February 2019
Doctor of Philosophy Degree Thesis

High-Performance Ternary Content Addressable Memory Architecture for FPGAs

Graduate School of Chosun University

Department of Computer Engineering

Inayat Ullah

February 2019

PhD_s Degree Thesis

High-Performance Ternary Content Addressable Memory Architecture for FPGAs

Inayat Ullah

High-Performance Ternary Content Addressable Memory Architecture for FPGAs

February 2019

Graduate School of Chosun University

Department of Computer Engineering

Inayat Ullah

High-Performance Ternary Content Addressable Memory Architecture for FPGAs

FPGA 기반 고성능 TCAM 구조

Supervised by Professor Jeong-A Lee

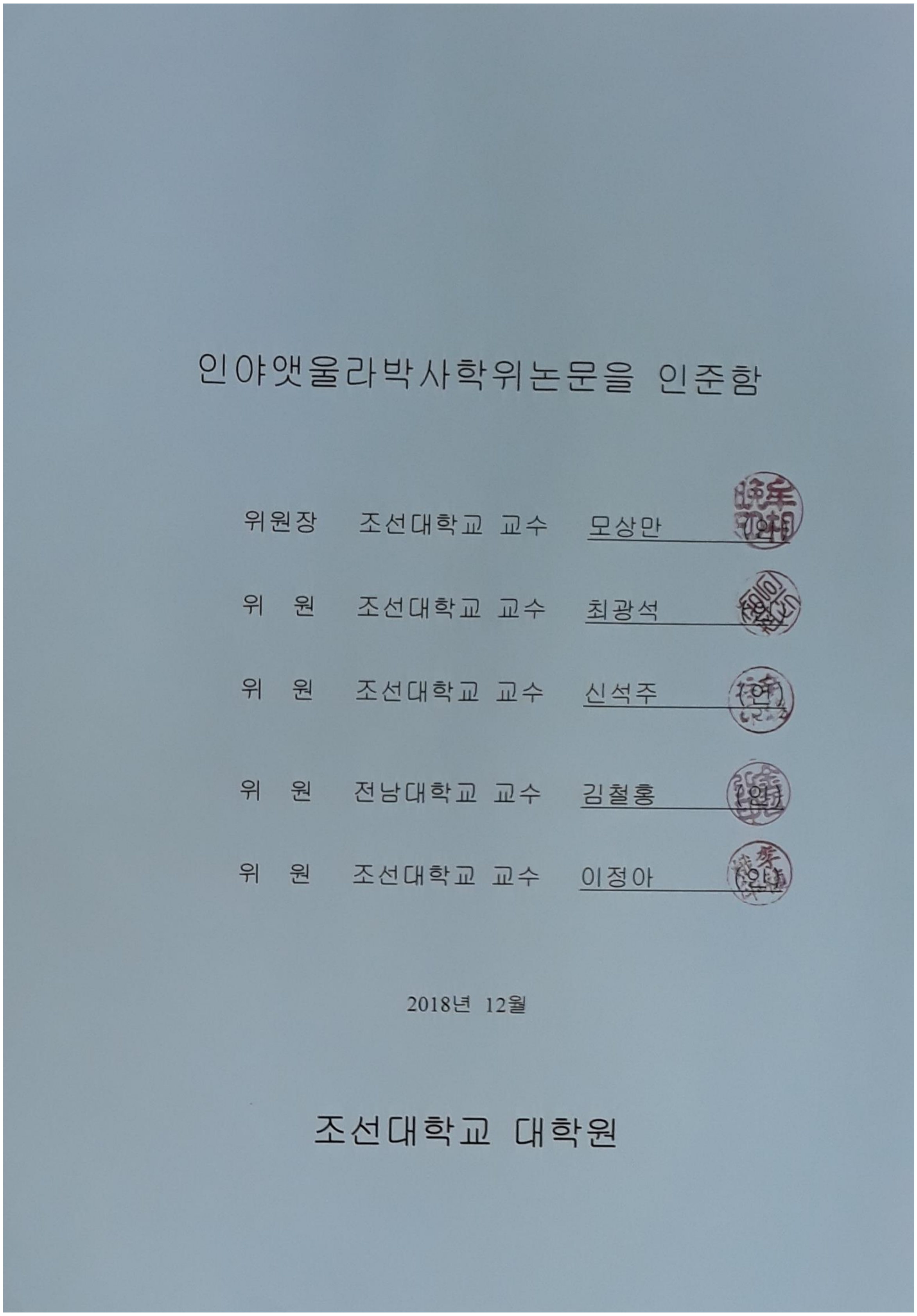
A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Engineering

25, February 2019

Graduate School of Chosun University

Department of Computer Engineering

Inayat Ullah



인야앳울라박사학위논문을 인준함

위원장	조선대학교 교수	모상만	(인)
위 원	조선대학교 교수	최광석	(인)
위 원	조선대학교 교수	신석주	(인)
위 원	전남대학교 교수	김철홍	(인)
위 원	조선대학교 교수	이정아	(인)

2018년 12월

조선대학교 대학원

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS iv

ABSTRACT ix

한 글 요 약 xii

I. INTRODUCTION 1

A. Background 1

B. Problem Statement 3

C. Motivations 6

D. Thesis Contributions 7

E. Related Work 8

F. Thesis Organization 11

II. MULTIPUMPING ENABLED MULTIPORTED SRAM-BASED TCAM ARCHITECTURE 13

A. Multipumping-Enabled Multiported SRAM 13

1. Basic Idea 14

2. Proposed Partitioning of Traditional TCAM Table . . . 16

3. Basic Architecture of the Proposed TCAM Memory . . 17

4. Modular Architecture 19

5. Effect of Multipumping SRAM on the Memory Usage and Throughput 20

B. Implementation Setup and Results 20

C. Performance Evaluation & Comparison 21

1. SRAM Memory Utilization 22

2. Throughput 23
3. Performance per Memory 24

III. PRE-CLASSIFICATION- BASED ENERGY-EFFICIENT SRAM-BASED TCAM ARCHITECTURE 28
A. Proposed Classification Scheme 28
B. EE-TCAM Proposed Architecture 34
C. EE-TCAM FPGA Implementation & Results 37
D. EE-TCAM Performance Evaluation & Comparison 38
1. Scalability of EE-TCAM 38
2. Performance Trade-Off with Increase in the Number of TCAM Sub-Tables (M) 41
3. Power Consumption 43
4. Power Consumption Per Performance Comparison . . . 44

IV. DYNAMICALLY RE-CONFIGURABLE ENERGY- AND RESOURCE- EFFICIENT TCAM ARCHITECTURE FOR FPGAS 49
A. Hardware Architecture of Proposed TCAM: DURE 49
1. Basic Idea 49
2. Building Blocks of DURE on FPGA 49
3. Architecture of the Proposed DURE 53
4. Dynamic Update 55
B. FPGA Implementation and Results 56
C. DURE Performance Evaluation and Comparison 58
1. Scalability of DURE 58
2. Search Latency 64
3. Update Rate 65

4. FPGA Resource Utilization 66
5. Performance per Area 69
6. Energy-Delay Product 74

V. CONCLUSIONS & FUTURE WORK **79**

REFERENCES **89**

ACKNOWLEDGEMENTS **90**

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application-specific integrated circuit
CAM	Content-addressable memory
BCAM	Binary Content-addressable memory
TCAM	Ternary content-addressable memory
SRAM	Static-random access memory
FPGA	Field-programmable gate array
BRAM	Block RAM
SL	Search line
MA	Match Address
ML	Match line
MLSA	Match line sense amplifier
PE	Priority encoder
LUT	Look-up table
LUTRAM	Look-up table RAM
CLB	Configurable logic block
SR	Slice register
TLB	Translation look-aside buffer
MDB	Maximum depth bound
ST	Sub table
EDP	Energy-delay product

LIST OF FIGURES

1 A 4×3 TCAM: (MLSAs: Match line sense amplifiers). 2

2 SRAM-based implementation of a 4×3 TCAM: (PE: priority encoder, and MA: match address) 3

3 Multipumping-based multiported memory: the SRAM block is clocked at an integral multiple of P, allowing P access during one external clock cycle. 13

4 (**a**) A conventional TCAM of 1×8; (**b**) An 16×1 SRAM without multipumping emulating 1×4 TCAM; (**c**) An 16×1 SRAM with a multipumping factor of $P = 2$ emulating 1×6 TCAM; (**d**) An 16×1 SRAM with a multipumping factor of $P = 4$ emulating 1×8 TCAM. 14

5 Proposed partitioning of the traditional TCAM table. 16

6 Basic architecture of the proposed TCAM memory. 17

7 Organization of the proposed TCAM memory units for a large storage capacity: (*IW*: input word, *PE*: priority encoder, *OPE*: overall priority encoder) 18

8 Timing diagram for the search operation in our proposed TCAM with a multipumping factor $P = 2$: (*IW*: input word, R_W: SRAM word read, *MW*: match word) 19

9 Example of the proposed classification algorithm: (**a**) Description of Classification based on set of bit positions $S_1 = \{b_0, b_1\}$. (**b**) Description of Classification based on set of bit positions $S_2 = \{b_1, b_3\}$. 29

10 Example of mapping proposed classiifcation scheme constructed TCAM-subtables to SRAM: (**a**) Mapping TCAM sub-tables contents to SRAM constructed based on set of bit positions $S_1 = \{b_0, b_1\}$. (**b**) Mapping TCAM sub-tables contents to SRAM constructed based on set of bit positions $S_2 = \{b_1, b_3\}$. 31

11 Architecture of the pre-classifier unit: (*S*: Select lines for extracting the pre-classification bits, log_2W: Number of bits in the , log_2M: Number of pre-classifier bits, *M*: Number of enable control bits for the *M* rows of SRAM blocks). 35

12 Proposed overall architecture: (*M*: Number of rows of SRAM blocks, *V*: Number of SRAM blocks in a row, *PE*: priority encoder unit, *MA*: matching address, log_2R_D: Address bits of the configured SRAM blocks, $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$: Configured SRAM blocks words width). 35

13 Trade-Off between the Number of TCAM Sub-Tables (M) and Power Consumption Performance 42

14 A simplified implementation of DURE. 50

15 Architecture of the basic memory block of DURE for implementing an 1×18 TCAM on FPGA. 51

16 Partitioning TCAM table contents into 6-bit chunks. 52

17 Architecture of the proposed DURE. 53

18 LUTRAM utilization trend for increasing TCAM depth (D) and width (W). 59

19 Slice registers usage trend for increasing TCAM depth (D) and width (W). 60

20 Clock rate trend for increasing TCAM depth (D) and width (W). 61

21 Power consumption trend for increasing TCAM depth (D) and width (W). 62
22 Update rate trend for increasing TCAM depth (D) and width (W). 63
23 Search latency comparison. 64
24 Update rate comparison. 65

LIST OF TABLES

1 List of basic notations used. 11
2 FPGA resource utilization of the proposed design. 21
3 Performance per memory comparison of the proposed TCAM with previous approaches. 26
4 A TCAM table of size 6×6. 29
5 FPGA resource utilization of EE-TCAM on Xilinx Virtex-6. . . 38
6 Performance trend for increasing EE-TCAM depth (D), width (W) in various configurations. 40
7 Comparison of the power consumption per performance with previous works. 46
8 A 3×4 TCAM table 49
9 FPGA resource utilization for proposed DURE 58
10 FPGA resource utilization for implementing various TCAM architectures 68
11 Performance per area comparison with existing FPGA-based TCAM architectures 72
12 Energy-delay product (EDP) comparison for various FPGA-based TCAM solutions 76

ABSTRACT

High-Performance Ternary Content Addressable Memory Architecture for FPGAs

Inayat Ullah
Advisor: Prof. Lee, Jeong-A, Ph.D.
Department of Computer Engineering
Graduate School of Chosun University

Ternary content-addressable memory (TCAM) is widely employed to design high-speed search engines and has applications in networking, artificial intelligence, and to accelerate various database search primitives. TCAMs are built either as application-specific integrated circuits (ASIC) natively or static random-access memory (SRAM)-based field-programmable gate arrays (FPGAs) are used to emulate TCAM by addressing the SRAM with TCAM contents. The search space demands of TCAM applications are constantly rising. However, existing realizations of TCAM on FPGAs suffer from storage inefficiency. Native TCAMs and SRAM-based TCAMs both have a high power consumption drawback. The SRAM-based TCAMs offer promising lookup performance, however, the update process in a TCAM table poses significant challenges to their efficient employment. SRAM-based TCAMs for FPGAs, suspend search operations during an already high-latency update operation rendering them infeasible in applications that require high-frequency updates. This thesis presents three approaches to overcome the aforementioned limitations in existing

FPGA-based TCAMs i.e. storage inefficiency, higher power consumption, and high-latency blocking updates.

We first propose a multipumping-enabled multiported SRAM-based TCAM design on FPGAs to achieve efficient SRAM utilization. Existing SRAM-based TCAM solutions reduce the impact of the increase in the traditional TCAM pattern width from an exponential growth in memory usage to a linear one with the use of cascaded block RAMs (BRAMs) on FPGA. However, BRAMs on state-of-the-art FPGAs have a minimum depth limitation, which limits the storage efficiency for TCAM bits. Our proposed solution avoids this limitation by mapping the traditional TCAM table divisions to shallow sub-blocks of the BRAMs towards memory-efficient TCAM design. Using the multipumping technique, the proposed solution operates the simple dual-port BRAMs of the design as multiported SRAM by clocking them with a higher internal clock frequency to access the sub-blocks of the BRAM in a single system cycle. We implemented the proposed design on a Virtex-6 FPGA device and compared with the existing FPGA-based TCAM designs, our proposed method achieves up to 2.85 times better performance per memory.

A pre-classifier-based architecture for a low-power SRAM-based TCAM is also presented. The first classification stage divides the TCAM table into several sub-tables of balanced size. The second SRAM-based implementation stage maps each of the resultant TCAM sub-tables to a separate row of SRAM blocks in the architecture. The proposed design selectively activates at most one row of SRAM blocks for each incoming TCAM word. Compared with the existing SRAM-based TCAM designs on FPGAs, the proposed design consumes significantly lower energy as it activates only a part of SRAM memory being used for lookup rather than the entire SRAM memory. We implemented the proposed approach on a Xilinx Virtex-6 FPGA and our experimental results showed that the proposed design achieved at least three times lower power consumption per performance than other SRAM-based TCAM architectures.

A dynamically updateable energy- and resource-efficient TCAM design (DURE) based on FPGAs is also presented in this thesis. DURE exploits the distributed RAM resources in FPGAs, more specifically, the look-up table RAMs (LUTRAMs) available in SLICEM resources are configured as quad-port RAM, which constitutes the basic memory (BM) block in the implementation of DURE. The contents of the TCAM table are divided into chunks of equal size and are mapped onto the LUTRAMs of the proposed BM blocks. DURE implements dynamic updates by reconfiguring the LUTRAMs of only those BM blocks that are associated with the word being updated, thereby allowing search and update operations to be performed simultaneously. Compared with existing SRAM-based TCAMs, DURE has a smaller single cycle search latency and achieves at least 2.5 times more energy-delay product efficiency and a 67% higher performance per area.

In this thesis, we have presented three high-performance TCAM architectures which achieve higher memory efficiency, better energy-delay product efficiency, and implement dynamic updates. Our proposed solutions are general and can be adapted in many applications. In the future, we will explore the adoption of these designs in various applications for further evaluation.

한 글 요 약

FPGA 기반 고성능 TCAM 구조

인야앳울라
지도 교수: 이정아
컴퓨터공학과
대학원, 조선대학교

TCAM (Ternary Content-Addressable Memory)은 고속 검색을 필요로 하는 네트워킹, 인공지능 등 다양한 분야에서 핵심엔진으로 이용된다. TCAM은 주문형 반도체(ASIC)로 바로 구현될 수 있으나 전력소모가 많은 단점이 있었으며, TCAM 내용을 SRAM의 주소로 이용하여 FPGA로 구현될 수 있었으나, 대용량 검색의 요구가 늘어감에 따라, 전력소모 증가와 스토리지 비효율성이란 문제가 있었다. 그리고 SRAM 기반으로 설계된 TCAM은 빠른 검색을 가능하게 하나, TCAM 테이블의 업데이트가 필요한 경우, 검색이 중단되는 문제점이 있었으며, 잦은 업데이트가 발생하는 응용에서는 활용이 어려웠다. 본 논문에서는 기존의 FPGA기반 TCAM 구조의 문제점인 스토리지 비효율, 높은 전력소비, 그리고 업데이트 발생 시 검색 중단으로 인한 대기시간 증가를 극복하기 위한 세 가지 해결방안을 제시한다.

첫 번째로 SRAM 기반 TCAM 구조에서 SRAM 메모리의 효율적인 활용을 위하여, 멀티펌핑 가능한 멀티포트 TCAM 구조를 제시한다. TCAM 검색워드 크기가 증가함에 따라 필요한 SRAM 메모리가 폭발적으로 증가하는 문제를 해결하기 위한, 기존의 해결책은 FPGA의 연결식 블록 RAM(BRAM)을 사용하는 것이었다. 그러나 최첨단 FPGA의 BRAM은 최소깊이 제한이 있어 TCAM 비트의 저장 효율성을 제한한다. 본 논문에서는 이러한 한계를 피하기

위하여, 기존의 TCAM 테이블영역을 구성된 BRAM의 얕은 하위블록에 매핑하여 메모리 효율적인 TCAM 메모리설계를 구현한다. 제안된 솔루션은 단일 시스템 사이클에서 BRAM의 하위블록에 액세스하기 위해 보다 높은 내부 클록주파수로 클러킹 함으로써 다중펌핑 기술을 사용하여 멀티포트 SRAM으로 구성된 BRAM을 작동시킨다. 제안된 디자인을 Xilinx Virtex-6 FPGA에 구현하여, 기존의 FPGA 기반 TCAM 설계와 비교하여 제안 된 방법이 메모리 당 최대 2.85 배의 성능을 달성함을 보였다.

그리고 SRAM 기반 TCAM 구조의 높은 전력소모를 해결하는 방안으로, 사전 분류기 기반 TCAM 구조를 제시한다. 첫 번째 분류를 통하여 TCAM 테이블을 균형 잡힌 크기의 몇 개의 서브-테이블로 분할한다. 이렇게 분할된 서브-테이블을 SRAM 블록의 분리된 개별 행에 매핑하여 구현한다. 본 구조는 TCAM 워드 검색시 전체가 아닌 일부에 해당하는 하나의 SRAM 블록 행만을 선택적으로 활성화하기 때문에 기존의 SRAM 기반 TCAM 구조와 비교할 때 에너지 소비를 크게 줄인다. 제안된 구조를 Xilinx Virtex-6 FPGA에 구현하여 기존의 SRAM 기반 TCAM 구조에 비하여 성능 당 적어도 3 배 이상 낮은 전력 소비를 달성함을 보였다.

세 번째로 검색을 하면서 동적 업데이트가 가능한 에너지 및 자원 효율적인 DURE TCAM 구조를 제시한다. DURE 구조에서는 FPGA의 분산 RAM인 SLICEM 리소스의 룩업테이블 RAM (LUTRAM)을 활용하여, 쿼드포트 RAM으로 구현된 LUTRAM들을 기본 메모리(BM) 블록으로 이용한다. TCAM 테이블의 내용은 동일한 크기로 분할 되어 제안 된 BM 블록에 있는 LUTRAM에매핑된다. DURE는 업데이트 되는 워드와 연관된 BM 블록의 LUTRAM만을 재구성하여 동적 업데이트를 구현하므로 검색 및 업데이트 작업을 동시에 수행할 수 있다. 기존의 SRAM 기반 TCAM 구조와 비교할 때, DURE는 검색소요시간이 짧으며 에너지와 수행시간를 모두 고려한 효율이 기존에 비하여 2.5 배 이상 향상 되고, 면적 당 67% 더 높은 성능을 달성한다.

