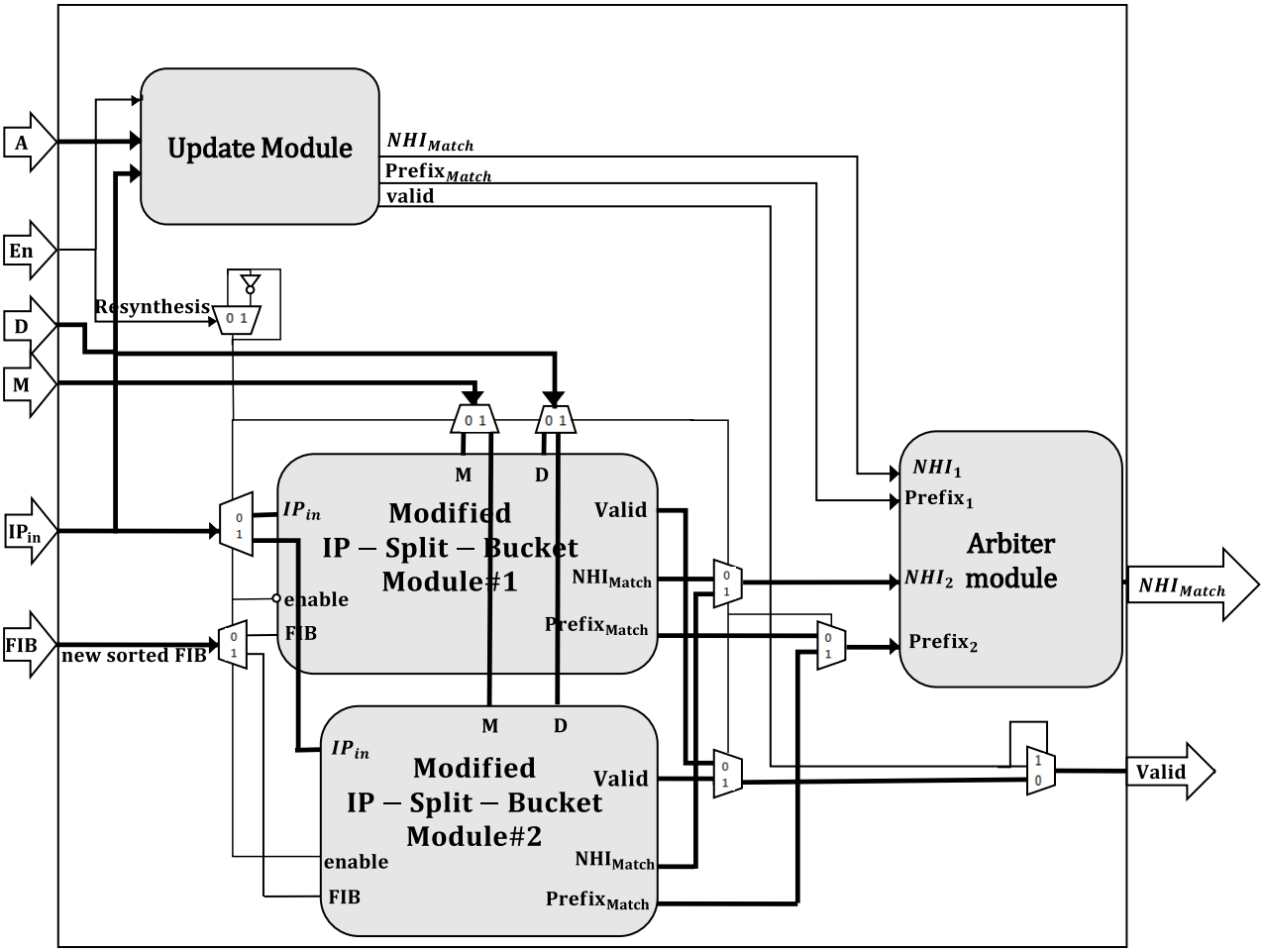

Figure 3.8: Update-enabled IP-Split-Bucket architecture

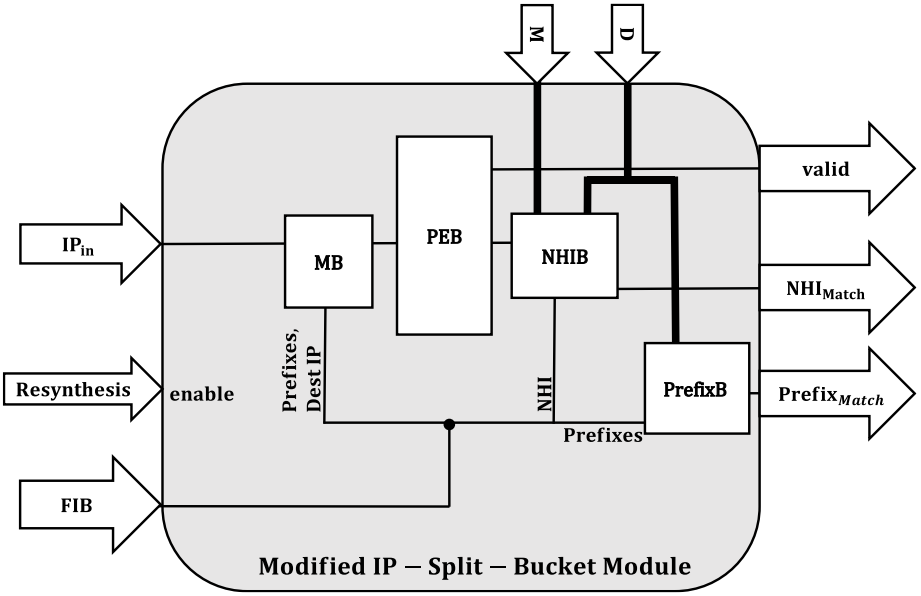

Figure 3.9: Modified IP-Split-Bucket architecture

CHAPTER 4 EXPERIMENTAL RESULTS AND DISCUSSION

This chapter presents the experimental results of the four proposed ALE architectures. The proposed architectures are evaluated in terms of hardware complexity and performance. Section 4.1 describes the synthesis results of hardware implementation of the Full-Serial architecture on FPGA. Section 4.2 provides the synthesis results of the Full-Parallel architecture on FPGA. Sections 4.3 and 4.4 present the detailed synthesis results of the IP-Split and the IP-Split-Bucket architectures on FPGA, respectively.

4.1 Full-Serial Architecture

The Full-Serial architecture was implemented for various sizes of real-world FIB extracted from Routing Information Services (RIS) raw data [8]. It was synthesized on a Virtex-5 XC5VLX50T FPGA using Xilinx ISE 13.4 synthesis tool. The Full-Serial architecture was implemented using two options: A and B. Figure 4.1 and Figure 4.2 illustrate the hardware consumption of both options in terms of LUTs and FFs, respectively. Option B has simpler architecture for its comparator than option A (see sub-sections 3.1.1 and 3.1.2). Therefore, it requires lower hardware consumption in terms of LUTs and FFs. Figure 4.3 shows a comparison of the clock period of this architecture using options A and B. Option B achieves better performance than option A.

In this architecture, all FIB is stored in on-chip Block RAM. Therefore, the determinative part of the resource consumption is the memory usage rather than the logical resource usage (LUTs and FFs). Consequently, the size of BRAMs in an FPGA will determine the maximum FIB size that can be supported by the Full-Serial architecture. Table 4.1 shows an estimation of the maximum size of FIB supported by various FPGAs using option A and option B. The last column of Table 4.1 shows the cost of each FPGA.

4.2 Full-Parallel Architecture

The Full-Parallel architecture was implemented on a Virtex-5 XC5VLX50T FPGA for a real-world FIB extracted from RIS raw data [8]. Table 4.2 illustrates the synthesis results of the Full-Parallel architecture for different FIB sizes up to 4 k using the Xilinx ISE 13.4 synthesis tool. The

results show that the number of LUTs and FFs increases linearly with the size of the FIB, whereas the clock period shows a sub-linear increase.

