Architecture of Block-RAM-Based Massively Parallel Memory Structures:

Multi-Ported Memories and Content-Addressable Memories

by

Ameer M. S. Abdelhadi

B.Sc. Computer Engineering, Technion, 2007

M.Sc. Electrical Engineering, Technion, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

September 2016

© Ameer M. S. Abdelhadi, 2016

Abstract

Since they were first introduced three decades ago, Field-Programmable Gate Arrays (FPGAs) have evolved from being merely used as glue-logic to implementing entire compute accelerators. These massively parallel systems demand highly parallel memory structures to keep pace with their concurrent nature since memories are usually the bottleneck of computation performance. However, the vast majority of FPGA devices provide dual-ported SRAM blocks only. In this dissertation, we propose new ways to build area-efficient, high-performance SRAM-based parallel memory structures in FPGAs, specifically Multi-Ported Random Access Memory and Content-Addressable Memory (CAM).

While parallel computation demands more RAM ports, leading Multi-Ported Random Access Memory techniques in FPGAs have relatively large overhead in resource usage. As a result, we have produced new design techniques that are near-optimal in resource overhead and have several practical advantages. The suggested method reduces RAM usage by over 44% and improves clock speed by over 76% compared to the best of previous approaches. Furthermore, we propose a novel switched-ports technique that allows further area reduction if some RAM ports are not simultaneously active. A memory compiler is proposed to generalize the previous approach and allow generating Multi-Switched-Ports Random Access Memory.

Content-Addressable Memories (CAMs), the hardware implementation of associative arrays, are capable of searching the entire memory space for a specific value within a single clock cycle. CAMs are massively parallel search engines accessing all memory content to compare with the searched pattern simultaneously. CAMs are used in a variety of scientific fields requiring high-speed associative searches. Despite their importance, FPGAs lack an area-efficient CAM implementation. We propose a series of scalable, area-efficient, and high-performance Binary Content-Addressable Memories (BCAMs) based on hierarchical search and data compression methods. Compared to current RAM-based BCAM architectures, our BCAMs require a maximum of 18% the RAM storage while enhancing clock speed by 45% on average, hence exhibiting a superior single-cycle search rate.

As a result, we can build faster and more cost-effective accelerators to solve some of the most important computational problems.

Preface

The major contributions of this dissertation have also been published in journal papers and conference proceedings [1–5] as outlined below.

For all of these publications, I proposed the ideas, carried out the research, performed all of the design methodology and implementation, conducted the experiments, data generation and the analysis of the results. Furthermore, I prepared the manuscripts for these publications under the supervision of Prof. Lemieux, who also provided editorial support for all of these manuscripts and provided advice on the research design methodology.

- Multi-ported memories
  - **Modular Multi-Ported SRAM-based Memories** [1]

    Published in the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2014). Parts of this publication appear in Chapter 3.
  - **Modular Switched Multi-ported SRAM-based Memories** [2]

    Accepted for publication in ACM Transactions on Reconfigurable Technology and Systems (TRETS) Special Issue on Reconfigurable Components with Source Code. Parts of this publication appear in Chapter 4.

  - **A Multi-Ported Memory Compiler Utilizing True Dual-port BRAMs** [3]

    To be published in the 2016 IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2016). Parts of this publication appear in Chapter 4.

- Content-addressable memories

  - **Deep and Narrow Binary Content-Addressable Memories using FPGA-based BRAMs** [4]

    Published in the 2014 International Conference on Field-Programmable Technology (ICFPT 2014). Parts of this publication appear in Chapter 5.

  - **Modular SRAM-based Binary Content-Addressable Memories** [5]

    Published in the 2015 IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2015). Parts of this publication appear in Chapter 6.

[1] A. M. S. Abdelhadi and G. G. F. Lemieux. Modular Multi-Ported SRAM-Based Memories. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 35–44, February 2014

[2] Ameer M.S. Abdelhadi and Guy G.F. Lemieux. Modular Switched Multi-Ported SRAM-Based Memories. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 9(3):22:1–22:26, July 2016

[3] A. M. S. Abdelhadi and G. G. F. Lemieux. A Multi-Ported Memory Compiler Utilizing True Dual-Port BRAMs. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 200–207, May 2016

[4] A. M. S. Abdelhadi and G. G. F. Lemieux. Deep and Narrow Binary Content-Addressable Memories Using FPGA-Based BRAMs. In *International Conference on Field-Programmable Technology (FPT)*, pages 318–321, December 2014

[5] A. M. S. Abdelhadi and G. G. F. Lemieux. Modular SRAM-Based Binary Content-Addressable Memories. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 207–214, May 2015

Table of Contents

List of Tables

List of Abbreviations

The following lists all abbreviations that have been used in the dissertation.

MSPRAM	Multi-Switched-Ports Random Access Memory
MPRAM	Multi-Ported Random Access Memory
LVT	Live-Value Table
I-LVT	Invalidation-Based Live-Value Table
ALM	Adaptive Logic Module
ESB	Embedded System Block
MFB	Multi-Function Block
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BRAM	Block-RAM

CAD	Computer-Aided Design
CAM	Content-Addressable Memory
BCAM	Binary Content-Addressable Memory
TCAM	Ternary Content-Addressable Memory
RCAM	Reconfiguration Memory Based Content-Addressable Memory
PE	Priority Encoder
PE	Processing Element
LPME	Longest-Prefix Match Encoder
LPM	Longest-Prefix Match
CGRA	Coarse-Grained Reconfigurable Array
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DFG	Data Flow Graph
DSP	Digital Signal Processing
FF	Flip-Flop
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface

IC	Integrated Circuit
LP	Linear Programming
ILP	Integer Linear Programming
IP	Intellectual Property
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LAB	Logic Array Block
MLAB	Memory Logic Array Block
LUT	Look-Up Table
MSB	Most Significant Bit
LSB	Least Significant Bit
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
SRAM	Static RAM
VLIW	Very Large Instruction Word

LZD	Leading Zero Detector
LZC	Leading Zero Counter
WAW	Write-After-Write
RAW	Read-After-Write
RDW	Read-During-Write
TLB	Translation Lookaside Buffer
TIRAM	Transposed Indicators RAM
STIRAM	Set Transposed Indicators RAM
BF-TI	Brute-Force Transposed Indicators
2D-HS-BCAM	2-Dimensional Hierarchical Search BCAM
II-HS-BCAM	Indirectly Indexed Hierarchical Search BCAM
BST	Binary Search Tree
BGP	Border Gateway Protocol
MSPS	Million Searches per Second
CMOS	Complementary Metal-Oxide-Semiconductor
NMOS	N-Type MetalOxideSemiconductor
ACM	Association for Computing Machinery

SIGDA	Special Interest Group on Design Automation
IEEE	The Institute of Electrical and Electronics Engineers
TRETS	Transactions on Reconfigurable Technology and Systems
FCCM	Field-Programmable Custom Computing Machines
ICFPT	International Conference on Field-Programmable Technology
NSERC	Natural Sciences and Engineering Research Council of Canada

List of Notations

The following lists all notations that have been used in the dissertation.

WAddr	Write Address
RAddr	Read Address
WData	Write Data Bus
RData	Read Data Bus
RWData	Bidirectional Read/Write Data Bus
RBankSel	Read Bank Selector
WPatt	Write Pattern
WAddr	Write Address
MPatt	Match Pattern
MAddr	Match Address
MIndc	Match Indicators

RmPatt	Removed Pattern
MultiPatt	Multiple Patterns
RefRAM	Reference RAM
SetRAM	Sets RAM
n_W	Number of Write Ports
n_R	Number of Read Ports
n_t	Number of True Ports
$n_{W,f}$	Number of Write Simple (Fixed) Ports
$n_{R,f}$	Number of Read Simple (Fixed) Ports
$n_{W,s}$	Number of Write Switched Ports
$n_{R,s}$	Number of Read Switched Ports
d	Memory Depth
w	Data Width
n_{M20K}	Number of M20K Blocks
n_{BReg}	Number of Bypass Registers
f_{fb}	I-LVT Feedback Function
f_{out}	I-LVT Output Extraction Function

P	Ports Group
W	Writes Group
R	Reads Group
E_s	Switched Edges Set
R_D	RAM Depth
C_D	CAM Depth
D_W	Data Width
P_W	Pattern Width
S_W	Set Width
$P_{W,opt}$	Optimal Cascaded Pattern Width
$I_{p,a}$	Match Indicator
A	Address Set
S	Set Set
$R_{W,max}$	Widest RAM Width
$R_{D,min}$	Shallowest RAM Depth
n_C	Number of Cascades

Acknowledgments

My gratitude begins with my research advisor Dr. Guy Lemieux, who have offered consistent support, guidance, dedication and patience throughout my research study and bringing this dissertation to fruition. Without his invaluable assistance, encouragement, advice, and technical insight, this work would not have been possible. I am very grateful for having him as my mentor and research advisor!

In addition, I would like to thank Dr. Mark Greenstreet, with whom I have worked on several exciting and productive project. His brilliance and expertise were a real inspiration. I am also thankful to Dr. Steve Wilton for the several discussions we had on academia, research and teaching. His deep knowledge, insights, support, and encouragement were invaluable.

I would also like to thank my external examiner, Dr. Paul Chow from the University of Toronto, for providing valuable comments that helped improve the quality of this dissertation and for being kind enough to travel and attend this dissertation defence in person. I thank my university examiners, Dr. Alexandra Fedorova and Dr. Jeremy Heyl and my supervisory committee Dr. Mieszko Lis and Dr. Mieszko Lis for their constructive feedback and suggestions during this dissertation defence.

I thank the System-on-Chip technical support staff, Roozbeh Mehrabadi and Dr. Roberto Rosales, for their help and dedication. I thank all my colleagues in the System-on-Chip research group, and other friends whom I have had the pleasure of working with. In particular, Rehan Ahmed, Usman Ahmed, Abdalrahman Arafeh, Samer Al-Kiswany, Mohammed Al-Taha, Alex Brant, Anas Bsoul, Assem Bsoul, Joydip Das, Stuart Dueck, Joseph Edwards, Abdullah Gharaibeh, Jeffrey Goeders, David Grant, Eddie Hung, Keith Lee, Zhiduo Liu, Hossein Omidian, Aaron Severance, Ahmad Sharkia, Douglas Sim, Tom Tang, Michael Yue, Chris Wang, and others.

Financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) is gratefully acknowledged. Thanks to Altera Corp. and CMC Microsystems for donating hardware and software licenses.

Last but most important, thanks and love goes to my family –my parents, Mohammad and Rasmia, my wife, Soaad, and our son, Mohammad– I am grateful for your unwavering love, gracious support and continuous encouragement. Thank you Soaad for being there for me again and again when I needed you the most. I would never have gotten to this point without you. Thank you for reminding me of my ability to accomplish this goal. I love you!

To my parents, Mohammad and Rasmia,

my wife, Soaad,

and our son, Mohammad.

Chapter 1

Introduction

The following introductory chapter is organized as follows. The need for Block-RAM-based massively parallel memory structures and the motivation behind our work is described in detail in Section 1.1. The thesis statement and research goals are provided in Section 1.2. Section 1.3 explains the feasibility problem of parallel memory structures in Field-Programmable Gate Arrays (FPGAs). Our research contributions are listed in Section 1.4. Section 1.5 describes the research methodology and evaluation metrics used to evaluate and compare our design to other techniques. The organization of the rest of this dissertation is summarized in Section 1.6

1.1 Motivation

Since they were first introduced three decades ago, Field-Programmable Gate Arrays (FPGAs) have evolved from being merely used as glue-logic to competing with custom-designed Application-Specific Integrated Circuits (ASICs). Modern FPGAs comprise hundreds of thousands of programmable logic gates augmented with thousands of configurable Digital Signal Processing (DSP) blocks and memory blocks, all on the same chip with flexible routing fabric. Routing and configuration flexibility of these numerous hardware blocks grants FPGAs their inherent parallelism; hence, FPGAs are exploited for massively parallel computing and can be tailored as an accelerator for specific applications.

These massively parallel systems demand highly parallel memory structures to keep pace with their concurrent nature since memories are usually the bottleneck of computation performance. In this dissertation, we propose new ways to build two key types of parallel memory structures in FPGAs, specifically Multi-Ported Random Access Memories (MPRAMs) and Content-Addressable Memories (CAMs), using the regular logic, registers and dual-ported memory blocks found in modern FPGAs.

FPGA devices provide SRAM blocks with only one or two access ports. This allows RAM content in one block to be accessed concurrently by one or two “users” at the same time. To allow more concurrent access, a method is required to allow potentially dozens of users to simultaneously read or write an SRAM in the same clock cycle as depicted in Figure 1.1.

In addition, modern FPGAs do not provide hard CAM blocks. Instead, they must be built from existing SRAM blocks; existing construction techniques do not scale well, making large CAMs very inefficient.

Below, we elaborate further on MPRAMs and CAMs.

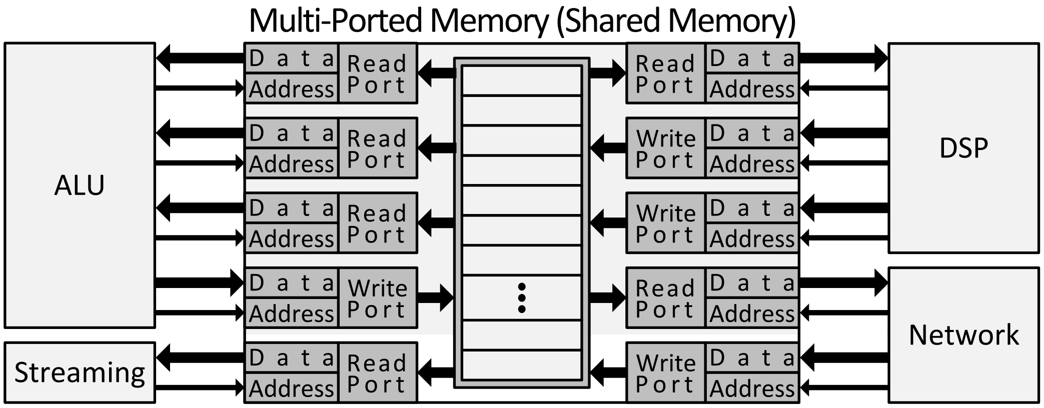


Figure 1.1: Multi-Ported Random Access Memory (MPRAM) abstraction as shared memory allowing concurrent access for several users.

1.1.1 Multi-Ported Random Access Memories (MPRAMs)

Multi-ported memories are the cornerstone of all high-performance Central Processing Unit (CPU) designs. They are often used in register files, but also in other shared-memory structures such as Translation Lookaside Buffers (TLBs), caches and coherence tags. Hence, high-bandwidth memories with multiple parallel reading and writing ports are required. In particular, multi-ported RAMs are often used by wide superscalar processors [6], Very Large Instruction Word (VLIW) processors [6, 7], multi-core processors [8, 9], vector processors, Coarse-Grained Reconfigurable Arrays (CGRAs) [10], and Digital Signal Processors (DSPs). For example, the second generation of the Itanium processor architecture employs a 20-port register file constructed from SRAM bit cells with 12 read ports and 8 write ports [8]. The key requirement for all of these designs is fast, concurrent, single-cycle access from multiple requesters. These multiple requesters require concurrent access for performance reasons. While there is demand for more RAM ports, the two leading multi-ported RAM techniques in FPGAs have relatively large overhead in (1) register usage or (2) total SRAM block count [11, 12]. This dissertation introduces two new design techniques that are near-optimal in resource overhead and have several practical advantages. Furthermore, it provides a mechanism to construct and optimize memory structures with time-switched ports. A RAM compiler has also been developed to automate the construction of these switched memories.

1.1.2 Content-Addressable Memories (CAMs)

CAMs, being a hardware implementation of associative arrays, are massively parallel search engines accessing all memory content to compare with the searched pattern simultaneously. An abstraction of a CAM is shown in Figure 1.2 where CAM patterns are stored in a memory. To find a specific pattern in the CAM, all CAM patterns are read simultaneously, then compared to the matched pattern. Comparing all CAM patterns to the matched pattern generates match indicators (or match lines) which indicated for each pattern in the CAM if this pattern matches the searched pattern. A priority-encoder will detect if there is a match and generate the address of the first matching pattern.

CAMs are considered heavy power consumers due to the very wide memory bandwidth requirement and the concurrent compare. While a standard RAM returns data located in a given memory address, a CAM returns an address containing a specific given datum, thus performing a memory-wide search for a specific value. To do this, it must perform a memory-wide search for a specific value, and there

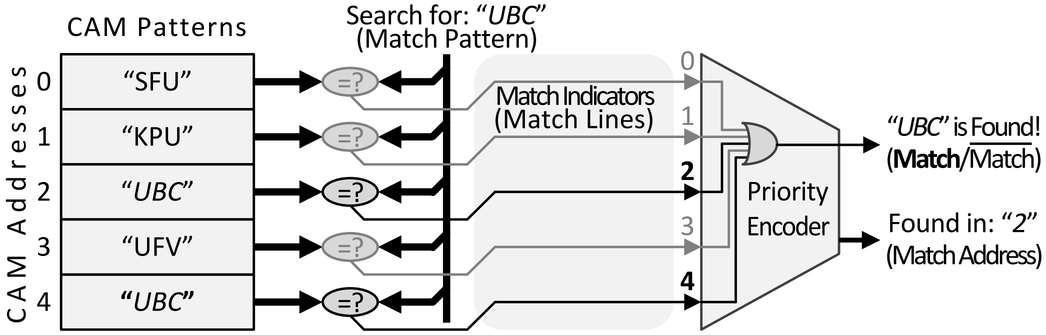


Figure 1.2: Content-Addressable Memory (CAM) abstraction as a massively parallel search engine accessing all memory content to compare with the searched pattern simultaneously.

may be multiple addresses that all match the data.

Since a CAM is actually a high-performance implementation of a very basic associative search, it can be used in many science fields [13]. CAMs are keystones of network processors [14–17], specifically used for Internet Protocol (IP) lookup engines for packet forwarding [18–24], intrusion detection [25–29], packet filtering and classification [30–32]. In high-performance processors, CAMs are used for memory management as coherence tag arrays for highly-associative caches [33] and Translation Lookaside Buffers (TLBs) [34, 35]. CAMs are also used to implement load and store queues in out-of-order instruction schedulers with a wide scheduling window [36]. In addition, CAMs are used for pattern matching [37–39], data compression [40], DSP [41, 42], databases [43, 44], bioinformatics [45, 46], and logic minimization [47]. A variety of other scientific fields also use CAMs as single-cycle associative search accelerators with millions of search entries.