본 논문에서는 FPGA 기반 고성능 TCAM 구조를 다양하게 제시하여, 메모리 효율을 높이고, 에너지와 수행시간을 모두 고려한 효율을 향상시켰으며, 동적 업데이트를 가능하게 하였다. 본 논문에서 제안 된 구조는 범용적인 것으로 다양한 어플리케이션에 적용이 가능하며, 이와 관련된 연구를 추후 진행할 예정이다.

I. INTRODUCTION

A. Background

Content addressable memory (CAM) selects a word among stored data based on its contents. CAM compares an input word with its entire stored data in parallel and outputs the matched word's address in a single cycle. CAM is mainly categorized into two: Binary CAM (BCAM) that store only two states, "0" and "1", and ternary CAM (TCAM) that also store and compare an additional third state, known as the "don't care state x". The don't care state "x" is used to store wild card entries in TCAM words.

TCAMs designed as a dedicated system in the application-specific integrated circuit (ASIC) are known as traditional TCAMs. It compares the search key with the entire stored TCAM words in parallel and outputs the address of the matching word in one cycle. The circuitry of a TCAM cell store and compare three states: "0", "1", and don't-care state "x". TCAM architecture is composed of an array of TCAM cells and a priority encoder (PE). Each TCAM cell comprises two SRAM cells storing a ternary bit and an associated comparison circuitry. Search lines (SLs) provide search key bits to the corresponding cells of the TCAM words. The comparison results of each TCAM word are placed on the match lines (MLs). In case the matching of the search key is successful with more than one TCAM word, the PE selects the matching address with the highest priority. Figure 1 shows a sample design of 4×3 traditional TCAM. For example, an input word of 101 is applied to the search lines of a 4×3 traditional TCAM shown in Figure 1. The TCAM words of Row 1 and 3 are found to match completely and the corresponding match lines ML1 and ML3 remain high. The PE selects the address of ML1 as the match address based on

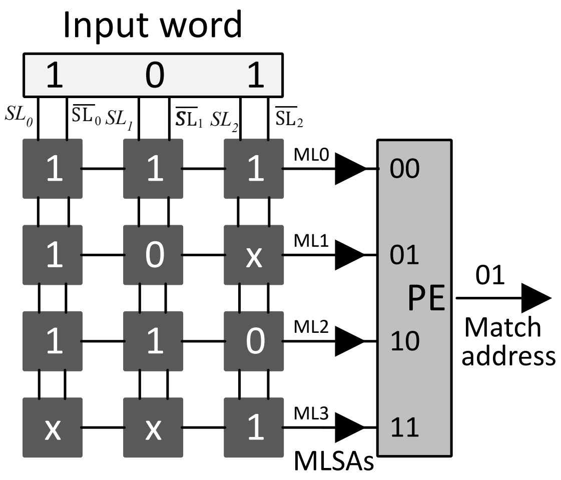


Figure 1: A 4×3 TCAM: (MLSAs: Match line sense amplifiers).

priority. CAM is widely employed to design perform high-speed search engines for a variety of emerging applications, in the areas of networking as look-up table [1, 2, 3, 4], in microprocessors as translation-look-aside buffer (TLB) cache [5, 6, 7, 8], in big-data analytics as database accelerators [9, 10, 6], and in image processing, pattern recognition, and DNA sequence matching as Local binary patterns recognition system [11, 12, 13, 14, 6]. The Internet-of-things and big-data processing devices employ TCAM as a filter when storing signature patterns, and achieve a substantial reduction in energy consumption by reducing wireless data transmissions of invalid data to cloud servers [9, 15].

Static random-access memory (SRAM)-based implementations of TCAM consist of an SRAM memory and a PE as shown in Figure 2. All the data of the original TCAM is mapped to the SRAM memory. For a TCAM table of size $A \times B$ (A words of B bits width), the SRAM-based implementation requires 2^B words of A bits width. Figure 2 shows the SRAM-based implementation of a 4×3 TCAM. It comprises of a 2^3 SRAM words of 4-bits each. The SRAM words

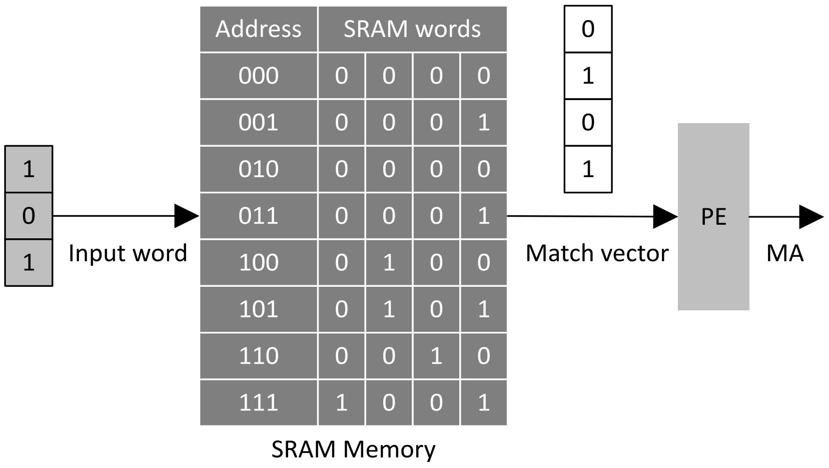


Figure 2: SRAM-based implementation of a 4 × 3 TCAM: (PE: priority encoder, and MA: match address)

store the match/ mismatch information on comparison with the TCAM words of all locations of the original TCAM for all possible combinations of input words. Bit positions 0, 1, 2, and 3 of the 1st SRAM word stores the match/mismatch information of the input word 000 with TCAM words of all four locations 0, 1, 2, and 3 respectively. Likewise, the SRAM words 0, 1, 2, ..., 7 store the match/mismatch information of all eight possible input words with TCAM words of all four locations at bit positions 0, 1, 2, and 3. The input word 101 matches successfully with TCAM words at locations 1 and 3. Accordingly, the SRAM memory for the 4 × 3 TCAM implementation stores high bits at bit positions 1 and 3 of the input word's 101 addressed location.

B. Problem Statement

Field-programmable gate arrays (FPGAs) emulate TCAM using SRAM, by addressing SRAM with TCAM contents. Each SRAM word corresponds to a specific TCAM pattern, and stores information on its existence for all possible data of the TCAM table. The increase in the number of TCAM pattern bits results in an exponential growth in memory usage. This exponential growth in memory usage has been reduced to linear growth by cascading multiple SRAM blocks in the design of TCAM on FPGA in previous work [16, 17].

Contemporary FPGAs implement block-RAM (BRAM) in the silicon substrate and offer a high speed. For example, Xilinx Virtex-6 xc6vlx760 FPGA contains 720 BRAMs of size 36 Kb [18], and provide operating frequencies of greater than 500 MHz [19]. Designers utilize these high-speed SRAM blocks to design SRAM-based TCAMs on FPGA. In existing SRAM-based solutions, the storage capacity of a BRAM for TCAM bits is limited by its higher SRAM/TCAM ratio $\frac{2^9}{9}$, because of its minimum depth limitation of 512×72 when configured in simple dual-port mode on FPGA [18]. For example, the design methodologies proposed in [20], [21], and [22], require a total of 56, 40, and 40 BRAMs of size 36 Kb, respectively, to implement an 18 Kb TCAM.

Excessive usage of BRAMs in the design of TCAM can result in a lack of BRAMs for other parts of the system on FPGA. Furthermore, the limited amount of BRAM resources on FPGA can compel designers to implement TCAMs in distributed RAM using SLICEM, resulting in the consumption of many slices, and a limitation on the maximum clock frequency of the design. This problem becomes more severe for the design of large storage capacity TCAMs. The efficient utilization of SRAM memory is imperative for the design of TCAMs on FPGAs.

Existing SRAM-based TCAMs on FPGAs suffer from higher energy consumption as they consume excessive power to energize the entire SRAM memory used per lookup. For example, the SRAM-based TCAM design methodologies presented in recent works [23, 24] consumed 2.5 W and 3.2 W to implement 89 kb and 150 kb TCAM tables using BRAMs on FPGA, respectively. The higher power consumption of SRAM-based TCAM designs becomes more severe for larger capacities.

A single update in SRAM-based TCAM requires rewriting all of the SRAM blocks configured by the design, owing to TCAM's support for wild card entries. The update process for a TCAM word in an SRAM-based TCAM architecture includes writing the update word and erasing the old TCAM word from SRAM blocks. This process involves the indexing of all the words of an SRAM block for writing. Such TCAMs implement dedicated update engines for writing the SRAM blocks in parallel, which introduces an excessive additional hardware overhead. Hence, the minimum number of cycles required for an update process is proportionate to the depth of the utilized SRAM blocks [17, 25].

In addition, during the update process in SRAM-based TCAM search operations remain suspended. Therefore, such TCAMs dedicate a substantial amount of clock cycles to supporting high-latency blocking updates in frequent update environments, thereby substantially lowering the arrival rate of input words for lookup [26]. Because lookup operations are locked during high latency updates, SRAM-based TCAMs require a large buffer space to store input words, in order to avoid the loss of these words [27, 28]. As such, despite the search performance of one search per cycle, these TCAMs are infeasible for use in high-churn-rate environments, where frequent TCAM updates are required. This critical TCAM table update issue in SRAM-based TCAM architectures has not so far been addressed. Therefore, further research on implementing dynamic updates in the FPGA-based TCAM architectures is required.

A TLB cache is an integral part of any processor architecture and is implemented using CAM [29, 5, 7]. Soft-core processors implemented in reconfigurable hardware such as FPGAs are widely employed in embedded system applications [30]. A low response time (single cycle) dynamically updateable CAM architecture is required in the design of a non-blocking TLB cache for soft-core processors [31, 32, 33]. However, existing RAM-based TCAM architectures on FPGAs suffer from high search latencies (high response times) and update operations during which searches are suspended.

C. Motivations

The significant advancement in CMOS technology has made modern FPGAs an attractive choice for implementing emerging systems because of their offered massive parallelism with flexibility through on-the-fly reconfiguration. Contemporary SRAM-based FPGA device such as the 40-nm Xilinx Virtex-6 FPGA consists of a large amount of embedded memory that operates at high speed with low power consumption.

The demand for a high-speed flexible (re-configurable) and adaptable (easy for integration) TCAM configurations renders the embedded memories (BRAMs and distributed RAM) on modern SRAM-based FPGAs attractive for the design of TCAMs. Configurable CAM architectures are desirable in embedded hardware, in order to be able to adapt to the requirements of the intended application to strike an optimal balance between the area, power, and speed [34, 35, 4].

D. Thesis Contributions

This thesis proposed three different approaches for designing memory-efficient, energy-efficient and dynamically updateable TCAM architectures for FPGAs.

The contributions of this thesis are summarized as follows:

- A novel multipumping-enabled multiported SRAM-based TCAM architecture, which achieves efficient memory utilization, is proposed. Compared to existing FPGA-based TCAM architectures, Our proposed design achieves a performance that is up to 2.85 times higher performance per memory.

- A pre-classifier-based architecture for SRAM-based TCAM design is proposed, which achieves a considerable reduction in energy consumption. Compared to prior work, our architecture exhibits at least 3 times lower power consumption per performance

- A dynamically updatable FPGA-based TCAM architecture using LUTRAM FPGA primitives is proposed. The proposed architecture targets TCAM applications where high-frequency updates are required. The main challenge in the efficient use of FPGA-based TCAM is the high latency blocking update operation, which is resolved by the proposed solution.

- The proposed dynamically updatable FPGA-based TCAM architecture is both energy and resource efficient. Compared with the existing state-of-the-art TCAM solutions, it achieves up to 67% more performance per area and at least 2.5 times better energy-delay product efficiency.

E. Related Work

Existing FPGA-based TCAM architectures are mostly RAM-based solutions employing block-RAM (BRAM) or distributed RAM resources for implementing TCAM. The BRAM-based TCAM solutions presented in previous work [20, 21, 22] store the knowledge about the presence of TCAM words and their address information in separate sets of BRAM units and thus suffer from inefficient memory usage. These works suffer from higher power consumption as the entire used excessive SRAM memory is activated for the incoming TCAM word lookup. The update process for such TCAMs includes updating a copy of the TCAM table, which is then remapped to the BRAM configured by the design on the FPGA. As such, all the BRAMs configured by the TCAM are rewritten, which is time-consuming and expensive in terms of the additional logic overhead.

The BRAM-based TCAM solutions presented in [16, 17, 25, 26, 36, 37] store the knowledge about the presence of the TCAM word and their address information in the same BRAM block. These approaches suffer from reduced memory efficiency because of the limited storage capacity of BRAM for storing the TCAM bits, owing to a higher SRAM/TCAM ratio of $2^9/9$ [17]. However, these works [16, 17, 25, 26, 36, 37] energize all used SRAM memory blocks in the design for each incoming TCAM word's lookup, and thus, consumes higher power consumption. The update process in the SRAM-based TCAMs from [16, 17, 20, 21, 22, 26, 36, 37] requires dedicated update engines to write all of the SRAM blocks used by the design in parallel, which results in an additional logic overhead. In this manner, the write cycle in the update process becomes proportional to the depth of the design-configured BRAMs, which requires 513 clock cycles [17, 25].

The distributed RAM-based TCAM solutions presented in previous work [17, 26, 36] achieve a higher memory efficiency, because of increased storage capacity for TCAM bits in distributed RAM with an SRAM/TCAM ratio of $2^5/5$. However, when implementing a large TCAM table, the bitwise-ANDing of a large number of wide bit vectors becomes time-critical, resulting in a reduced achievable throughput. Although these designs have a relatively small update latency of 33 cycles, they suffer from an expensive rewriting of the entire configured SRAM memory, on account of supporting wild card entries.

Xilinx has presented several implementation techniques for binary CAM using BRAM and for TCAM using the shift register (SRLE16) on FPGAs in its application note [37]. The BRAM-based binary CAM technique is storage-inefficient. The TCAM implementation technique, which employs LUT resources as shift registers, has reduced resource efficiency, as only two TCAM bits are encoded and are mapped on a single 16-bit shift register LUT primitive SRLE16. For larger TCAM table implementations, this technique suffers from timing problems arising because of routing congestion.

A fast content updating mechanism for SRAM-based TCAM is presented in [38]. Its update latency depends on the number of “don’t care” bits in the sub-words of an update word. However, this TCAM approach, which is implemented using Xilinx BRAM resources, has the worst case update latency, being 513 cycles.

The multiplexer-based TCAM design presented in the patent [39] suffers from a degraded system throughput owing to the sequential searching of the TCAM data for the input word, i.e., a single TCAM word is searched per cycle. This technique stores the contents of the TCAM table in the LUTRAMs from the SLICEM resources, while the comparison circuitry is implemented using programmable FPGA interconnects. The outputs of the comparison circuits are further *AND* cascaded to implement multiple bits of the TCAM word.

RAM-based associative CAM was presented in a U.S. patent [40]. Its memory requirement increases exponentially with the increase in the pattern bits of CAM. For large bit patterns, it does not scale well in terms of the memory requirement, power consumption, and cost. Thus, it is not feasible to implement it on ASIC or FPGA platforms.

A set-associative memory architecture implemented in hardware using the well-known hashing method was presented in [41]. It used RAM memory to implement CAM. However, in order to support TCAM functionality, the architecture suffered from inefficient memory utilization as it required two bits to encode ternary bits.

In contrast to the existing SRAM-based TCAMs, our proposed multipumping-enabled multiported SRAM-based TCAM design exploits the efficient utilization of SRAM memory by mapping TCAM divisions to shallow sub-blocks of BRAMs on FPGA. It operates high-speed BRAMs in the design as multipumping-enabled multiported SRAM, maintaining a high system throughput.

In this thesis, we present a pre-classifier-based architecture for an energy-efficient SRAM-based TCAM design (EE-TCAM). We first classify a TCAM table into several TCAM sub-tables, which are further partitioned vertically. Each partitioned TCAM sub-table is implemented as a row of cascaded SRAM blocks in the proposed architecture. For each input TCAM word, at most one row of SRAM blocks is activated in the proposed design, significantly reducing the dynamic power consumption compared with the existing SRAM-based TCAMs.

In contrast to previous related SRAM-based TCAM designs, this thesis presents a dynamically updateable energy- and resource-efficient TCAM design (DURE). DURE maps the contents of a TCAM table onto the LUTRAMs available in SLICEM resources, which together with the available fast carry chains implements the desired TCAM functionality, achieving a single cycle search latency. DURE dynamically reconfigures only the LUTRAMs associated with the word being updated, without suspending search operations.

Table 1: List of basic notations used.

Notations	**Description**
$\boldsymbol{D}$	Depth of traditional TCAM
$\boldsymbol{W}$	Width of traditional TCAM
R_D	Depth of the configured SRAM blocks
R_W	Width of the configured SRAM blocks
$log_2(R_D)$	Address bits of the SRAM block
$\boldsymbol{P}$	Number of sub-blocks in an SRAM block / Multipumping factor
R_D/P	Depth of the sub-blocks in an SRAM block
$\boldsymbol{M}$	Rows of the TCAM divisions / Rows of the TCAM memory units
$\boldsymbol{N}$	Columns of the TCAM divisions / Columns of the TCAM memory units

F. Thesis Organization

This thesis is organized as follows. Chapter II describes multipumping-enabled multiported SRAM-based TCAM architecture, its FPGA implementation results and comparison with existing FPGA-based TCAM architectures. Chapter III details the pre-classification based energy efficient TCAM architecture and evaluates its performance based on its comparison with other existing TCAM solutions on FPGAs. Chapter IV explains the dynamically re-configurable LUTRAM-based energy- and resource- efficient TCAM architecture and its performance evaluation is detailed. Finally, Chapter V concludes this work. Table 1 lists the description of the basic notations used in the thesis.

II. MULTIPUMPING ENABLED MULTIPORTED SRAM-BASED TCAM ARCHITECTURE

A. Multipumping-Enabled Multiported SRAM

The multipumping technique multiplies the ports of a dual ported SRAM block by internally clocking it at an integral multiple of the external system clock [42, 43, 44, 45]. The addresses and data are registered and provided access to the SRAM block in a circular order by using mod *P* counter bits as shown in Figure 3. Several designs utilize multipumping for the implementation of efficient multiported memory [46, 47].

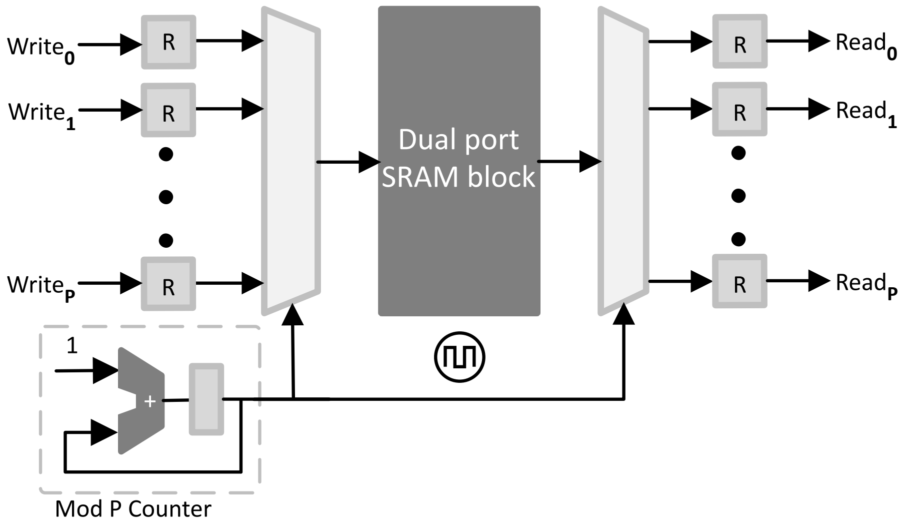


Figure 3: Multipumping-based multiported memory: the SRAM block is clocked at an integral multiple of *P*, allowing *P* access during one external clock cycle.