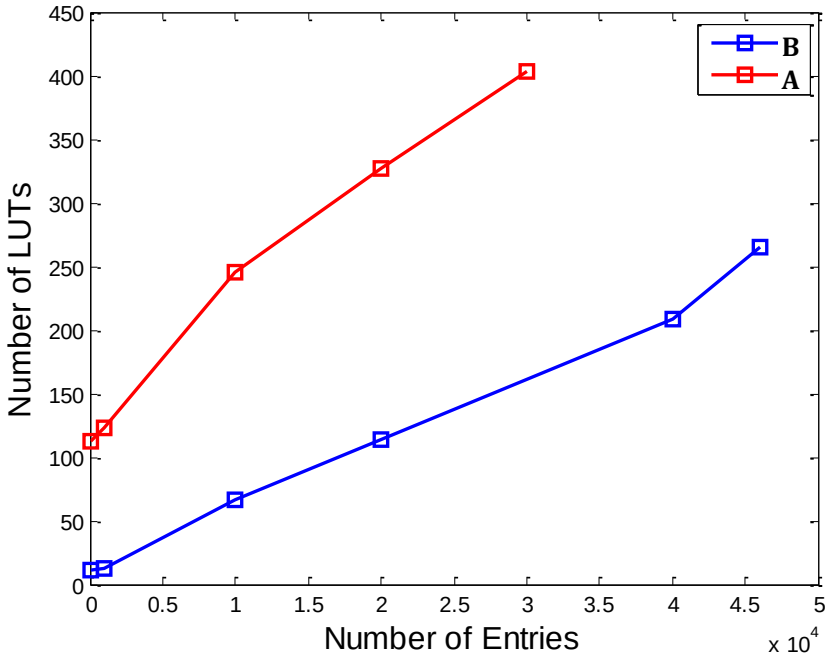

Figure 4.1: Comparison of option A and option B in terms LUTs utilization

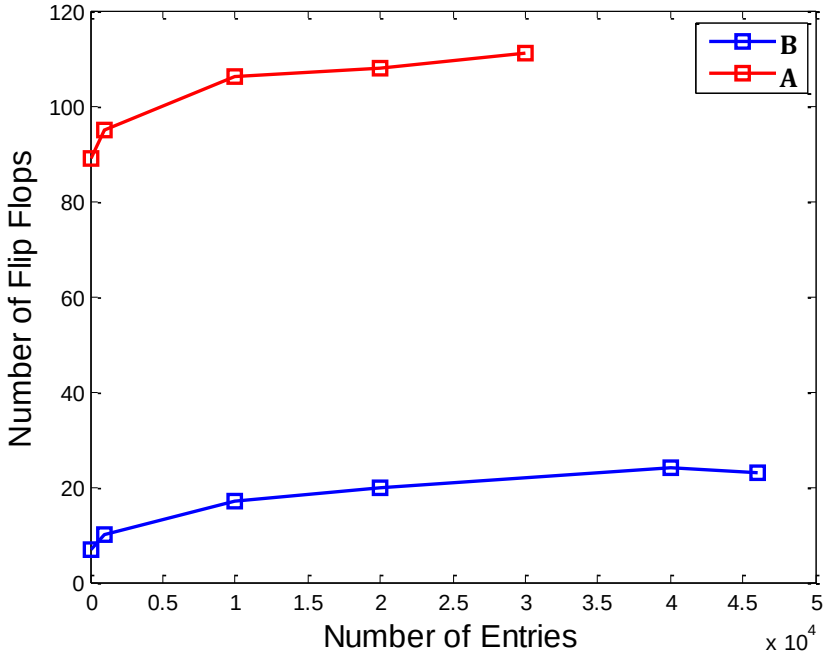

Figure 4.2: Comparison of option A and option B in terms FFs utilization

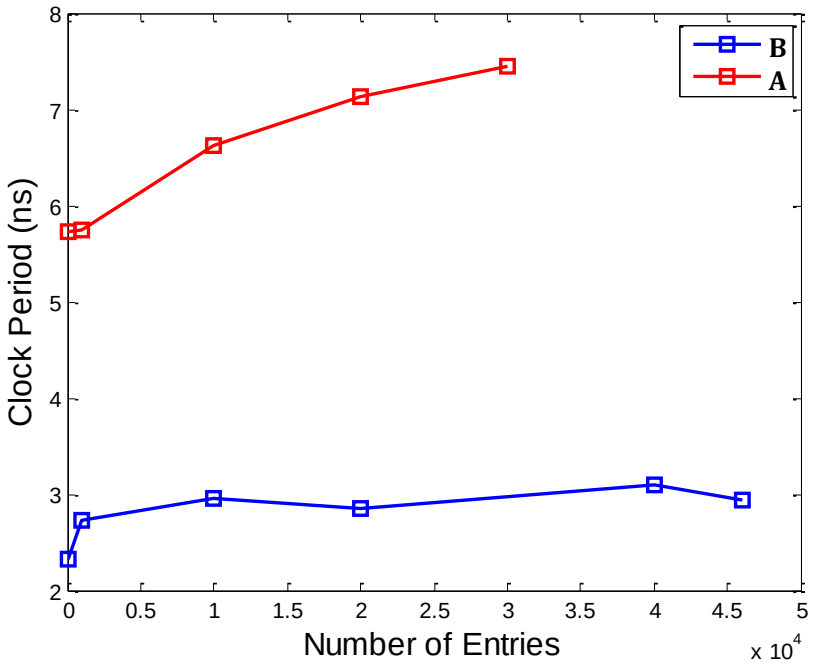

Figure 4.3: Comparison of option A and option B in terms of clock period

Table 4.1: Maximum FIB size supported for Full-Serial with FPGAs

FPGA	Block RAM (Kb)	Max N Supported (Option A)	Max N Supported (Option B)	Unit Price ($CAD)
Virtex 5 : XC5VLX50T	2160	30 k	46 k	750
Virtex 5 : XC5VFX200T	16416	228 k	356 k	8500
Kintex 7 : XC7K480T	34380	477 k	747 k	4750
Virtex 7 : XC7VX1140T	67680	940 k	1471 k	23000
Kintex UltraScale : KU115	75900	1054 k	1650 k	unavailable
Virtex UltraScale : VU190	132900	1845 k	2889 k	unavailable

In the Full-Parallel architecture, all FIB entries (known as $cells$ in section 3.2) are stored in the logical resources of the FPGA. Moreover, there are N parallel comparators in the MB implemented in logical resources. Therefore, the number of LUTs in an FPGA determines the maximum supported size of the FIB for the Full-Parallel architecture.

Table 4.3 lists the largest Xilinx FPGAs of different families with their number of available LUTs and their price. In this table, an estimation on the maximum supported size of the FIB is provided for each FPGA.

4.3 IP-Split Architecture

The complexity of the IP-Split architecture is dependent on the predefined values of its design parameters shown in Figure 3.5: $v, k, (m_1 : m_N)$, N. Therefore, it is required to find optimal values for the design parameters. The value of $(m_1 : m_N)$ and N is dependent on the FIB, which is data set dependent. The value of (v, k) in the DB have an impact on the complexity of the CB. Apart from (v, k) values, the prefix distribution of the FIB determines the size of the AND operations in the CB. Since the objective is to find the most efficient design parameter values for the IP-Split architecture, we conducted a design space exploration for (v, k) to estimate the complexity of the CB for an existing FIB. Figure 4.4 illustrates the prefix distribution of an existing IPv4 FIB extracted from RIS raw data that corresponds to the normal prefix distribution of the existing FIBs [8].

Table 4.2: Full-Parallel synthesis results for different sizes of FIB on Virtex-5

N	LUTs	FFs	Clock Period	LUTs/N
31	0.29 k	0.10 k	2.7	9.2
128	1 k	0.19 k	2.8	9.1
1 k	8 k	1 k	2.8	8.3
2 k	16 k	2 k	2.9	8.0
4 k	34 k	4 k	2.9	8.3

Table 4.3: Maximum FIB size supported for Full-Parallel with FPGAs

FPGA	LUTs	Max N Supported	Unit Price ($CAD)
Virtex 5 : XC5VLX50T	28 k	3 k	0.75 k
Virtex 5 : XC5VLX330T	207 k	23 k	18 k
Kintex 7 : XC7K480T	298 k	33 k	4 k
Kintex UltraScale : KU115	663 k	73 k	unavailable
Virtex 7 : XC7VX1140T	712 k	79 k	23 k
Virtex 7 : XC7V2000T	1 M	135 k	25 k
Virtex UltraScale : VU440	2 M	281 k	unavailable

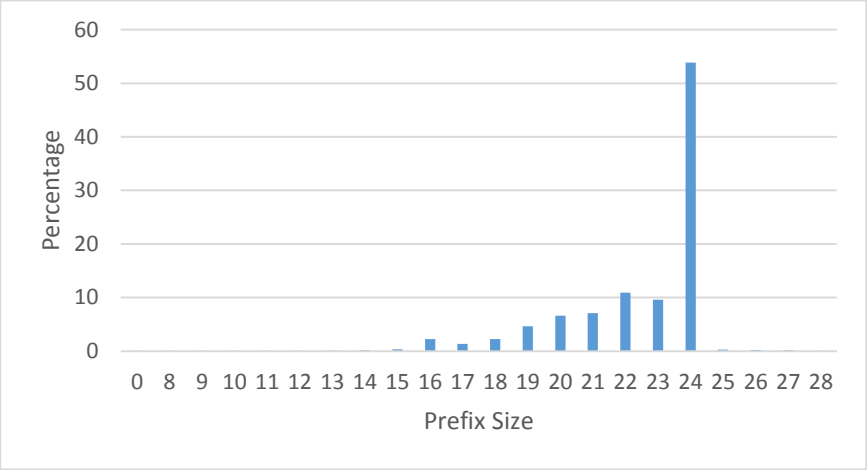

Figure 4.4: Prefix distribution of a real-world IPv4 FIB [8]

We propose a 2-step estimation process on the complexity of the CB using the prefix distribution shown in Figure 4.4 and a specific value of (v, k). In the first step, we evaluate the size of all AND operations using Equation 3.1. In the second step, the number of consumed LUTs for every AND operation is estimated. It is assumed in the second step of estimation that recent FPGA families with 6-input LUTs are used. For example, Table 4.4 shows an estimation of the complexity of the CB when $k = 7$ and $v = 4$. The first and the second columns show the prefix size and the repetition of that prefix size in the FIB. The third column shows the size of every AND operation calculated by Equation 3.1. The third column stores the total number of

consumed LUTs for all IP addresses with that specific prefix size. The calculation of the values stored in the fifth row of Table 4.4 is described below:

- The prefix size is equal to 19.
- The repetition is the number of IP addresses with prefix size equal to 19. It is equal to 27033 as shown in Figure 4.4.
- The size of AND operation for an IP address with prefix size of 19 is equal to $(\left\lfloor \frac{19}{7} \right\rfloor + 19 \bmod 7 = 7)$ according to Equation 3.1.
- The number of consumed LUTs is equal to $(27033 \times 2 = 54066)$, since every AND operation of 7-bits requires two 6-input LUTs.