Despite their importance, the high implementation cost of CAMs means they are used sparingly. As a result, FPGA vendors do not provide any dedicated CAM circuitry or any special infrastructure to enable a construction of efficient CAMs. Furthermore, FPGAs lack an area-efficient soft CAM implementation. Current CAM approaches in vendor IP libraries achieve a maximum of 64K entries and utilize all the resources of a modern FPGA device. Instead, designers tend to use algorithmic search heuristics causing a dramatic performance degradation.

1.2 Thesis Statement and Research Goals

This dissertation addresses the memory bottleneck of massively parallel reconfigurable systems by providing efficient, parallel and customizable embedded memory structures. Although MPRAMs and CAMs are important, their high implementation cost means they are used sparingly. As a result, FPGA vendors only provide standard dual-ported memories to handle the majority of usage patterns, and rely upon a simplistic soft CAM implementation that does not scale. This dissertation describes a novel, efficient and modular approach to construct MPRAMs and CAMs out of basic dual-ported RAM blocks, logic and registers.

The main goal is to address scaling issues with both designs, to permit use of deeper MPRAMs with more ports, as well as deeper and wider CAMs that can scale better. These new designs must be practical as well, meaning they achieve high clock rates and provide complete functionality such as initialization, bypassing, and fast updates.

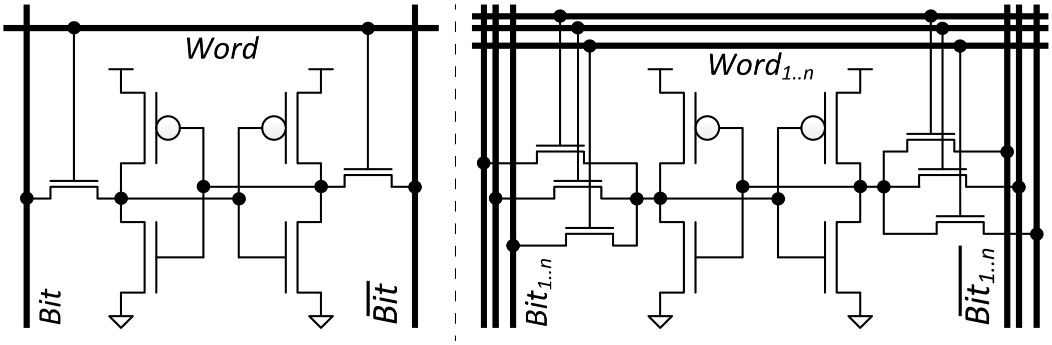


Figure 1.3: (left) A single-port SRAM cell using two cross-coupled CMOS inverters. (right) Multi-ported access using additional pass-gates, word and bit lines.

1.3 Research Problem: On the feasibility of Parallel memory structures in FPGAs

In this section, we describe the limitations of creating area-efficient and performance-oriented parallel memory structures in FPGAs, specifically multi-ported memories and content-addressable memories.

1.3.1 Multi-Ported Random Access Memories (MPRAMs)

A Static RAM (SRAM) cell is a Complementary Metal-Oxide-Semiconductor (CMOS) bi-stable circuit built out of CMOS cross-coupled inverters and access pass-gates as illustrated in Figure 1.3 (left). To provide more access ports, the basic basic SRAM bit cell can be altered to provide more bit lines, word lines, and access transistors as described in Figure 1.3 (right), however, the area growth is quadratic with the number of ports [48]. Furthermore, this requires a custom design for each unique set of parameters (e.g., number of ports, width and depth of RAM).

Since FPGAs must fix their RAM block design for generic, most common usage cases, it is too costly to provide highly specialized RAMs with a large number of ports. In FPGAs, one way of synthesizing a multi-ported RAM is to build it from registers and logic. However, this is only feasible for very small memories. Other techniques, covered in Chapter 2, are inefficient and do not scale well for deep or wide MPRAMs. Hence, a method of composing arbitrary, multi-ported RAMs from simpler RAM blocks is required.

1.3.2 Content-Addressable Memories (CAMs)

CAMs are usually custom-designed at the transistor level [49–53]. As depicted in Figure 1.4, four additional transistors over the standard 6-transistor SRAM cell are required. The additional transistors form a comparison circuit, an XOR with NMOS-stack only, since its output (the match line) is pulled-up. Other variations of the BCAM cells are also available, e.g., the 9-T NAND-type BCAM cell [49]. However, the custom approach requires more engineering effort and longer delays in time-to-market. This custom approach is unsuitable for FPGAs.

Older FPGA devices, including Altera's FLEX, Mercury and APEX devices [54] employed minor architectural features to directly construct small CAM blocks. However, FPGA vendors do not provide dedicated hard cores for CAMs in modern devices. These have been replaced with soft CAM cores that employ existing Block-RAMs in a brute-force approach described in this dissertation as the traditional or transposed-indicators RAM approach.

While modern databases can easily contain millions of entries, the area growth

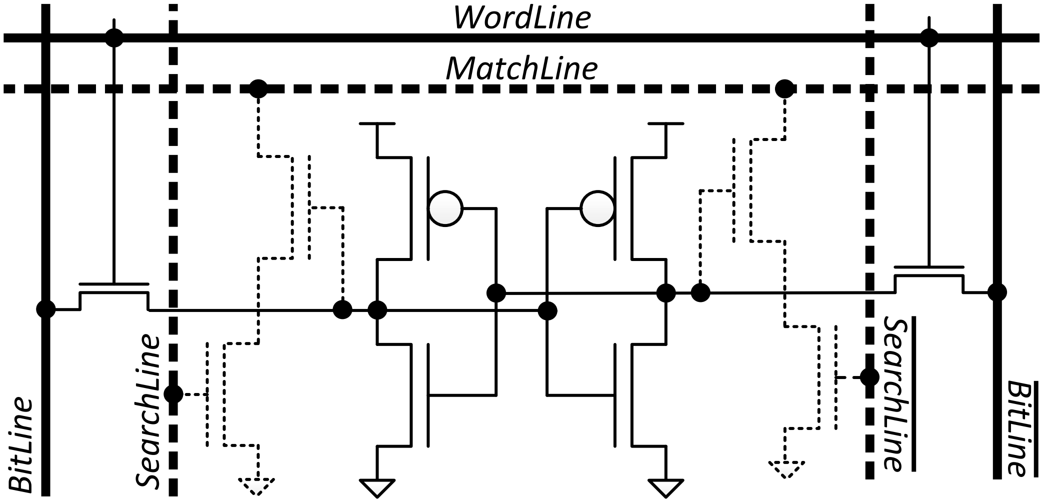


Figure 1.4: CMOS 10-T NOR-type BCAM cell; solid lines form a 6-T SRAM cell.

of traditional CAM techniques in FPGAs is high and is currently limited to 64k entries. Wide and shallow RAMs are needed to efficiently implement brute-force CAMs. Shallow RAMs are required because each extra bit in the CAM pattern width doubles the required RAM depth, resulting in poor efficiency. In addition, deeper CAMs can be built by increasing RAM width. However, FPGA RAM block width is growing slowly. For example, M4K blocks in Stratix II devices have minimal depth of 128 with maximal width of 36, M9K blocks in Stratix III and Stratix IV devices have minimal depth of 256 and maximal width of 36, M20K blocks in Stratix V devices [55] have minimal depth of 512 and maximal width of 40. With the increasing depth of RAMs, and limited width growth, the brute-force approach is getting less efficient.

1.4 Research Contributions

This section lists the contributions of my dissertation. The multi-ported memory contributions can be categorized into the Invalidation-Based Live-Value Table (I-LVT) work in Chapter 3 and the switched ports work in Chapter 4. Whereas our preliminary work on BCAMs is the 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) work in Chapter 5 and the follow-up work is the Indirectly Indexed Hierarchical Search BCAM (II-HS-BCAM) in Chapter 6.

1.4.1 Invalidation-Based Live-Value Table (I-LVT)

Recently, a few FPGA-based multi-ported RAM designs have been proposed. They use a Live-Value Table (LVT) together with multi-banking for data storage [11]. While each writing port writes to a different bank, the LVT tracks the last-written bank for each memory address, allowing reading of the latest data. Since the LVT is composed of registers, it cannot build deep MPRAMs efficiently. Alternatively, the XOR-based method retrieves the latest written data by utilizing cancellation properties of the XOR operator[12]. The XOR-based method overcomes the register-based memory limitation by storing all data in BRAMs; however, this incurs excessive memory overhead since wide memories are required to accommodate all the XOR-ed data.

In Chapter 3, an SRAM-based LVT is proposed by utilizing an invalidation-table data structure [1]. Similar to a regular LVT, the Invalidation-Based Live-Value Table (I-LVT) determines the correct bank to read from, but it differs in how updates to the table are made.

While previous approaches use registers to implement a live-value-table, the breakthrough of this approach is that it is near-optimal and purely SRAM-based, allowing it to efficiently scale to large depths. Compared to other multi-ported approaches, the I-LVT also provides improved overall performance. This dissertation demonstrates the viability, area reduction, and performance benefits of the proposed approach and provide an open source library with a fully tested, generic and modular implementation that can be adopted in parallel computing systems.

1.4.2 Switched Ports

Switched ports, first introduced in [2], are a generalization of true (bidirectional) ports, where a certain number of write ports can be dynamically switched into a different number of read ports using a common read/write control signal. While a true port is a pair of read/write ports, switched ports are best described as a set of read ports and set of write ports. Furthermore, a given application may have multiple sets, each set with a different read/write control. While previous work generates multi-ported RAM solutions that contain only true ports [56], or only simple (unidirectional) ports, this research demonstrates that using only two models is too limiting and prevents optimizations from being applied. In particular, the use of switched ports can lead to a reduction in the amount of storage needed by the data banks. It does this by using the underlying true-dual-port capability of FPGA Block-RAMs.

The general problem of switched ports is optimized by solving the corresponding set cover problem via Integer Linear Programming (ILP). This is the first time such an optimization model is used to construct multi-ported memories. A memory compiler that automates the construction of a multi-ported RAM with switched ports was released as an open source library.

1.4.3 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM)

Due to the increasing amount of processed information, modern BCAM applications demand a deep searching space. However, traditional BCAM approaches in FPGAs suffer from storage inefficiency. The 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) [4] is a novel and efficient technique for constructing BCAMs out of standard SRAM blocks in FPGAs. To achieve high storage efficiency, the suggested technique first searches for a row containing multiple patterns, then a search is done within the row for a precise match.

Using Altera's Stratix V device, the traditional design method achieves up to a 64K-entry BCAM while the proposed technique achieves up to 4M entries. For the 64K-entry test-case, the traditional method consumes 43 times more Adaptive Logic Modules (ALMs), 18 times longer mapping runtime, and achieves only one-third of the F_{max} of the proposed method. A fully parameterized Verilog implementation is being released as an open source hardware library. The library has been extensively tested using ModelSim and Altera's Quartus tools.

1.4.4 Indirectly Indexed Hierarchical Search BCAM (II-HS-BCAM)

The hierarchical search just described does not allow wide CAMs to be constructed; it incurs an exponential growth as pattern width increases. The II-HS-BCAM [5] approach solves this by indirectly storing match indicators. This allows II-HS-BCAM blocks to be cascaded in pattern width, allowing for linear growth. Our method exhibits high storage efficiency and is capable of implementing up to nine times wider BCAMs compared to other approaches.

1.5 Research Methodology and Evaluation Metrics

In the section, we describe the research methodology, including design methodology, simulation and synthesis used throughout the dissertation. Furthermore, we review the evaluation metrics used to evaluate and compare our design to other techniques.

1.5.1 Verilog Description

All architectures which have been proposed in this dissertation, together with previous and standard approaches, have been fully developed as parameterized Verilog modules. The Verilog implementation is used to verify correctness and evaluate characteristics of the suggested architectures and compare to standard approaches and previous attempts.

Some modules could not be described with Verilog directly. Alternatively, a Verilog generator was developed to generate these modules based on given design parameters. For instance, the Priority Encoder (PE) which was used in Chapter 5 and Chapter 6 to generate CAM match addresses has a recursive definition. Since Verilog is not capable of describing recursive modules, a generator was used to create the Verilog code recursively. In addition, the memory compiler in Chapter 4 requires solving a set-cover problem via Linear Programming (LP) to optimize the generated modules. Since performing this optimization process in Verilog is not possible, a Verilog generator was used to create Verilog code based on the optimized solution.

1.5.2 Simulation and Synthesis

In each chapter, a large variety of different architectures and design parameters are swept and simulated in batch using Altera's ModelSim version 10.1e, each with over a million random cycles. All different design modules were compiled, mapped and fit using Altera's Quartus II version 14.0 [57] on Altera's Stratix V `5SGXMABN1F45C2` device [55]. This is a high-end performance-oriented speed grade 2 device with 360k ALMs, 2640 M20K blocks, and 1064 I/O pins. Half of the ALMs can be used to construct Memory Logic Array Blocks (MLABs), where a single MLAB consists of 10 ALMs.

A run-in-batch simulation and synthesis flow manager that simulates and synthesizes various designs with various parameters in batch using Altera's ModelSim and Quartus II is also provided. The Verilog modules, the Verilog generators, the algorithmic scripts, and the flow manager are available online as an open source contribution [58].

1.5.3 Results Collection

A general sweep is performed to test all combinations. Following this, the full set of results is analyzed. In this dissertation, we omit many of the in-between settings because they behaved as one might expect to see via interpolation of the endpoints.

The run-in-batch flow manager was also used to collect design results of the proposed and previous techniques. Performance evaluation (e.g., $\mathrm{F}_{\mathrm{max}}$) was generated by TimeQuest, the Quartus II Static Timing Analysis (STA) engine. Resource consumption, such as registers, Look-Up Tables (LUTs), ALMs, Logic Array Blocks (LABs), MLABs, and Block-RAMs (BRAMs), was collected after the design was fit to the Stratix V device by the Quartus II fitter.

1.5.4 BRAM Packing Approximation

As shown in Figure 1.5, Altera's M20K blocks can be configured into several RAM depth and data width configurations [55]. The total amount of utilized SRAM bits can be either 16Kbits, or 20Kbits. Assuming that the RAM packing process minimizes the number of blocks cascaded in depth to avoid additional address decoding, each 16K lines will be packed into single bit-wide blocks, and the remainder will be packed into the minimal required configuration.

An estimation of the number of packed M20K blocks required to construct a RAM with a specific depth, d, and data width, w, is $n_{M20K}(d,w)$. This value is described as follows

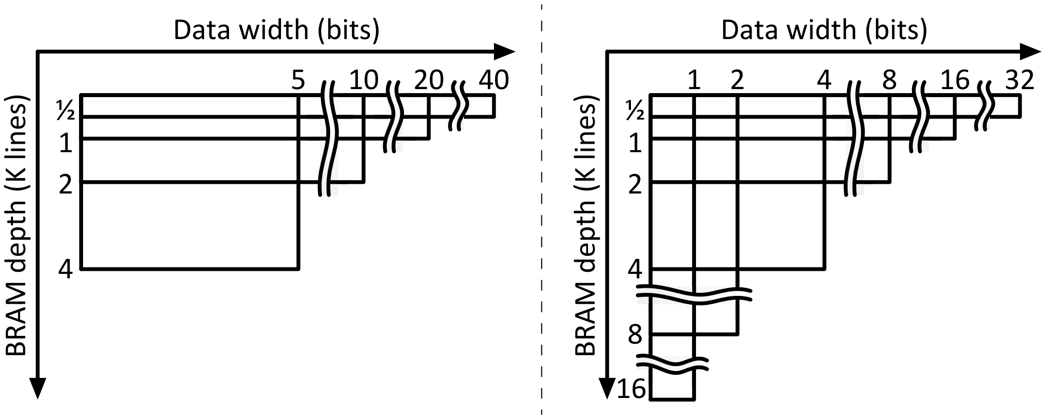


Figure 1.5: Altera M20K configuration (left) 20Kb (right) 16Kb.

$$n_{M20K}(d,w) = \left\lfloor \frac{d}{16k} \right\rfloor \cdot \begin{cases} w & d\%16k > 8k \\ \lfloor w/2 \ \rfloor & 8k \geq d\%16k > 4k \\ \lfloor w/5 \ \rfloor & 4k \geq d\%16k > 2k \\ \lfloor w/10 \rfloor & 2k \geq d\%16k > 1k \\ \lfloor w/20 \rfloor & 1k \geq d\%16k > \frac{1}{2}k \\ \lfloor w/40 \rfloor & \frac{1}{2}k \geq d\%16k \end{cases} . \qquad (1.1)$$

1.6 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 provides a background on multi-ported memories and content-addressable memories in FPGAs. Standard and previous methods are surveyed and their limitations are discussed. Chapter 3 provides a detailed description of our near-optimal Invalidation-Based

Live-Value Table (I-LVT) multi-ported memory. Multi-ported memories with switched port and the Multi-Switched-Ports Random Access Memory (MSPRAM) compiler are described in Chapter 4. The 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM), an efficient and deep BCAM for narrow patterns is provided in Chapter 5. Chapter 6 describes the Indirectly Indexed Hierarchical Search BCAM (II-HS-BCAM), an area-efficient and high-performance BCAM architecture, based on hierarchical search and compression techniques that supports wide patterns. Future directions and conclusions are drawn in Chapter 7.

Chapter 2

Background and Related Work

This chapter provides a background on multi-ported memories and content-addressable memories in FPGAs. Standard and previous methods are surveyed and their limitations are discussed.

2.1 RAM Multi-Porting Techniques in Embedded Systems

This section provides a review of current methods of creating multi-ported memories in embedded systems. Creating multi-ported access to register-based memories is described in Section 2.1.1. Multi-pumping is described in Section 2.1.2. Replicating a memory bank to increase the number of read and write ports is described in Section 2.1.3 and Section 2.1.4, respectively.

2.1.1 Register-based RAM

Multi-ported RAM arrays can be constructed using basic Flip-Flop (FF) cells and logic. As depicted in Figure 2.1, each writing port uses a decoder to steer the relevant written data into the addressed row. Each read port uses a mux to choose the relevant register output. This method is not practical for large memories due to area inflation, fan-out increase, performance degradation, and a decline in routability.

2.1.2 RAM Multi-Pumping

A time-multiplexing approach can be applied to an SRAM block to reuse access ports and share them among several clients, each during a different time slot. As depicted in Figure 2.2, addresses and data from several clients are latched then given round-robin access to a dual-ported RAM. The RAM must operate at a higher frequency than the rest of the circuit. If the maximum RAM frequency is similar to the pipe frequency, or a large number of access ports are required, then multi-pumping cannot be used.