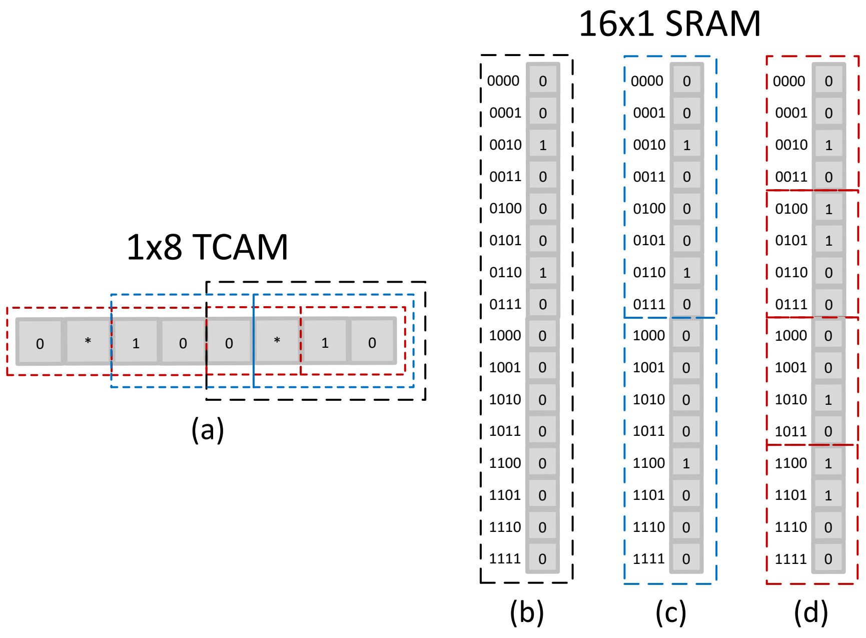


Figure 4: (**a**) A conventional TCAM of 1×8; (**b**) An 16×1 SRAM without multipumping emulating 1×4 TCAM; (**c**) An 16×1 SRAM with a multipumping factor of $P = 2$ emulating 1×6 TCAM; (**d**) An 16×1 SRAM with a multipumping factor of $P = 4$ emulating 1×8 TCAM.

1. Basic Idea

In the SRAM-based implementation of TCAM, the depth of the traditional TCAM determines the width of SRAM memory, and the width of the traditional TCAM is encoded as the address of the SRAM memory. The basic concept of the proposed multipumped SRAM-based TCAM implementation achieving increased memory efficiency is shown in Figure 4. Figure 4(a) shows a 1×8 traditional TCAM table, and Figure 4(b) shows the implementation of the four TCAM bits (0*10) by using a 16×1 SRAM block. Figure 4(c) shows the implementation of six TCAM bits (100*10) by using 16×1 SRAM block, which has been multipumped two times, each SRAM sub-block of size 8×1 emulating three TCAM bits. Figure 4(d) shows the implementation of eight TCAM bits (0*1000*10) by using 16×1 SRAM block, which has been multipumped four times, each SRAM sub-block of size 4×1 emulating two TCAM bits. Thus, designing TCAM using multipumping-enabled multiported SRAM in Figure 4(c) and (d) achieved a higher SRAM memory efficiency (i.e. fewer SRAM bits are utilized per TCAM bit) when compared with that of multipumping-less SRAM-based TCAM design in Figure 4(b). The TCAM bits storage capacity of the SRAM block increases with multipumping.

A multiported SRAM block of size $R_D \times R_W$ with a multipumping factor of P implements a traditional TCAM table of size $Plog_2(R_D/P) \times R_W$, each SRAM sub-block of size $(R_D/P) \times R_W$ emulating $log_2(R_D/P) \times R_W$ TCAM data, as shown in Figure 5 and 6. Our proposed design achieves increased TCAM bits storage capacity with an increase in multipumping factor P.

2. Proposed Partitioning of Traditional TCAM Table

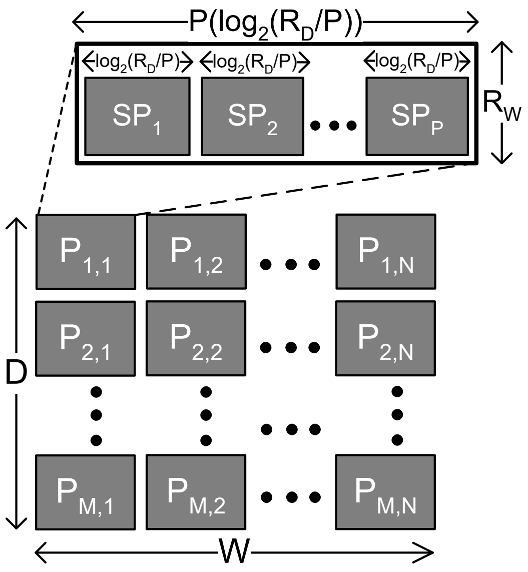


Figure 5: Proposed partitioning of the traditional TCAM table.

We partition the traditional TCAM table of size $D \times W$ into $M \times N$ partitions such that each partition consists of P parts of $log_2(R_D/P) \times R_W$ size as shown in Figure 5. Our proposed TCAM design uses its configured SRAM blocks of $R_D \times R_W$ size as multiported SRAM, constituting P sub-blocks of size $(R_D/P) \times R_W$ as shown in Figure 6.

Each sub-block of the SRAM stores $log_2(R_D/P) \times R_W$ size divisions of the traditional TCAM. Consequently the P sub-blocks of the multiported SRAM memory in our proposed design stores a traditional TCAM division of size $Plog_2(R_D/P) \times R_W$ as shown in the Figures 5 and 6. Similarly, the $M \times N$ TCAM divisions of size $Plog_2(R_D/P) \times R_W$ are mapped to the SRAM blocks of the $M \times N$ TCAM memory units in the proposed design, as shown in Figures 5 and 7.

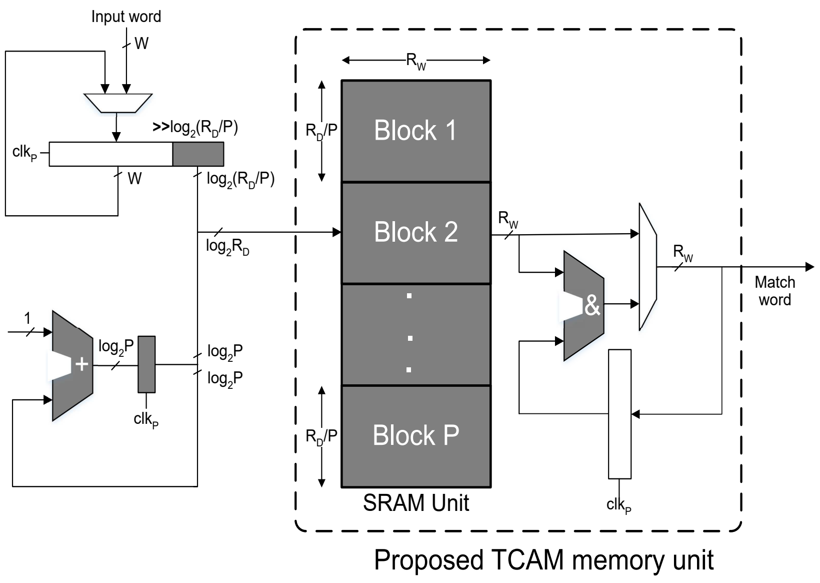


Figure 6: Basic architecture of the proposed TCAM memory.

3. Basic Architecture of the Proposed TCAM Memory

The basic architecture of our proposed TCAM memory design is shown in Figure 6. It is operated by two fully synchronized clocks, a system clock clk_S and internal clock clk_P, such that clk_P is P times faster than clk_S. An incoming TCAM word is registered in a W-bit shift register using the system clock clk_S. The log_2P-bit counter generates a sequence of log_2P-bit numbers in P internal clock cycles. It is initialized to zero upon reset and it rolls over after every P internal clock cycles. The log_2P-bits from the counter are concatenated with the $log_2(R_D/P)$ bits from the shift register to make the log_2R_D-bit address space of the SRAM. At the positive edge of the internal clock clk_P, the SRAM address is executed such that log_2P-bits from the counter constitute its most significant bits, and points to the start of the corresponding sub-block in SRAM and the lower

$log_2(R_D/P)$ bits from the shift register selects an SRAM word in the sub-block.

The read SRAM words are *AND*-accumulated for each cycle in an R_W-bit register using clk_P. Similarly, the look-up is completed for a W-bit input word by reading and *AND*-accumulating SRAM words from each sub-block of the SRAM in P internal clock cycles or one system cycle. Consequently, the P *AND*-accumulated SRAM words are produced as match word using clk_S. The timing diagram in Figure 8 elaborates the search operation of the proposed TCAM memory architecture shown in Figure 6 with a multipumping factor of $P = 2$.

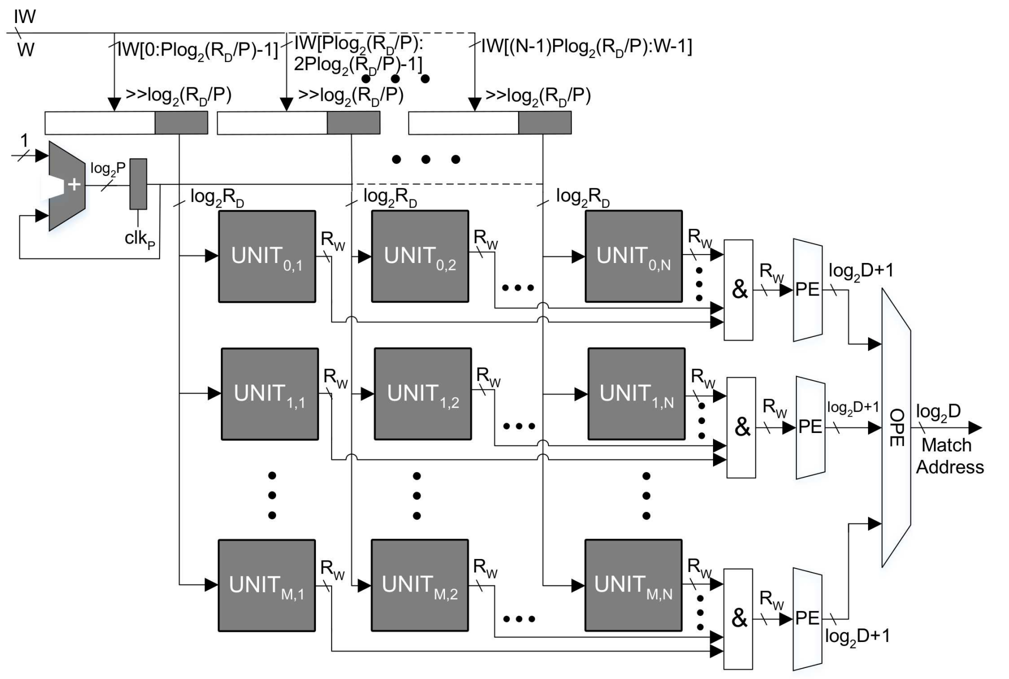


Figure 7: Organization of the proposed TCAM memory units for a large storage capacity: (*IW*: input word, *PE*: priority encoder, *OPE*: overall priority encoder)

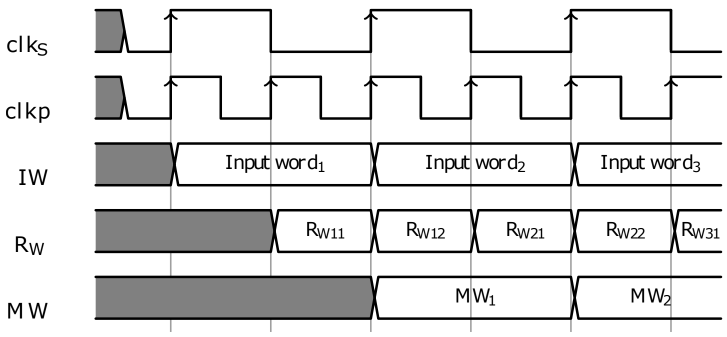


Figure 8: Timing diagram for the search operation in our proposed TCAM with a multipumping factor $P = 2$: (IW: input word, R_W: SRAM word read, MW: match word)

4. Modular Architecture

TCAM design of large storage capacity is implemented as a cascade of $M \times N$ proposed design TCAM memory units as shown in Figure 7. An incoming W-bit TCAM word is divided into N sub-words of $Plog_2(R_D/P)$-bits with the bit ranges shown in Figure 7. The resultant sub-words are stored in N shift registers of size $Plog_2(R_D/P)$-bits on clk_S. The $log_2 R_D$-bit indexes from the N shift registers are provided to the corresponding M TCAM memory units of the N columns of the proposed design in parallel using clk_P, as shown in Figure 7. All TCAM memory units of the design operate in parallel using clk_P. The R_W-bit match words from each row of the TCAM memory units are bit-wise *ANDed* on clk_S, and the results are provided to the associated priority encoder (PE) units. The $log_2 D$-bit match address and the match information from each PE unit are provided to the overall priority encoder unit, which eventually forwards a match address based on the priority. The proposed TCAM design registers an input word and produces a match word as output on clk_S.

The update of a TCAM word is performed in each TCAM memory unit of the design in parallel. The worst-case update latency of the proposed design comprises R_D/P system cycles.

5. Effect of Multipumping SRAM on the Memory Usage and Throughput

Multipumping results in a useful reduction in SRAM memory usage for the design of TCAM on FPGA. The configured SRAM memory blocks in our proposed design with the multipumping factor of P implements traditional TCAM divisions of size $Plog_2(R_D/P) \times R_W$ as shown in Figure 6. The TCAM bits storage capacity of SRAM blocks in the proposed design increases with an increase in P. The upper bound on the multipumping factor P is $R_D/2$, i.e. $R_D/2$ sub-blocks in the SRAM and each sub-block consists of two SRAM words.

Multipumping divides the achievable internal clock frequency of the design by the multipumping factor, to obtain the operating frequency of the overall system [42, 43, 44, 48]. Although an increase in the multipumping factor P results in a higher memory efficiency for the design of TCAM, only the use of small multipumping factors is practical in order to avoid a significant drop in the operating frequency of the overall system. Overall multipumping factor P controls a tradeoff between the SRAM memory efficiency and speed of the proposed design.

B. Implementation Setup and Results

To verify our proposed design we implemented it on a Xilinx Virtex-6 FPGA device (xc6vlx760). The proposed design was implemented using the Xilinx ISE 14.7 design tool, and verified through behavioral and post-route simulations using an ISim simulator.

We implemented our proposed design cases I and II on the Xilinx Virtex-6 FPGA device for 512×28 (14 Kb) and 512×32 (16 Kb) TCAM tables, with multipumping factors of $P = 4$ and $P = 2$, respectively. Our proposed design CASE-III implements a large TCAM table of size 1024×140 (140 Kb), with a multipumping factor of $P = 4$. We have selected small multipumping factors of $P = 4$, 2, and 4, in our proposed design cases I, II, and III, to avoid lower operating frequencies of the overall system.

Table 2 lists the FPGA resource utilization slice registers (SRs), look-up tables, and BRAMs for the implementation of our proposed design cases I, II, and III. The post place & route results show that the proposed design cases I, II, and III could achieve internal clock frequencies of 475 MHz, 475 MHz, and 349 MHz and multipumping factors of $P = 4$, 2, and 4, giving the system clock frequencies of 119 MHz, 237 MHz, and 87 MHz, respectively.

Table 2: FPGA resource utilization of the proposed design.

Proposed design	**TCAM size** ($D \times W$)	**Slice registers**	**LUTs**	**BRAMs (36 Kb)**
CASE-I ($P = 4$)	512×28	536	968	8
CASE-II ($P = 2$)	512×32	1593	1515	16
CASE-III ($P = 4$)	1024×140	6287	7516	80

C. Performance Evaluation & Comparison

The performance of our proposed design is evaluated based on its comparison with the existing SRAM-based TCAM solutions on FPGAs.

1. SRAM Memory Utilization

SRAM-based TCAM solutions implement a traditional TCAM of depth D and width W by cascading SRAM blocks of size $R_D \times R_W$ on FPGAs. The minimum overall SRAM memory requirement of the existing SRAM-based TCAM solutions on FPGAs can be formulated as (6) shown below:

$$\sum_{M=1}^{\frac{D}{R_W}} \sum_{N=1}^{\frac{W}{log_2 R_D}} (R_D \times R_W) = \left(\frac{D}{R_W}\right)\left(\frac{W}{log_2 R_D}\right)(R_D \times R_W)$$

$$= DW\left(\frac{R_D}{log_2 R_D}\right) \quad (1)$$

The overall memory requirement of the proposed design for the implementation of a $D \times W$ size traditional TCAM using $R_D \times R_W$ size SRAM blocks is devised as (7) shown below:

$$\sum_{M=1}^{\frac{D}{R_W}} \sum_{N=1}^{\frac{W}{Plog_2(R_D/P)}} (R_D \times R_W)$$

$$= \left(\frac{D}{R_W}\right)\left(\frac{W}{Plog_2(R_D/P)}\right)(R_D \times R_W)$$

$$= DW\left(\frac{R_D}{Plog_2(R_D/P)}\right) \quad (2)$$

Equation (7) describes that the SRAM memory usage of our proposed design is $\frac{R_D}{Plog_2(R_D/P)}$ times that of the corresponding traditional TCAM table of size $D \times W$.

Our proposed design achieves a considerable reduction in the SRAM memory usage by a factor of $\frac{1}{P[1-log_2 P/log_2 R_D]}$, when compared with that of the existing approaches as described using (3) as follows:

$$\frac{DW\left(\frac{R_D}{Plog_2(R_D/P)}\right)}{DW\left(\frac{R_D}{log_2R_D}\right)} = \frac{log_2R_D}{Plog_2(R_D/P)}$$

$$= \frac{log_2R_D}{P[log_2R_D - log_2P]} = \frac{1}{P[1 - log_2P/log_2R_D]} \quad (3)$$

The usage of BRAMs in our proposed design is compared with those of previous approaches in Column 5 of Table 3. Our proposed TCAM design CASE-I emulates a 14 Kb traditional TCAM, achieving a lower BRAMs utilization of 8 BRAMs compared with the usage of 56, 40, 40, 32, and 64 BRAMs for previous approaches in [20], [21], [22], [37], and [25], respectively for an 18 Kb traditional TCAM emulation. The proposed design CASE-III emulates a large TCAM of size 1024×140 using 80 BRAMs. It achieves a lower BRAMs utilization compared with the large TCAM implementations of size 1024×150 and 504×180 in the previous approaches [17] and [36], using 272 and 140 BRAMs, respectively.

2. Throughput

The operational speed of our proposed design is compared with those of previous approaches in column 4 of Table 3. Our proposed design cases I and II emulates traditional TCAM of size 14 Kb and 16 Kb achieving operating frequencies of 119 MHz and 237 MHz with multipumping factors of $P = 4$ and 2 respectively. The operating frequency of our proposed design CASE-II is higher than previous works in [20, 21, 22, 25, 37] for an 18 Kb traditional TCAM emulation.

Our proposed design methodology is more useful for the design of large storage capacity TCAMs. The TCAM memory units of our proposed design *AND*-accumulate SRAM words from the sub-blocks of the SRAM blocks in each system cycle, reducing the complexity of the *AND* operation units of the overall architecture, as shown in Figures 6 and 7. This further prevents the *AND* operation units from limiting the operating frequency of wide pattern TCAMs designs on FPGA. Our proposed design uses fewer BRAMs, thus alleviating the overall routing complexity of the design on FPGA. The divided *AND* operation complexity and reduced routing complexity makes our proposed design more practical for large storage capacity TCAMs.

The system frequency of our proposed design CASE-III emulating a large capacity TCAM of 140 Kb is 87 MHz, which is comparable with the maximum achievable frequency 97 MHz in previous work [17] implementing a large size TCAM of 150 Kb. While the SRAM memory usage of our proposed design CASE-III is 70% lower than that of [17].

Our proposed design provides increased design flexibility in terms of the speed vs memory usage tradeoff. The designer must consider the important design factors such as the required storage capacity, relative availability of BRAMs on the target FPGA, and required throughput for the selection of the multipumping factor in our proposed design.