The total number of consumed LUTs for design parameters $k = 7$ and $v = 4$ is equal to the summation of the last column of Table 4.4.

We conducted a design space exploration on the decoder size by applying the 2-step estimation process. We proposed 32 test cases that estimate the LUT consumption of the CB for different size of decoders as shown in Figure 4.5. For every test case, the size of decoders (k) is incremented by one and the value of v for is equal to $\left\lfloor \frac{32}{k} \right\rfloor$. The estimation results illustrate that the optimal values for (v, k) among the test cases are (3, 10) that lead to the lowest LUT consumption.

As shown in Figure 4.4, most of the IP addresses in a FIB have prefix size of 24. For such prefix size, when (v, k) is (3, 10), we require a 6-input AND operation $(\left\lfloor \frac{24}{10} \right\rfloor + 24 \bmod 10 = 6)$. However, we require a 3-input AND operation $(\left\lfloor \frac{24}{8} \right\rfloor = 3)$ for (4, 8). Therefore, the choice (4, 8) results in a simpler AND operation compared to (3, 10). Moreover, the estimation results shown in Figure 4.5 does not show a significant difference in their LUT consumption. Therefore, we have applied four decoders of 8-to-2^8 in all our experiments.

The IP-Split architecture was synthesized on a Virtex-5 xcvlc50t FPGA with Synplify 15.09 synthesis tool for various sizes of real-world FIBs extracted from the RIS raw data [8]. The hardware description of the IP-Split architecture is more complex compared to the previous ones, and thus, the Synplify tool is used instead of Xilinx in this section to save the synthesis time for

large FIB size. To evaluate the complexity of every block in terms of resource usage and performance, the synthesis results of every block is presented in following sections.

Table 4.4: Estimation of the number of used LUTs while applying 4 of 7-to-2^7 decoders

Prefix Size	Repetition	Size of AND operation	LUTs
15	1792	3	1792
16	13080	4	13080
17	7817	5	7817
18	13110	6	13110
19	27033	7	54066
20	38278	8	76556
21	41172	3	41172
22	63423	4	63423
23	55974	5	55974
24	313482	6	313482
25	1428	7	2856
26	1221	8	2442
27	765	9	1530
28	199	4	199
29	268	5	268
30	295	6	295
31	29	7	58
32	527	8	1054

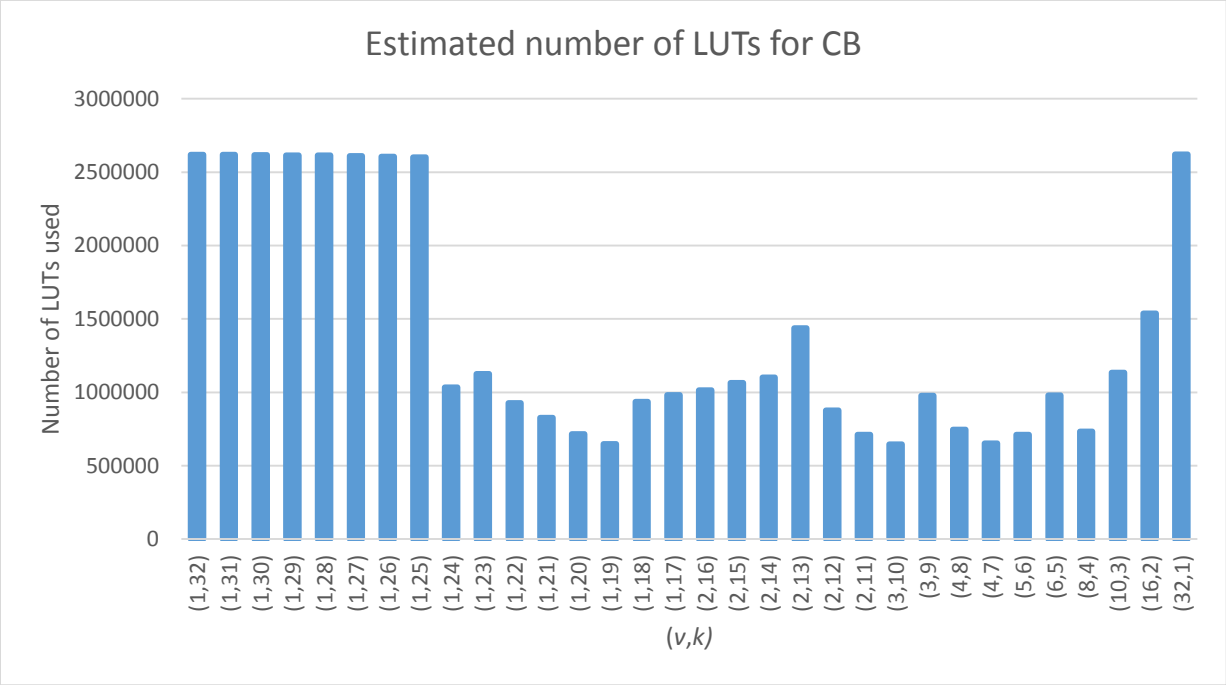

Figure 4.5: Design space exploration for the size of decoders

4.3.1 Synthesis Results of the Decoder Block

This section presents the synthesis results of the DB. The DB consists of four 8-to-256 decoders. Table 4.5 illustrates the hardware complexity of the DB in terms of LUT and FF consumption. The last column of Table 4.5 shows that this block supports a clock frequency of 1191 MHz. Since the complexity of the DB is not dependent on the FIB size, N is not a metric in the presented results. Therefore, the IP-Split architecture requires a small, fixed-size DB regardless of the FIB size.

Table 4.5: Synthesis results of the DB

DB	LUTs	FFs	Clock Period (ns)
4 Decoders of 8-to-256	1044	1024	0.839

4.3.2 Synthesis Results of the Comparator Block

This section presents the synthesis results of the CB for different FIB sizes. Figure 4.6 shows the number of consumed LUTs and FFs of this architecture. Based on the theoretical analysis of the CB growth, it is estimated that a FIB with around 500 k entries requires 652 k FFs and 586 k LUTs in its CB. Figure 4.7 illustrates the clock period of the DB that has a near-linear increase with the size of the FIB.