In FPGAs the process of generating multi-pumped memories can be automated and tailored for heterogeneous applications [59–62]. Furthermore, the multi-pumping methods can be combined with other techniques to save area [63]. A number of designs utilize multi-pumping to gain additional access ports while keeping area overhead minimal [64, 65]. The 2.3GHz Wire-Speed POWER processor uses double-pumping to double the writing ports [66].

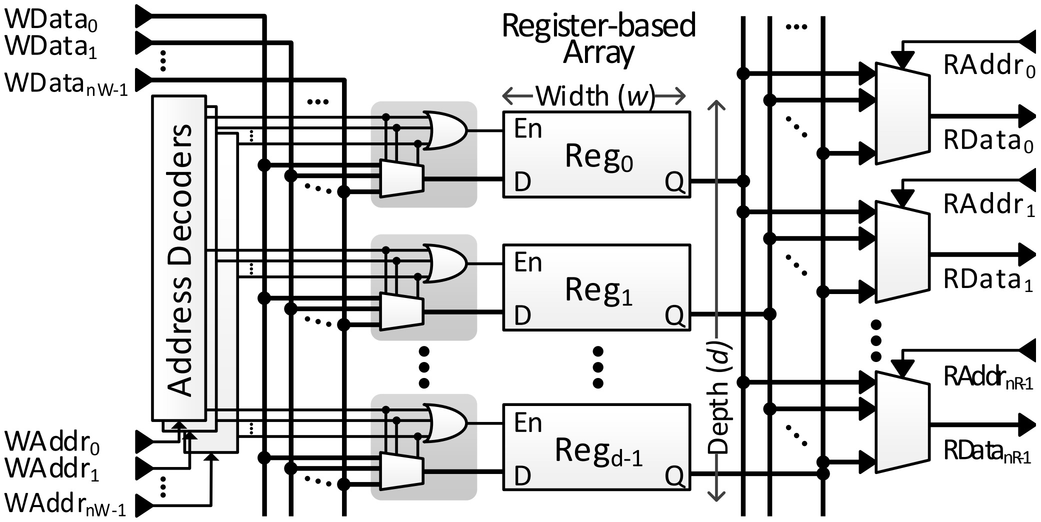


Figure 2.1: Register-based multi-ported memory.

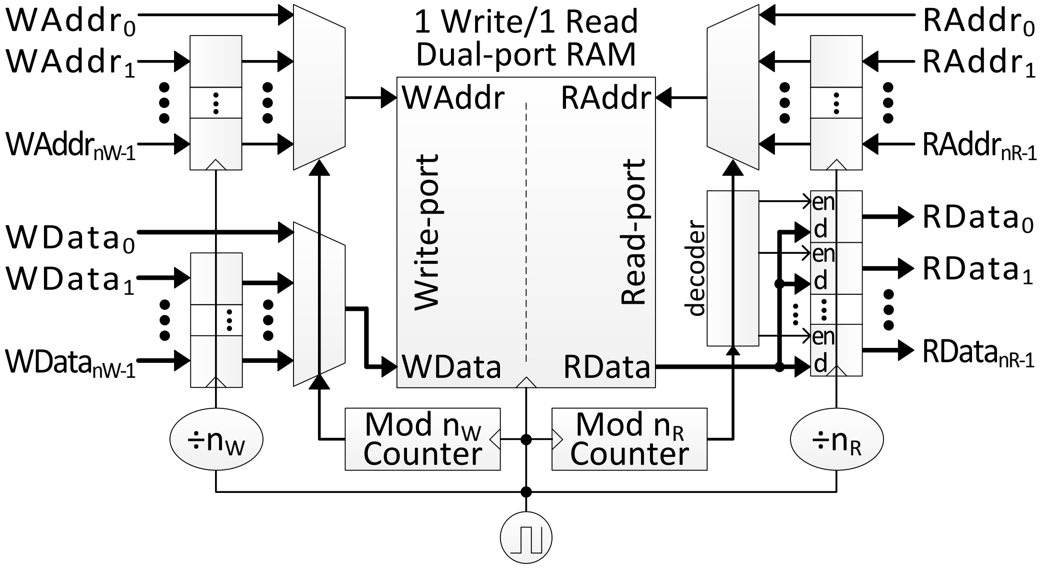


Figure 2.2: Multi-pumping: RAM is overclocked, allowing multiple access during one pipeline cycle.

2.1.3 Multi-Read RAM: Bank Replication

To provide more reading ports, the whole memory bank can be replicated while keeping common write address and data as shown in Figure 2.3. Data will be written to all bank replicas at the same address, hence reading from any bank is equivalent. This method incurs high SRAM area and consumes more power. However, the replication approach has two strong advantages over other multi-porting approaches. The first is the simplicity and modularity of bank replication. The second is that read access time is unaffected as the number of port increases; only write delays increase due to fan-out, but this can be hidden by pipelining and bypassing. The bank replication technique is commonly used in state-of-the-art processors to increase parallelism. The 2.3GHz Wire-Speed POWER processor replicates a 2-read SRAM bank to achieve 4 read ports [66]. Each of the two integer clusters of the Alpha 21264 processor has a replicated 80-entry register file, thus doubling the number of read ports to support two concurrent integer units. Similarly, the 72-entry floating-point register file is duplicated, supporting two concurrent floating-point units [67].

2.1.4 Multi-Write RAM: Emulation via Multi-banking

Multi-ported memories are very expensive in terms of area, delay, and power for a large number of ports. The overhead of multi-porting can be reduced by multi-banking if one relaxes the guaranteed access delay constraint. As depicted in Figure 2.4, the total RAM capacity can be divided into several banks, each with few ports (e.g., dual-port). A fixed hashing scheme is used to match each address

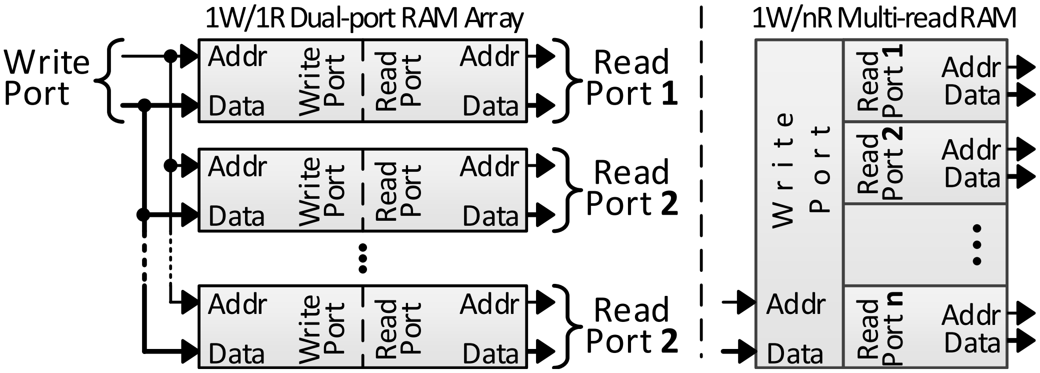


Figure 2.3: (left) Replicated dual-ported banks with a common write port. (right) Multi-read RAM symbol used in this dissertation.

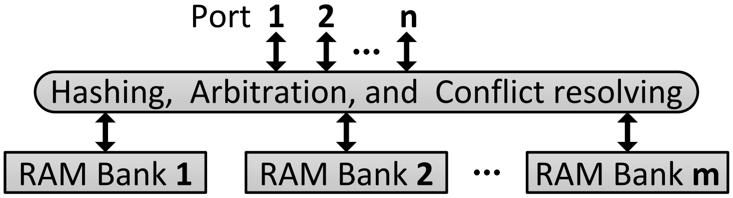


Figure 2.4: Multi-banking: RAM capacity is divided into several banks. Ports access with a fixed hashing scheme.

to a single bank; often, the address MSBs are used. Arbitration logic steers access from multiple ports to each bank. Since two ports can request access to the same bank at the same time, a conflict resolving circuit determines which port grants access to a specific bank. The other port will miss the arbitration and is required to request access again. Not only does this approach provide unpredictable access latency due to the arbitration miss, but it also increases delay due to the additional circuitry. Several approaches have been proposed to improve multi-banking [6, 68–71]. State-of-the-art memory controllers and caches are based on multi-banking due to area and power efficiency. For example, Intel's i486 CPU has a data cache with 8 interleaved banks and two access ports [72].

2.2 Multi-ported SRAM-based Memories: Prior Work

In this section a review of three previous multi-ported SRAM-based memories are provided. The first approach is based on multi-banking with a LVT [11, 73] and is described in Section 2.2.1. The second approach retrieves the latest written data by utilizing logical XOR properties [12, 73] and is described in Section 2.2.3. Constructing data banks with true ports support are provided in Section 2.2.2. These methods are surveyed and their limitations are discussed.

2.2.1 LVT-Based Multi-ported RAM

As depicted in Figure 2.5 (left), LVT-based multi-ported memories are compromised of two portions, LVT and data storage. In the data storage, each storage bank has a single write port and n_R read ports. This bank is replicated n_W times, where n_W and n_R are the number of write ports and read ports respectively. When a write port writes data to a specific address, the LVT is updated to indicate the port of this latest write operation. When this address is later read, the LVT provides this write port ID to retrieve the correct data.

For each RAM address, the LVT stores the ID of the bank replica that holds the latest data. The data storage uses a different bank replica for each writing port, while each bank replica has several reading ports. All banks are accessed by all read addresses in parallel; the LVT helps to steer the read data out of the correct bank since it holds the ID of last accessed (written) bank for each address.

Actually, the LVT itself is a multi-ported memory with the same depth and

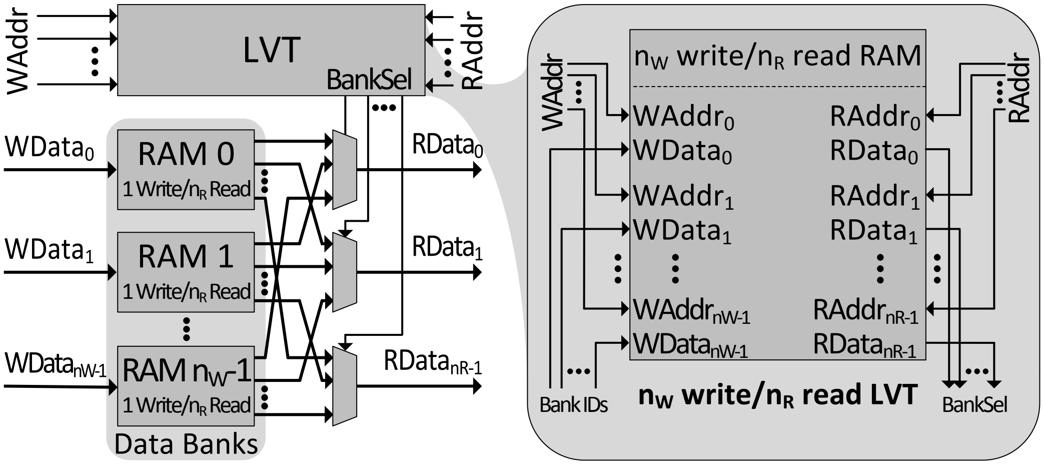


Figure 2.5: (left) LVT-based multi-ported memory. (right) An LVT implemented using multi-ported memory.

number of writing ports as the implemented multi-ported RAM. However, since the LVT stores only bank IDs, the data (line) width of the LVT table is only $\lceil \log_2 \rceil$ of writing ports. As described in Figure 2.5 (right), the LVT doesn't have write data, instead it writes a fixed bank ID for each port.

Since an LVT is a narrow multi-port memory, it is implemented as a register-based multi-ported memory. As explained in Section 2.1.1, register-based RAM is not suitable for building deep memories. While the LVT width is only $\log_2$ of the number of writing ports, the depth of the LVT is still identical to the depth of the original RAM. *This is the main cause of the area overhead.* In Chapter 3 we present two methods for building the LVT out of Block-RAMs instead of registers, allowing deep and wide multi-ported memories to be constructed much more efficiently.

Assuming that bank IDs are binary encoded, the total number of registers required to implement the LVT is

$$d \cdot \lceil \log_2 n_W \rceil, \tag{2.1}$$

where d is the depth of the memory.

For deep memories, the large number of registers and huge read multiplexers make register-based LVTs impractical. For example, a Stratix V GX A5 device (185k ALMs) can fit up to 16k-deep memory with four write ports.

For the data storage part, a total of n_W multi-read banks are required for each write port. Each multi-read bank supports n_R read ports, allowing the LVT to select the required data block. The total number of SRAM cells in the data storage is

$$d \cdot w \cdot n_W \cdot n_R, \tag{2.2}$$

where w is width of data.

Using Altera's Stratix M20K BRAMs , the total number of required M20K blocks for the data storage is

$$n_{M20K}(d, w) \cdot n_W \cdot n_R. \tag{2.3}$$

2.2.2 Multi-Ported Random Access Memories with True Dual-Ports

In the previous subsection, the data storage portion is built with simple read and simple write parts. Two papers [56, 73] introduced a modification to the data banks to build a multi-ported memory where all ports are true (bidirectional) ports. This is achieved by utilizing the bidirectional functionality of underlying true-dual-port BRAMs in the FPGA. Figure 2.6 (left) illustrates a generalization of this method. Each port either writes to a set of data banks, or reads from them. *Every pair of ports has one data bank in common*, hence, when a port reads, it can access data written from any other port. A register-based LVT determines which bank holds the latest data. Figure 2.6 (right) describes an example of a RAM with 3 true-ports. This problem is identical to the mathematical handshakes problem, where each port must connect (handshake) to all other ports via a RAM. Hence, a total of

$$\frac{1}{2} \cdot n_t \cdot (n_t - 1) \tag{2.4}$$

data copies are required, where n_t is the number of bidirectional (true) ports. In contrast, the original LVT approach [11] requires n_t^2 data copies. Nevertheless, this true-port RAM architecture is still based on a register-based LVT, hence it suffers from the same shortcomings.

2.2.3 XOR-Based Multi-Ported RAM

While the two LVT-based multi-ported memories just shown implement their LVTs as a register-based multi-ported memory, the XOR-based multi-ported memory is implemented using SRAM blocks [12, 73]. This makes it more efficient for deep memories. However, as will be shown, it is inefficient for wide memories.

The XOR-based method utilizes the special properties of the XOR function

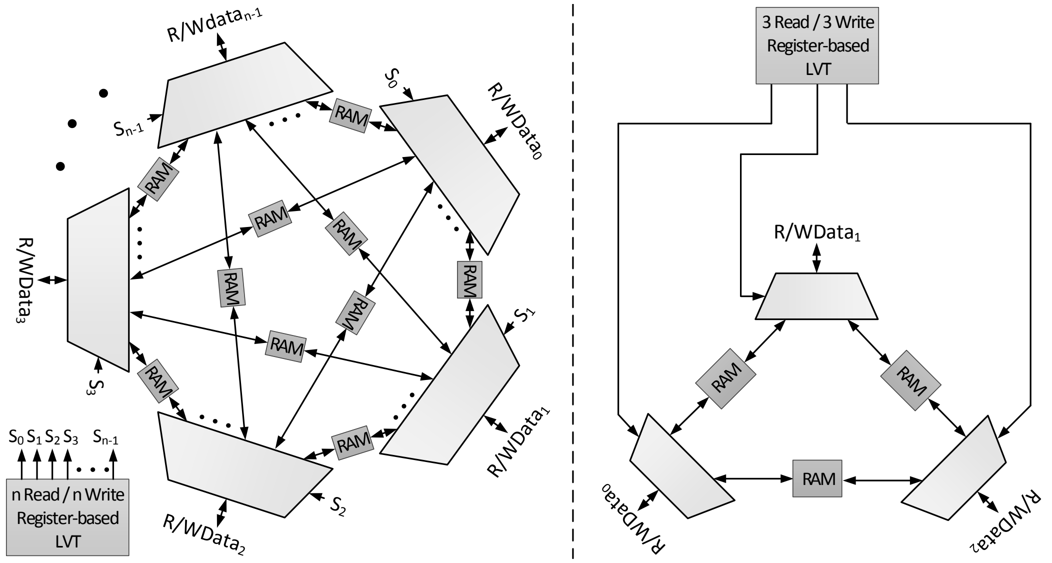


Figure 2.6: Data banks with bidirectional true-ports support. Each port share a single BRAM with any other port.(left) Generalized approach. (right) A 3 true-ports example.

to retain the latest written data for each write port [74]. XOR is commutative $a \oplus b = b \oplus a$, associative $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, zero is the identity $a \oplus 0 = a$, and the inverse of each element is itself $a \oplus a = 0$.

Like the previous methods, the XOR-based method contains two structures: one replaces the LVT and is used to encode written data, while the other is similar to the data storage but it stores encoded data.

As illustrated in Figure 2.7, each write port has a bank with multi-read and a single write. Some of the read ports are used as a feedback to encode the data to be rewritten, while the remaining read ports generate the data output. To perform a write, the new data is XOR'ed together with all the old data from the other banks; this encoded value is written to the corresponding bank. Hence if an address A is written through write port i with data $WData_i$, $Bank_i$ will be written with

$$Bank_i[A] \leftarrow Bank_0[A] \oplus Bank_1[A] \oplus \cdots \oplus WData_i \oplus \cdots \oplus Bank_{n_W-1}[A]. \quad (2.5)$$

A read is performed by XOR'ing all the data for the corresponding read address from all the banks, hence,

$$RData_i[A] \leftarrow Bank_0[A] \oplus Bank_1[A] \cdots \oplus Bank_{n_W-1}[A]. \quad (2.6)$$

Substituting $Bank_i[A]$ from (Equation 2.5) into (Equation 2.6) and applying commutative and associative properties of the XOR shows that each bank appears twice in the XOR equation, hence will be cancelled since XORing similar elements is 0. The only remaining item will be $WData_i$, the required data.