3. Performance per Memory

Considering the time-space tradeoff, we used the performance evaluation metric *performance per memory* from [49], given by (8).

$$\frac{Throughput(Gb/s)}{Normalized\ Memory\ [Memory(Kb)/TCAM\ Depth]} \tag{4}$$

Table 3 compares the performance per memory of our design with previous FPGA-based TCAMs. The depth and pattern width of traditional TCAMs implemented in previous studies are listed in the third column. For a fair comparison, the speed results of the compared works with technology differences are normalized to 40 nm, using (9) from [50]. The speed results in parenthesis represent the original data reported in the respective papers.

$$T^* = T \times \left[\frac{40(nm)}{Technology(nm)}\right] \times \left[\frac{VDD}{1.0}\right] \tag{5}$$

where T represents the original delay time, and T^* denotes the normalized delay time for 40 nm CMOS technology with a supply voltage of 1.0 V. The proposed design cases I and II implemented 14 Kb and 16 Kb traditional TCAMs using 288 Kb and 576 Kb SRAM memory with operating frequencies of 119 MHz and 237 MHz, respectively. The proposed design cases I and II achieved a performance per memory of 5.78 $((Gb/s \times TCAMDepth)/Kb)$ and 6.58 $((Gb/s \times TCAMDepth)/Kb)$, respectively.

Table 3: Performance per memory comparison of the proposed TCAM with previous approaches.

Architecture	**FPGA**	**TCAM size** ($D \times W$)	**Speed** (MHz)	**BRAMs** (36 Kb)	**Memory usage** (Kb)	**Throughput** (Gb/s)	**P/M**$^{(a)}$
Locke-[37]	Virtex-6	512×36	166	64	2304	5.84	1.3
Jiang-[17]	Virtex-7	1024×150	97 (139)	272	9792	14.26	1.49
Qian-[36]	Virtex-6	504×180	133	140	5040	23.38	2.34
REST-[25]	Kintex-7	72×28	35 (50)	1	36	0.96	1.92
HP-TCAM-[20]	Virtex-6	512×36	118	56	2016	4.15	1.05
Z-TCAM-[21]	Virtex-6	512×36	159	40	1440	5.59	1.99
E-TCAM-[22]	Virtex-6	512×36	164	40	1440	5.77	2.05
UE-TCAM-[16]	Virtex-6	512×36	202	32	1152	7.1	3.16
Proposed CASE-I	Virtex-6	512×28	119	8	288	3.25	5.78
Proposed CASE-II	Virtex-6	512×32	237	16	576	7.41	6.59
Proposed CASE-III	Virtex-6	1024×140	87	80	2880	11.90	4.25

$^{(a)}$ **P/M**: Performance per memory $((Gb/s \times TCAM\ Depth)/Kb)$

Table 3 shows that the performance per memory of the proposed design cases I and II are 1.83 times higher than that of UE-TCAM [16], which was the highest among the existing methods. Our proposed design CASE-III emulates a large TCAM of size 1024×140, achieving the performance per memory of 4.25 $((Gb/s \times TCAMDepth)/Kb)$, which is 2.85 times higher than for large TCAM of size 1024×150 in the existing study [17].

Our proposed design scales well in terms of the performance when evaluated for the design of a large storage capacity. Table 3 shows that the performance per memory of our proposed design CASE-III is slightly lower than the proposed design CASE-I (with the same multipumping factor of $P = 4$) while the implemented TCAM size of CASE-III is ten times greater than that of CASE-I.

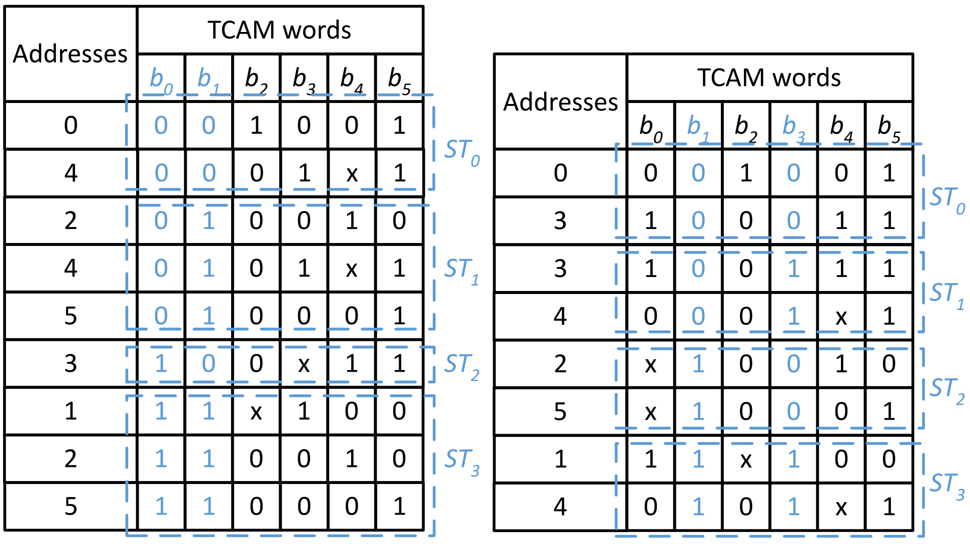
III. PRE-CLASSIFICATION- BASED ENERGY-EFFICIENT SRAM-BASED TCAM ARCHITECTURE

A. Proposed Classification Scheme

The proposed design uses the bits extracted from specific bit positions of the TCAM words to classify the TCAM table words into groups called as TCAM sub-tables. In the proposed partitioning scheme we extract log_2M classification bits from the specified bit positions of the TCAM words to produce M sub-tables. For example, suppose two bits are used for the classification of a sample TCAM table of size 6×6 is presented in Table 4. The TCAM table presented in Table 4 is classified using two different set of bit positions $S_1 = \{b_0, b_1\}$ and $S_2 = \{b_1, b_3\}$ as shown in Figure 9a,b, respectively. The sub-tables constructed based on the bit values (00, 01, 10, and 11) of bit positions $\{b_0, b_1\}$ are shown in Figure 9a. The number of TCAM words in the constructed sub-tables ST_0, ST_1, ST_2 and ST_3 varies based on the pattern of bits in the bit positions $\{b_0, b_1\}$ selected for the classification of Table 4. TCAM words with ‘x’ as bit value in the classification bit positions are stored in more than one sub-table. For example, the TCAM word at address 2 has the bit values of ‘x1’ at $\{b_0, b_1\}$, and is thus stored in both sub-tables ST_1 and ST_3. This redundancy expands the resultant TCAM sub-tables. The classification of any realistic dataset based on a specific set of bit positions may not necessarily produce sub-tables of the same size.

Table 4: A TCAM table of size 6 $\times$ 6.

Address	**TCAM Words**
0	001001
1	11x100
2	x10010
3	100x11
4	0x01x1
5	x10001



Addresses	TCAM words						
	b_0	b_1	b_2	b_3	b_4	b_5	
0	0	0	1	0	0	1	ST_0
4	0	0	0	1	x	1	
2	0	1	0	0	1	0	ST_1
4	0	1	0	1	x	1	
5	0	1	0	0	0	1	
3	1	0	0	x	1	1	ST_2
1	1	1	x	1	0	0	ST_3
2	1	1	0	0	1	0	
5	1	1	0	0	0	1	

(a)

Addresses	TCAM words						
	b_0	b_1	b_2	b_3	b_4	b_5	
0	0	0	1	0	0	1	ST_0
3	1	0	0	0	1	1	
3	1	0	0	1	1	1	ST_1
4	0	0	0	1	x	1	
2	x	1	0	0	1	0	ST_2
5	x	1	0	0	0	1	
1	1	1	x	1	0	0	ST_3
4	0	1	0	1	x	1	

(b)

Figure 9: Example of the proposed classification algorithm: (**a**) Description of Classification based on set of bit positions $S_1 = \{b_0, b_1\}$. (**b**) Description of Classification based on set of bit positions $S_2 = \{b_1, b_3\}$.

The proposed solution is based on the concept that the classification effectiveness of a set of bit positions varies from that of other bit positions for a specific dataset when the target of the classification is to construct balanced-size sub-tables. The classification example of the TCAM table in Table 4 using two different sets of bit positions is illustrated in Figure 9. It shows that the classification using $S_2 = \{b_1, b_3\}$ is more effective for the TCAM table presented in Table 4, as the constructed sub-tables are of balanced size (2), when compared with the unbalanced size sub-tables constructed for bit positions $S_1 = \{b_0, b_1\}$ (2, 3, 1, and 3) as explained above.

The constructed TCAM sub-tables are further mapped to the distinct rows of SRAM blocks in the proposed design. Figure 10 shows the mapping of the proposed classification scheme constructed sub-tables to SRAM memory. The contents of the four bit places $S_2 = \{b_2, b_3, b_4, b_5\}$ of the unbalanced size sub-tables shown in Figure 9a and that of bit places $S_2 = \{b_0, b_2, b_4, b_5\}$ of the balanced size sub-tables shown in Figure 9b are vertically partitioned into width of two, further mapped to the SRAMs with depth $D = 4$ shown in Figure 10a,b, respectively. It clearly illustrates that the SRAM memory requirement for storing balanced size TCAM sub-tables constructed for bit positions $S_2 = \{b_1, b_3\}$ is lower than that of unbalanced size sub-tables constructed for bit positions $S_1 = \{b_0, b_1\}$. The SRAM memory utilization overhead for pre-classifying the TCAM table contents in the proposed approach is minimal as balanced size sub-tables are constructed based on effective classification bits.

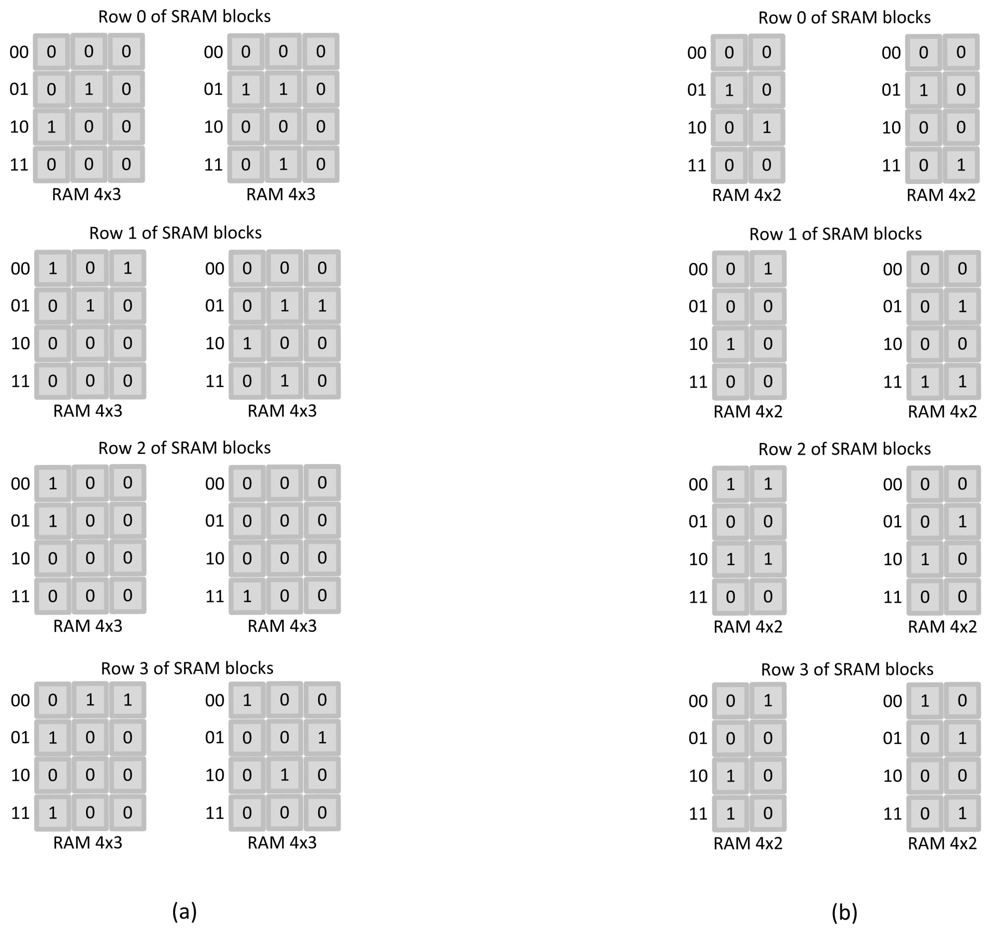


Figure 10: Example of mapping proposed classiifcation scheme constructed TCAM-subtables to SRAM: (**a**) Mapping TCAM sub-tables contents to SRAM constructed based on set of bit positions $S_1 = \{b_0, b_1\}$. (**b**) Mapping TCAM sub-tables contents to SRAM constructed based on set of bit positions $S_2 = \{b_1, b_3\}$.

Algorithm 1 describes the proposed classification scheme. It classifies the TCAM words of the $D \times W$ TCAM table into M sub-tables based on the comparison with the $log_2 M$-bit values extracted from specific bit positions. The resultant sub-tables formed are tested for the maximum depth bound (MDB) of

$(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$, where R_W is the width of the configured SRAM blocks of the design on FPGA and α is a scaling factor with integer values of $\alpha \geq 1$. If the number of TCAM words in the resultant sub-tables exceeds MDB, a subsequent set of bit positions is used for the classification of the TCAM table. In the worst-case scenario, all subsets of log_2M bit positions from W bit positions are used to classify the TCAM table. The worst-case classification complexity of the proposed Algorithm 1 is reduced by using a relaxed MDB for the construction of sub-tables. An increase in the value of α by one increases the MDB of the sub-tables by R_W. A relaxed MDB of the sub-tables results in an increased RAM memory usage, as the resultant sub-tables are mapped to the SRAM blocks of the proposed design. The value of α provides a trade-off between the time complexity of the proposed classification algorithm and the overall RAM memory usage of the proposed design.

The words of the M TCAM sub-tables are mapped to the M rows of the SRAM blocks of the architecture and the corresponding classification bit positions are used to configure the pre-classifier bit positions in the proposed architecture.

Algorithm 1 Algorithm for the classification of the TCAM table into M sub-tables.

```
INPUT: $D$ ternary words of $W$ bits: $T_{i,j}$, where $T_{i,j} \in \{0,1,\ \mathrm{x}\}^W, i = 0,1,\ldots,D-1$, All possible subsets of $log_2M$ bit positions from $W$ bit positions: $S_{u,v}$, where $u = 1,2,\ldots,\binom{W}{log_2M}$, $v = 0,1,\ldots,log_2M-1$.
OUTPUT: $M$ sub-tables (STs) with identification addresses of $A_M = 0,1,2,\ldots,M-1$, and each ST of $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$ ternary words of $W$ bits: $ST_{i,j}$, where $ST_{i,j} \in \{0,1,\ \mathrm{x}\}^W, i = 0,1,\ldots,(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W - 1$.
for $u = 1,2,\ldots,\binom{W}{log_2M}$ do
    for $i = 0,1,\ldots,D-1$ do
// Check for the maximum depth bound
        if ($Size_{A_M} == (\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$) then
            break
        else
// Extraction of classification bits & construction of sub-tables
            $C_{bits} \leftarrow Extract(S_{u,v}, T_{i,j})$
            if ($A_M == C_{bits}$) then
                $Add\_ST(A_M, T_{i,j})$
                $Size_{A_M} \leftarrow Size_{A_M} + 1$
            end if
        end if
    end for
end for
$C_{bits}$: Extracted classification bits
$Size_{A_M}$: Size of constructed sub-tables
```

B. EE-TCAM Proposed Architecture

The TCAM table of $D \times W$ size is classified into M sub-tables using the proposed classification scheme in Algorithm 1. The W-bit TCAM words of M sub-tables constructed are further divided into V sub-words of log_2R_D-bits. The resultant $M \times V$ sub-partitions of the TCAM table are mapped to the M rows of the V SRAM blocks in the proposed architecture as shown in Figure 12. Each TCAM table sub-partition of size $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W \times log_2R_D$ is implemented using an SRAM block. The SRAM block is a cascade of $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)$ number of $R_D \times R_W$ size SRAM blocks.

The proposed architecture comprises a pre-classifier unit and an SRAM-based TCAM. The pre-classifier unit of the proposed architecture is shown in Figure 11. The bit positions of the pre-classifier bits are specified using Algorithm 1 provided log_2M number of pre-classification bit positions. The pre-classifier bits are extracted from the log_2M bit positions of the incoming TCAM words using the log_2M number of select lines of W-to-1 multiplexers as shown in Figure 11. The extracted log_2M bits are further decoded to get an M-bit control signal that selectively activates at most one row of SRAM blocks of the proposed architecture.

The proposed SRAM-based TCAM architecture is shown in Figure 12. The incoming W-bit TCAM word is divided into V sub-words of log_2R_D-bits. The V sub-words are provided as addresses to the selected row of V SRAM blocks in parallel and V SRAM words are read. The V SRAM words read undergo a bit-wise *AND* operation and the resultant matching information bit vector is provided to the associated PE. The PE unit encodes the highest-priority matching bit position with the level high as the matching address.

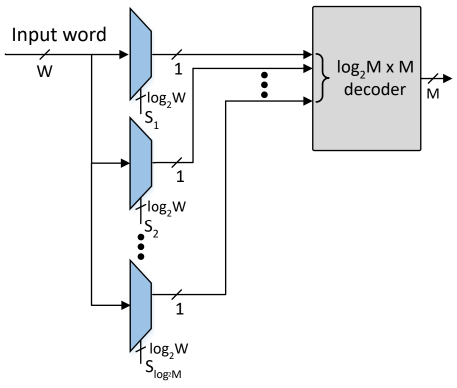


Figure 11: Architecture of the pre-classifier unit: (S: Select lines for extracting the pre-classification bits, log_2W: Number of bits in the , log_2M: Number of pre-classifier bits, M: Number of enable control bits for the M rows of SRAM blocks).

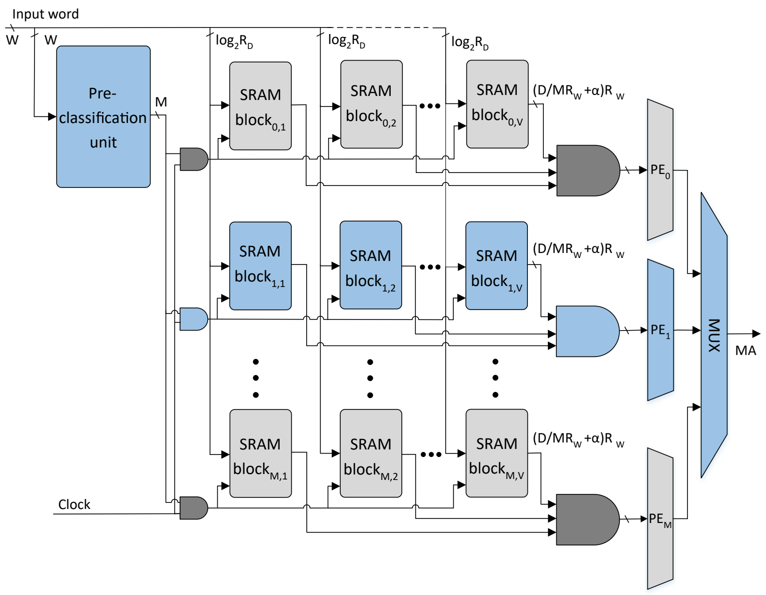


Figure 12: Proposed overall architecture: (M: Number of rows of SRAM blocks, V: Number of SRAM blocks in a row, PE: priority encoder unit, MA: matching address, log_2R_D: Address bits of the configured SRAM blocks, $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$: Configured SRAM blocks words width).