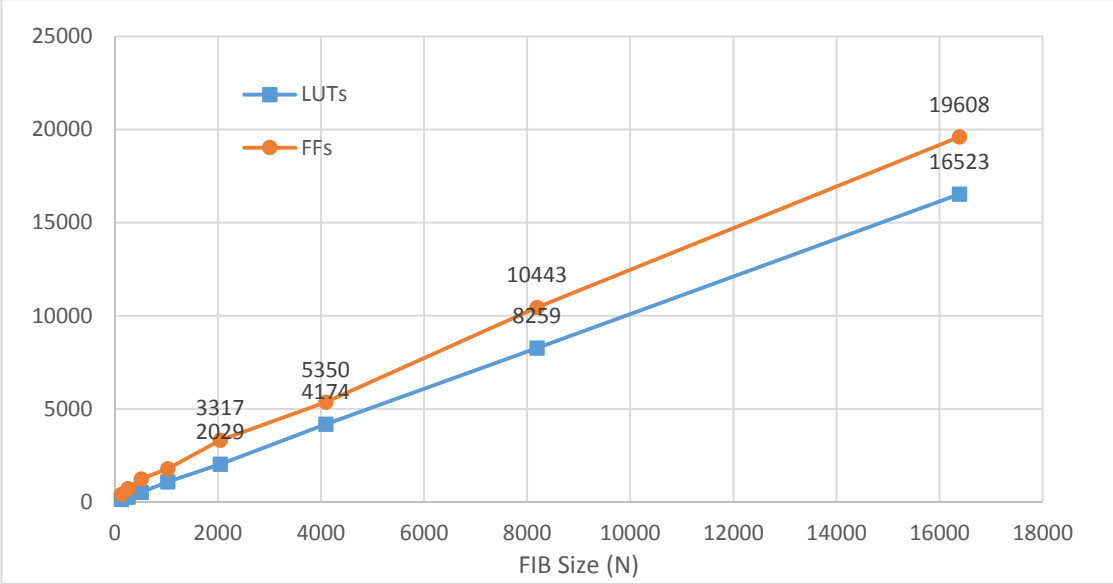

Figure 4.6: Hardware resource usage of the comparator block

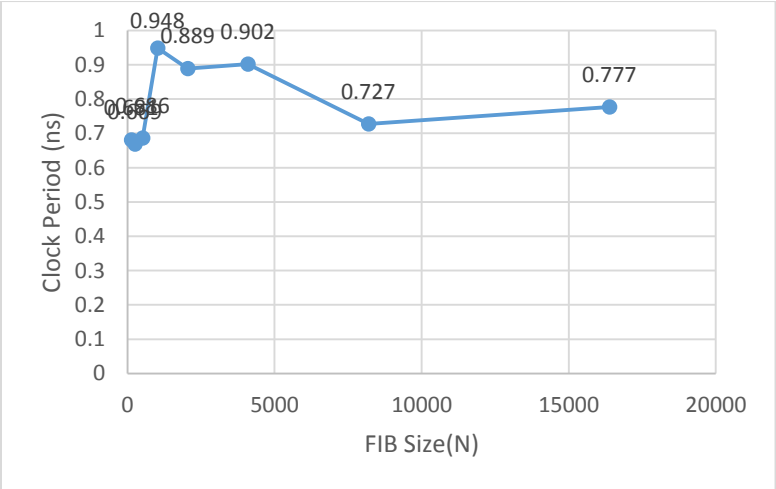

Figure 4.7: Clock period of the comparator block

4.3.3 Synthesis Results of Priority Encoder Block

This section presents the synthesis results of the PEB for various FIB sizes. The priority encoder has N inputs where N is equal to the number of FIB entries. Therefore, the size of the PEB increases linearly with the size of the FIB. Figure 4.8 illustrates the synthesis results of the PEB in terms of LUT and FF consumption. Based on the theoretical analysis of the PEB growth, the PEB consumes 582 k FFs and 1 M LUTs for a FIB size around 500 k. Figure 4.9 shows the clock period of the PEB for different FIB sizes. PEB is the bottleneck of IP-Split architecture in terms of clock period. According to the synthesis results, it is necessary to minimize the size of the PEB for large FIBs, which is the main goal of the IP-Split-Bucket architecture.

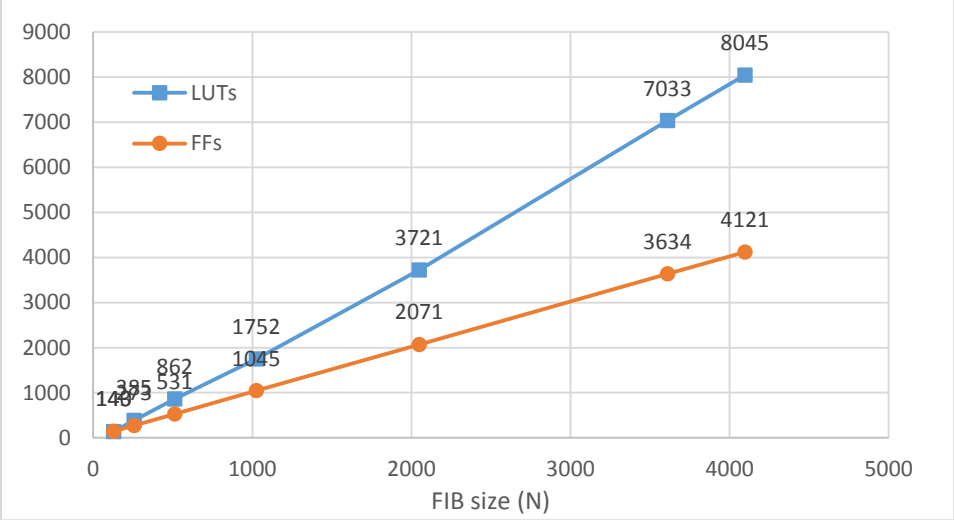

Figure 4.8: Hardware resource usage of the priority encoder block

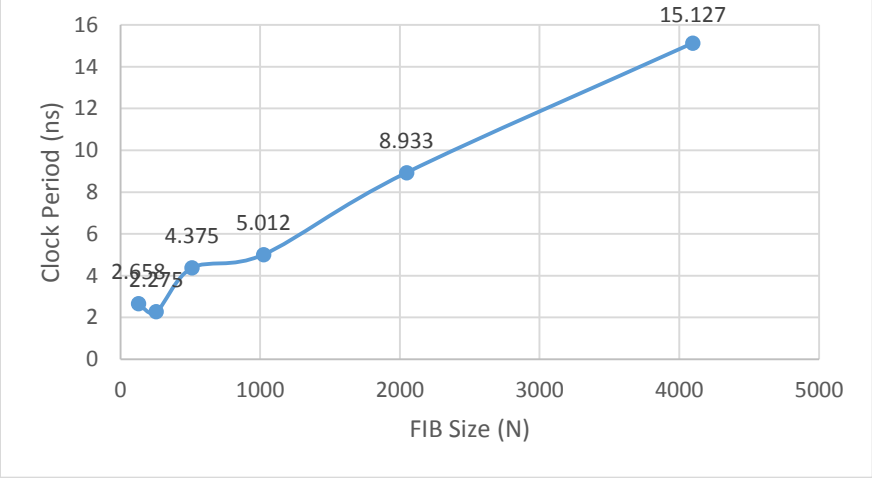

Figure 4.9: Clock period of the priority encoder block

4.3.4 Synthesis Results of the NHIB

The NHIB is implemented in a BRAM of size $N \times 8$ bits containing 8-bit output ports. Therefore, a FIB with around 500 k entries requires an FPGA with ability to support 4 Mb block RAMs. The NHIB consumes neither LUTs nor FFs.

4.3.5 Discussion

The IP-Split architecture provides some improvements compared to previously proposed architectures. It proposes deterministic latency with high performance, which is supported by the Full-Serial architecture. Furthermore, the IP-Split architecture avoids repetitive redundant comparisons handled in the Full-Serial and the Full-Parallel architectures. However, some drawbacks still exist in the IP-Split architecture in the matter of area consumption for large FIBs. It is required to minimize the resource usage of the IP-Split architecture to be able to fit the whole architecture inside the existing FPGA. Therefore, it is required to minimize the size of each block:

1. The DB requires no modification or change, since it is static for every FIB size and has negligible resource usage compared to the other blocks.
2. The CB is a large block containing simple AND operations that its size is dependent on the FIB size.
3. The PEB is the bottleneck of the design for large FIBs. Finding the match with largest prefix size among N inputs is the main challenge of the IP-Split architecture. For instance, to support a FIB with 500 k entries, it is required to synthesize a priority encoder with 500 k inputs.

We need to minimize the LUTs and FFs usage of each block to be able to fit the whole design inside an existing FPGA. The PEB is the main challenging block among all the blocks. It is the bottleneck of the design in terms of clock period. Therefore, to respect the network constraints in terms of throughput we need to minimize the clock period of the PEB. For that matter, the IP-Split-Bucket architecture is proposed. It minimizes the size of the problem by dividing the whole FIB into buckets based on the value of the certain portion of the IP addresses. Accordingly, the CB consists of several buckets where only one bucket contains the viable match result at a time

for each incoming IP address. Therefore, the size of the PEB will decrease from the whole FIB size to the size of the maximum bucket.

4.4 IP-Split-Bucket Architecture

In this section, we used a different FPGA for all IP-Split-Bucket experiments compared to the previously proposed architectures. A Virtex-7 FPGA was employed, due to its higher performance and larger capacity compared to the Virtex-5 family. For large N FIBs implemented on the IP-Split-Bucket architecture, the Synplify optimization tool leads to an internal error. The largest FIB size supported with the Synplify synthesis tool was 16 k for the IP-Split-Bucket architecture. Therefore, we used the Xilinx ISE synthesis tool for the IP-Split-Bucket experiments. The target IPv4 FIB used for all experiments was extracted from RIS raw data [8]. Four set of experiments were performed for the IP-Split-Bucket architecture. In the first set of experiments described in section 4.4.1, the 6-level pipelined architecture of IP-Split-Bucket was implemented for different FIB sizes. In the second set of experiments presented in section 4.4.2, the IP-Split-Bucket architecture is compared with some existing trie-based and CAM-emulation approaches. In the two remaining set of experiments, the objective is to find an efficient set of values for the design parameters (BI_s, n, v, and k) of the IP-Split-Bucket architecture, since their values affect its complexity. In the third set of experiments described in section 4.3.3, efficient values for BI_s and n are selected by evaluating the resource utilization of the design for various values of N, BI_s and n. In the fourth set of experiments presented in section 4.4.4, a decoder generator is proposed to find efficient values the remaining design parameters. Therefore, it determines a series of decoders in the DB using the values selected in section 4.4.3 for BI_s and n.

4.4.1 Synthesis Results of the IP-Split-Bucket Architecture

In the first set of experiments, the hardware area and the performance achieved with the IP-Split-Bucket architecture were evaluated using different IPv4 FIB sizes (N). In order to achieve a uniform distribution of IP addresses in the buckets, the bucket identifier was chosen based on the approach proposed by Zheng et al. [11]. In this experiment, we chose $n = 8$ that partitions the FIB into 256 buckets. Using Zheng's approach for 256 buckets, bits 9 to 16 were found to be the most appropriate bucket identifier ($BI_s = 9$ and $BI_e = 16$). Figure 4.10 shows the distribution of IP addresses when Zheng's approach is applied for 256 buckets ($n = 8$) on a FIB size of

581851 extracted from RIS raw data [8]. In Figure 4.10, the x-axis determines the ID group number of each bucket and the y-axis determines the number of IP addresses stored in the corresponding bucket.