The XOR-based multi-ported memory requires n_W multi-read banks for each write port. Each multi-read bank supports $n_W - 1$ read ports for feedback, and n_R read ports. Each feedback read port is of width d, to match the write data, so these feedback memories can be quite large. The number of required SRAM cells for the entire multi-ported memory is

$$d \cdot w \cdot n_W \cdot (n_R + n_W - 1). \quad (2.7)$$

Using Altera's Stratix M20K BRAMs , the total number of required M20Ks is

$$n_M20K(d,w) \cdot n_W \cdot (n_R + n_W - 1). \quad (2.8)$$

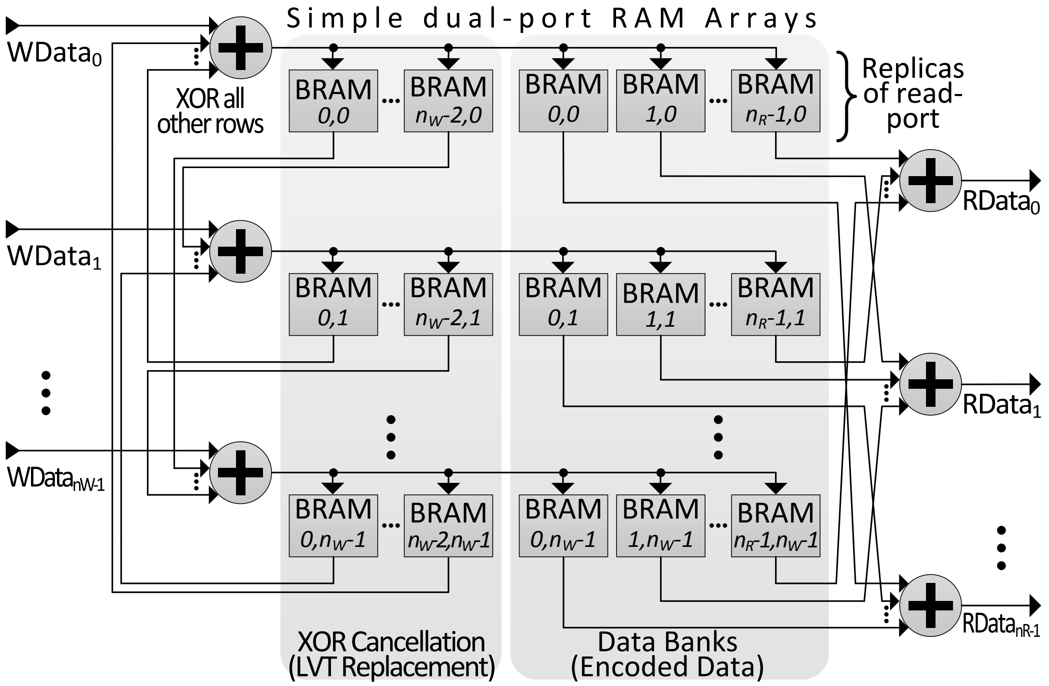


Figure 2.7: XOR-based multi-ported memory.

Since FPGA Block-RAM is synchronous, data feedbacks are read with a one-cycle read delay. Hence, the written data, their addresses and write-enables must be retimed to match the feedback data. This requires the following number of registers

$$n_W \cdot (w + \lceil \log_2 d \rceil + n_W). \tag{2.9}$$

2.3 FPGA-Based Binary Content-Addressable Memories (BCAMs)

CAMs are usually designed at the transistor level [49–53]. Older FPGA devices, including Altera's FLEX, Mercury and APEX devices [54], employed minor architectural features to support small CAM blocks. However, FPGA vendors do not provide dedicated hard cores for CAMs in modern devices. These have been replaced with soft CAM cores that employ a brute-force approach. While the address space in modern databases can easily exceed millions of entries, traditional CAM techniques in FPGAs cannot satisfy these requirements. Wide and shallow RAMs are needed to efficiently implement brute-force BCAMs, yet BRAM width is growing slowly.

CAMs can be classified into two major classes: Binary Content-Addressable Memories (BCAMs) and Ternary Content-Addressable Memories (TCAMs). While BCAMs hold binary values only, TCAMs can hold don't care values (X's) that can match any value of the corresponding pattern bit. In this dissertation, we focus on CAM architecture in FPGAs. As depicted in Figure 2.8, CAM architecture in FPGAs can be classified into three categories, based on the FPGA memory resources they utilize. The first is register-based where registers are used to store patterns and concurrently compare all register values, however, register resources are limited, a state-of the art Altera's Stratix V device [55] can provide up to 32K-entries of a single byte pattern BCAM. SRAM blocks are utilized in the second approach to store for each pattern if it exists in any of the address individually, hence the name transposed RAM, also known as the brute-force approach. The same Altera Stratix V device can realize a CAM with byte-wide patterns up to 64K in depth. The third method is reconfigurable Reconfiguration Memory Based Content-Addressable Memory (RCAM), where LUT configuration memory is utilized. It is possible to implement a 128K-line single byte pattern RCAM on the

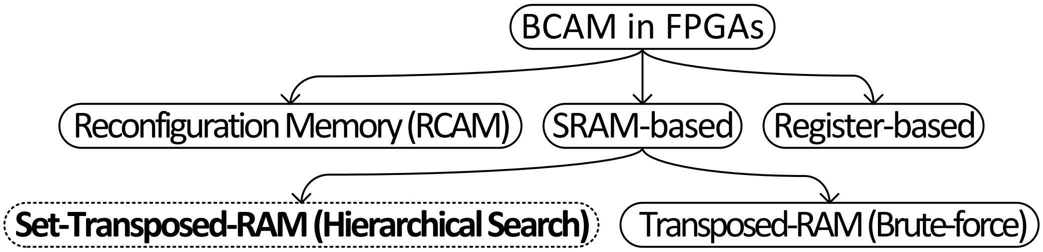

BCAM in FPGAs
Reconfiguration Memory (RCAM)
SRAM-based
Register-based
Set-Transposed-RAM (Hierarchical Search)
Transposed-RAM (Brute-force)

Figure 2.8: CAM classification in FPGAs.

same Stratix V device. However, a LUT's SRAM configuration is written serially, which adversely affects writing throughput. The following subsections provide a detailed review for these categories.

This section provides a review of current BCAM architectures in FPGAs. Using registers to create BCAMs is described in Section 2.3.1. The traditional brute-force BRAM-based approach is described in Section 2.3.2. BCAM pattern width cascading is described in Section 2.3.3. Reconfiguration Memory Based Content-Addressable Memories (RCAMs) are explained in Section 2.3.4. Alternative algorithmic searches are briefly described in Section 2.3.5. A review of FPGA vendors' supports of BCAMs is placed in Section 2.3.6.

2.3.1 Register-Based BCAMs

The flexibility of reading and writing flip-flops makes it possible to concurrently read and compare all the patterns as depicted in Figure 2.9. Similar to a register-based RAM, an address decoder is used to generate one-hot decoded address lines, enabling a single line for writing. Each registered pattern is compared with the Match Pattern (MPatt) simultaneously; the comparison results drive the match

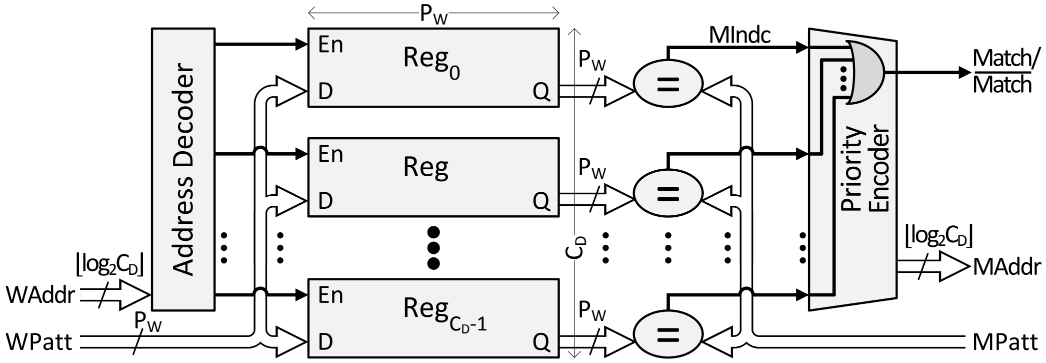


Figure 2.9: Register-based BCAM.

line, also called Match Indicators (MIndc) followed by a priority-encoder to detect the first Match Address (MAddr). The high demand for limited resources such as registers, comparators, address decoder and priority encoder (proportional to BCAM depth), make this approach infeasible for deep BCAMs; using Altera's high-end Stratix V device [55], only a one byte-wide, 32K-depth BCAM can be generated.

2.3.2 Brute-Force Approach via Transposed Indicators RAM (TIRAM)

The brute-force approach creates BCAMs out of standard BRAMs. These BRAMs are addressed with the match pattern, thus allowing a single cycle pattern match. This approach is utilized by FPGA vendor IP libraries or application notes. As depicted in Figure 2.10, a BRAM is addressed by the match pattern while each bit of the RAM data bits indicates the existence of the pattern. The data bit position corresponds to the BCAM address location. Thus, the depth of the CAM, C_D, must match the width of the data RAM, D_W. Also, the pattern width, P_W, of the CAM

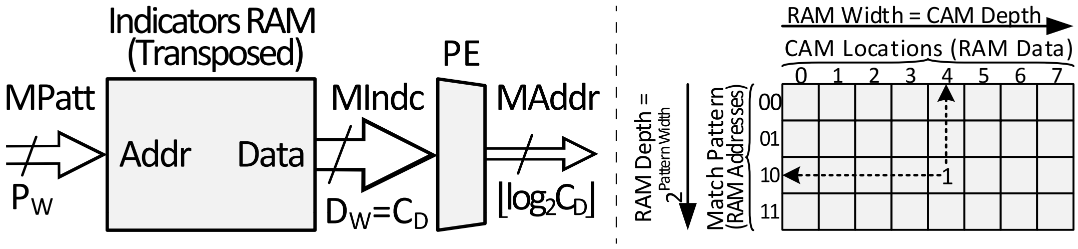


Figure 2.10: (left) CAM as a Transposed-RAM followed by a Priority Encoder (PE). (right) Example of a pattern indicator for pattern '10' in address 4.

must match the address width of the data RAM, i.e., $P_W = \lceil log_2 R_D \rceil$, where R_D is RAM depth.

In this dissertation, the Transposed Indicators RAM (TIRAM) structure in Figure 2.10 (left) is described as a matrix of indicators

$$TIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \cdots & I_{0,C_D-1} \\ I_{1,0} & I_{1,1} & \cdots & I_{1,C_D-1} \\ \vdots & \vdots & \ddots & \vdots \\ I_{|P|-1,0} & I_{|P|-1,1} & \cdots & I_{|P|-1,C_D-1} \end{bmatrix} \tag{2.10}$$

$$\forall a \in A, p \in P : I_{p,a} = \big(RAM\,[a] \;\; EQ \;\; P\big),$$

Where A is the address space set and P is the pattern set in the corresponding RAM structure.

The complete system of the brute-force TIRAM approach is described in Figure 2.11. Reading from the TIRAM is performed by providing the Match Pattern (MPatt) as address to read the Match Indicators (MIndc) for the entire BCAM address space for this specific match pattern. A priority encoder detects the first Match Address (MAddr) from the match indicators. However, writing (also called or updating) to the TIRAM structure requires more computation since it requires setting the new match indicator and clearing the old match indicator. Appendix A provides the detailed writing mechanism of the brute-force TIRAM BCAM.

To implement a BCAM with C_D entries and P_W pattern width, namely a $C_D \times P_W$ BCAM, The brute-force TIRAM approach requires $C_D \cdot P_W$ SRAM cells for the Reference RAM (RefRAM), shown in Figure 2.11, which stores a copy of the Pattern written to at each CAM location. In contrast, the TIRAM requires $2^{P_W} \cdot C_D$ SRAM cells, for a total of:

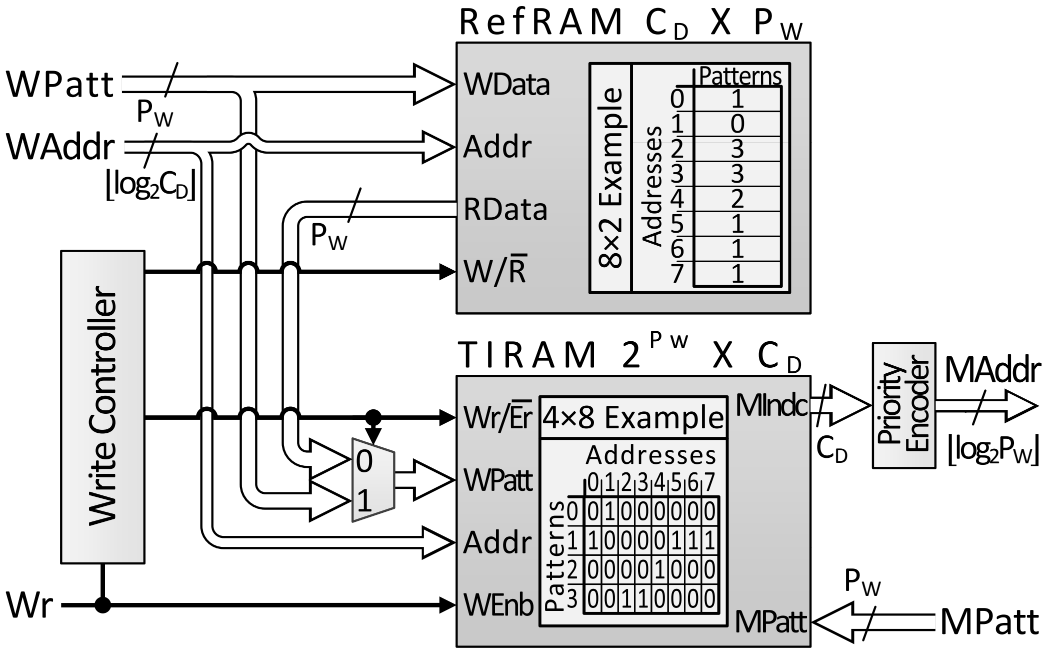
$$C_D \cdot P_W + 2^{P_W} \cdot C_D. \tag{2.11}$$

Assuming that RefRAM is fully utilized, and the TIRAM uses the widest BRAM configuration, the BRAM count is estimated as

$$\left\lceil \frac{C_D \cdot P_W}{R_{D,min} \cdot R_{W,max}} \right\rceil + \left\lceil \frac{2^{P_W}}{R_{D,min}} \right\rceil \cdot \left\lceil \frac{C_D}{R_{W,max}} \right\rceil, \tag{2.12}$$

where $R_{W,max}$ and $R_{D,min}$ are BRAM parameters indicating the maximum width, and minimum depth, respectively.

This BRAM-based brute-force approach is adopted by Xilinx [75–78] and Altera [79] to create soft CAMs as described in their application notes. Zerbini and Finochietto [80] apply the pattern width cascaded brute-force approach to emulate TCAMs for packet classification; however updating the TCAM content is not dis-



Addresses	Patterns
0	1
1	0
2	3
3	3
4	2
5	1
6	1
7	1

Patterns \ Addresses	0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	0	0
1	1	0	0	0	0	1	1	1
2	0	0	0	0	1	0	0	0
3	0	0	1	1	0	0	0	0

Figure 2.11: Brute-force TIRAM approach with 8×2 example.

cussed. Jiang [81] also uses the brute-force approach to emulate TCAMs, however, a pattern update requires a sequential rewriting of all RAM addresses for each pattern. Ullah *et al.* [82–85] also use the brute-force approach, but their TCAMs can be partitioned, allowing the search of specific TCAM fragments. However, the required storage is still similar to the brute-force approach. Furthermore, rewriting the TCAM requires a serial rewriting of all RAM locations.

2.3.3 BCAM Pattern Width Cascading and Scaling

As shown in Equation 2.11, SRAM cell usage for the brute-force TIRAM approach is exponential to pattern width P_W. A wide pattern width will make the SRAM requirements infeasible. BCAM pattern width cascading relaxes this SRAM growth from exponential into linear. As depicted in Figure 2.12, the BCAM pattern (both matched and written pattern) is divided into smaller pattern segments; each segment is associated with a separate BCAM. The BCAM pattern width cascading technique is used by Xilinx[75–78] and Altera [79] to create soft scalable BCAMs as described in their application notes.

The write operation writes every pattern segment into its corresponding BCAM, while match operation matches each pattern segment with its corresponding BCAM. A match for the entire pattern is found if a match is found for all segments individually at the same BCAM location (hence the bitwise AND). The optimal pattern segment width is determined by the minimal depth $R_{D,min}$ of the BRAM, namely the shallowest and widest configuration, since choosing a wider pattern requires exponential growth. The optimal pattern width is therefore $P_{W,opt} = \left\lfloor log_2\left(R_{D,min}\right)\right\rfloor$ and the total number of BCAM pattern segments cascades is $n_C = \left\lceil \frac{P_W}{P_{W,opt}} \right\rceil$.

One stage of TIRAM will consume $\left\lceil \frac{C_D}{R_{W,max}} \right\rceil$ BRAMs and the total BCAM consumption of the TIRAM is therefore

$$n_C \cdot \left(\left\lceil \frac{C_D}{R_{W,max}} \right\rceil + \left\lceil \frac{C_D \cdot P_{W,opt}}{R_{D,min} \cdot R_{W,max}} \right\rceil \right). \qquad (2.13)$$

The linear relation to P_W and C_D in Equation 2.13 is clear, in contrast to the uncascaded version in Equation 2.12 where the relation to P_W is exponential.

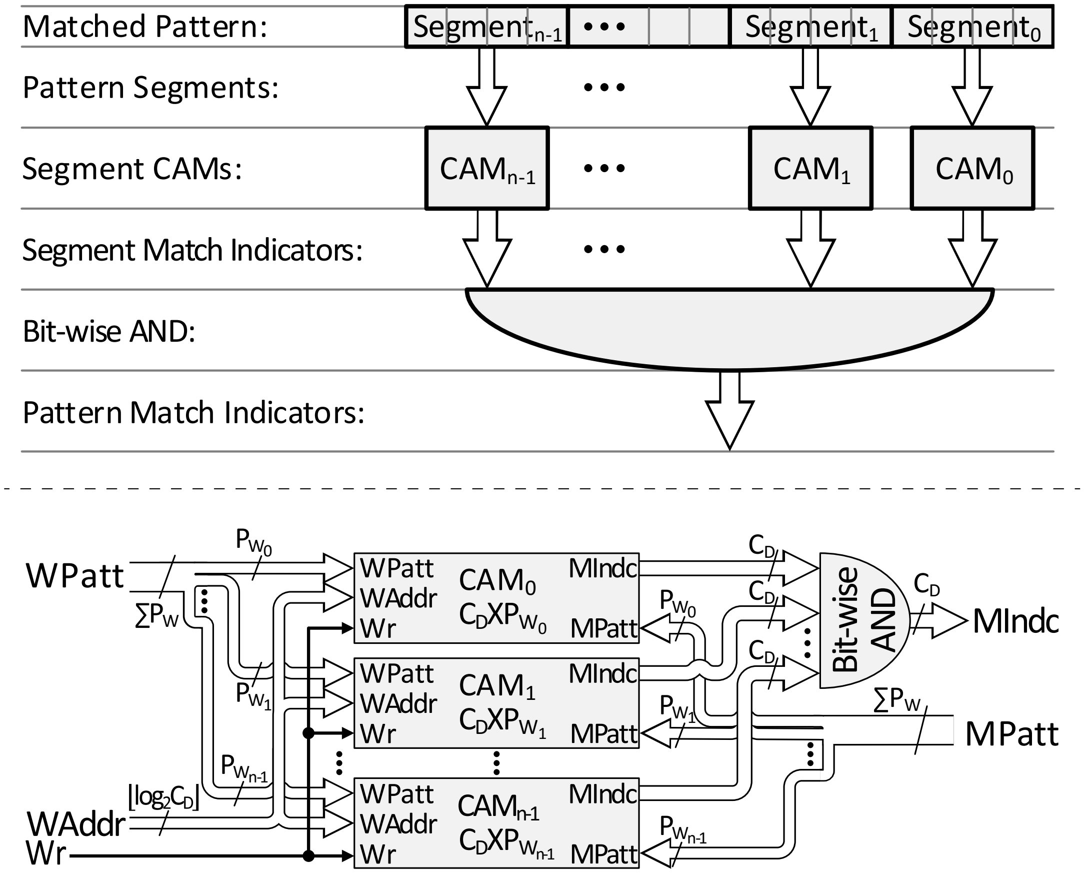


Figure 2.12: BCAM pattern width cascading: (top) abstraction, (bottom) details.