The proposed TCAM design maps a new TCAM table dataset of the same size to the SRAM blocks of the configured architecture on FPGA. The SRAM blocks of the architecture has storage space for $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$ number of $log_2 R_D$-bit TCAM sub-words. Algorithm 1 finds the set of classification bit positions, which makes TCAM sub-tables considering the maximum depth limitation of the configured architecture SRAM blocks on FPGA. The updated TCAM sub-tables are mapped to the SRAM blocks of the design on FPGA. The corresponding classification bit positions from Algorithm 1 configure the pre-classifier unit. The classification bits are now extracted from the updated set of bit positions for the incoming TCAM words. The proposed solution performs reconfiguration of the hardware design on FPGA in two cases: first, when the number of TCAM words in the Algorithm 1 constructed TCAM sub-tables exceeds the storage space of the configured architecture SRAM blocks, resulting in a relaxed MDB on the updated TCAM sub-tables. Second, when a TCAM table of different size is implemented in the proposed design.

During run-time, an update process of a TCAM word in proposed design includes the writing of the update word to the respective TCAM sub-table first, and the updated sub-table is then written to the corresponding row of SRAM blocks in the proposed architecture. The number of TCAM words in updated sub-tables is tested for the MDB of $(\left\lfloor \frac{D}{MR_W} \right\rfloor + \alpha)R_W$ as this is the storage capacity of a row of configured SRAM blocks in proposed architecture. Owing to the presence of the don't-care bits (x) in the TCAM words, EE-TCAM in the worst case writes the entire used SRAM memory to complete the update process of a TCAM word. The partitioned sub-tables are written in parallel to the corresponding SRAM blocks in the proposed architecture, and the depth R_D of the configured BRAMs determines the update latency of EE-TCAM design. The update latency of EE-

TCAM is 513 cycles. While native TCAMs have also comparable worst case TCAM write time of O(N) for updating a TCAM word, where N is the number of words in the TCAM table [27, 38, 51, 52].

C. EE-TCAM FPGA Implementation & Results

We used a Xilinx Virtex-6 XC6VLX760-2FF1760 FPGA device for the proposed EE-TCAM design of a 512×36 TCAM. Three design cases of 512×36 TCAM were implemented with the following design parameters: $[M = 4, V = 4]$ denoted EE-TCAM-I and $[M = 8, V = 4]$ denoted EE-TCAM-II using 36 kb BRAMs and $[M = 16, V = 4]$ denoted EE-TCAM-III using 18 kb BRAMs. Algorithm 1 was used to classify 512 TCAM words into $M =$ 4, 8, and 16 TCAM sub-tables.

The minimum depth limitation on the configuration of BRAM is $512 = 2^9$. The storage capacity of BRAMs is maximum when used in minimum depth configurations of 512×36 for 18 kb size and 512×72 for 36 kb size [23]. Thus, the contents of the TCAM sub-tables formed were further divided into $V = 4$ sub-words of width$= 9$, which addresses the configured BRAMs of depth $2^9 = 512$. The resultant $M \times V$ sub-partitions were mapped to the configured BRAMs of the EE-TCAM designs on FPGA. The proposed EE-TCAM designs were implemented using Xilinx ISE 14.7 and were verified via behavioural and post-route simulations using the Xilinx ISim simulator. The Xilinx Place and Route report was used to evaluate the FPGA resource utilization and speed. The Xilinx Xpower Analyzer tool [53] was used for the estimation of the dynamic power consumption of the design. Table 5 lists the FPGA resource utilization parameters such as slice registers, LUTs, and BRAMs of the EE-TCAM I, II, and III designs for implementing a 512×36 TCAM table.

Table 5: FPGA resource utilization of EE-TCAM on Xilinx Virtex-6.

Proposed Design Cases	**Slice Registers**	**LUTs**	**BRAMs**
EE-TCAM-I [M = 4]	652	1535	32
EE-TCAM-II [M = 8]	687	1455	32
EE-TCAM-III [M = 16]	712	1419	32

D. EE-TCAM Performance Evaluation & Comparison

1. Scalability of EE-TCAM

We have evaluated the scalability of EE-TCAM using important performance evaluation metrics, like memory utilization, clock rate, and power consumption implementing various sizes of TCAM tables. Table 6 lists the memory utilization, clock rate, and power consumption for the TCAM tables of width W = 36, 54, and 72 for depth D = 256, 512, and 1024 with TCAM sub-tables of M = 4, 8, and 16 configurations implemented using EE-TCAM.

Table 6 shows that the memory usage of EE-TCAM grows proportionally with an increase in the width or depth of the implemented TCAM table size. The memory usage of EE-TCAM remains same with an increase in the number of TCAM sub-tables M.

Table 6 shows that EE-TCAM achieves high clock rates of (233 MHz–346 MHz) for the specified sizes of TCAM tables. EE-TCAM achieve a linear decrease in the clock rate with a linear increase in the depth or width of the implemented TCAM tables when M is constant. The delay in EE-TCAM increases because of the increase in the number of wide bit-wise ANDing operations and associated routing delay. The EE-TCAM showed an increase in the clock rate with an increase in the number of TCAM sub-tables M owing to the reduced bit-wise ANDing complexity of a row of SRAM blocks.

Table 6 illustrates that the dynamic power consumption of EE-TCAM increases linearly with an increase in the width and depth of the implemented TCAM table when M is constant. The memory usage of a row of SRAM memory blocks increases with an increase in the size of the implemented TCAM table, which results in an increase in the power consumption of EE-TCAM. The dynamic power consumption of EE-TCAM decreases with an increase in the number of TCAM sub-tables M owing to the activation of reduced size SRAM memory of a row of SRAM blocks.

The EE-TCAM design implementations for depth $D = 256$ and width $W = 36$, 54, 72 with $M = 16$ does not follow the same performance trend in memory usage and power consumption as shown in Table 6. The memory usage of EE-TCAM implementations for the specified sizes with $M = 16$ gets double from that of the same sizes implementations with $M = 8$. This is because the memory size of a row of SRAM blocks for $M = 16$ remains the same (72 kb) as that of $M = 8$ and the number of rows of SRAM-blocks gets double (16 from 8). This is due to the minimum 36 bit width configuration for depth = 512 of 18 kb BRAM on Xilinx FPGAs [18]. This increase in the memory usage of a row of SRAM-blocks results in the increased power consumption of proposed EE-TCAM implementations for specified sizes with $M = 16$ as illustrated in the Table 6.

Table 6: Performance trend for increasing EE-TCAM depth (D), width (W) in various configurations.

No. of TCAM	**TCAM Depth**	**256**			**512**			**1024**		
Sub-Tables	**TCAM Width**	**36**	**54**	**72**	**36**	**54**	**72**	**36**	**54**	**72**
		Memory utilization (kb)								
$M = 4$		576	864	1152	1152	1728	2304	2304	3456	4608
$M = 8$		576	864	1152	1152	1728	2304	2304	3456	4608
$M = 16$		1152	1728	2304	1152	1728	2304	2304	3456	4608
		Clock rate (MHz)								
$M = 4$		326	313	299	276	260	243	264	252	233
$M = 8$		338	323	308	321	306	261	297	272	246
$M = 16$		346	335	317	336	316	270	310	293	251
		Power consumption (mW)								
$M = 4$		26.3	37.3	55.2	33.7	52.9	70	62	89.1	134
$M = 8$		23.6	36.8	51.3	27.7	50.3	64.5	36.1	59.1	80
$M = 16$		32.1	54.2	72.7	27.7	47.9	61.4	30.9	55.5	72.1

2. Performance Trade-Off with Increase in the Number of TCAM Sub-Tables (M)

The number of TCAM sub-tables M determines the depth of the sub-tables formed, and when mapped to the SRAM memory determines the RAM memory usage of each row of SRAM blocks in the EE-TCAM design. An increase in M reduces the RAM memory usage of each row of SRAM blocks in the EE-TCAM design. This leads to a reduction in the overall power consumption of the proposed design with an increase in M as the reduced memory of at most one row of SRAM blocks is energized per lookup for incoming words.

To evaluate the impact of the change in M on the performance of the proposed EE-TCAM design, we implemented three EE-TCAM design cases I, II, and III, with $M = 4$, 8, and 16, respectively, using Xilinx BRAM resource for a 512×36 TCAM table. The RAM memory usage of a row of SRAM blocks in the EE-TCAM design shows a decreasing trend with the increase in the value of M. The proposed design cases EE-TCAM I, II, and III, activate at most one row of SRAM blocks per lookup i.e., 288 kb (8 BRAMs of size 36 kb), 144 kb (4 BRAMs of size 36 kb), and 72 kb (4 BRAMs of size 18 kb), respectively. The individual BRAM resource power consumption of the EE-TCAM design cases I, II, and III shows a decreasing trend 18.41 mW, 9.03 mW, and 4.16 mW, respectively, as shown in Figure 13. The increase in the number of TCAM sub-tables M has a decreasing impact on the size of the activated BRAM resource in the proposed EE-TCAM design. Thus, the individual power consumption of the BRAM resource in proposed EE-TCAM shows a decreasing trend with an increasing value of M.

The overall power consumption of the EE-TCAM design III does not show an exactly decreasing trend. This is because of the increased individual clock and signal resource power consumption as shown in Figure 13. Compared to EE-TCAM I with $M = 4$, the number of design units in EE-TCAM II ($M = 8$) and EE-TCAM III ($M = 16$) doubles and quadruples, respectively. Thus, EE-TCAM II and EE-TCAM III show an increased clock distribution and interconnect power consumption as shown in Figure 13.

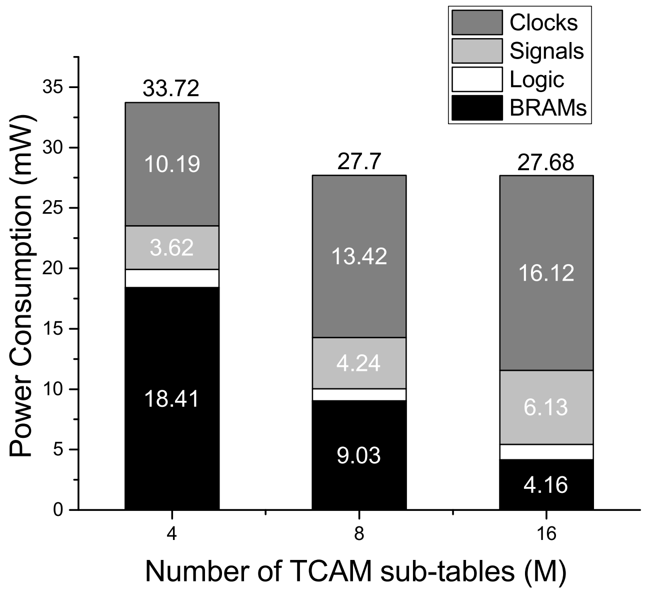


Figure 13: Trade-Off between the Number of TCAM Sub-Tables (M) and Power Consumption Performance

The operational frequencies of the EE-TCAM design listed in Table 6 shows an increasing trend with the increase in the number of TCAM sub-tables M, owing to the reduced complexity of bit-wise *AND* operation and PE units, as the size of the SRAM words decreases with the increase in the value of M.

Overall, the trade-off between the number of TCAM sub-tables M and the power consumption and throughput performance of the proposed EE-TCAM design showed that considerable improvement in performance is achieved with an increase in M.

3. Power Consumption

RAM memory accounts for a major proportion of the power consumption in the design of SRAM-based TCAMs. In the SRAM-based solutions of TCAM, the depth of TCAM table determines the width of the SRAM memory and is implemented as a cascade of $\frac{D}{R_W}$ number of SRAM blocks. The width of TCAM table is encoded as the address of the SRAM memory and implemented as a cascade of $\frac{W}{log_2 R_D}$ number of SRAM blocks. The minimum achieved power consumption of the existing SRAM-based TCAM design methodologies on FPGAs in terms of the SRAM blocks used can be formulated using Equation (6) as follows:

$$P_E = \left\lceil \frac{D}{R_W} \right\rceil \left\lceil \frac{W}{log_2(R_D)} \right\rceil \times P_{SU} \tag{6}$$

where P_{SU} denotes the power consumption of an SRAM block. The proposed design activates at most $1/M$ number of entire used SRAM blocks to complete the lookup operation for an incoming TCAM word. Thus, achieves a considerable reduction in the power consumption by a factor of M as expressed using Equation (7).

$$P_P = \frac{1}{M} \left\lceil \frac{D}{R_W} \right\rceil \left\lceil \frac{W}{log_2(R_D)} \right\rceil \times P_{SU} \tag{7}$$

Column 7 of Table 7 shows the power consumption comparison of our proposed EE-TCAM design with previous works. It shows that the power consumption of the proposed EE-TCAM design is at least two times lower than that of [22, 54, 55, 56, 57] for implementing a 18 kb size 512×36 TCAM table.

4. Power Consumption Per Performance Comparison

Table 7 presents a comparison of the proposed EE-TCAM designs with the existing state-of-the-art FPGA-based TCAMs with respect to the power consumption per performance from [49] given by Equation (8).

$$\frac{\text{Normalized power consumption } [\mu\text{W/ TCAM Depth}]}{\text{Performance [Gb/s]}} \tag{8}$$

For a fair comparison with [25] and [23], we normalized their reported frequency and power consumption results to 40-nm CMOS technology using Equation (9), modified from previous study in [58].

$$\begin{aligned} F^* &= F \times \left(\frac{Technolgy(nm)}{40(nm)}\right) \times \left(\frac{1.0}{VDD}\right) \\ P^* &= P \times \left(\frac{40(nm)}{Technology(nm)}\right) \times \left(\frac{1.0}{VDD}\right)^2 \end{aligned} \tag{9}$$

The normalized frequency and power consumption factors are denoted as F^* and P^*, respectively, when considering 40-nm CMOS technology and a supply voltage of 1.0 V. The original results of the respective works are reported in parentheses.

EE-TCAM designs I, II, and III implemented a 512 × 36 (18 kb) TCAM table consuming 33.72 mW, 27.70 mW, and 27.68 mW, respectively, and achieved the speed of 276 MHz, 321 MHz, and 336 MHz, respectively. Thus, the power consumption per performance of EE-TCAM I, II, and III was 6.8 μW/(TCAMDepth.Gb/s), 4.8 μW/(TCAMDepth.Gb/s), and 4.6 μW/(TCAMDepth.Gb/s), respectively. Table 7 shows that the power consumption per performance of the proposed EE-TCAM designs is at least three times lower than that of UE-TCAM [54], which is the lowest among the existing FPGA implementations of SRAM-based TCAMs.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar

Table 7: Comparison of the power consumption per performance with previous works.

Architecture	FPGA	TCAM Size [D × W]	Speed [MHz]	AM/L [(a)]	Throughput [Gb/s]	Power [mW]	PC/P [(b)]
Locke [57]	Virtex-6	512 × 36	166	2304	5.8	253	85
Qian [24]	Virtex-6	504 × 180	133	5040	23.4	2548	216
REST [25]	Kintex-7	72 × 28	35 (50)	36	1.4	161 (113)	2329
HP-TCAM [55]	Virtex-6	512 × 36	118	2016	4.2	188	89
Z-TCAM [56]	Virtex-6	512 × 36	159	1440	5.6	109	38
E-TCAM [22]	Virtex-6	512 × 36	164	1440	5.8	91	31
UE-TCAM [54]	Virtex-6	512 × 36	202	1152	7.1	78	21
Jiang [23]	Virtex-7	1024 × 150	97 (139)	9792	20.4	4587 (3211)	315
EE-TCAM-I	Virtex-6	512 × 36	276	288	9.7	33.72	6.8
EE-TCAM-II	Virtex-6	512 × 36	321	144	11.3	27.7	4.8
EE-TCAM-III	Virtex-6	512 × 36	336	72	11.8	27.68	4.6

(a) **AM/L**: Activated memory/lookup [kb]; (b) **PC/P**: Power consumption/performance μW/(TCAMDepth.Gb/s).

lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

IV. DYNAMICALLY RE-CONFIGURABLE ENERGY- AND RESOURCE- EFFICIENT TCAM ARCHITECTURE FOR FPGAS

A. Hardware Architecture of Proposed TCAM: DURE

1. Basic Idea

Table 8: A 3×4 TCAM table

Address	TCAM words
0	1001
1	x100
2	01xx

The basic design of the DURE TCAM implementation technique is illustrated in Fig. 14, which shows a simplified implementation of the 3×4 TCAM table presented in Table 8. The TCAM bits of each word are mapped to a specific row of AND-cascaded RAM. When an input word is applied to the rows of RAM blocks, the match/mismatch information read is AND-cascaded to obtain the final match result.

2. Building Blocks of DURE on FPGA

DURE utilizes the on-chip distributed RAM available in state-of-the-art Xilinx FPGAs. The SLICEM resources of a Xilinx FPGA comprise four LUTRAM

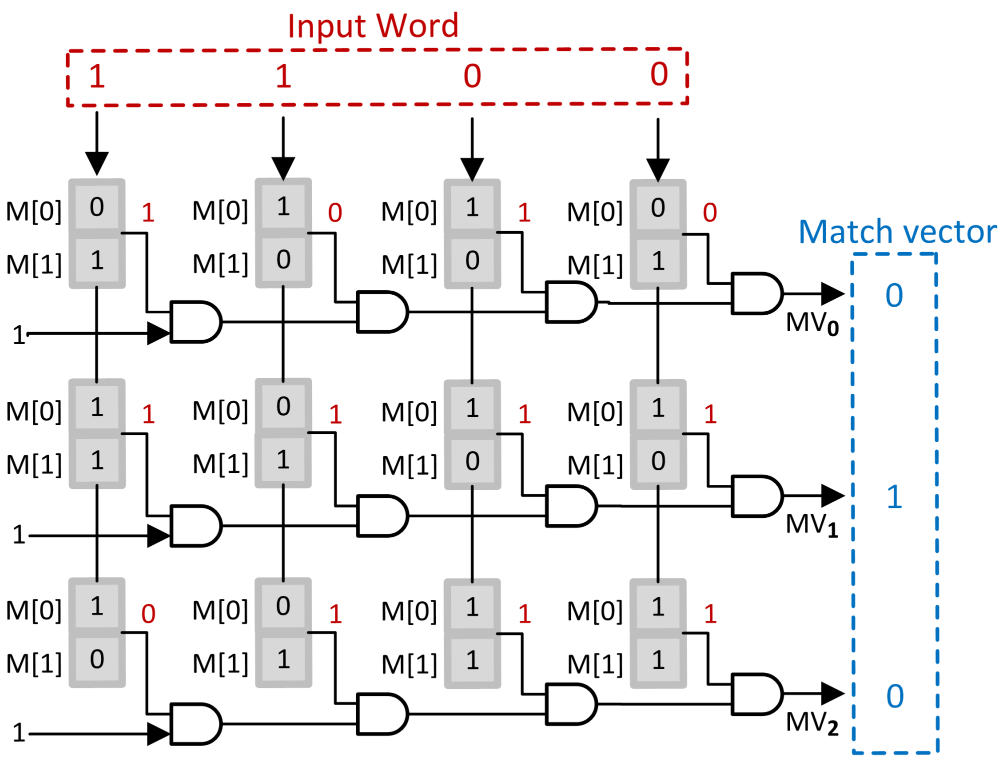


Figure 14: A simplified implementation of DURE.

cells, which are configured as quad-port RAM (three read/one write) in DURE. The four LUTRAM cells are called LUTA, LUTB, LUTC, and LUTD in Xilinx terminology, as shown in Figure 15. The LUTRAM cells from the same SLICEM share a common write address port. Four distinct bits are written to these LUTRAM cells using data from the ports DIA, DIB, DIC, and DID, while the write address is applied to LUTD. At the same time, the address ports of the three LUTRAM cells LUTA, LUTB, and LUTC are available for reading. In this manner, the structure creates quad-port RAM (three read/one write) with three read ports (read addresses are applied in parallel to LUTA, LUTB, and LUTC) and one write port (the write address is applied to LUTD).