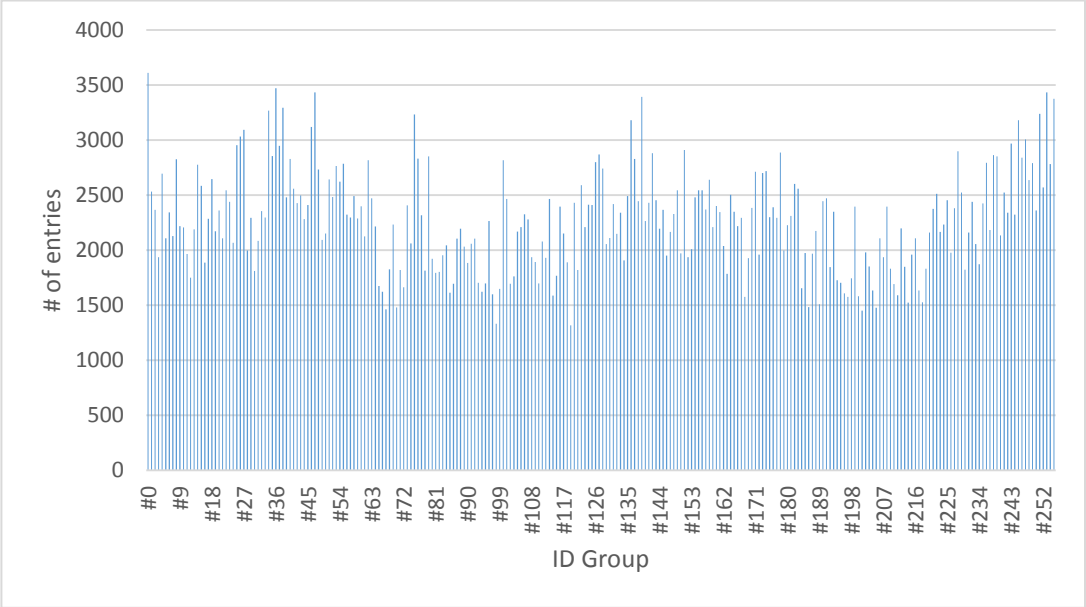

Figure 4.10: IP address distribution into 256 buckets (BI_s = 9, BI_e=16)

The remaining design parameters have the following values: $w = 32$, $k = 8$ and $v = 4$. Since $k = 8$ and $v = 4$, the second chosen segment is equivalent to the bucket identifier, the second decoder in DB is redundant and can be removed from the design. Indeed, the target IP addresses are distributed among the buckets based on the bucket identifier, i.e., the second chosen segment in this case. Hence, the comparisons of the second segment (bits 9 to 16) are implicitly performed by the described partitioning scheme.

The hardware area consumption of the IP-Split-Bucket architecture depends on the target FIB size. Therefore, the total available resources in the target FPGA determine the largest supported FIB size. Table 4.6 shows the area consumption expressed in terms of the number of LUTs and FFs for various FIB sizes. The number of consumed LUTs increases almost linearly with the problem size, whereas the number of consumed FFs shows a sub-linear increase. The proposed architecture was evaluated using a target FIB of up to 524 k entries. The results show that implementing the largest FIB on a XC7V2200T FPGA consumes 23% and 22% of the available LUTs and FFs, respectively. Since the 6-level pipelined IP-Split-Bucket architecture provides

one result every clock cycle, the frequency column, shown in Table 4.6, can be considered as the throughput of the architecture.

Table 4.6: Synthesis results of IP-Split-Bucket architecture for different FIB sizes on Virtex-7

N	m	Frequency (MHz)	Number of LUTs	Number of FFs	Latency (ns)
4 k	40	246	6.6 k	5 k	24.3
8 k	59	245	12 k	9 k	24.4
16 k	115	189	23.7 k	18 k	31.6
18 k	126	184	25.9 k	19.7 k	32.5
20 k	143	181	28.6 k	21.7 k	33.0
32.7 k	220	159	46.8 k	35 k	37.6
65.5 k	427	136	93 k	69 k	43.8
90 k	591	120	102 k	69.8 k	49.7
131 k	846	119	115.9 k	70 k	50.1
524 k	3316	103	282.3 k	549.7 k	57.9

4.4.2 Comparison of IP-Split-Bucket Architecture and Existing Work

This section compares some existing trie-based and CAM-emulation approaches with the IP-Split-Bucket architecture. Table 4.7 demonstrates the FPGA synthesis results of various designs using CAM-emulation and trie-based techniques.

We compare the IP-Split-Bucket architecture with two existing memory-less CAM-emulation approaches [21], [29]. Xilinx developed an FPGA-based TCAM core that can be configured to use either SRLs or RAMs [29]. Since IP-Split-Bucket avoids using memory resources, we compared it with the 32-bit wide SRL-based TCAM core as shown in the sixth column of Table 4.7. The IP-Split-Bucket architecture was evaluated by implementing test cases of comparable sizes on XC5VLX220 FPGAs. The results show that, for a FIB with 1024 IP prefixes, the proposed architecture consumes 83.4% fewer LUTs and offers 3.5× higher throughput compared to the Xilinx TCAM core [29]. Clark and Schimmel [21] evaluated their design using the largest

test cases among BCAM-emulation works. Therefore, we implemented a comparable table size of 18 k on the IP-Split-Bucket. Table 4.7 shows that the IP-Split-Bucket consumes 39% and 64% fewer LUTs and FFs, respectively.

Three Trie-based approaches are included in Table 4.7 [15], [16], [17]. Matoušek's [16] paper presented the work that is most recent and most comparable work to ours. We implemented an equivalent table size of 442.7 k on XC6VSX475T FPGA, using different tested FIBs. The results show that while the IP-Split-Bucket architecture does not use memory resources, Matoušek's approach consumes 7.7 Mb of internal memory. On the other hand, Matoušek's approach requires 85% and 90.5% fewer LUTs and FFs. The IP-Split-Bucket architecture achieves 90.8% lower latency compared to Matoušek et al [16] while having 54% lower throughput. Yang et al. [15] presented a memory-based Trie-based approach that supports a very large FIB (9.5 M entries). Although, Yang et al.'s approach [15] achieves better performance, it consumes a large amount of internal and external memories. Yang et al. have not reported the size of the utilized external memories, whereas the IP-Split-Bucket architecture avoids using any memory resources (~0 Kb).

4.4.3 The Size and the Starting Bit Selection for the Bucket Identifier

Two methods are proposed to find efficient values for BI_s and n for a FIB size of N to minimize the complexity of the IP-Split-Bucket architecture. In the first method, we estimate the IP-Split-Bucket complexity for various values of n and N and fixed size for BI_s. In the second method, an estimation function is proposed that finds efficient values of BI_s, n for a fixed size of N.

In the first method, the resource utilization was characterized for different values of n using a fixed value of BI_s. To improve efficiency, all buckets should contain approximately the same number of entries of the FIB. Therefore, it is required to choose a fixed value of BI_s for this method that leads to equalizing the bucket sizes. In order to achieve a uniform distribution of IP addresses in the buckets, the bucket identifier was chosen based on the approach proposed by Zheng *et al.* [11]. In [11], consecutive bits of 10 to 13 are chosen for the bucket identifier that leads to a normal distribution of IP addresses into 16 groups. The authors tested this bucket identifier for multiple real-world FIBs that leads to a uniform group division. Therefore, using Zheng's approach, we apply the fixed value of $BI_s = 10$.

Table 4.7: Detailed comparison of existing work with IP-Split-Bucket

Metrics / Approaches	Device	# of Patterns × # of char	LUTs	FFs	Memory (Kb)	Frequency (MHz)	Throughput (MLPS)	Latency (ns)	Technique
[17] **2006**	Virtex-2 XCVP100	602 × 20.4	6 k	6 k	~ 0	216	216	37	Trie-based
[15] **2011**	Virtex-6 SX475T	9.5 M × 32	7 k	22 k	28,044 (BRAM) + 4 External SRAMs	156	312	13	Trie-based
[16] **2013**	Virtex-6 XC6VSX475T	442,748 × 32	88 k	44 k	7780	127	254	472	Trie-based
[21] **2004**	Virtex2-8000	17,537× 32	55 k	55 k	~ 0	219	233	146	BCAM-emulation
[29] **TCAM core**	Virtex-5 XC5VLX220	1024 × 32	13 k	63	~ 0	80	80	12	TCAM-emulation
IP-Split-Bucket	Virtex-6 XC6VSX475T	442,747 × 32	593 k	464 k	~ 0	116	116	43	TCAM-emulation
	Virtex-5 XC5VLX220	1024 × 32	2.2 k	1.9 k	~ 0	362	362	13.5	
	Virtex-7 XC7V2000T	1024 × 32	2.3 k	1.9 k	~ 0	415	415	12	
		18,001 × 32	22 k	19 k	~ 0	184	184	27	
		524,287 × 32	282 k	550 k	~ 0	103	103	48	

There are nine test cases with different values of n changing in the range of 2 to 10. Changing the value of n does not significantly affect the hardware complexity of the DB, CB and NHIB blocks. The PEB is the main block that varies by changing n. The PEB consists of a multiplexer, a priority encoder and an adder. The adder combines two values of length $log_2 m$ bits. Since increasing n leads to a negligible growth of m, ignoring variations in resource utilization of the adder in the final estimation does not significantly change the result. Hence, the hardware complexity is estimated by summing up the LUT usage of the multiplexer and the priority

encoder for each tested value of n. The complexity of the multiplexer is also dependent on the value m, while the complexity of the priority encoder is only dependent on m. Therefore, we evaluate the maximum bucket size (m) for every test case and FIB size, shown in Table 4.8.