2.3.4 Reconfiguration Memory Based Content-Addressable Memories (RCAMs)

FPGA configuration memory is an SRAM chain loaded with the configuration bit-stream via a serial link and is used to configure the device functionality, mainly routing and logic (LUT) functionality. A segment of this configuration chain is shown in Figure 2.13, where it used to configure a 4-input LUT. A modern FPGA device accommodates several Mbits of configuration SRAM cells, for instance, Altera's Stratix V E device contains 22Mbits of configuration bits for LUTs only [55].

The SRAM reconfiguration memory in FPGAs can be utilized as a wide and shallow memory to generate Reconfiguration Memory Based Content-Addressable Memories (RCAMs). As depicted in Figure 2.13, a LUT MUX chooses one SRAM cell for each specific input that represents the LUT function for this input. Alternatively, the LUT input is used as a BCAM match pattern, and the LUT configuration SRAM cells indicates for a single CAM address if this specific pattern exists. Hence, a 4-inputs LUT fits a single address BCAM with 4-bits pattern, namely a 1×4 BCAM. The address space can be increased by searching concurrently the same pattern for several BCAMs, i.e., sharing LUT inputs. The pattern width can be increased by cascading BCAM blocks as described in Section 2.3.3.

Due to its inherent nature, a variety of device-dependent RCAM approaches has been proposed [86–92]. All these device-dependent methods utilize Xilinx devices reconfiguration capabilities via JBits [93], a Java based Application Programming Interface (API) to reconfigure Xilinx devices by accessing and modifying the configuration bit-stream. However, RCAM writing requires rewriting the entire bit-stream or parts of it, in case partial reconfiguration is supported. In addition, using LUTs as RCAMs will block logic resources.

On the other hand, Altera's Stratix devices provides full accessibility to LUT configuration memory as SRAM blocks with decoded addresses called MLABs [55]. However, the LUT configuration memory can be used either for LUT configuration or as part of the MLABs. For example, each ALM of the Stratix V device can accommodate a 6-input LUT, hence 64 configuration bits. Each 10 ALMs (a single LAB) creates a simple dual-ported 64 × 10 or 32 × 20 MLAB block. MLABs can be utilized to create BCAMs in a similar method as the reconfiguration memory, although the writing mechanism is different.

Other approaches cascade *logic LUTs* to generate area-efficient CAM [94–97]. However, writing to these CAMs is either not supported or requires rewriting the LUT content. Furthermore, logic resources are limited and incur increased delays.

The RCAM approach suffers from several drawbacks that make it impractical in many cases. The RCAM writing requires rewriting the entire bit-stream or parts of it, in case partial reconfiguration is supported. While some applications require a single-cycle writing capability, RCAMs require several cycles to rewrite the bit-stream, depending on the configuration mechanism. In addition, using LUTs as RCAMs will consume logic resources; this is not the case in BRAM-based BCAMs where the logic resource and the SRAM blocks are separate resources. Finally, RCAMs are device-dependent, due to the usage of internal LUT parameters and configuration. A redesign should be applied for different devices, which requires more engineering effort.

2.3.5 Algorithmic Heuristics

Algorithmically, BCAMs are functionally equivalent to associative array data structures where a set of keys is provided and each key is associated with a specific value. Providing a specific key, an associative array will search for the key and return the respective data associated with this key.

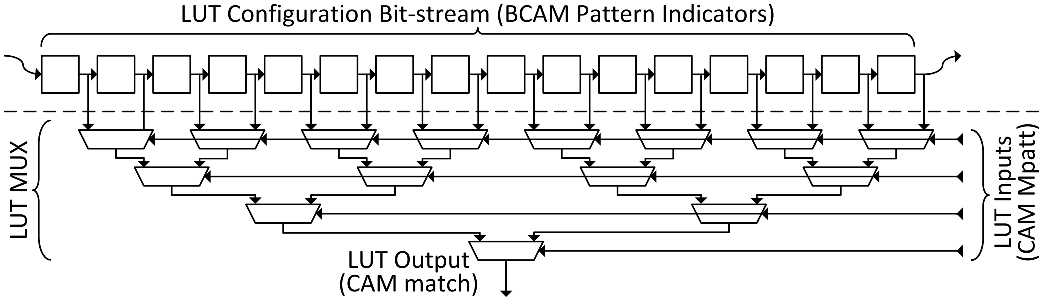


Figure 2.13: FPGA LUT as BCAM.

A linear search or a scan will traverse the memory space sequentially, hence a worst case runtime of $O(n)$. Hash-tables [98] distribute entries across the memory and reduce the average computation into $O(1)$, while the worst case scenario is $O(n)$. Self-balancing or height-balanced Binary Search Tree (BST), e.g., AVL trees and red-black trees [98], can also be used to algorithmically construct associative arrays, with a worst case runtime of $O(\log_n)$.

Algorithmic heuristics CAM emulation for specific applications are widely available, often requiring multiple cycles per lookup. For instance, Trie and BST based IP lookup engines [18–22], Bloom filters [99, 100], and Hash-table based associative arrays [30, 33].

2.3.6 Vendor Support for BCAMs

Modern FPGAs provide plenty of embedded hard-coded blocks, such as Block-RAMs, external memory controllers, processors, DSP blocks/multipliers, and fast I/O transceivers. However, hard CAM blocks do not exist in modern FPGAs presumably due to their high area and power consumption, and their highly specialized nature. While most FPGA vendors provide simple register-based or brute-force

SRAM-based CAMs, some old devices provide partial support for CAM construction. Altera's legacy FLEX, Mercury and APEX [54] device family integrates intrinsic BCAM support into their Embedded System Blocks (ESBs). The ESB can be configured into several modes; a 2Kbits RAM/ROM mode with configurable width and depth, 32 product terms with 32 literal inputs, or a 32×32 BCAM. These BCAM blocks can be used in parallel to increase the address space, and can be cascaded as described in the previous subsection to increase pattern width. Since ESBs are limited to a few hundred blocks in these devices, and due to routing complexity, deep CAMs are infeasible. Furthermore, BCAMs can only be implemented as soft IP in modern Altera devices.

On the other hand, Xilinx devices do not provide native support for BCAMs. However, partial configuration capabilities in Xilinx Virtex devices can be utilized to create a CAM as described in Xilinx application notes [75, 101, 102], however this approach is very slow at writing new patterns. Other Xilinx application notes suggest utilizing the brute-force approach to create BCAMs [75, 76, 78].

Lattice ispXPLD devices [103] have an integrated support for CAMs via their Multi-Function Blocks (MFBs) which can be configured into 128×48 Ternary CAM block (with don't care values). Alternatively, Actel application notes [104] recommend using multi-cycle CAMs by searching BRAM in parallel. For a single-cycle CAM, using registers is suggested.

Chapter 3

Multi-Ported Random Access Memories via Invalidation-Based Live-Value Table (I-LVT)

In this chapter, a novel and modular approach is proposed to construct multi-ported memories out of basic dual-ported RAM blocks. Like other multi-ported RAM designs, each write port uses a different RAM bank and each read port uses replication within a bank. The main contribution of this work is an optimization that merges the previous live-value-table (LVT) and XOR approaches into a common design that uses a generalized, simpler structure we call an invalidation-based live-value-table (I-LVT). Like a regular LVT, the I-LVT determines the correct bank to read from, but it differs in how updates to the table are made; the LVT approach requires multiple write ports, often leading to an area-intensive register-based implementation, while the XOR approach uses wider memories to accommodate the XOR-ed data and suffers from lower clock speeds. Two specific I-LVT implementations are proposed and evaluated, binary and thermometer coding. The I-LVT approach is especially suitable for larger multi-ported RAMs because the table is implemented only in SRAM cells. The I-LVT method gives higher performance while occupying less block RAMs than earlier approaches: for several configurations, the suggested method reduces the block RAM usage by over 44% and improves clock speed by over 76% compared to the best of previous approaches. To assist others, we are releasing our fully parameterized Verilog implementation as an open source hardware library. The library has been extensively tested using ModelSim and Altera's Quartus tools.

3.1 Introduction

In this section, a modular and parametric multi-ported RAM is constructed out of basic dual-ported RAM blocks while keeping minimal area and performance overhead. The suggested method significantly reduces SRAM use and improves performance compared to previous attempts. To verify correctness, the proposed architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II. A large variety of different memory architectures and parameters, e.g., bypassing, memory depth, data width, number of reading or writing ports are simulated in batch, each with over one million random memory access cycles. Stratix V, Altera's high-end performance-oriented FPGA, is used to implement and compare the proposed architecture with previous approaches.

Major contributions of this chapter are:

- A novel I-LVT architecture to produce modular multi-ported SRAM-based memories. It is built out of dual-ported SRAM blocks only, without any register-based memories. To the authors' best knowledge, compared to other multi-ported approaches, the I-LVT consumes the fewest possible SRAM cells. It also provides improved overall performance.
- A fully parameterized Verilog implementation of the suggested methods, together with previous approaches is released as an open source hardware library. A flow manager to simulate and synthesize various designs with various parameters in batch using Altera's ModelSim and Quartus II is also provided. The Verilog modules and the flow manager are available online [105].

The rest of this section is organized as follows. The proposed invalidation-based live-value-table method is described in detail and compared to previous methods in Section 3.2. The experimental framework, including simulation and synthesis and results, are discussed in Section 3.3, and conclusions are drawn in Section 3.4.

3.2 Invalidation Table

As described in Section 2.2.3, to build an MPRAM, the XOR-based approach requires $n_W \cdot (n_R + n_W - 1)$ encoded copies of the RAM content, while the LVT approach requires another register-based multi-ported memory with the same number of read and write ports for bank IDs.

This work proposes to implement LVTs using SRAM blocks only, which has a major scaling advantage over register-based LVTs and a lower SRAM area compared to the XOR-based approach. Instead of requiring multiple write ports to each multi-read bank in the regular LVT method, we suggest a design with a single write port each like the XOR method. This makes it feasible to implement the LVT using standard dual-ported RAMs. However, writing an ID to one bank requires also invalidating the IDs in the other banks, which produces the need for the multiple write ports. Instead, we suggest writing an ID to only one specific bank and invalidating all the other IDs with a single write by using an *invalidation table*. Since the *invalidation table* has the same functional behavior as an LVT, we call it an invalidation-based LVT, or I-LVT.

The I-LVT doesn't require multiple writes to indicate the last-written bank. Instead, as shown in Figure 3.1, the I-LVT reads all other bank IDs as feedback, embeds the new bank ID into the other values through a feedback function f_{fb}, then rewrites the specific bank. To extract back the latest written bank ID, all banks are read and data is processed with the output function f_{out} to regenerate the required ID. Selection of the f_{fb} and f_{out} functions is what distinguishes different I-LVT implementations.

The I-LVT requires n_W multi-read banks, each with n_R read ports for output extraction. Furthermore, an additional $n_W - 1$ read ports are required in each bank for feedback rewriting. The data width of these read ports varies depending on the feedback method and the bank ID encoding. In this section, two bank ID encoding methods are presented, binary and thermometer. The binary method employs exclusive-OR functions to embed the bank IDs, while the second uses mutually-exclusive conditions to invalidate table entries and generate one-hot-coded bank selectors. The two methods are described in Section 3.2.1 and Section 3.2.2, respectively.

3.2.1 Bank ID Embedding: Binary-Coded Bank IDs and Selectors

This approach attempts to reduce the SRAM cell count in the I-LVT by employing binary-coded bank IDs. The special properties of the exclusive-OR function are utilized to embed the latest written bank ID, hence invalidating all other IDs. The current written bank ID is XOR'ed with the content of all the other banks from the same write address as described in the following feedback function,

$$f_{fb,k} = k \bigoplus_{0 \leq i < n_W ; i \neq k} Bank_i[WAddr_k], \tag{3.1}$$

where k is the ID of the currently written bank.

Similar to the XOR-based method described in Section 2.2.3, the last written bank ID is extracted by XOR'ing the content of all the banks from the same read address as described in the following output extraction function

$$f_{out,k} = \bigoplus_{0 \leq i < n_W} Bank_i[RAddr_k]. \tag{3.2}$$

Without loss of generality, if address A in bank k is written with the feedback

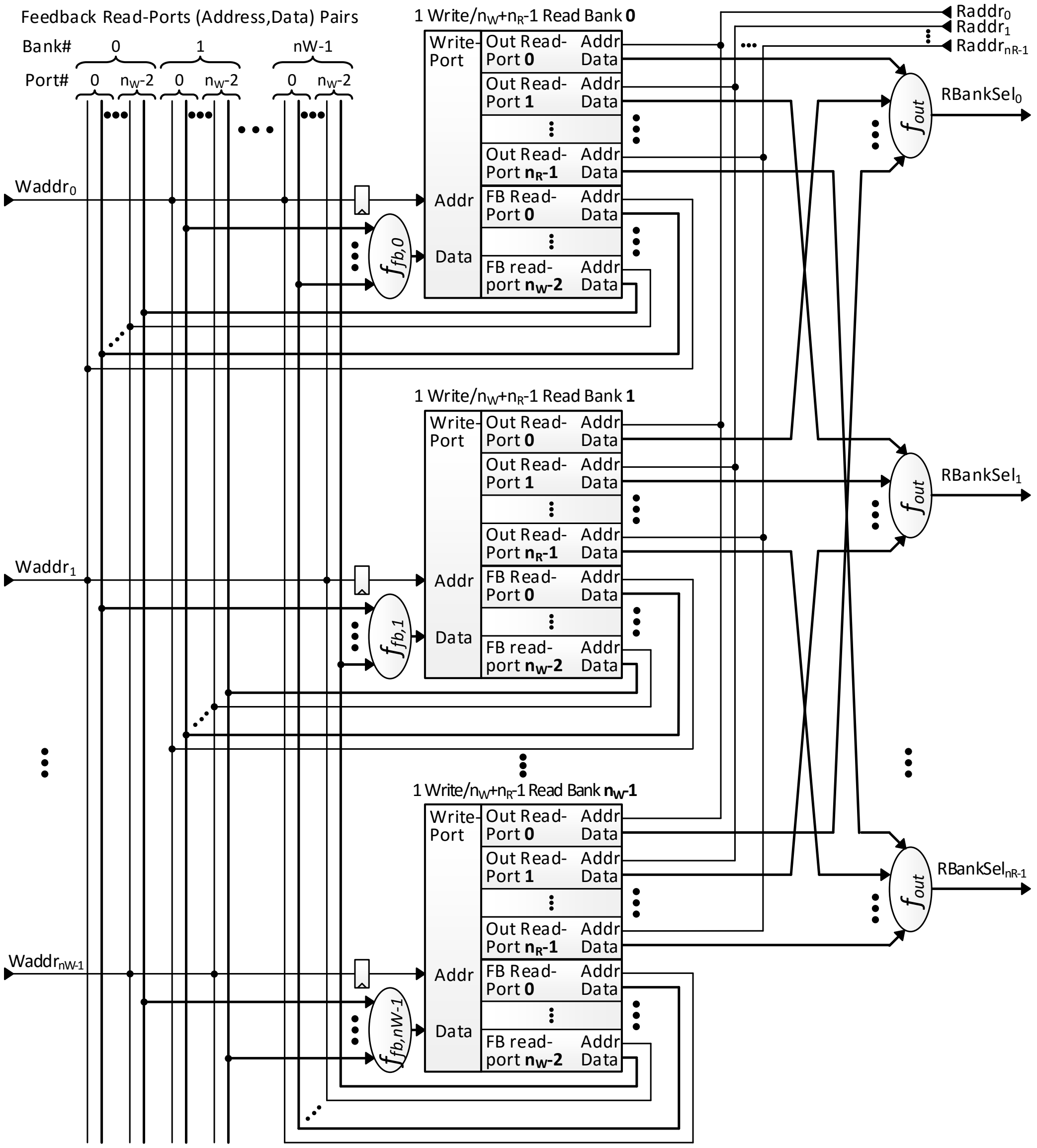


Figure 3.1: Generalized approach for building the I-LVT.

function from Equation 3.1, then

$$Bank_k[A] = k \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A]. \tag{3.3}$$

If one of the read ports, say read port r, is trying to read from the same address, namely $RAddr_r = A$, then the read bank selector will be generated using the same output extraction function from Equation 3.2, hence

$$RBankSel_r = \bigoplus_{0 \le i < n_W} Bank_i[A]. \tag{3.4}$$

Due to XOR operation associativity, $RBankSel_r$ from Equation 3.4 can be expressed as

$$RBankSel_r = Bank_k[A] \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A], \tag{3.5}$$

Substituting $Bank_k[A]$ from Equation 3.3 into Equation 3.5 provides

$$RBankSel_r = k \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A] \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A]. \tag{3.6}$$

The last two series in Equation 3.6 can be reduced revealing that $RBankSel_r = k$, the ID of the latest writing bank into address A, as required.

Figure 3.2 provides an example of a 2W/2R binary-coded I-LVT. As will become apparent in the next section, in case of 2 write ports only, the binary-coded and thermometer-coded I-LVTs are identical. Figure 3.3 shows a 3W/2R binary-coded I-LVT.