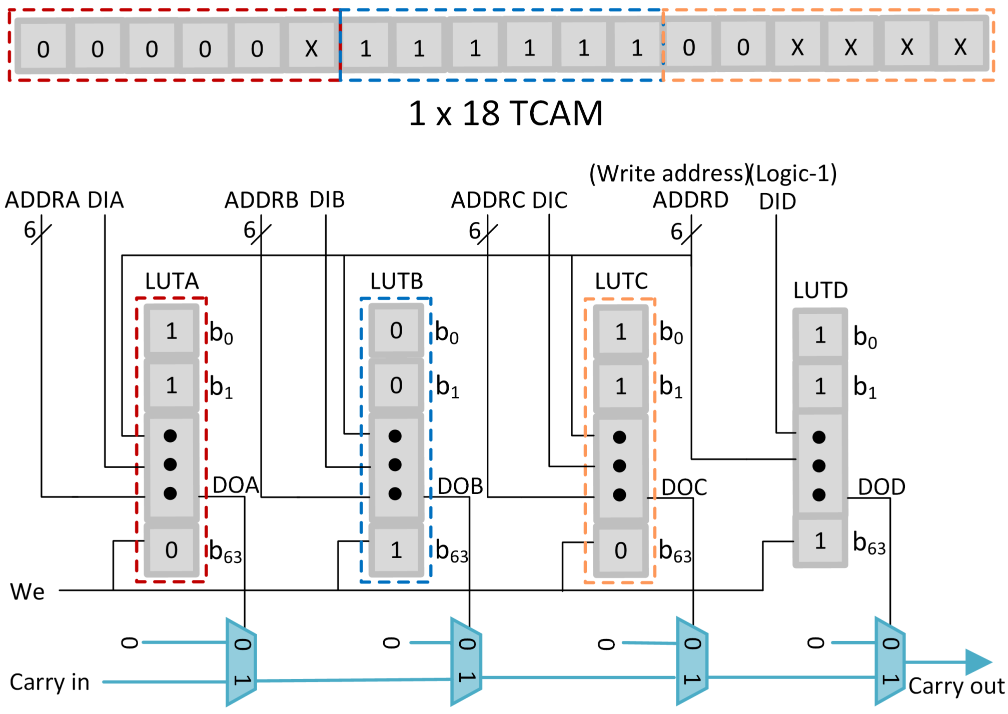


Figure 15: Architecture of the basic memory block of DURE for implementing an 1×18 TCAM on FPGA.

The quad-port RAM structure illustrated in Figure 15 implements an 1×18 TCAM using the LUTA, LUTB, and LUTC LUTRAM cells. Each of these 64 bit LUTs implements six bits of the TCAM. The input words are applied to the address ports of the LUTRAMs LUTA, LUTB, and LUTC in parallel, and match information bits are read at their outputs. The hardwired fast carry chain structure present in the slice is then used for ANDing match bits to yield the final match result. This SLICEM configuration constitutes a basic memory (BM) block in DURE. The architecture of the BM block dictates that LUTD cannot be used for storing TCAM bits, as the address port of LUTD acts as the common write address port. To implement wide TCAM words, this implementation scheme

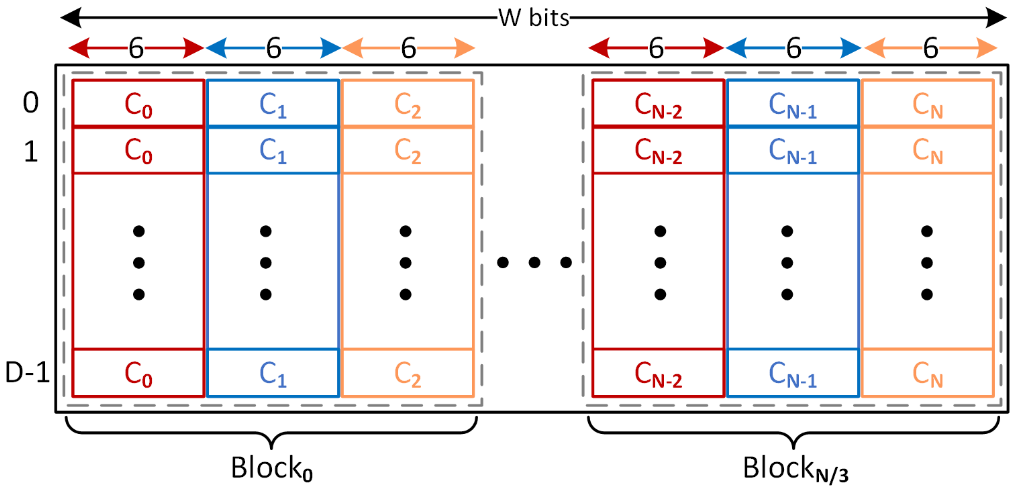


Figure 16: Partitioning TCAM table contents into 6-bit chunks.

must transfer the partial match results from the carry chain in the current slice to the carry chain in the next one. To forward the match result to the carry chain structure of the next slice, the output of the LUTD DOD must be permanently set to logic-1. For this purpose, all the bits of the LUTD are initialized to “1,” and a logic-high is connected to its DID input.

3. Architecture of the Proposed DURE

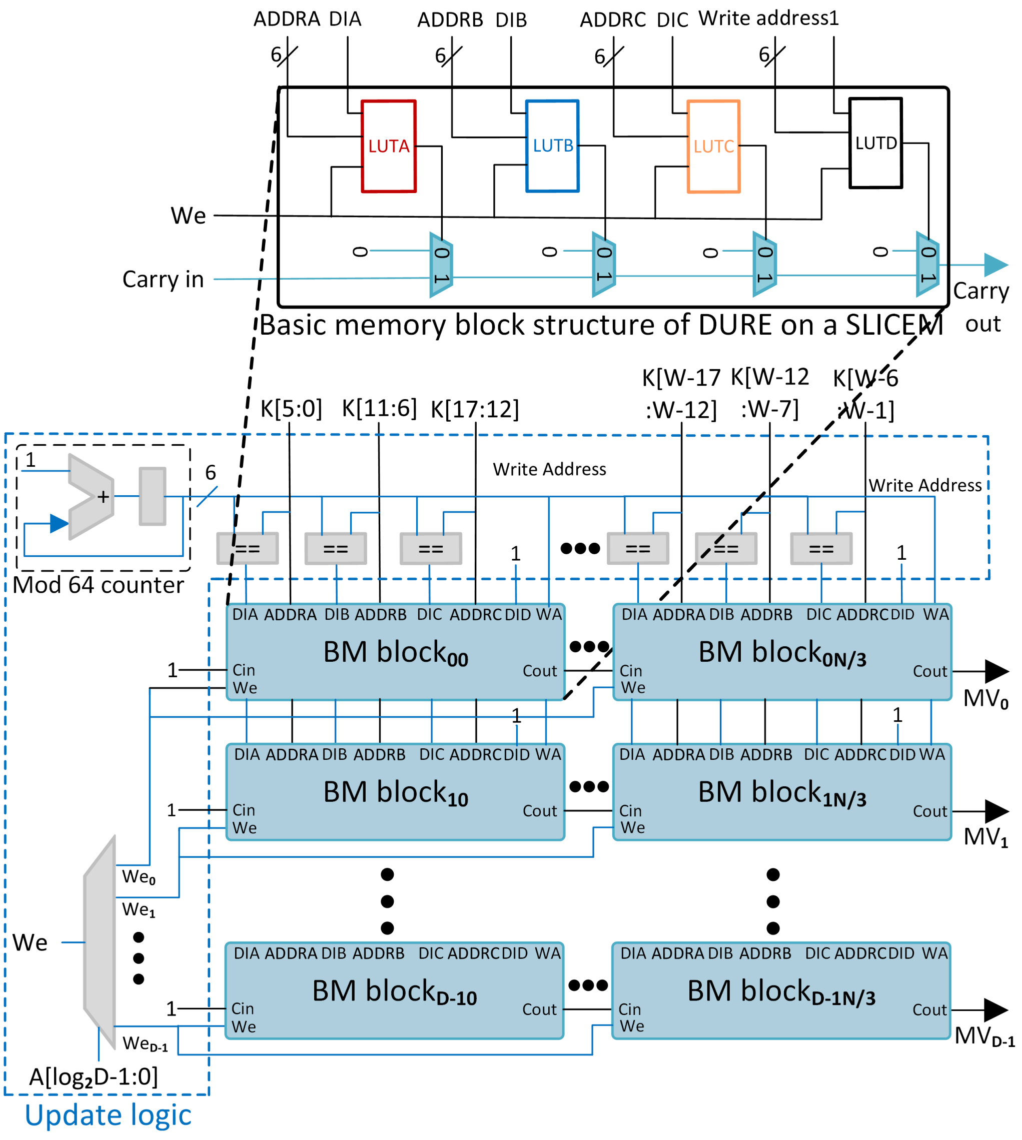


Figure 17: Architecture of the proposed DURE.

A TCAM table of word width W and depth D is illustrated in Figure 16, where it is partitioned into chunks of equal size. The D words of this TCAM table are divided into chunks of six bits, denoted by C_0, C_1, ..., C_{N-1}, where $N = W/6$. Each three successive chunks of the TCAM table are annotated as basic blocks. Therefore, each basic block consists of D sub-words with a word width of 18 bits.

The architecture of the proposed DURE is illustrated in Figure 17. This primarily includes a cascaded array of the proposed BM blocks implementing the TCAM and a separate update logic. The entire contents of the $D \times W$ TCAM table are mapped onto the array of BM blocks on the FPGA. The sub-words of the basic blocks are mapped onto the LUTRAMs of the BM blocks, with each six-bit chunk of the TCAM mapped onto a 64-bit LUTRAM cell. For example, the three chunks of the first basic TCAM block, C_0, C_1, and C_2, are implemented using LUTA, LUTB, and LUTC from the BM block, as shown in Figures 16 and 17.

The input word bits are applied in parallel to the D rows of the $N/3$ BM blocks. The match bits from the LUTRAMs LUTA, LUTB, and LUTC of each BM block are AND-cascaded through the dedicated hardwired carry chain structures, which extend into multiple slices [59, 60]. This implementation technique cascades the BM blocks of each row, BM Block$_0$, BM Block$_1$, ..., BM Block$_{N/3}$, through carry chains. Consequently, the match bits from the LUTRAMs, which each implement a TCAM word, are AND-cascaded to obtain the final match result. The use of these extended fast computation structures allows the cascading operations the be performed without consuming additional logic, thus also saving any associated routing resources.

4. Dynamic Update

Algorithm 2 Update operation in DURE.

```
    INPUT: A $W$-bit update TCAM word: $U_0, U_1, ..., U_{W-1}$.
    INPUT: A $log_2 D$-bit update address $A$.
    INPUT: Update operation: Add or Delete.
    OUTPUT: Updated $D \times W/k$, $2^k$ bit arrays, where $k$=6: $M_{i,j} = b_{i,j,0}, b_{i,j,1}, ..., b_{i,j,2^{k-1}}, i = 0, 1, ..., D-1, j = 0, 1, ..., W/k-1$
 1: for $i = 0, 1, ..., D-1$ do
 2:     for $j = 0, 1, ..., W/k-1$ do
 3:         for $l = 0, 1,‘..., 2^k-1$ do
 4:             if (Operation==Add) then
 5:                 if $(U[j*k:(j+1)*k]$ matches $l)$ then
 6:                     $M_{A,j,l} \leftarrow' 1'$
 7:                 else
 8:                     $M_{A,j,l} \leftarrow' 0'$
 9:                 end if
10:             else if (Operation==Delete) then
11:                 $M_{A,j,l} \leftarrow' 0'$
12:             end if
13:         end for
14:     end for
15: end for
```

The procedure for updating a TCAM word in DURE is presented in Algorithm 2. The update logic in DURE includes a MOD-64 counter, decoder logic, and N=$W/6$ number of six-bit comparison modules, as shown in Figure 17. The update address A of the log_2D-bits is decoded to give the write enable signals, which select the LUTRAM cells of the rows of the intended BM blocks for writing. This writing operation takes 65 cycles. The MOD-64 counter generates a new sequence of six bits for every cycle, which is matched with the six-bit chunks of the update word. If the match result is "TRUE," then a bit value of "1" is stored in the corresponding LUTRAM on the index specified by the six-bit sequence from the MOD-64 counter. The write address is applied using the common write address port of the LUTRAMs (the address port of LUTD) in each BM block, as explained above.

During the update process, the BM blocks are available for search operations, thereby allowing dynamic updates in DURE. The DURE TCAM implementation technique is highly efficient because all the LUTs from an occupied SLICEM are utilized and connected to the available three read/one write ports. Thus, the existing interconnections are exploited for the search and update processes with very little overhead for update logic in the proposed DURE architecture.

DURE modifies all the LUTRAMs from the BM blocks corresponding to the update word in parallel, thus taking 65 cycles. Moreover, DURE is able to support the longest prefix matching through a dynamic update of the priority encoder, when the priority of the update word is different from that of the old TCAM word.

B. FPGA Implementation and Results

DURE was implemented in a Xilinx Virtex-6 FPGA device (XC6VLX760). This FPGA has 33,120 SLICEMs each of which contains four six-input LUTs, constituting a total of 132,480 six-input LUTRAM cells. The LUTRAM cells and the carry chain structure available in the SLICEM resources are configured using DURE's BM blocks, as described in Figure 15. This BM block structure cannot be inferred using synthesis and implementation tools. We implemented the proposed BM blocks by instantiating the LUTRAM cells as RAM64M primitives and initialized them with the appropriate bits by defining their INIT attributes. The DOA, DOB, DOC, and DOD outputs of the LUTRAMs were connected as select inputs to the carry chains by defining the CARRY4 primitives. For example, the implementation of an 1×18 TCAM data (00000x11111100xxxx), shown in Figure 15, requires the instantiation of the LUT64M primitives with (**64'h**0000000000000003) for LUTA, (**64'h**8000000000000000) for LUTB, and (**64'h**000000000000ffff) for LUTC.

We employed the Xilinx ISE 14.7 synthesis and implementation tool to design and evaluate our proposed architecture. The Xilinx post-place-and-route results were used to report the maximum achievable clock rate and resource consumption of DURE. The power consumption for this implementation was estimated using the XPower Analyzer tool.

Table 9 lists the FPGA resource utilization for two different TCAM table sizes (case I: 512×36, case II: 1024×144) using the DURE technique. This implementation of DURE does not contain a priority encoder. The DURE implementation cases I and II utilize 4096 and 32768 LUTRAM cells, respectively, for storing the contents of the respective TCAM tables, with 624 and 1220 LUTs, respectively, for implementing the update logic. As can be observed, the resource utilization overhead for implementing the update logic using the proposed DURE is considerably small.

Table 9: FPGA resource utilization for proposed DURE

TCAM size	TCAM			Update logic	
(D×W)	LUTRAMs	SLICEM	FFs	LUTs	SLICEL
512×36	4096	1024	1165	624	210
1024×144	32768	8192	2690	1220	582

C. DURE Performance Evaluation and Comparison

The performance of DURE is evaluated for TCAM table implementations of different sizes to perform its scalability study. We compared the proposed DURE architecture with existing TCAM architectures for FPGAs in terms of the search latency, update rate, FPGA resource utilization, performance per area, and energy efficiency. We implemented two TCAM sizes of 18 kbit (case I: 512×36) and 144 kbit (case II: 1024×144) in DURE for the sake of appropriate comparisons with the related works.

1. Scalability of DURE

The scalability of DURE is evaluated through important metrics, such as the FPGA resource utilization, clock rate, power consumption, and update rate for TCAM table implementations of different sizes. To this end, TCAM tables of width $W = 36$, 72, 108, and 144 against depths of $D = 64$, 128, 256, and 512 are compared in Figures 18,19,20,21, and 22.

The utilization of FPGA resources such as LUTRAM and slice registers for different TCAM sizes implemented using DURE are shown in Figures 18 and

19 using a logarithmic scale. It is clear from these figures that the hardware consumption of DURE increases proportionally with an increase in the width or depth of the TCAM table size. DURE efficiently utilizes the LUTRAM resources for storing TCAM bits by implementing 18 TCAM bits in the four LUTRAM cells of a SLICEM. The LUTRAM consumed in the implementation of a TCAM table of depth D and width W using DURE can be calculated from equation 10.

$$No.\ of\ LUTRAMs\ used = D \times [(W/18) \times 4] \tag{10}$$

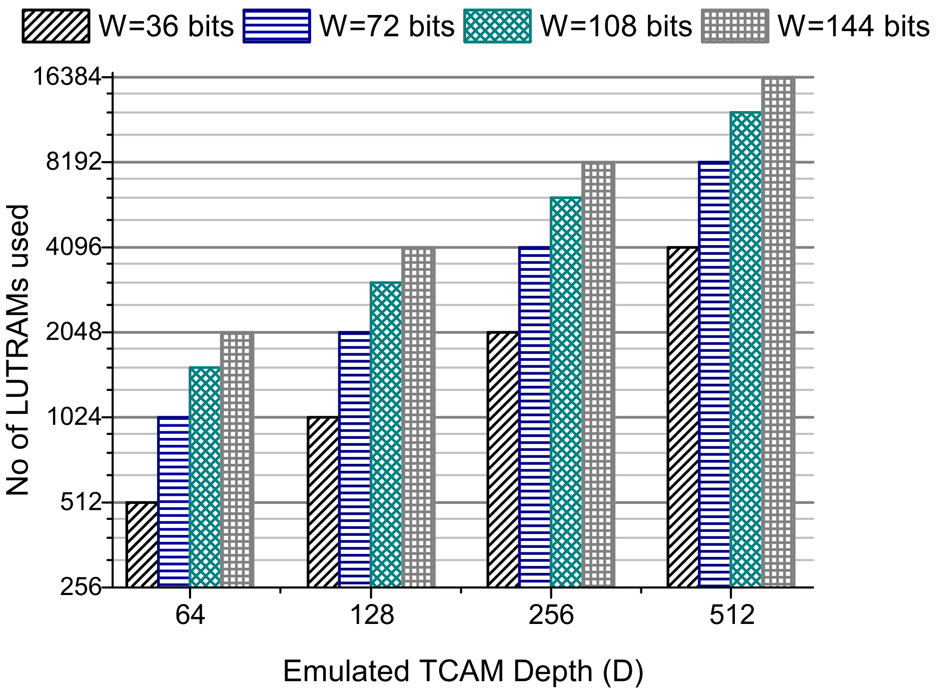


Figure 18: LUTRAM utilization trend for increasing TCAM depth (D) and width (W).

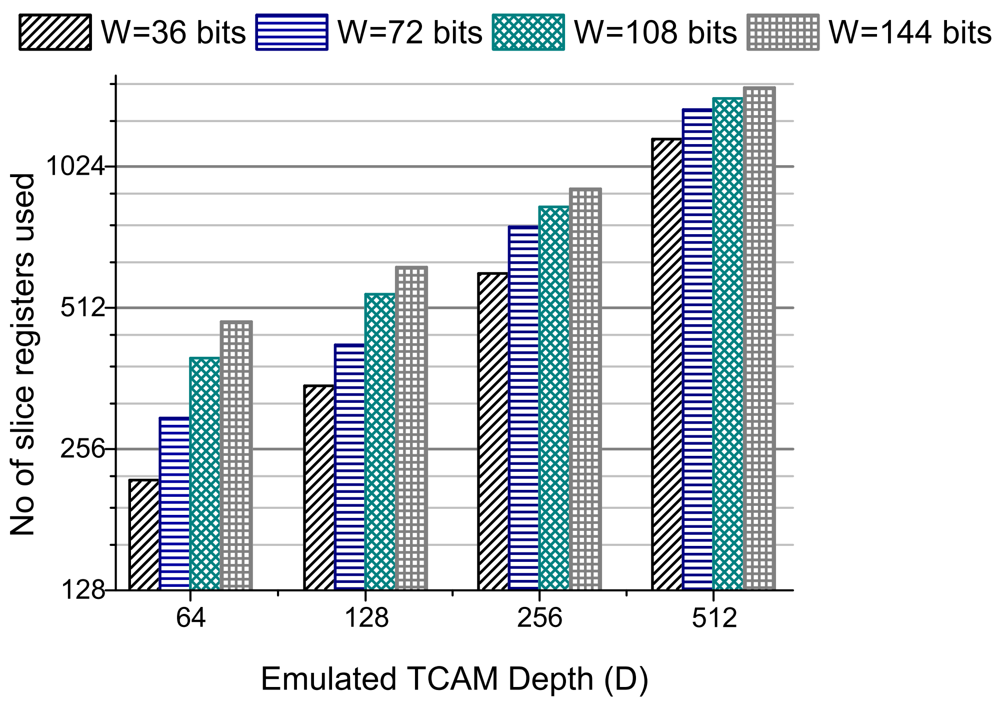


Figure 19: Slice registers usage trend for increasing TCAM depth (D) and width (W).