Figure 4.11.a and Figure 4.11.b show the LUT consumption of the multiplexer and priority encoder for the nine test cases, respectively. According to Figure 4.11.a, the multiplexer has a near-linear increase in the LUT usage as n increases. For a given number of table entries N, increasing the number of buckets 2^n means that the buckets will be smaller, hence smaller m. Since the priority encoder size only depends on the value of m, a smaller m leads to a smaller priority encoder. Figure 4.11.b shows that when n increases, the resource utilization of the priority encoder is reduced. Figure 4.12 compares the final hardware complexity for different values of n and N for the multiplexer and the priority encoder. These results show that changing the number of buckets from 16 ($n = 4$) to 128 ($n = 7$) has a limited impact on the overall LUT consumption. Among the test cases, the optimal choice for the bucket identifier size is $n = 5$, which minimizes resource utilization for almost all FIB sizes.

Table 4.8: Maximum bucket size (m) for variable test cases with variable FIB sizes

FIB Sizes	**Case #1**	**Case #2**	**Case #3**	**Case #4**	**Case #5**	**Case #6**	**Case #7**	**Case #8**	**Case #9**
	n = 2	**n = 3**	**n = 4**	**n = 5**	**n = 6**	**n = 7**	**n = 8**	**n = 9**	**n = 10**
N = 65536	17429	8741	4558	2504	1325	703	427	225	149
N = 32768	8729	4404	2282	1273	679	358	220	120	79
N = 18001	4748	2402	1281	705	379	199	126	69	45
N = 16384	4304	2178	1157	639	348	183	115	63	43
N = 8192	2123	1085	573	316	166	96	59	34	26
N = 4096	1083	544	306	165	97	53	40	19	15

In the second method, we vary the position of the bucket identifier starting index (BI_s) and for each one, we measure the complexity for various values of n. The CB and PEB blocks are the main parts of the architecture that vary when changing BI_s and n. Hence, the resource utilization

is estimated by summing up the LUT consumption of the CB and PEB blocks for each tested value of BI_s and n using a moderate size FIB (16 k).

We propose two steps of estimation for the CB and PEB. The first step estimates the complexity of the CB. The CB consists of $N + E$ number of AND operations, where N is the FIB size and E is number of expansions in the FIB. Every IP address with a prefix size ($prefix$) smaller than the ending index of a bucket identifier (BI_e) is expanded into E IP addresses with prefix size of BI_e. The number of expansions (E) is calculated by $2^{BIe-prefix}$. For instance, suppose $BI_s = 10$, $BI_e = 15$, $IP_1 =$ "234.84.212.153" and $prefix_1 = 13$. In such case, IP_1 requires an expansion into 2^{15-13}=4 IP addresses with prefix size of 15. According to the example, it is stated that the value of the BI_e determines E in a FIB. Since BI_e is equal to $BI_s + n - 1$, a prefix_expansion function is proposed that determines the value of E required for a FIB using the values of BI_s and n. Therefore, the complexity of the CB is estimated by adding the value of E and N. The second step estimates the complexity of the PEB that consists of an $m \times 2^n$multiplexer and an m–input priority encoder. The value of m is determined after an expansion and bucket division of the FIB for specific BI_s and n. Therefore, the summation of the LUT consumption of the multiplexer and priority encoder determines the complexity of the PEB.

The estimation function sweeps the value of BI_s and n in the ranges of (1:23) and (2:11), respectively and estimates the total LUT consumption of the PEB and CB. Next, the design parameters with the lowest value of total LUT consumption are selected as the optimal values. Figure 4.13 shows the design space exploration on BI_s and n for the moderate FIB size of 16383. The optimal value ($minCost$) occurs for a $BI_s = 9$ and $n = 7$.

Figure 4.14 is a zoomed-in section of Figure 4.13. In the reported synthesis results of the IP-Split-Bucket architecture in section 4.4.1, we applied the values of 9 and 8 for BI_s and n, respectively. Figure 4.14 demonstrates there is not a significant difference in the total estimated LUT consumption for a design with $BI_s = 9$ and $n = 8$ compared to the optimal point of $BI_s = 9$ and $n = 7$.

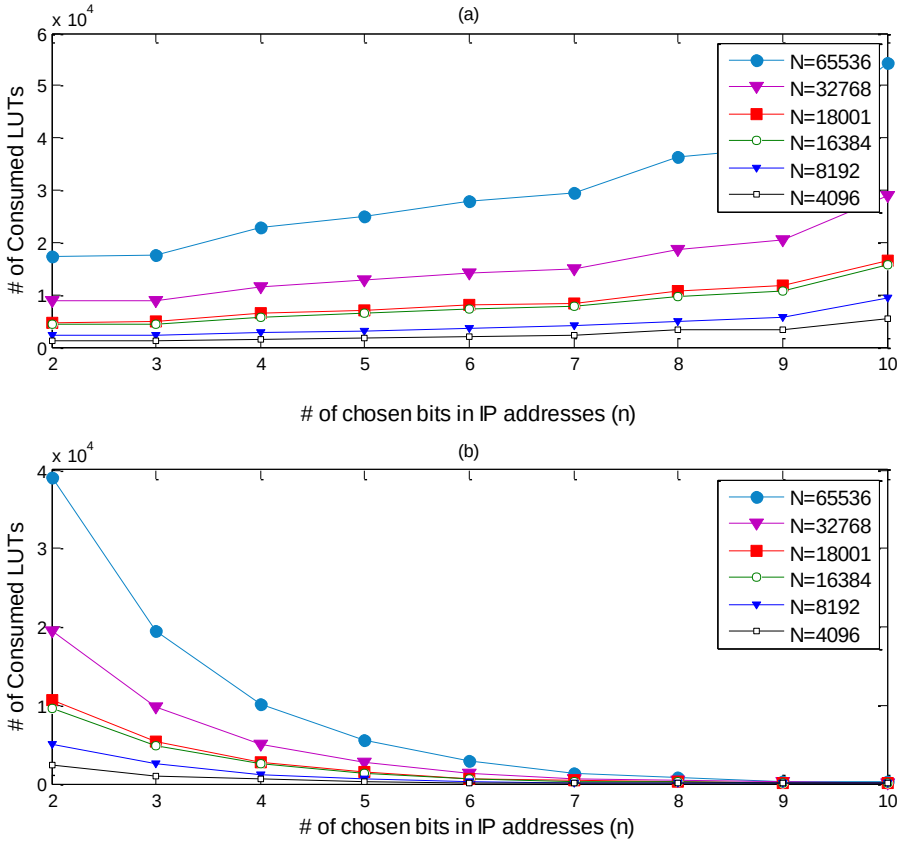

Figure 4.11: LUT consumption of multiplexer (a) and priority encoder (b) of the PEB