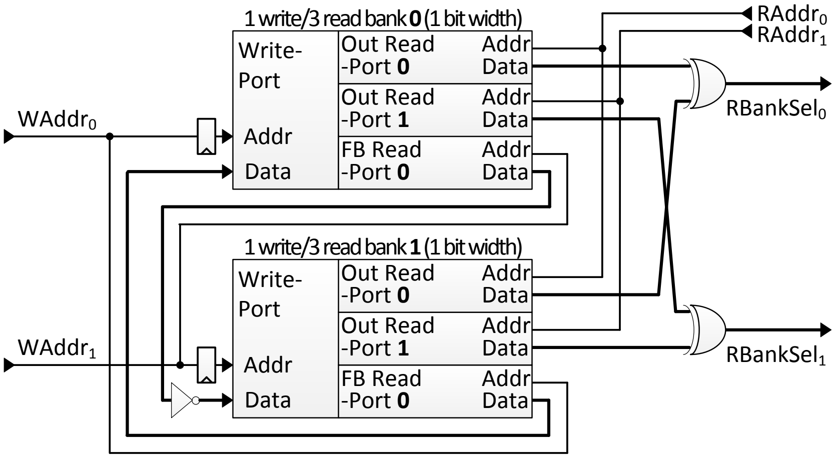


Figure 3.2: A 2W/2R SRAM-based I-LVT; identical for binary-coded or thermometer-coded bank IDs.

The required data width of the I-LVT SRAM blocks is $\lceil \log_2 n_W \rceil$. Also, n_W multi-read banks are required each with n_R output ports for ID extraction and $n_W - 1$ feedback ports for ID rewriting. Hence, the number of required SRAM cells is

$$d \cdot \lceil \log_2 n_W \rceil \cdot n_W \cdot (n_W + n_R - 1). \tag{3.7}$$

Hence, the number of required M20K block RAMs is

$$n_{M20K}(d, \lceil \log_2 n_W \rceil) \cdot n_W \cdot (n_W + n_R - 1). \tag{3.8}$$

Similarly, the number of registers required for retiming is

$$n_W \cdot (\lceil \log_2 d \rceil + 1). \tag{3.9}$$

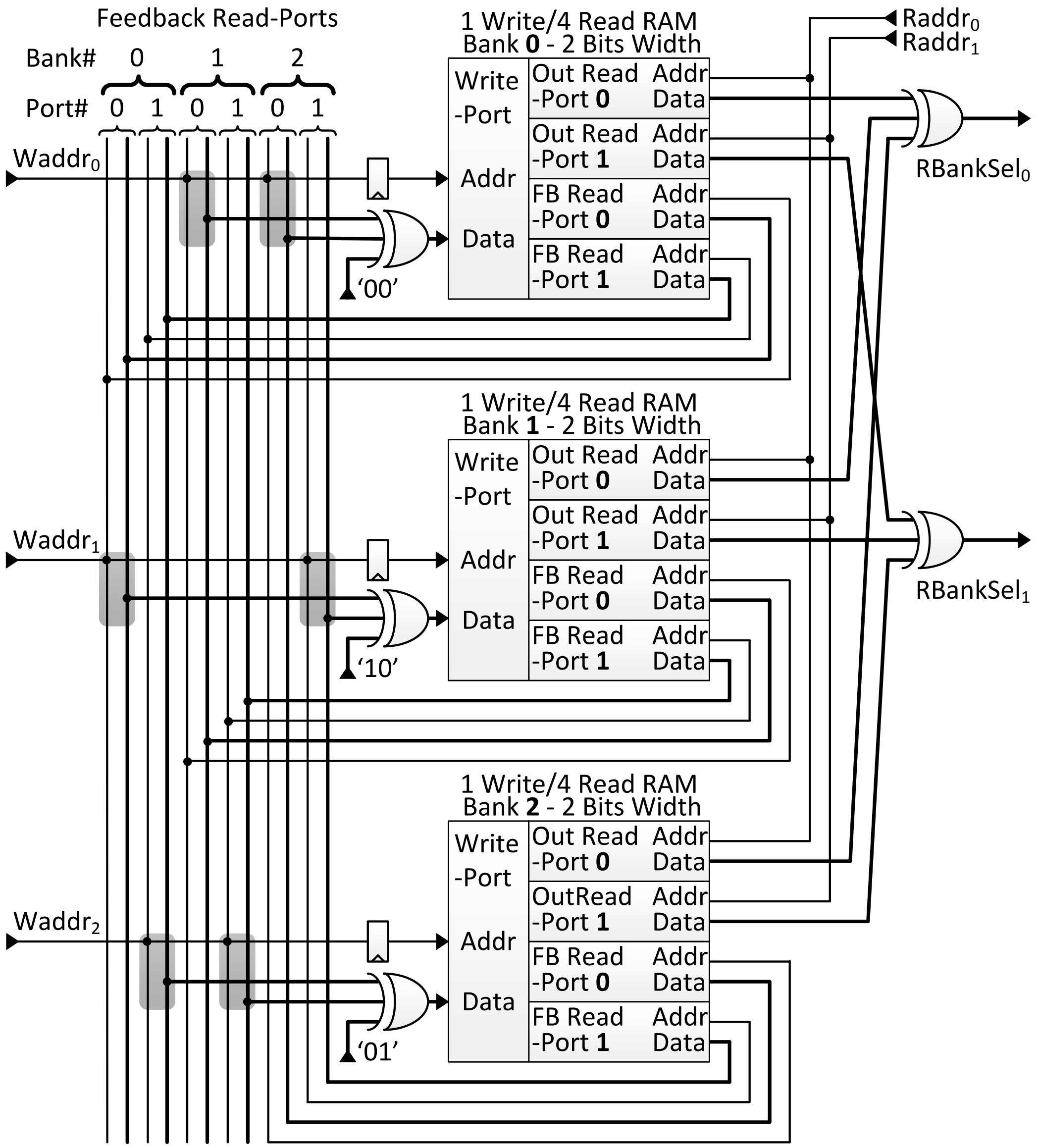


Figure 3.3: A 3W/2R SRAM-based I-LVT with binary-coded bank IDs.

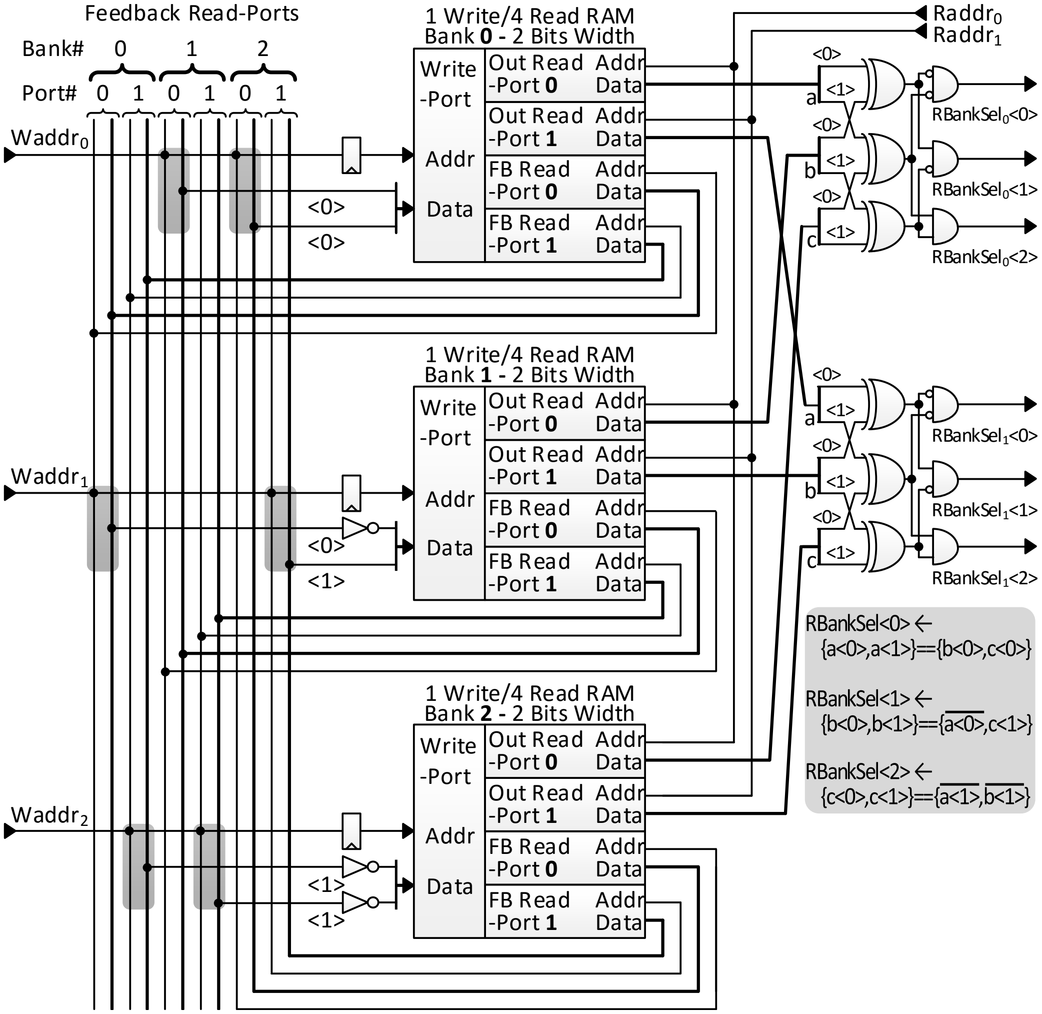


Figure 3.4: A 3W/2R SRAM-based I-LVT with thermometer-coded bank IDs.

3.2.2 Mutually-Exclusive Conditions: Thermometer-Coded Bank IDs with One-hot-Coded Selectors

The previous binary-coded I-LVT incurs a long path delay through the feedback and output extraction functions due to the n_W-wide XOR gates used to generate these functions, which causes a performance reduction in structures with more ports. On the other hand, employing a thermometer ID encoding reduces the feedback paths to a single inverter at most, compared to the n_W-wide XOR used earlier.

Mutually-exclusive conditions are used to rewrite the RAM contents. A specific bank is written data that contradicts all the other banks, hence only this specific bank will be valid and all the others are invalid. By checking the appropriate mutually-exclusive condition for each bank, only the latest written bank will hold the valid data.

Equation 3.10, Equation 3.11 and Equation 3.12 describe mutually-exclusive feedback functions for n_W values 1, 2, and 3, respectively. These feedback functions are also illustrated in Figure 3.5. The angle brackets in theses equations are used for bit selection and concatenation, while the square brackets in other equations are used for RAM addressing. As can be seen from these equations, writing to one bank will invalidate all the other banks at the same address since one mutual negated bit is shared between each two lines. For example, writing to $bank_1$ when $n_W = 3$ (Equation 3.11) will write $Bank_1\langle 0\rangle \leftarrow \overline{Bank_0\langle 0\rangle}$ which will invalidate bank 0, and $Bank_1\langle 1\rangle \leftarrow Bank_2\langle 1\rangle$ which will invalidate $bank_2$.

$$n_W = 2 : \begin{cases} f_{fb,0} : Bank_0\langle 0\rangle \leftarrow Bank_1\langle 0\rangle \\ f_{fb,1} : Bank_1\langle 0\rangle \leftarrow \overline{Bank_0\langle 0\rangle} \end{cases} \tag{3.10}$$

$$n_W = 3 : \begin{cases} f_{fb,0} : Bank_0\langle 1:0\rangle \leftarrow \langle Bank_2\langle 0\rangle, Bank_1\langle 0\rangle\rangle \\ f_{fb,1} : Bank_1\langle 1:0\rangle \leftarrow \langle Bank_2\langle 1\rangle, \overline{Bank_0\langle 0\rangle}\rangle \\ f_{fb,2} : Bank_2\langle 1:0\rangle \leftarrow \langle \overline{Bank_1\langle 1\rangle}, \overline{Bank_0\langle 1\rangle}\rangle \end{cases} \tag{3.11}$$

$$n_W = 4 : \begin{cases} f_{fb,0} : Bank_0\langle 2:0\rangle \leftarrow \langle Bank_3\langle 0\rangle, Bank_2\langle 0\rangle, Bank_1\langle 0\rangle\rangle \\ f_{fb,1} : Bank_1\langle 2:0\rangle \leftarrow \langle Bank_3\langle 1\rangle, Bank_2\langle 1\rangle, \overline{Bank_0\langle 0\rangle}\rangle \\ f_{fb,2} : Bank_2\langle 2:0\rangle \leftarrow \langle Bank_3\langle 2\rangle, \overline{Bank_1\langle 1\rangle}, \overline{Bank_0\langle 1\rangle}\rangle \\ f_{fb,3} : Bank_3\langle 2:0\rangle \leftarrow \langle \overline{Bank_2\langle 2\rangle}, \overline{Bank_1\langle 2\rangle}, \overline{Bank_0\langle 2\rangle}\rangle \end{cases} \tag{3.12}$$

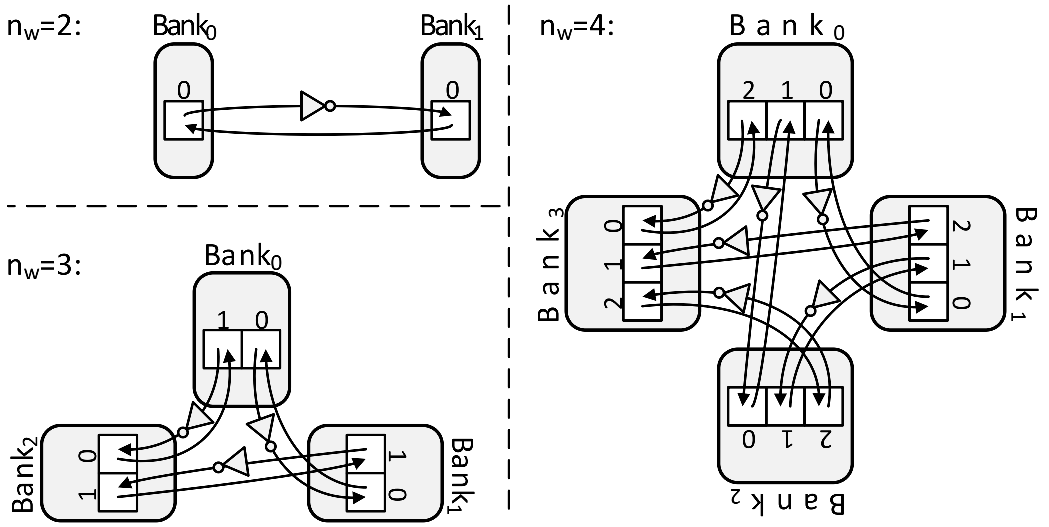


Figure 3.5: Feedback updates for n_W = 2, 3 and 4.

Equation 3.13 generalizes the feedback function to

$$f_{fb,k}\langle i\rangle|_{0\leq i<n_W-1} : Bank_k[WAddr_k]\langle i\rangle \leftarrow \begin{cases} \overline{Bank_i[WAddr_k]\langle k-1\rangle} & i<k \\ Bank_{i+1}[WAddr_k]\langle k\rangle & \text{otherwise} \end{cases} \tag{3.13}$$

This equation shows that each bank is using bits from all other banks to write its own content. To prove that each two banks are mutually exclusive, one bit of these banks should be mutually negated. Suppose $0 \leq k_0 \leq n_W - 1$ a bank ID, and $0 \leq i_0 \leq n_W - 1$ a bit index. From Equation 3.13 if $i_0 \geq k_0$ then another bank ID k_1 and bit index i_1 exist such that $Bank_{k_0}\langle i_0\rangle \leftarrow Bank_{k_1}\langle i_1\rangle$, $k_1 = i_0 + 1$, and $i_1 = k_0$. Hence, $i_1 < k_1$ and from Equation 3.13 $Bank_{k_1}\langle 1\rangle \leftarrow \overline{Bank_{k_0}\langle i_0\rangle}$ as required. The proof in case of $i_0 < k_0$ is identical.

The output extraction function checks for each one-hot output selector if the read data from a specific bank matches the mutually-exclusive case. Hence, only one case will match due to exclusivity. The output extraction function consists of an $n_W - 1$ bit wide comparator for each one-hot selector.

An example of a 2W/2R thermometer-coded I-LVT is shown in Figure 3.2, while a 3W/2R thermometer-coded I-LVT is depicted in Figure 3.4.

The thermometer-coded I-LVT requires $n_W - 1$ SRAM bits to save the mutually exclusive cases. However, the feedback read ports require only one bit, since only one bit is used by the feedback function from each bank. n_W multi-read banks are required each with n_R output ports for one-hot selectors extraction and $n_W - 1$ feedback ports for mutually-exclusive case rewriting. Hence, the number of required SRAM cells is

$$d \cdot (n_W - 1) \cdot n_W \cdot n_R + d \cdot n_W \cdot (n_W - 1). \tag{3.14}$$

Thus, the number of required M20K block RAMs is

$$n_{M20K}(d, n_W - 1) \cdot n_W \cdot n_R + B_{M20K}(d, 1) \cdot n_W \cdot (n_W - 1). \tag{3.15}$$

Similarly, the number of registers required for retiming is equal to the binary-coded case and is described by Equation 3.9.

3.2.3 Data Dependencies and Bypassing

The new I-LVT structure and the previous XOR-based multi-ported RAMs incur some data dependencies due to feedback functions and the latency of reading the I-LVT to decide about the last written bank. Data dependencies can be handled by employing bypassing, also known as forwarding.

Figure 3.6 illustrates two types of bypassing based on write data and address pipelining. Bypassing is necessary because dual-port block RAMs in FPGAs cannot internally forward new data when one port reads and the other port writes the same address on the same clock edge, constituting a read-during-write (RDW) hazard. Both bypassing techniques are functionally equivalent, allowing reading of the data that is being written on the same clock edge, similar to single register functionality. However, the fully-pipelined two-stage bypassing shown in Figure 3.6 (right) can overcome an additional cycle latency. This capability is required if a block RAM has pipelined inputs (e.g., cascaded from another block RAM) that need to be bypassed.

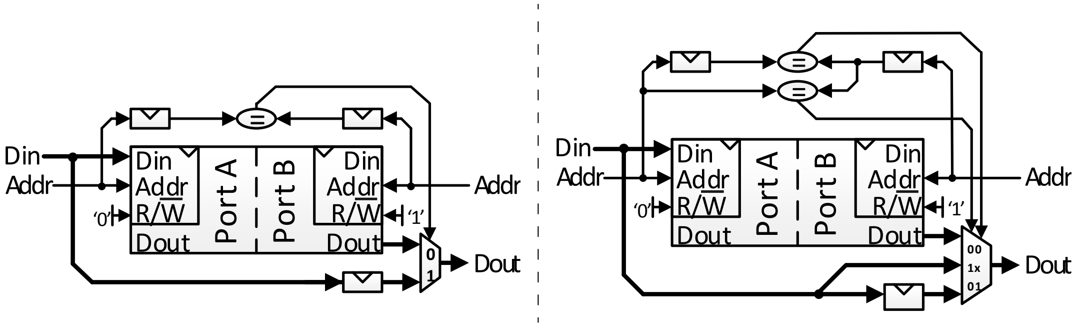


Figure 3.6: RAM bypassing (left) single-stage (right) 2-stages fully pipelined.