The clock rates achieved by DURE for different TCAM sizes are shown in Figure 20. DURE is able to achieve a linear decrease in clock frequency with a linear increase in the depth or width of the implemented TCAM. The critical path delay in DURE increases owing to an increase in the lengths of the longest wires connecting the LUTRAM cells. Figure 20 also shows that DURE can achieve considerably high operating speeds from 221 MHz up to 475 MHz for the given TCAM sizes.

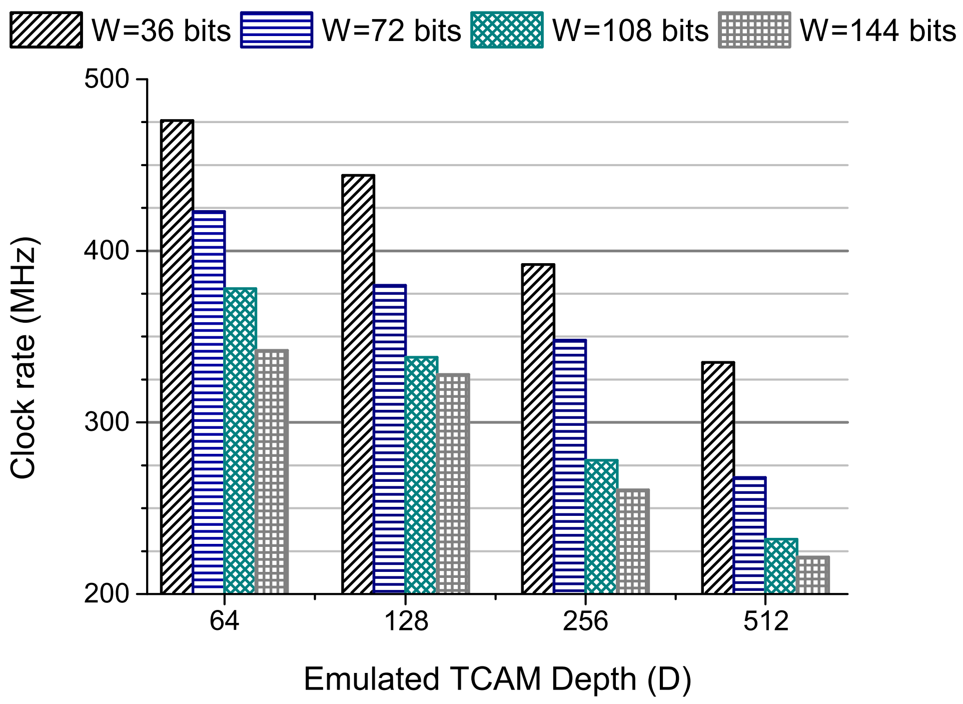


Figure 20: Clock rate trend for increasing TCAM depth (D) and width (W).

Figure 21 shows that the dynamic power consumption of DURE increases linearly with an increase in the TCAM table width or depth. In SRAM-based TCAM implementations, the primary source of power consumption is the SRAM memory, which stores the TCAM contents. As previously mentioned, DURE efficiently utilizes the LUTRAM resources to store the TCAM contents. Moreover, avoiding the use of LUTs and their associated programmable interconnects for AND cascading the potential match bits and instead using the carry chains for this purpose leads to minimal power consumption in DURE.

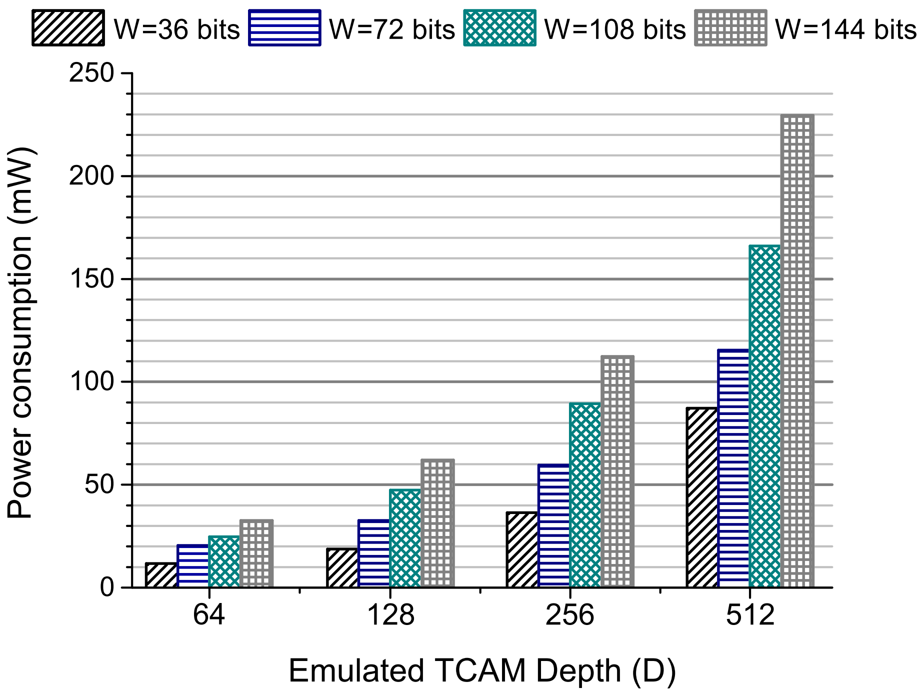


Figure 21: Power consumption trend for increasing TCAM depth (D) and width (W).

The update rate performance achieved in DURE is illustrated in Figure 22, and is calculated according to equation 11.

$$Update\ rate = \frac{Clock\ rate\ (\text{MHz})}{\#\ of\ cycles\ consumed\ per\ update} \tag{11}$$

The update operation takes 65 clock cycles. The update rate also decreases linearly with an increase in the width or depth of the implemented TCAM tables. DURE can achieve high update rates of 7.3∼3.4 million updates per second.

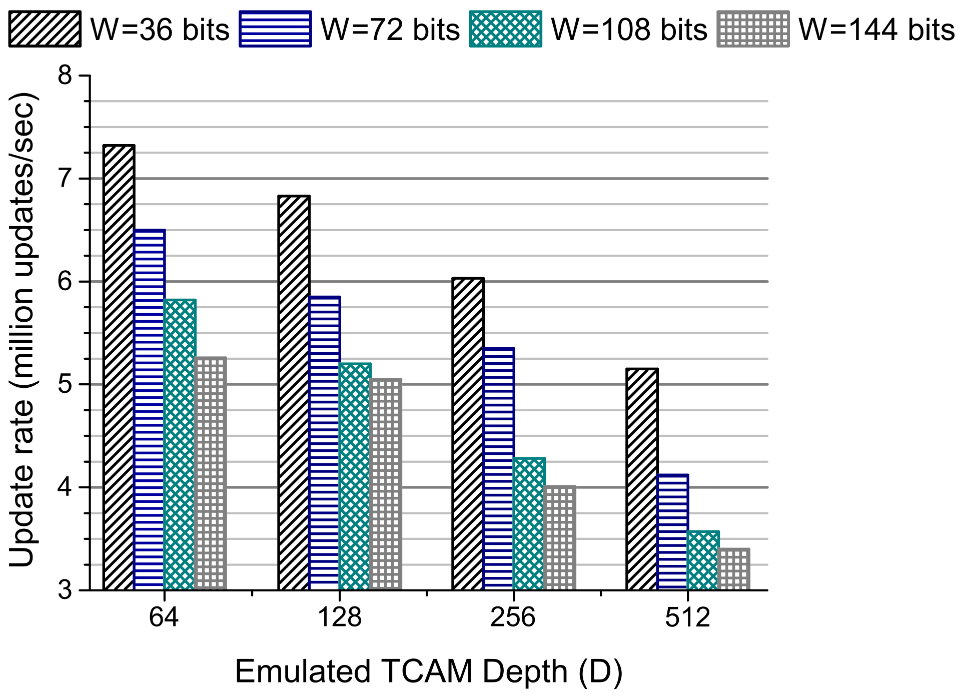


Figure 22: Update rate trend for increasing TCAM depth (D) and width (W).

2. Search Latency

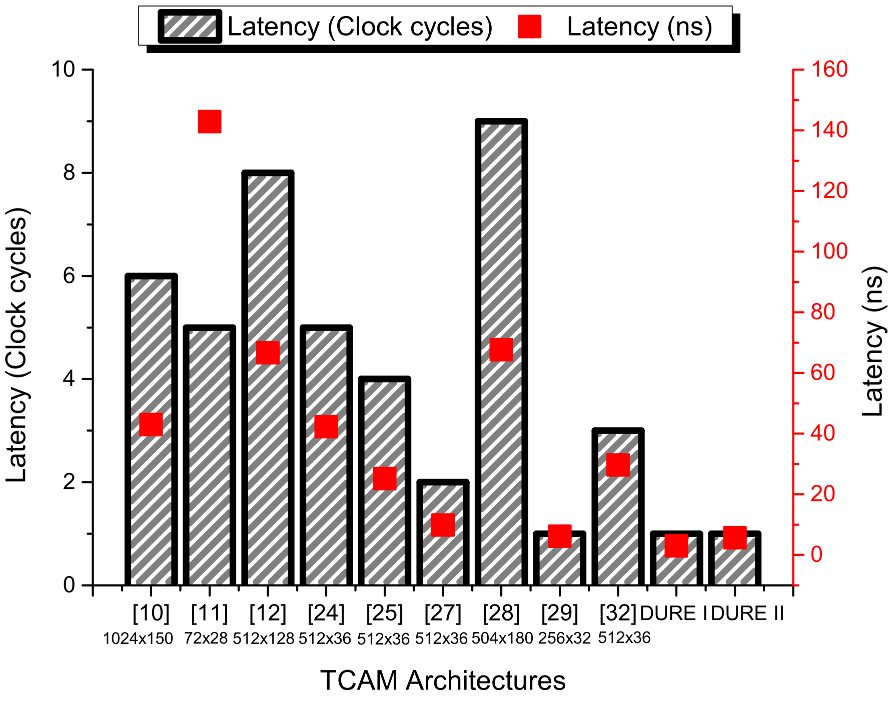


Figure 23: Search latency comparison.

The comparison of DURE with other FPGA-based TCAM architectures in terms of the search latency is shown in Figure 23. The FPGA-based TCAM architectures in [17], [25], [26], [20], [21], [16], [36], [37], and [38] achieved search latencies of 6, 5, 8, 5, 4, 2, 9, 1, and 3 clock cycles, respectively, and thus the match results against input words are processed in 43 ns, 143 ns, 67 ns, 42 ns, 25 ns, 10 ns, 68 ns, 6 ns, and 29.7 ns respectively. DURE is able to achieve a search latency of a single clock cycle. For the two cases implemented in DURE, the delay time achieved in case I is 2.987 ns, and that in case II is 5.723 ns. Thus, the time required by the search operation for DURE is the lowest among the existing FPGA-based TCAM approaches. Thus, the single cycle low access time search makes the proposed DURE method a promising architecture for implementing applications for which low response times are desired, such as TLB caches.

3. Update Rate

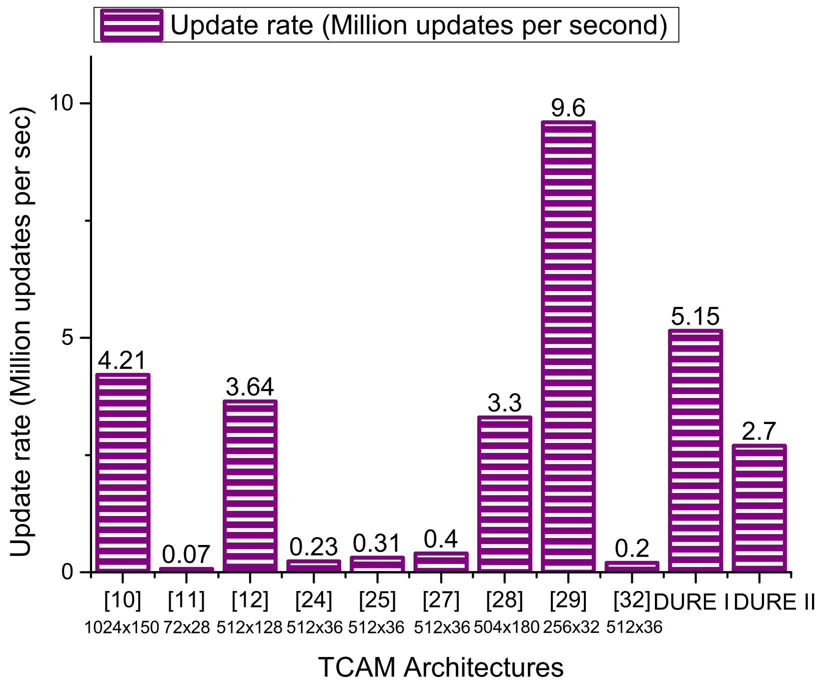


Figure 24: Update rate comparison.

DURE is compared in terms of the update rate with the related TCAM architectures in Figure 24. DURE cases I and II, implementing 512 × 36 and 1024 × 144 sizes of TCAM tables respectively, achieve clock rates of 335 MHz and 175 MHz. The update operation in DURE is performed in 65 clock cycles.

Thus, the update rates achieved in DURE cases I and case II are 5.15 million updates per second and 2.7 million updates per second respectively.

The BRAM-based TCAM architectures presented in previous works [25], [20], [21], [16], and [38] achieve clock rates of 35, 118, 159, 202, and 109 MHz, respectively, while the high update latency of 513 cycles in BRAM-based TCAM architectures, results in lower update rates of 0.07, 0.23, 0.31, 0.4, and 0.2 million updates per second, respectively. The TCAM architectures [17], [26], and [36], implemented in distributed RAM with an update latency of 33 cycles achieve update rates of 4.21, 3.64, and 3.3 million updates per second, respectively. The 512×32 size TCAM table implemented in [37] with an update latency of 17 cycles, achieves an update rate of 9.6 million updates per second. It should be noted that the non-blocking update operation implemented in DURE does not affect the search rate of the system since the search operations can be performed simultaneous to the update operations in DURE. However, the blocking update operations in existing FPGA-based TCAMs lower the effective search rates as search operations remain suspended during the update operations.

4. FPGA Resource Utilization

Table 10 compares the FPGA resource utilization in terms of LUTRAMS, LUTLs, slice registers, and BRAMs along with the corresponding percentage utilization for the various TCAM architectures. Column 2 lists the FPGA device used for implementing the TCAM table sizes listed in column 3. The LUTRAM utilization of architectures implemented in distributed RAM is listed in Column 4 while the BRAM utilization of BRAM-based architectures is given in Column 7. The maximum TCAM bit storage capacity of the TCAM architectures [17, 26, 36] implemented in distributed RAM is higher because it configures the LUTRAMs of a SLICEM as a 32×6 simple dual port RAM, thereby implementing 30 TCAM bits per SLICEM compared with 18 TCAM bits per SLICEM in the case of DURE. However, the given architectures achieve this higher TCAM bit storage capacity at the cost of higher logic consumption in bitwise ANDing of a large number of wide bit vectors as shown in Column 5. On the other hand, the proposed DURE uses the carry chain structures available in FPGA slices for AND-cascading the match bits from the LUTRAMs thereby saving the extra logic and associated routing resources consumed in bitwise ANDing.

Table 10: FPGA resource utilization for implementing various TCAM architectures

Architecture	FPGA	TCAM size $(D \times W)$	LUTRAMs usage (%)	LUTLs usage (%)	Slice registers usage (%)	BRAMs (36 kbit) usage (%)	FPGA Slices usage(%)
Jiang[17]	XC7V2000T	1024×150	20480 (5.94%)	61624 (7%)	37556 (1.54%)	0	20526 (6.72%)
REST[25]	XC7K70T	72×28	8 (0.06%)	130 (0.47%)	390 (0.48%)	1 (0.74%)	77 (0.75%)
Xilinx 2017[26]	XC7V2000T	512×128	8875 (2.58%)	27559 (3.14%)	35068 (1.44%)	3 (0.23%)	12011 (3.93%)
HP-TCAM[20]	XC6VLX760	512×36	0	6546 (1.92%)	2670 (0.28%)	56 (7.78%)	1637 (1.38%)
Z-TCAM[21]	XC6VLX760	512×36	0	4462 (1.30%)	2178 (0.23%)	40 (5.56%)	1116 (0.94%)
UE-TCAM[16]	XC6VLX760	512×36	0	3652 (1.07%)	1758 (0.19%)	32 (4.44%)	913 (0.77%)
Qian[36]	XC6VLX75T	504×180	15552 (93%)	24715 (83%)	35342 (38%)	0	10067 (86%)
Locke[37]	XC5VLX220	256×32	4096 (11.23%)	1527 (1.50%)	341 (0.25%)	0	1406 (4.07%)
Syed[38]	XC6VLX760	512×36	–	3013 (0.88%)	552 (0.06%)	32 (4.44%)	754 (0.64%)
DURE case I	XC6VLX760	512×36	4096 (3%)	1605 (0.47%)	1174 (0.13%)	0	1668 (1.40%)
DURE case II	XC6VLX760	1024×144	32768 (24%)	3039 (0.89%)	2700 (0.28%)	0	9654 (8.14%)

5. Performance per Area

The performance of the TCAM architectures is metricised in terms of processing throughput Mbit/s and area which is measured as the number of FPGA slices required per word for the implemented TCAM table. Since the FPGA-based TCAM architectures under consideration were implemented in different embedded memory resources such as BRAM or distributed RAM, in order to fairly compare the given TCAM architectures, we measured their normalized area based on the observation that 384 bits of BRAM bits correspond to one 6-input LUT, while 1536 BRAM bits are similar to one slice comprising four 6-input LUTs on a Virtex-6 FPGA. This is described using equation 12 which has been adapted and modified from [49, 61, 62].

$$\begin{aligned} Normalized\ Area(slices) &= \#\ of\ FPGA\ slices \\ &\quad + [\#\ of\ BRAMs\ (36\ \text{kbit})\ \times\ 24] \end{aligned} \tag{12}$$

Table 11 presents the comparison of DURE in terms of performance per area metric which is calculated according to equation 13 modified from [49, 61, 62].

$$\frac{Performance}{Area} = \frac{Throughput\ (\text{Mbit/s})}{\frac{Normalized\ Area\ (Slices)}{Number\ of\ TCAM\ words}} \tag{13}$$

The clock rates of the TCAM designs implemented in different CMOS technologies are normalized using equation 14 which has been modified from [50, 58].

$$F^{*} = F \times \left[\frac{Technology\ (\text{nm})}{40\ (\text{nm})}\right] \times \left[\frac{1.0}{VDD}\right] \tag{14}$$

where F represents the clock rate, and F^{*} denotes the normalized clock rate for 40 nm CMOS technology (Virtex-6) and a supply voltage of 1.0 V.

Case I, where a 512×36 size TCAM table is implemented in 1668 FPGA slices achieves a throughput of 12.06 Gbit/s. Therefore, the performance per area achieved in case I is 3.7 Mbit/s/*Slice*. Thus, DURE case I is able to achieve a 67% higher performance per area than [16], which was already the highest among the related TCAM architectures.