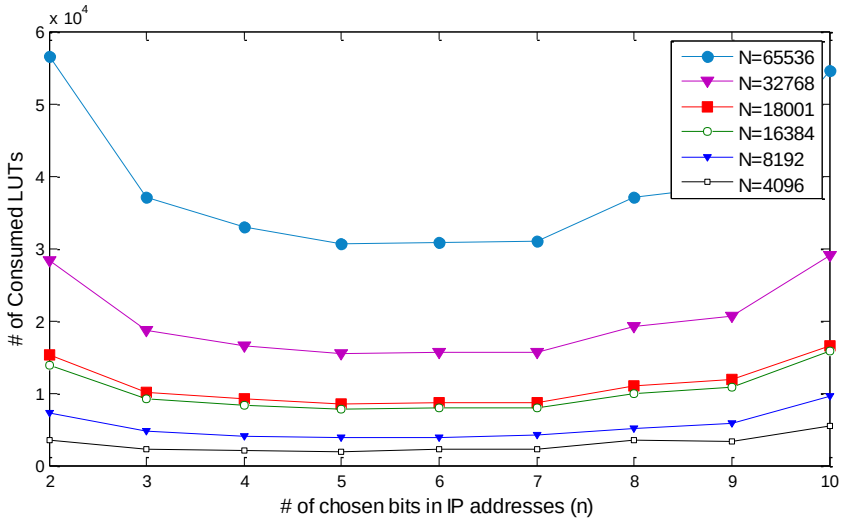

Figure 4.12: PEB resource utilization estimation as a function of n and m

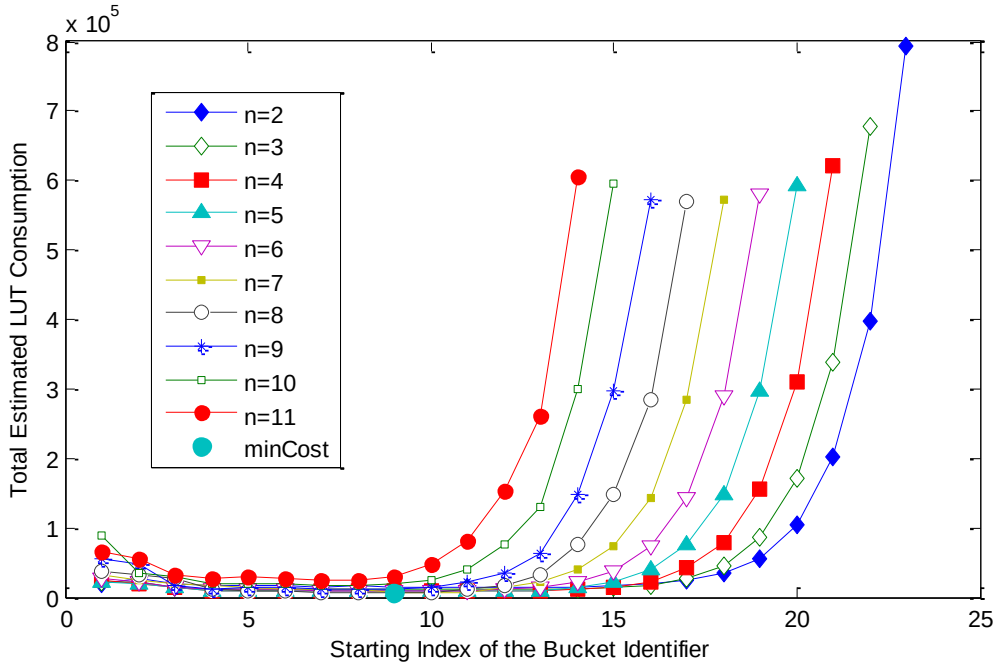

Figure 4.13: A design space exploration on BI_s and n

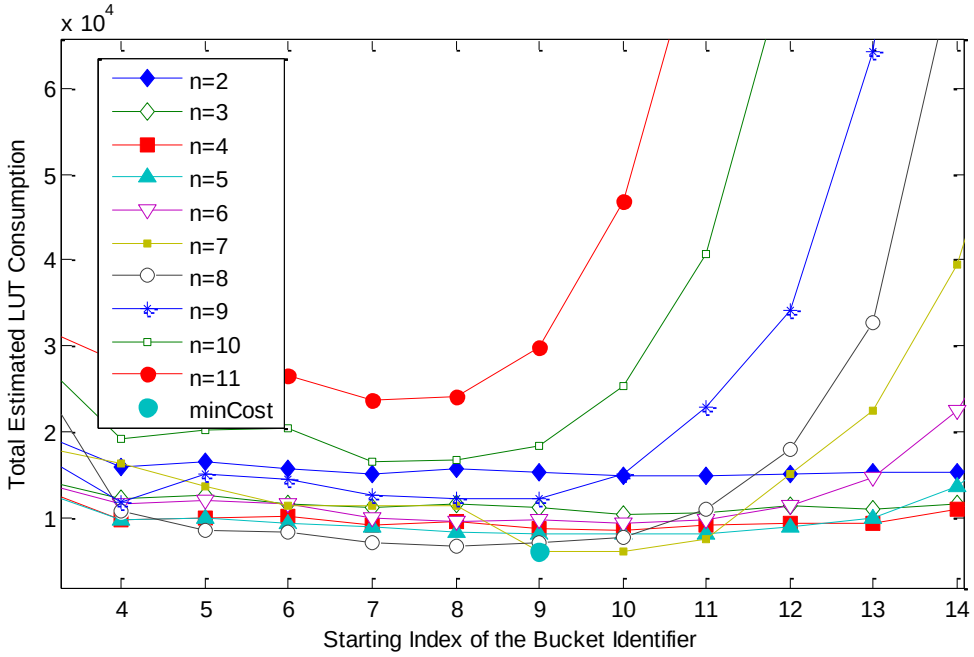

Figure 4.14: A zoomed-in section of the design space exploration on BI_s and n

4.4.4 Decoders Selection

Using the optimal values of BI_s and n obtained in section 4.4.3, we searched for a series of decoders for the IP-Split-Bucket architecture that lead to the lowest total LUT consumption. A random decoder generator is proposed that generates an array (dec) containing a random number of decoders with random sizes. In the random decoder generator, it is assumed that the summation of the decoder sizes ($SUM(dec)$) and the bucket identifier size (n) should be equal to the size of the IPv4 address (32). An array ($rand$) with 16 entries and for which each entry has a random value between 2 to 10 is generated. The first d entries of the generated array construct the dec, where the summation of the d entries ($SUM(rand(1:d))$) is equal to $32-n$. If the summation of the first d entries is more than $32-n$, $d-1$ first entries and a value of $32-n-(SUM(rand(1:d-1)))$ will generate the dec. Next, an evaluation function is proposed that calculates the total LUT consumption of the IP-Split-Bucket architecture with the following design parameters: BI_s, n and dec. The evaluation function determines its design parameters by the result of the estimation function (BI_s, n) and the random decoder generator (dec). Later, this function is called iteratively and an efficient dec is selected among the iterations that leads to the lowest total LUT consumption. Figure 4.15 illustrates the results given by the evaluation function for 3000 iterations. Each point, in this figure, demonstrates the estimated LUT consumption of the best solution found in the corresponding number of random tests. After 3000 iterations, the most efficient design parameters found by the evaluation function for $N = 16383$ are: $BI_s = 9$, $n = 7$ and $dec = [5\ 4\ 4\ 4\ 3\ 2\ 3]$. For this design, the evaluation function estimates the LUT consumption of 22.5 k where $E = 323$ and $m = 197$.

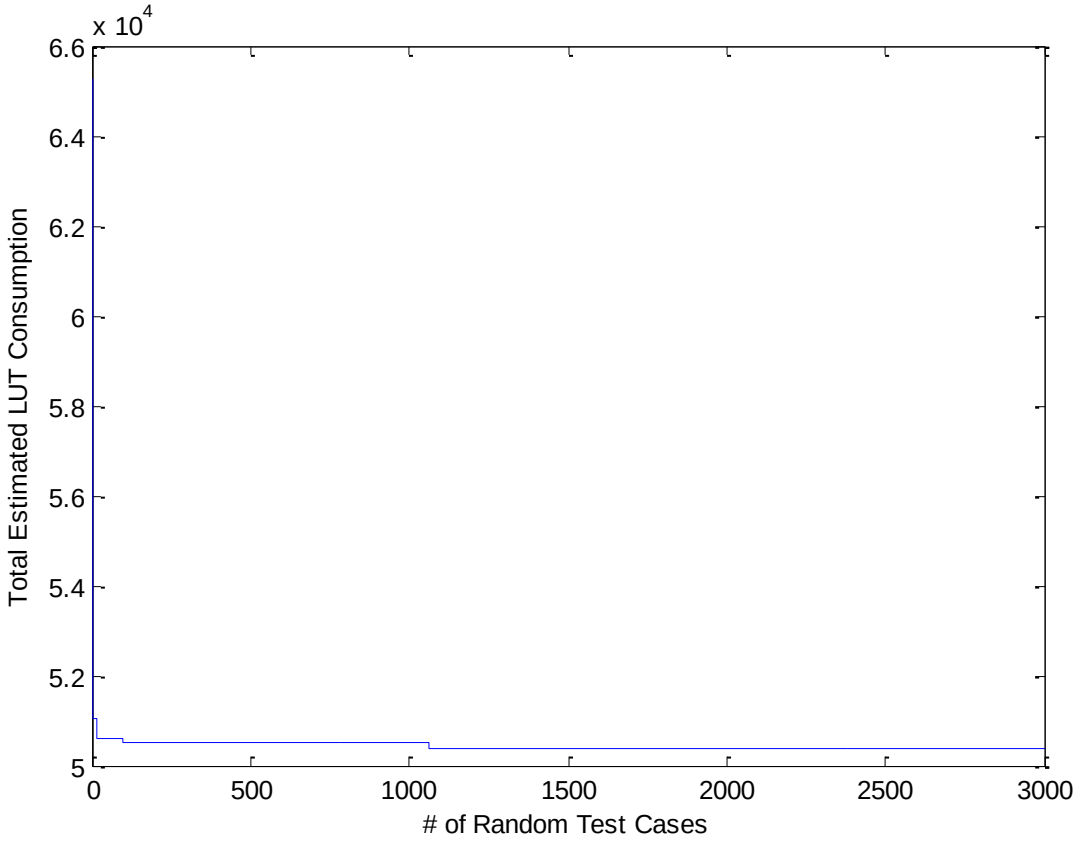

Figure 4.15: Evaluation function results for 3000 iterations

CHAPTER 5 CONCLUSION AND FUTURE WORK

Due to ever-increasing number of IP addresses in existing FIBs, there is a high demand for fast, high performance ALEs supporting large FIBs. In this thesis, four ALE architectures were proposed and implemented in FPGA: The Full-Serial, the Full-Parallel, the IP-Split, the IP-Split-Bucket and a fifth on the Update-enabled IP-Split-Bucket was proposed. The drawbacks of every proposed architecture are avoided in the subsequent one.

The Full-Serial architecture performs a serial search on the FIB entries stored in the memory using one comparator. The comparator is implemented using two different architectures: option A and option B. The lookup time is proportional to $N/2$ in the Full-Serial architecture. The Full-Serial architecture has some drawbacks in terms of memory consumption for large FIBs. Moreover, it is a slow architecture for which the lookup time is data set dependent.

These drawbacks are avoided in the second proposed architecture. The Full-Parallel architecture performs a parallel search on the FIB entries using multiple parallel comparators instead of one comparator. It has a constant latency and throughput for any FIB regardless of the incoming IP address. Since all FIB entries are stored into the logical resources of the FPGA, it has a high complexity in terms of LUT and FF consumption. In a large FIB with thousands of IP addresses, a parallel comparison on the FIB may contain several redundant comparisons on equivalent entries. This issue is resolved in the next proposed architecture.