The single-stage and the two-stage bypass circuitry for a Block-RAM with w bits data width and d lines depth requires w registers for data bypassing, two $\lceil \log_2 d \rceil$ wide address registers and one enable register, for a total of

$$n_{BypReg}(d,w) = w + 2\lceil \log_2 d \rceil + 1. \tag{3.16}$$

The most severe data dependency the I-LVT design suffers from is Write-After-Write (WAW), namely, writing to the same address that has been written before in the previous cycle. This dependency occurs because of the feedback reading and writing latency. A single-stage bypassing for the feedback banks should solve this dependency.

Two types of reading hazards are introduced by the proposed I-LVT design, Read-After-Write (RAW) and Read-During-Write (RDW). RAW occurs when the same data that have been written in the previous clock edge are read in the current clock edge. RDW occurs when the same data are written and read on the same clock edge.

Due to the latency of the I-LVT, reading from the same address on the next clock edge after writing (RAW) will provide the old data. To read the new data instead, the output banks of the I-LVT should be bypassed by a single-stage bypass to overcome the I-LVT latency.

The deepest bypassing stage is reading new data on the same writing clock edge (RDW), which is similar to single register stage latency. This can be achieved by 2-stage bypassing on the output extract ports of the I-LVT or the XOR-based design to allow reading on the same clock edge. The data banks, which are working in parallel with the I-LVT should also be bypassed by a single-stage bypass to provide new data. Table 3.1 summarizes the required bypassing for data banks, feedback banks and output banks for each type of bypassing of the XOR-based, binary-coded and thermometer-coded I-LVT designs.

Since XOR-based multi-ported RAM requires bypassing for all the $n_W \cdot (n_W + n_R - 1)$ banks to read new data when RAW or RDW, the additional registers required for the bypassing are

$$n_W \cdot (n_W + n_R - 1) \cdot n_{BypReg}(d, w). \tag{3.17}$$

RAW for binary-coded method requires bypassing the I-LVT only. Since the I-LVT is built out of $n_W \cdot (n_W + n_R - 1)$ blocks, each with $\lceil \log_2 n_W \rceil$ bits width data, the following amount of additional registers is required

$$n_W \cdot (n_W + n_R - 1) \cdot n_{BypReg}(d, \lceil \log_2 n_W \rceil). \tag{3.18}$$

RAW for thermometer-coded method requires bypassing the whole I-LVT, $n_W \cdot (n_W - 1)$ feedback banks with 1 bit width and $n_W \cdot n_R$ output banks with $n_W - 1$ bits width, hence a total registers of

$$n_W \cdot (n_W - 1) \cdot n_{BypReg}(d, 1) + n_W \cdot n_R \cdot n_{BypReg}(d, n_W - 1). \tag{3.19}$$

RDW for both binary and thermometer-coded methods require bypassing the $n_W \cdot n_R$ data banks in addition to the I-LVT, hence the following amount of registers is added to the previous count in Equation 3.18 and Equation 3.19

$$n_W \cdot n_R \cdot n_{BypReg}(d, w). \tag{3.20}$$

3.2.4 Initializing Multi-Ported RAM Content

Since some applications require to initializing RAM content to a specific value on power up to enable processing of this initial data during runtime. Dual-ported BRAMs (e.g., Alteras M20K BRAMs [55]) allow initialization with a specific content on power up. However, initializing multi-ported memories requires special handling.

For the XOR-based multi-ported RAM, the first multi-read bank should be

Table 3.1: Bypassing for XOR-based and binary/thermometer-coded I-LVT multi-ported memories.

	XOR-based		I-LVT-based		
	Feedback banks	**Output banks**	**Data banks**	**Feedback banks**	**Output banks**
Allow WAW	1-stage	None	None	1-stage	None
New data RAW	1-stage	1-stage	None	1-stage	1-stage
New data RDW	1-stage	2-stage	1-stage	1-stage	2-stage

initialized to the required initial content; all the other multi-read banks should be initialized to zero.

On the other hand, LVT-based multi-ported memories require storing the initial content into a specific data bank (e.g., Bank 0), then initializing the LVT to the same bank ID (zeros) points to the location of the initial data.

Considering that the initial data will be stored in data bank 0, the thermometer-coded I-LVT-based multi-ported RAM requires initializing all the I-LVT banks with zeros. The binary-coded I-LVT will generate a selector to the first data bank (indexed zero), since XOR'ing all the initial values (zeros) will generate zero.

Similarly, the thermometer-coded I-LVT will be initialized to the first mutually exclusive case, hence the first bank will be selected. Only the first bank holds the initial data; the remaining banks are left uninitialized. The initial values of each bank in the binary/thermometer-coded I-LVT and XOR-based designs are shown in Figure 3.7.

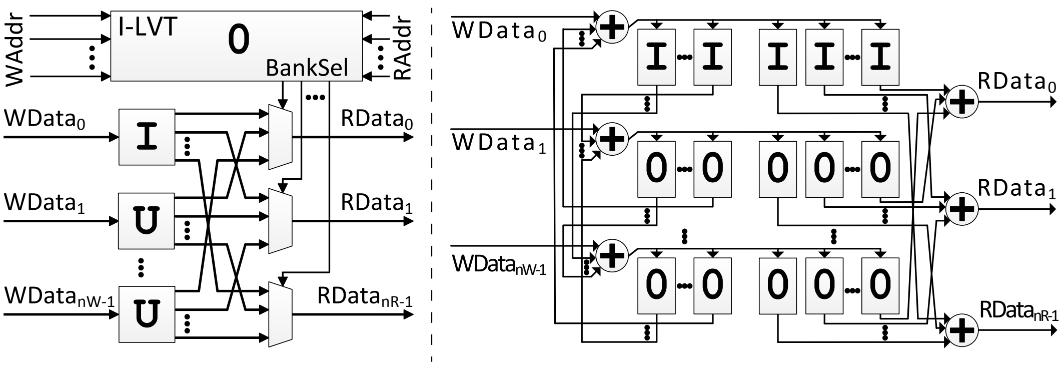


Figure 3.7: Initial value for (left) I-LVT-based (right) XOR-based. (0: zeros, I: initial content, U: uninitialized).

3.2.5 Comparison and Discussion

In this section, we compare SRAM and register consumption of our proposed approaches with previous approaches based on RAM architecture and ports functionality. A usage guideline based on the required RAM parameters is also provided. These analytical results are in agreement with experimental results in Section 3.3.

3.2.5.1 SRAM Usage based on RAM Architecture

In this section, we compare the previous LVT and XOR approaches to the new I-LVT approaches for building multi-port memories. Using the equations provided, we will illustrate why the I-LVT approach is superior in terms of number of BRAMs required, and number of registers required. Also, between the two I-LVT methods proposed, we will inspect the number of BRAMs and registers used by each bypassing method.

Table 3 summarizes SRAM resource usage for each of the three multi-ported RAM approaches: the XOR-based and the binary/thermometer-coded I-LVT. Both the general SRAM cell count and the number of Altera's M20K blocks are described. Comparing the SRAM cell counts, the XOR-based approach consumes fewer SRAM cells than the thermometer-coded I-LVT if

$$w < n_R + 1. \tag{3.21}$$

This inequality is unlikely to be satisfied, since for a single byte data width, the number of reading ports n_R would need to be larger than 8, which is very rare except for systems with multiple requesters requiring a concurrent access to a few bits of data (e.g., mutex or mailbox system). Hence, for typical use cases, the thermometer-coded I-LVT approach will consume fewer SRAM cells.

Comparing the XOR-based approach to the binary-coded approach, the XOR-based approach consumes fewer SRAM cells only if

$$w < \left. \frac{\lceil \log_2 n_W \rceil \cdot (n_W + n_R - 1)}{n_W - 1} \right|_{n_W > 1}. \tag{3.22}$$

Both Equation 3.21 and Equation 3.22 show that the XOR-based approach will consume less SRAM cells only for a very narrow data widths which are uncommonly used. Hence, the I-LVT approach will be the choice for most applications. Comparing the two I-LVT approaches, Table 3.2 shows that the thermometer-coded I-LVT consumes fewer SRAM cells than the binary-coded I-LVT if

$$1 < n_W \leq 3 \;\; \text{OR} \;\; n_R < \left. \frac{(n_W - 1) \cdot (\lceil \log_2 n_W \rceil - 1)}{(n_W - 1) - \lceil \log_2 n_W \rceil} \right|_{n_W > 3}. \tag{3.23}$$

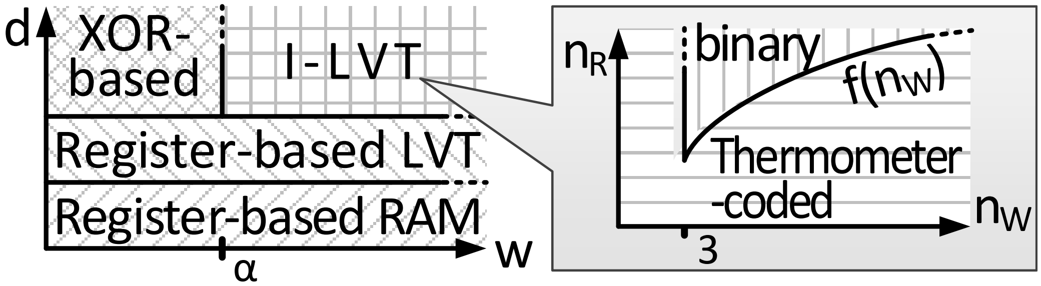


Figure 3.8: Multi-ported RAM usage guideline. α is determined by the minimum of Equation 3.21 and Equation 3.22, while the trend $f(n_W)$ is determined by Equation 3.23.

Figure 3.8 illustrates a guideline for choosing a multi-ported RAM architecture based on area efficiency. For shallow memories, register-based RAM and LVT offer the best area efficiency. However, their area rapidly inflates as RAM goes deeper. The usage of BRAMs alleviates the problem since SRAM-based memories have higher capacities that register-based memories. The XOR-based method is suitable for memories narrower than α which is determined by the minimum of Equation 3.21 and Equation 3.22. Choosing binary-coded or thermometer-coded I-LVT is based on n_W and n_R only and is determined by Equation 3.23.

3.2.5.2 Register Usage based on RAM Architecture

Table 3.4 summarizes register usage for all multi-ported RAM architectures and bypassing. Only the register-based LVT architecture is directly proportional to memory depth. As a consequence, it consumes much more registers than other architectures, making register-based LVTs impractical for deep memories.

With a single-stage bypassing, the XOR-based design consumes fewer registers than the binary-coded if

$$w < \lceil \log_2 n_W \rceil . \tag{3.24}$$

Equation 3.24 is unlikely to be satisfied. Even if the data width is just one byte ($w = 8$), the number of write ports n_W would need to be larger than 256, which is impractical.

On the other hand, with a single-stage bypass, the XOR-based design consumes fewer registers than the thermometer-coded I-LVT design if

$$w < \left. \frac{1+n_R}{1+\frac{n_R}{n_W-1}} \right|_{n_W>1} . \tag{3.25}$$

In a typical compute-oriented design, $n_R = 2 \cdot n_W$. Assuming that $n_R = 2 \cdot (n_W - 1)$ requires that $3 \cdot w - 1 < n_R$; even for a one byte data width, this requires $23 < n_R$ to satisfy Equation 3.25, which is impractical. Therefore, for a single-stage bypass, the I-LVT based designs will consume fewer registers than the XOR-based design.

Considering two-stage bypassing, I-LVT based designs will consume $n_W \cdot n_R \cdot n_{BReg}(d,w)$ more registers, as described in Equation 3.20. In this case, XOR-based design consumes fewer registers than the binary-coded I-LVT design only if

$$w < \lceil \log_2 n_W \rceil \cdot \left(1 + \frac{n_R}{n_W - 1} \right) . \tag{3.26}$$

On the other hand, XOR-based design consumes fewer registers than the thermometer-coded I-LVT design only if

$$w < n_R + 1. \tag{3.27}$$

Similar to Equation 3.21, which is equal to Equation 3.27, this is unlikely to be satisfied in practical designs. Hence, in the case of two-stage bypassing, the I-LVT-based design will consume fewer bypassing registers than the XOR-based method.

In the next sections, we will show these analytical results are in agreement with experimental results.

Table 3.2: Summary of SRAM bits usage.

	Data banks	**LVT feedback banks**	**LVT output banks**
Register-based LVT	$d\ w\ n_W \quad n_R$	N/A	N/A
XOR-based	$d\ w\ n_W\ (n_R + n_W - 1)$	N/A	N/A
Binary-coded I-LVT	$d\ w\ n_W \quad n_R$	$d\ \lceil \log_2 n_W \rceil\ n_W\ (n_W - 1)$	$d\ \lceil \log_2 n_W \rceil\ n_W\ n_R$
Thermometer-coded I-LVT	$d\ w\ n_W \quad n_R$	$d \qquad\qquad n_W\ (n_W - 1)$	$d\ (n_W - 1\ )\ n_W\ n_R$

Table 3.3: Summary of M20K blocks usage[1].

	Data banks	**LVT feedback banks**	**LVT output banks**
Register-based LVT	$n_{M20K}(d,w)\ n_W\quad n_R$	N/A	N/A
XOR-based	$n_{M20K}(d,w)\ n_W\ (n_R+n_W-1)$	N/A	N/A
Binary-coded I-LVT	$n_{M20K}(d,w)\ n_W\quad n_R$	$n_{M20K}(d,\lceil\log_2 n_W\rceil)\ n_W\ (n_W-1)$	$n_{M20K}(d,\lceil\log_2 n_W\rceil)\ n_W\ n_R$
Thermometer-coded I-LVT	$n_{M20K}(d,w)\ n_W\quad n_R$	$n_{M20K}(d,\qquad\qquad)\ n_W\ (n_W-1)$	$n_{M20K}(d,n_W-1\quad)\ n_W\ n_R$

Table 3.4: Summary of register usage[2].

	No bypass	**Additional registers for RAW bypass**	**Additional for RDW**
Register-based LVT	$d \lceil \log_2 n_W \rceil$	None	None
XOR-based	$n_W\ (w + \lceil \log_2 d \rceil + 1)$	$n_W (n_R + n_W - 1) n_{BReg}(d, w \qquad)$	None
Binary-coded I-LVT	$n_W\ (\qquad \lceil \log_2 d \rceil + 1)$	$n_W (n_R + n_W - 1) n_{BReg}(d, \lceil \log_2 n_W \rceil)$	$n_W\ n_R\ n_{BReg}(d, w)$
Thermometer-coded I-LVT	Same as binary-coded	$n_W((n_W - 1)\ n_{BReg}(d,1) + n_R\ n_{BReg}(d, n_W - 1))$	Same as binary-coded

3.3 Experimental Results

In order to verify and simulate the suggested approach and compare to previous attempts, fully parameterized Verilog modules have been developed. Both the previous XOR-based multi-ported RAM method, and the proposed I-LVT method have been implemented. To simulate and synthesize these designs with various parameters in batch using Altera's ModelSim version 10.1e and Quartus II version 14.0, a run-in-batch flow manager has also been developed. The Verilog modules and the flow manager are available online [105]. To verify correctness, the proposed architecture is simulated using Altera's ModelSim. A large variety of different memory architectures and parameters are swept, e.g., bypassing, memory depth, data width, number of ports, and simulated in batch, each with over one million random memory access cycles.

All different multi-ported design modules are implemented using Altera's Quartus II on Altera's Stratix V `5SGXMA5N1F45C1` device. This is a high-performance device with 185k ALMs, 2304 M20K blocks and 1064 I/O pins. We performed a general sweep and test all combinations of the following parameters:

- Writing ports (n_W): 2, 3 and 4 writing ports.
- Reading ports (n_R): 3, 4, 5 and 6 reading ports.
- Memory depth (d): 16 and 32 K-lines.
- Data width (w): 8, 16, and 32 bits.
- Bypassing: No bypassing, RAW and RDW.

Following this, we analyze the full set of results. In this section, we omit many of the in-between settings because they behaved as one might expect to see via interpolation of the endpoints.

Figure 3.9 plots the maximum frequency derived from Altera's Quartus II STA at 0.9V and temperature of 0 °C. The results show a higher F_{max} for binary/thermometer-coded I-LVT compared to the XOR-based approach for all design cases. With three or more writing ports, the thermometer-coded I-LVT supports a higher frequency compared to all other design styles. Compared to the XOR-based approach, the thermometer-coded I-LVT improves F_{max} by 38% on average for all design configurations, while the maximum F_{max} improvement is 76%.

Figure 3.10 (top) plots the number of Altera's M20K blocks used to implement each multi-ported RAM configuration. The proposed binary/thermometer-coded I-LVT consumes the least BRAM blocks in all cases. The average reduction of the best of binary/thermometer-coded I-LVT compared to the XOR-based approach is 19% for all tested design configurations, while it can reach 44% for specific configurations. The difference of consumed Altera's M20Ks between binary-coded I-LVT and thermometer-coded I-LVT is less than 6%. To clarify the difference in BRAM consumption, Figure 3.10 (bottom) shows the percentage of BRAM overhead above the register-based LVT, which uses the fewest possible BRAMS overall. The XOR-based design consumes more BRAMs in all cases, up to twice the BRAMs compared to register-based LVT. On the other hand, I-LVT-based methods consume only 12.5% more BRAMs in the case of 32-bit wide memories.

Figure 3.11 shows the number of ALMs consumed by each design with different bypassing methods. New data RAW bypassing consumes more ALMs than the non-bypassed version due to address comparators and data muxes. On the other hand, new data RDW bypassing requires an additional address comparator and a wider mux; hence it consumes more ALMs than a new data RAW bypass. In all bypass modes, as memory data width goes higher, the XOR-based method consumes more ALMs than the I-LVT methods due to wider XOR gates.

The number of registers required for various designs and bypassing styles is shown in Figure 3.12. The I-LVT-based methods consume fewer registers compared to the XOR-based method for no bypassing or new data RAW bypass. For new data RDW bypass, the I-LVT based methods must bypass the data banks, hence the register consumption goes higher than the XOR-based method. However, the register consumption of the register-based LVT method is the highest overall and can be three orders of magnitude higher since it is directly proportional to memory depth. Furthermore, register-based LVT memories with 4 write ports and over 16k-entries failed to synthesize on our Stratix V with 185k ALMs.