Case II implements a larger TCAM table of size 144 kbit (1024×144) in 9654 FPGA slices and achieves a throughput of 25.2 Gbit/s. Therefore, it achieves a performance per area of 2.67 Mbit/s/*slice*. The similar similar sized TCAM design in [17] achieve a performance per area of 1.04 Mbit/s/*slice*. Table 11 illustrates that DURE case II is able to achieve about 2.5 times higher performance per area than [17], which was already the highest among other large sized TCAM architectures [17, 26, 36]. Thus, the whole DURE provides a high-speed and resource- efficient TCAM architecture. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in

Table 11: Performance per area comparison with existing FPGA-based TCAM architectures

Architecture	TCAM size ($D \times W$)	FPGA Device (Technology)	BRAMs (36 kbit)	Slices usage	Normalized Area (slices)	Speed (MHz)	Normalized speed (MHz)	Throughput (Gbit/s)	P/A[(a)]
Jiang[17]	1024×150	Virtex-7 (28 nm)	0	20526	–	199	139	20.9	1.04
REST[25]	72×28	Kintex-7 (28 nm)	1	77	101	50	35	0.98	0.7
Xilinx 2017[26]	512×128	Virtex-7 (28 nm)	3	12011	12083	171	120	15.36	0.65
HP-TCAM[20]	512×36	Virtex-6 (40 nm)	56	1637	2981	118	–	4.25	0.73
Z-TCAM[21]	512×36	Virtex-6 (40 nm)	40	1116	2076	159	–	5.72	1.41
UE-TCAM[16]	512×36	Virtex-6 (40 nm)	32	913	1681	202	–	7.26	2.21
Qian[36]	504×180	Virtex-6 (40 nm)	0	10067	–	109	–	19.26	0.98
Locke[37]	256×32	Virtex-5 (65 nm)	0	1406	–	100	163	5.2	1.13
Syed[38]	512×36	Virtex-6 (40 nm)	32	754	1522	101	–	3.64	1.22
DURE case I	512×36	Virtex-6 (40 nm)	0	1668	–	335	–	12.06	3.7
DURE case II	1024×144	Virtex-6 (40 nm)	0	9654	–	175	–	25.2	2.67

[(a)] **P/A**: Performance per Area ((Mbit/s)/*Slice*) sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

6. Energy-Delay Product

Table 12 compares the performances of the two cases implemented with DURE with other FPGA-based TCAM solutions in terms of the energy-delay product (EDP). The power consumption results of previous methods for implementing the TCAM architectures in different CMOS technologies are normalized using equation 15, which has been taken from [58] and modified.

$$P^* = P \times \left[\frac{40\ (\mathrm{nm})}{Technology\ (\mathrm{nm})}\right] \times \left[\frac{1.0}{VDD}\right]^2 \tag{15}$$

where P represents the dynamic power consumption, while P^* represents the normalized dynamic power consumption for 40 nm CMOS technology (Virtex-6) and a supply voltage of 1.0 V.

Case I consumes a dynamic power of 52 mW. Thus, the energy consumption in DURE is 28 fJ/bit/search and the EDP is 84 ns.fJ/bit/search. The EDP achieved in case I is 2.5 times lower than that of UE-TCAM [16], which is the lowest among the other FPGA-based TCAM architectures. Case II, which implements a larger TCAM, consumes a dynamic power of 483 mW and has a delay time of 5.72 ns. Thus, its EDP is 187 ns.fJ/bit/search, which is about seven times lower than that of [17], which is 150 kbit size. Thus, DURE is also a highly energy-delay product efficient TCAM architecture. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac

Table 12: Energy-delay product (EDP) comparison for various FPGA-based TCAM solutions

Architecture	TCAM size (D × W)	FPGA (Technology)	Delay (ns)	Search cycles	Power (W)	Normalized delay T* (ns)	Energy (fJ/bit/search)	EDP (ns.fJ/ bit/search)
Jiang[17]	1024×144	Virtex-7 (28 nm)	5.03	6	1.9(2.7)	7.18	125.8	1290
REST[25]	72×28	Kintex-7(28 nm)	20	5	0.11(0.16)	28.57	559.7	22817
Xilinx 2017[26]	512×128	Virtex-7 (28 nm)	5.85	8	1.01(1.442)	8.34	220	1834
HP-TCAM[20]	512×36	Virtex-6 (40 nm)	8.47	5	0.19	–	102.2	865
Z-TCAM[21]	512×36	Virtex-6 (40 nm)	6.29	4	0.11	–	58.9	371
UE-TCAM[16]	512×36	Virtex-6 (40 nm)	4.95	2	0.08	–	42.3	210
Qian[36]	504×180	Virtex-6 (40 nm)	7.52	9	2.5	–	280.9	2111
Locke[37]	256×32	Virtex-5 (65 nm)	10	1	0.09(0.055)	6.15	68	416
Syed[38]	512×36	Virtex-6 (40 nm)	9.9	3	0.043	–	23.3	231
DURE case I	512×36	Virtex-6 (40 nm)	2.99	1	0.05	–	28	84
DURE case II	1024×144	Virtex-6 (40 nm)	5.72	1	0.48	–	32.8	187

habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

V. CONCLUSIONS & FUTURE WORK

In this thesis, we have presented three different high-performance TCAM architectures for FPGAs.

- A memory-efficient TCAM design, based on multipumping-enabled multiported SRAM, that operates the SRAM blocks in the design at a frequency that is multiple times higher than that of the overall system frequency. This allows reading from the sub-blocks of the SRAM to take place within one system cycle. The FPGA implementation results show that the performance per memory of the proposed design is up to 2.85 times higher than the existing SRAM-based TCAM solutions on FPGA.

- A pre-classifier-based architecture for an energy-efficient SRAM-based TCAM, which selectively activates at most one row of SRAM blocks for lookup rather than activating the entire SRAM memory as in the existing architectures. The experimental results showed that the proposed design exhibits at least three times lower power consumption per performance than the existing FPGA realizations of TCAM.

- A technique for implementing a dynamically reconfigurable TCAM in SRAM-based FPGAs with a higher energy consumption and resource utilization efficiency. The proposed TCAM design methodology employs an FPGA's distributed RAM resources and configures the LUTRAMs of a SLICEM as quad-port RAM in its implementation. It dynamically reconfigures only the LUTRAMs associated with the word being updated, and at the same time allows lookup operations to be performed. DURE achieved a better performance than other methods in terms of the lookup and update operations when tested on a Virtex-6 FPGA device. It achieved an energy-delay product efficiency and performance per area that were at least 2.5 times and 67% better, than those of the other FPGA-based TCAM solutions.

In the future, we have the plan to implement soft-error protection for SRAM-based TCAMs. Additionally, we have also the plan to use our proposed high-performance CAM architectures for implementing efficient translation look-aside buffer (TLB) cache for soft-core processors.

REFERENCES

[1] K. Etzel, “Answering ipv6 lookup challenges,” *Technical Article, Cypress Semiconductor Corporation*, 2004.

[2] D. E. Taylor, “Survey and taxonomy of packet classification techniques,” *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 238–275, 2005.

[3] J.-Y. Huang and P.-C. Wang, “TCAM-Based IP Address Lookup Using Longest Suffix Split,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 976–989, 2018.

[4] G. Brebner and W. Jiang, “High-speed packet processing using reconfigurable computing,” *IEEE Micro*, vol. 34, no. 1, pp. 8–18, 2014.

[5] J. R. Haigh and L. T. Clark, “High performance set associative translation lookaside buffers for low power microprocessors,” *Integration, the VLSI Journal*, vol. 41, no. 4, pp. 509–523, 2008.

[6] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, “Emerging trends in design and applications of memory-based computing and content-addressable memories,” *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1311–1330, 2015.

[7] I. Arsovski, A. Patil, R. M. Houle, M. T. Fragano, R. Rodriguez, R. Kim, and V. Butler, “1.4 Gsearch/s 2-Mb/mm 2 TCAM Using Two-Phase-Pre-Charge ML Sensing and Power-Grid Pre-Conditioning to Reduce Ldi/dt Power-Supply Noise by 50%,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 155–163, 2018.

[8] I. Arsovski, R. M. Houle, M. T. Fragano, A. Patil, and V. D. Butler, “Ternary content addressable memory (TCAM) for multi bit miss detect circuit,” Mar. 13 2018, uS Patent 9,916,896.

[9] L.-Y. Huang, M.-F. Chang, C.-H. Chuang, C.-C. Kuo, C.-F. Chen, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, K.-L. Su *et al.*, “ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing,” in *VLSI Circuits Digest of Technical Papers, 2014 Symposium on*. IEEE, 2014, pp. 1–2.

[10] X.-T. Nguyen, T.-T. Hoang, H.-T. Nguyen, K. Inoue, and C.-K. Pham, “An FPGA-Based Hardware Accelerator for Energy-Efficient Bitmap Index Creation,” *IEEE Access*, vol. 6, pp. 16 046–16 059, 2018.

[11] O. Mujahid, Z. Ullah, H. Mahmood, and A. Hafeez, “Fast pattern recognition through an lbp driven cam on fpga,” *IEEE Access*, vol. 6, pp. 39 525–39 531, 2018.

[12] K.-J. Lin and C.-W. Wu, “A low-power cam design for lz data compression,” *IEEE Transactions on Computers*, vol. 49, no. 10, pp. 1139–1145, 2000.

[13] M. Imani, A. Rahimi, and T. S. Rosing, “Resistive configurable associative memory for approximate computing,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 1327–1332.

[14] H.-J. Tsai, K.-H. Yang, Y.-C. Peng, C.-C. Lin, Y.-H. Tsao, M.-F. Chang, and T.-F. Chen, “Energy-efficient TCAM search engine design using priority-

decision in memory technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 962–973, 2017.

[15] M.-F. Chang, C.-C. Lin, A. Lee, Y.-N. Chiang, C.-C. Kuo, G.-H. Yang, H.-J. Tsai, T.-F. Chen, and S.-S. Sheu, "A 3T1R Nonvolatile TCAM Using MLC ReRAM for Frequent-Off Instant-On Filters in IoT and Big-Data Processing," *IEEE Journal of Solid-State Circuits*, 2017.

[16] Z. Ullah, M. K. Jaiswal, R. C. C. Cheung, and H. K. H. So, "UE-TCAM: An ultra efficient SRAM-based TCAM," in *TENCON 2015 - 2015 IEEE Region 10 Conference*, Nov 2015, pp. 1–6.

[17] W. Jiang, "Scalable ternary content addressable memory implementation using FPGAs," in *Proc. of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, 2013, pp. 71–82.

[18] Xilinx, "Virtex-6 FPGA memory resources user guide," [Online]. Available: http://www.xilinx.com, 2014.

[19] "Xilinx WP335 Creative Uses of Block RAM, author=Alfke, Peter, journal=White Paper: Virtex and Spartan FPGA Families, Xilinx, year=2008."

[20] Z. Ullah, K. Ilgon, and S. Baeg, "Hybrid Partitioned SRAM-Based Ternary Content Addressable Memory," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 12, pp. 2969–2979, 2012.

[21] Z. Ullah, M. Jaiswal, and R. Cheung, "Z-TCAM: An SRAM-based architecture for TCAM," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 402–406, Feb 2015.

[22] Z. Ullah, M. K. Jaiswal, and R. C. Cheung, “E-TCAM: An efficient SRAM-based architecture for TCAM,” *Circuits, Systems, and Signal Processing*, vol. 33, no. 10, pp. 3123–3144, 2014.

[23] W. Jiang, “Scalable ternary content addressable memory implementation using FPGAs,” in *Proc. of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, 2013, pp. 71–82.

[24] Z. Qian and M. Margala, “Low power RAM-based hierarchical CAM on FPGA,” in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*. IEEE, 2014, pp. 1–4.

[25] A. Ahmed, K. Park, and S. Baeg, “Resource-Efficient SRAM-Based Ternary Content Addressable Memory,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1583–1587, 2017.

[26] Xilinx, “Ternary Content Addressable Memory (TCAM) Search IP for SDNet v1.0; Xilinx Product Guide PG190,” November, 2017.

[27] Z. Wang, H. Che, M. Kumar, and S. K. Das, “CoPTUA: Consistent policy table update algorithm for TCAM without locking,” *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1602–1614, 2004.

[28] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie, “Partial Order Theory for Fast TCAM Updates,” *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 1, pp. 217–230, 2018.

[29] S. Mittal, “A survey of techniques for architecting TLBs,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 10, p. e4061, 2017.

[30] S. Tamimi, Z. Ebrahimi, B. Khaleghi, and H. Asadi, "An Efficient SRAM-based Reconfigurable Architecture for Embedded Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[31] K. Aasaraai and A. Moshovos, "NCOR: An FPGA-friendly nonblocking data cache for soft processors with runahead execution," *International Journal of Reconfigurable Computing*, vol. 2012, p. 6, 2012.

[32] M. Biglari, K. M. Barijough, M. Goudarzi, and B. Pourmohseni, "A fine-grained configurable cache architecture for soft processors," in *Computer Architecture and Digital Systems (CADS), 2015 18th CSI International Symposium on*. IEEE, 2015, pp. 1–6.

[33] H. Park, H. So, and H. Lee, "Application specific cache design using STT-RAM based block-RAM for FPGA-based soft processors," *IEICE Electronics Express*, vol. 15, no. 10, pp. 20 180 330–20 180 330, 2018.

[34] T. Adegbija, A. Rogacs, C. Patel, and A. Gordon-Ross, "Microprocessor optimizations for the Internet of Things: a survey," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 7–20, 2018.

[35] A. Gordon-Ross, F. Vahid, and N. D. Dutt, "Fast configurable-cache tuning with a unified second-level cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 80–91, 2009.

[36] Z. Qian and M. Margala, “Low power RAM-based hierarchical CAM on FPGA,” in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*. IEEE, 2014, pp. 1–4.

[37] K. Locke, “Xilinx Application Note: XAPP1151 - Parameterizable Content-Addressable Memory,” http://www.xilinx.com, 2011.

[38] F. Syed, Z. Ullah, and M. K. Jaiswal, “Fast Content Updating Algorithm for an SRAM based TCAM on FPGA,” *IEEE Embedded Systems Letters*, 2017.

[39] P. Maidee, “Multiplexer-based ternary content addressable memory,” May 16 2017, uS Patent 9,653,165.

[40] M. Somasundaram, “Circuits to generate a sequential index for an input number in a pre-defined list of numbers,” Patent, Dec. 26, 2006, US Patent 7,155,563.

[41] S. Cho, J. Martin, R. Xu, M. Hammoud, and R. Melhem, “CA-RAM: A high-performance memory substrate for search-intensive applications,” in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, 2007, pp. 230–241.

[42] C. E. LaForest and J. G. Steffan, “Efficient multi-ported memories for FPGAs,” in *Proceedings of the 18th annual ACM/SIGDA int. symposium on Field programmable gate arrays*. ACM, 2010, pp. 41–50.

[43] H. E. Yantir, S. Bayar, and A. Yurdakul, “Efficient implementations of multi-pumped multi-port register files in FPGAs,” in *Digital System Design (DSD), 2013 Euromicro Conf. on*. IEEE, 2013, pp. 185–192.

[44] C. E. LaForest, “Multi-Ported Memories for FPGAs, Accessed on Nov. 13, 2017,” [Online]. Available: http://fpgacpu.ca/multiport/index.html.

[45] N. Manjikian, “Design issues for prototype implementation of a pipelined superscalar processor in programmable logic,” in *Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on*, vol. 1. IEEE, 2003, pp. 155–158.

[46] H. Yokota, “Multiport memory system,” May 29 1990, uS Patent 4,930,066.

[47] B. A. Chappell, T. I. Chappell, M. K. Ebcioglu, and S. E. Schuster, “Virtual multi-port ram employing multiple accesses during single machine cycle,” Jul. 30 1996, uS Patent 5,542,067.

[48] A. Abdelhadi and G. G. Lemieux, “Modular Switched Multiported SRAM-Based Memories,” p. 22, 2016.

[49] H. Nakahara, T. Sasao, H. Iwamoto, and M. Matsuura, “LUT Cascades Based on Edge-Valued Multi-Valued Decision Diagrams: Application to Packet Classification,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 1, pp. 73–86, 2016.

[50] P.-T. Huang and W. Hwang, “A 65 nm 0.165 fJ/Bit/Search 256 x 144 TCAM Macro Design for IPv6 Lookup Tables,” *IEEE Journal of Solid-State Circuits*, vol. 46, no. 2, pp. 507–519, 2011.

[51] D. Shah and P. Gupta, “Fast incremental updates on ternary-cams for routing lookups and packet classification,” in *Proceedings of Hot Interconnects*, 2000.

[52] ——, “Fast updating algorithms for tcam,” *IEEE Micro*, vol. 21, no. 1, pp. 36–47, 2001.

[53] Xilinx, “Xilinx Xpower Analyzer,” [Online]. Available: http://www.xilinx.com.

[54] Z. Ullah, M. K. Jaiswal, R. C. C. Cheung, and H. K. H. So, “UE-TCAM: An ultra efficient SRAM-based TCAM,” in *TENCON 2015 - 2015 IEEE Region 10 Conference*, Nov 2015, pp. 1–6.

[55] Z. Ullah, K. Ilgon, and S. Baeg, “Hybrid partitioned SRAM-based ternary content addressable memory,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 12, pp. 2969–2979, 2012.

[56] Z. Ullah, M. K. Jaiswal, and R. C. Cheung, “Z-TCAM: an SRAM-based architecture for TCAM,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 402–406, 2015.

[57] K. Locke, “Parameterizable content-addressable memory,” *Xilinx Application Note XAPP1151*, 2011.

[58] O.-C. Chen and R.-B. Sheen, “A power-efficient wide-range phase-locked loop,” *IEEE Journal of Solid-State Circuits*, vol. 37, no. 1, pp. 51–62, 2002.

[59] Xilinx, “Virtex-6 FPGA Configurable Logic Block User Guide UG364 (v1.2),” [Online]. Available: http://www.xilinx.com, 2012.

[60] H. Parandeh-Afshar, G. Zgheib, P. Brisk, and P. Ienne, “Routing wire optimization through generic synthesis on FPGA carry chains,” in *Proceedings of the 20th International Workshop on Logic and Synthesis, San Diego, Calif*, 2011.

[61] H. Nakahara, T. Sasao, and M. Matsuura, "An update method for a cam emulator using an lut cascade based on an evmdd (k)," in *Multiple-Valued Logic (ISMVL), 2014 IEEE 44th International Symposium on*. IEEE, 2014, pp. 1–6.

[62] I. Sourdis, D. N. Pnevmatikatos, and S. Vassiliadis, "Scalable multigigabit pattern matching for packet inspection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 156–166, 2008.

ACKNOWLEDGEMENTS

Praise be to Allah, the Cherisher and Sustainer of the Worlds.

I would like to express my sincere gratitude to my advisor Professor Jeong-A Lee for exceptional guidance and support during my doctoral study and research. Besides my advisor, my sincere thanks also go to Professor Zahid Ullah for sharing his expertise, sincere and valuable guidance and encouragement throughout my research period, which steered me in the right direction. I am thankful to my thesis evaluation committee members Prof. Sangman Moh, Prof. Choi Goang Seog, Prof. Seokjoo Shin and Prof. Cheol Hong Kim for their insightful comments and critical review.

I would like to thanks department of computer engineering, college of engineering, Chosun University for the research facilities and funding. I must pay my esteem regards to National Research Foundation of Korea for its financial support during the period of my doctoral studies.

I am thankful to Mr Umar Afzaal for providing me with personal and professional help. I am grateful to my lab members: Sami Hassan, Tooba Arifeen, and Sana Ullah with whom I have had the pleasure to work during this project. I am extremely thankful to Mr Ijaz Ahmed, Without your precious support and friendship, my stay at Korea won't be this smooth.

I must express my very profound gratitude to my beloved daughter and my wife Sehrish Jalal for providing me unconditional love with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

I dedicate this thesis with love and eternal appreciation to my great parents, Nasreen Bibi and my Late father Mahmood Khan, who never stop giving of

themselves in countless ways. I dedicate this thesis to my beloved brothers and sisters who always stand by with me when things look bleak.
And finally, to all my teacher at Pakistan who enabled me to stand in place.
Thanks for all your encouragement!

Inayat Ullah

February, 2019
Chosun University