The IP-Split architecture employs an additional block of decoders to prevent equivalent repetitive comparisons. The main shortcoming of this architecture is the PEB size that is dependent on the FIB size. For large FIBs, it requires a large priority encoder, which is the bottleneck of this architecture.

The IP-Split-Bucket architecture is a modified version of the IP-Split architecture. This architecture reduces the priority encoder size by dividing the FIB into buckets, where only one bucket contains viable comparison results at a time. Therefore, the priority encoder size is reduced from the FIB size to the size of the largest existing bucket. The incoming IP address specifies the corresponding bucket. IP-Split-Bucket is a memory-less high performance architecture supporting large FIBs.

The Update-enabled IP-Split-Bucket architecture is the last architecture proposed to improve the applicability of the IP-Split-Bucket architecture for high-update-rate IP address lookup. This architecture employs a modified version of the IP-Split-Bucket architecture with a parallel update system to support all types of updates.

The proposed architectures were synthesized for various sizes of real-world FIBs taken from the RIS raw data set [30]. They were implemented on Virtex-7 and Virtex-5 families of FPGAs using Xilinx and Synplify synthesis tools. The Full-Serial architecture was implemented on a Virtex-5 XC5VLX50T FPGA using Xilinx ISE 13.4 synthesis tool for FIBs up to 46 k entries. This architecture was implemented using two proposed options for the comparator architecture: A and B. The Full-Parallel architecture was implemented on a Virtex-5 XC5VLX50T FPGA using Xilinx ISE 13.4 synthesis tool for FIBs up to 4 k entries. Since this architecture employs one comparator for each entry of the FIB, the logical resources of the existing FPGAs determine the maximum supported FIB size.

The IP-Split architecture consists of four blocks of DB, CB, PEB and NHIB. All the blocks of IP-Split architecture were synthesized for a Virtex-5 XC5VLX50T FPGA with 15.9 Synplify synthesis tool. The synthesis results evaluations illustrate that the PEB is the bottleneck of the IP-Split architecture. The IP-Split-Bucket architecture was implemented on a Virtex-7 FPGA using the Xilinx ISE 13.4 synthesis tool. The results show that implementing a table of 524 k entries on a XC7V2200T FPGA consumes 23% and 22% of the available LUTs and FFs, respectively. The IP-Split-Bucket architecture has some design parameters that determine its complexity. Therefore, a design exploration was performed to choose efficient values for each block parameters.

The fourth proposed architecture, the IP-Split-Bucket architecture, avoids the shortcomings of the previously proposed ones. It is a generic scalable memory-less high performance architecture. Its main feature is the ability to handle large FIBs (524 k entries) while being memory-less. Additionally, the proposed architecture does not face the limitations of trie-based techniques, including nondeterministic latency and external memory access. Moreover, as the FIB is hardcoded in the logical resources of the FPGA, the IP-Split-Bucket architecture does not require TCAMs or other internal or external memory as opposed to existing TCAM-based and TCAM-emulation approaches. In addition to high-performance, post-fabrication flexibility and scalability

are other crucial factors for emerging technologies such as Software Defined Networking (SDN). Fast and low-cost realization of new communication protocols in SDN demands highly flexible architectures that have the capacity to adapt to new protocols. TCAM-based techniques have difficulty providing the required flexibility, whereas the IP-Split-Bucket architecture meets the scalability factor by its design parameters.

Compared to previously reported memory-less approaches, when configured for similar moderate size tables (18 k entries), the complexity of our solution is 60% less. Moreover, the closest recently reported solution that can handle comparable size tables requires a large (7.7 Mb) internal memory, while IP-Split-Bucket requires no memory. Future work will focus on the Update-enabled IP-Split-Bucket architecture implementation on FPGAs and improving its capability of supporting fast updates.

REFERENCES

[1] P. Ivan, "IPSpace," 13 August 2013. [Online]. Available: http://blog.ipspace.net/2013/08/management-control-and-data-planes-in.html.

[2] G. Trotter, "Terminology for Forwarding Information Base (FIB) based Router," December 2001. [Online]. Available: https://tools.ietf.org/html/rfc3222.

[3] E. Z. Liebeherr, "http://www.cs.virginia.edu/~cs458/," March 2005. [Online]. Available: http://www.cs.virginia.edu/~cs458/slides/module09b-routers.pdf.

[4] "Join us on the journey to 5g," Ericsson, 5 5 2016. [Online]. Available: http://www.ericsson.com/spotlight/5g.

[5] "5G Vision: 100 Billion connections, 1 ms Latency, and 10 Gbps Throughput," huawei, [Online]. Available: http://www.huawei.com/minisite/5g/en/defining-5g.html.

[6] V. Singh, "What is 5G Network? In Simple Words," CompleteGate, 26 January 2016. [Online].

[7] S. Lofgren, "IEEE," 10 March 2015. [Online]. Available: http://iot.ieee.org/newsletter/march-2015/iot-future-proofing-device-communications.html.

[8] "RIPE network coordination centre," [Online]. Available: https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data.

[9] B. White and A. Robertson, "The Internet Corporation for Assigned Names and Numbers," 3 February 2011. [Online]. Available: https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf.

[10] L. Smith and I. Lipner, "Free Pool of IPv4 Address Space Depleted," Number Resource Organization, 3 Februrary 2011. [Online]. Available: https://www.nro.net/news/ipv4-free-pool-depleted.

[11] K. Zheng, C. Hu, H. Lu and B. Liu, "An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine," Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies INFOCOM, vol. 3, pp. 1984-1994, 2004.

[12] D. Pao, "TCAM Organization for IPV6 Address Lookup," The 7th International Conference on Advanced Communication Technology, vol. 1, pp. 26-31, 2005.

[13] Y. Sun and M. S. Kim, "A Hybrid Approach to CAM-Based Longest Prefix Matching for IP

Route Lookup," Global Telecommunications Conference, pp. 1-5, 2010.

[14] H. le, W. Jiang and V. K. Prasanna, "Scalable High Throughput SRAM-based Architecture for IP-Lookup Using FPGA," International Conference on Field Programmable Logic and Applications (FLP), pp. 134-142, 2008.

[15] Y.-H. E. Yang, O. Erdem and V. K. Prasanna, "High performance IP lookup on FPGA with combined length-infix pipelined search," 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 77-80, 2011.

[16] J. Matoušek, M. Skačan and J. Kořenek, "Memory efficient IP lookup in 100 GBPS networks," 23rd International Conference on Field programmable Logic and Applications, pp. 1-8, 2013.

[17] Z. K. Baker and V. K. Prasanna, "Automatic synthesis of efficient intrusion detection systems on FPGA," IEEE Transactions on Dependable and Secure Computing, pp. 289-300, 2006.

[18] A. Rasmussen, A. Kragelund, M. Berger, H. Wessing and S. Ruepp, "TCAM-based High Speed Longest Prefix Matching with Fast Incremental Table Updates," International Conference on High Performance Switching and Routing, pp. 43-48, 2013.

[19] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAM for efficient and high-speed NIDS pattern matching," Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on, pp. 258-267, 2004.

[20] Z. Ullah, M. Kumar Jaiswal, Y. Chan and R. C. Cheung, "FPGA Implementation of SRAM-based Ternary Content Addressable Memory," International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pp. 383-389, 2012.

[21] C. R. Clark and E. D. Schimmel, "Scalable Pattern Matching for High Speed Networks," in 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004. FCCM 2004., 2004.

[22] J. Ditmar, K. Torkelsson and A. Jantsch, "A Dynamically Reconfigurable FPGA-based Content Addresable Memory for Internet Protocol Characterization," 10th International FPL conference, pp. 19-28, 2002.

[23] "UMC," SiberCore Technologies and UMC, [Online]. Available: www.umc.com.

[24] S. Yan and S. K. Min, "A Hybrid Approach to CAM-Based Longest Prefix Matching for IP Route Lookup," Global Telecommunications Conference, pp. 1-5, 2010.

[25] W. Jiang, "Scalable Ternary Content Addressable Memory Implementation Using FPGAs," IEEE Symposium on Architecture for Networking and Communications Systems (ANCS),

pp. 71-82, 2013.

[26] J. H. Mun and H. Lim, "New Approach for Efficient IPAddress Lookup Using a Bloom Filter in Trie-Based Algorithms," in IEEE Transactions on Computers, 2016.

[27] S. A. Guccione, D. Levi and D. Downs, "A Reconfigurable Content Addressable Memory," Custom Integrated Circuits Conference, pp. 24.1/1-24.1/4, 1990.

[28] B. Gamache, Z. Pfeffer and S. Khatri, "A Fast Ternary CAM Design for IP Networking Applications," International Conference on Computer Communications and Networks, pp. 434-439, 2003.

[29] K. Locke, "Parameterizable Content-Addressable," Xilinx Application note, 2011.

[30] "RIS Raw Data," [Online]. Available: http://data.ris.ripe.net.

[31] L. Bin and C. H. Jonathan, in High Performance Switches and Routers, New Jersey, John Wiley & Sons, Inc, 2007, pp. 6-8.

[32] R. Margaret, "TechTarget," March 2013. [Online]. Available: http://searchsdn.techtarget.com/definition/data-plane-DP.

[33] M. Rouse, "WhatIs," TechTarget, November 2014. [Online]. Available: http://whatis.techtarget.com/definition/latency.