Since the register-based LVT approach is not feasible with the provided deep memory test-cases, the register-based LVT trends are derived analytically from Table 3.2, Table 3.3 and Table 3.4 and not from experimental results. Hence, the register-based LVT trend was added as a reference baseline to Figure 3.10 and Figure 3.12 only.

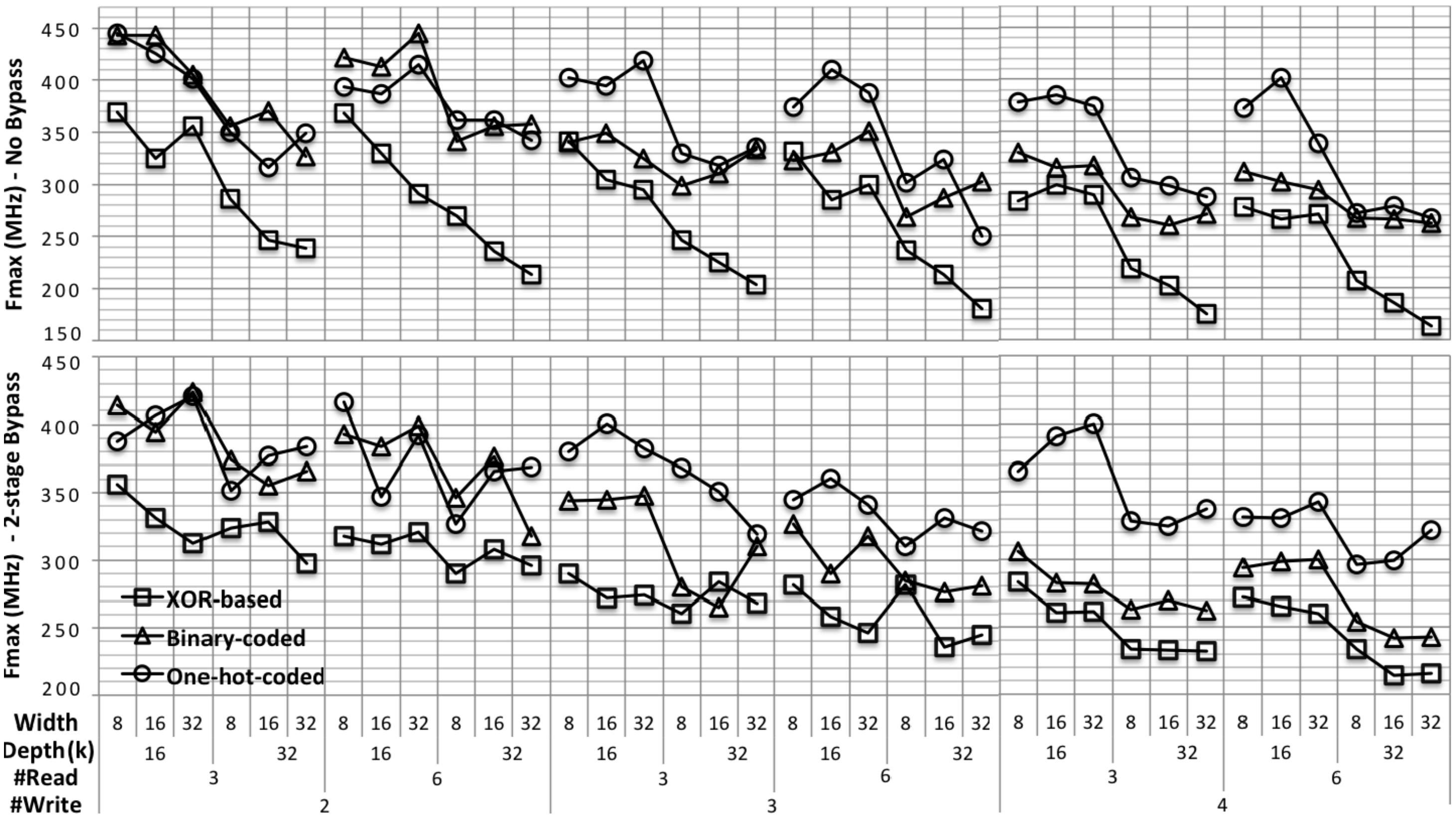


Figure 3.9: Fmax (MHz) T=0C (top) No bypass (bottom) new data RDW bypass.

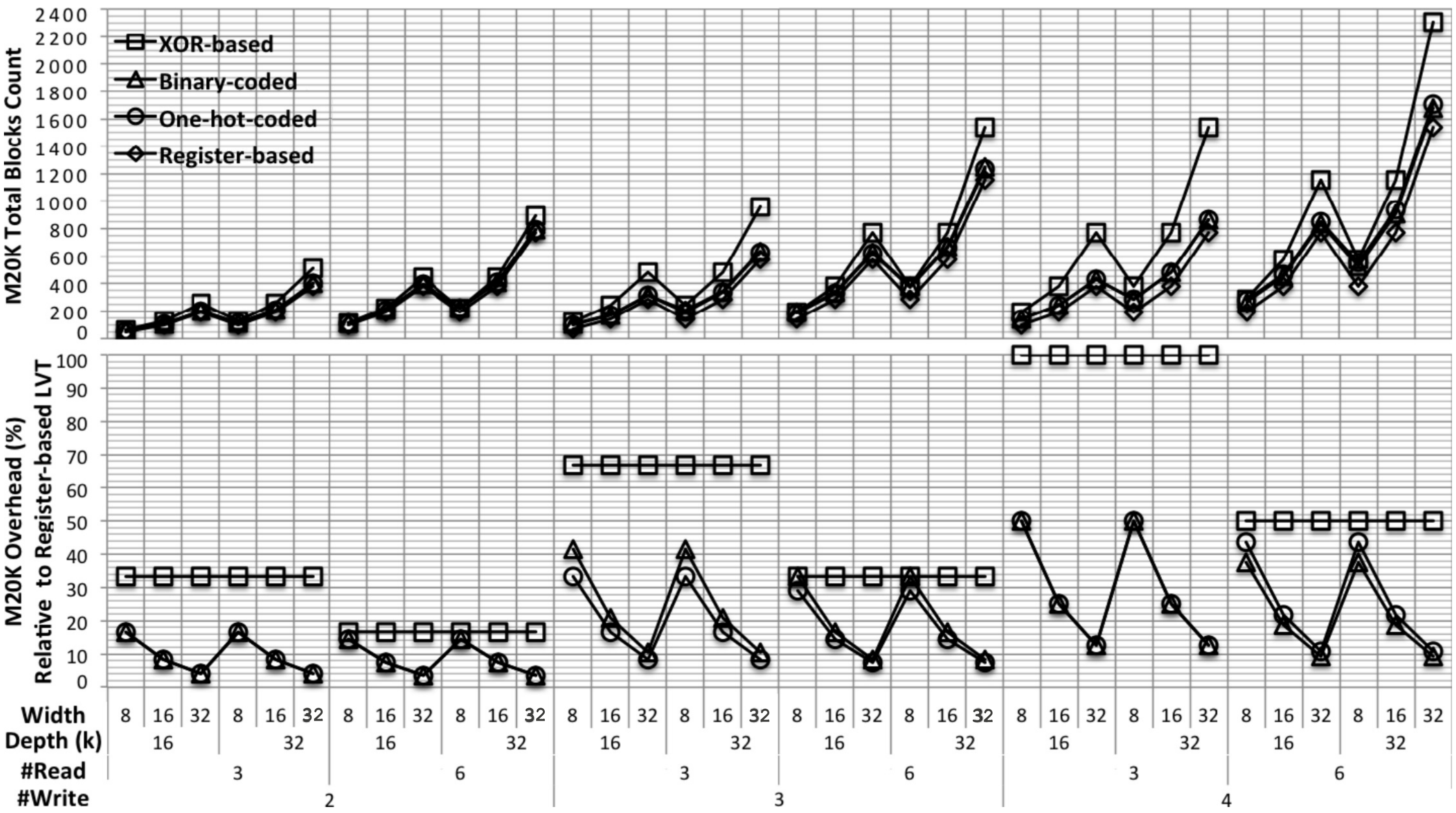


Figure 3.10: M20K blocks (top) total count (bottom) overhead percentage relative to register-based LVT.

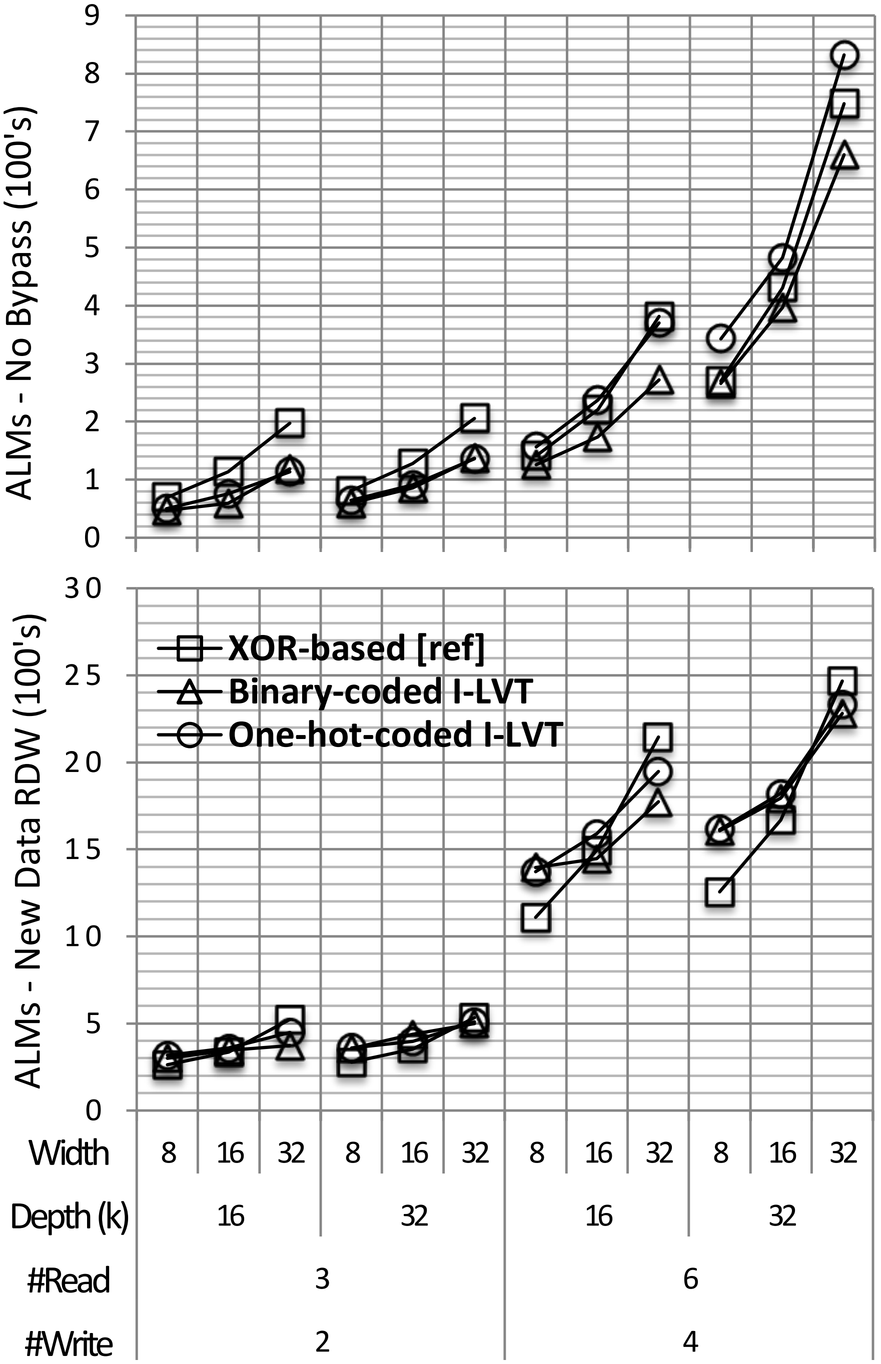


Figure 3.11: ALMs (top) no-bypass (bottom) new data RDW bypass.

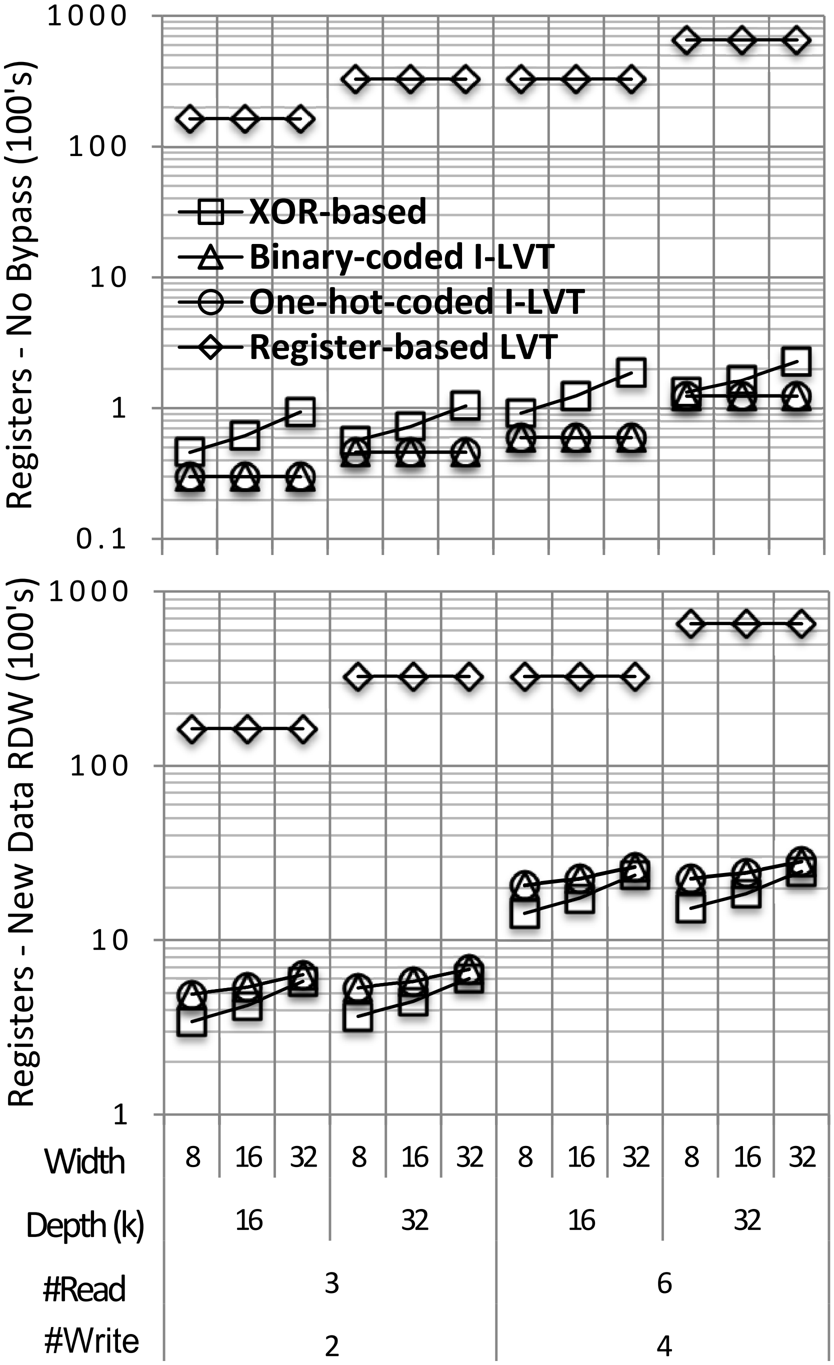


Figure 3.12: Registers count (top) no-bypass (bottom) new data RDW bypass.

3.4 Conclusions

In this chapter, we have proposed the use of an invalidation-based live-value-table, or I-LVT, to build modular SRAM-based multi-ported memories. The Invalidation-Based Live-Value Table (I-LVT) is used to determine the latest written data banks. This generalizes and replaces two prior techniques, the LVT and XOR-based approaches. A general I-LVT is described, along with two specific implementations: binary-coded and thermometer-coded. Both methods are purely SRAM based, hence they scale efficiently with required memory depth. A detailed analysis and comparison of resource consumption of the suggested methods and previous methods is provided. The original LVT approach can use an infeasible number of registers. In contrast, the I-LVT register usage is not directly proportional to memory depth; hence it requires orders of magnitude fewer registers. Furthermore, the proposed I-LVT method can reduce BRAM consumption up to 44% and improve F_{max} by up to 76% compared to the previous XOR-based approach. The thermometer-coded I-LVT method exhibits the highest F_{max}, while keeping BRAM consumption within 6% of the minimal required BRAM count. Meanwhile, the binary-coded I-LVT uses fewer BRAMs than the one-hot coded when there are more than 3 write ports. A detailed analysis and comparison of resource consumption of the suggested methods and previous methods is provided. With this information we develop a guideline for choosing the most area efficient approach. Generally, past approaches of XOR and LVT are only recommended for narrow data widths or shallow depths, respectively. In all other cases, the new I-LVT approaches are superior. A fully parameterized and generic Verilog implementation of the suggested methods is provided as open source hardware [105].

Chapter 4

Multi-Ported Random Access Memories with Switched Ports

Recent attempts to create FPGA-based multi-ported memories suffer from low storage utilization. While most approaches provide simple unidirectional ports with a fixed read or write, others propose true bidirectional ports where each port dynamically switch read and write. True RAM ports are useful for systems with transceivers and provide high RAM flexibility; however, this flexibility incurs high BRAM consumption. In this chapter, a novel, modular and BRAM-based switched multi-ported RAM architecture is proposed. In addition to unidirectional ports with fixed read/write, this switched architecture allows a group of write ports to switch with another group of read ports dynamically, hence altering the number of active ports. Switched ports are a generalization of true ports, where a certain number of write ports can be dynamically switched into a possibly different number of read ports using one common read/write control signal. While a true port is made up of a single read port and a single write port sharing a single read/write control, switched ports are best described as a set. Furthermore, a given application may have multiple sets, each set with a different read/write control. While previous work generates multi-port RAM solutions that contain only true ports, or only simple ports, we contend that using only these two models is too limiting and prevents optimizations from being applied. The proposed switched-ports architecture is less flexible than a true-multi-ported RAM where each port is switched individually. Nevertheless, switched memories can dramatically reduce BRAM consumption compared to true-ports for systems with alternating port requirements. The I-LVT can be used with fixed ports, true-ports or the proposed switched ports architectures.

Most previous work has focused on the design of the LVT to reduce area and improve performance. In this chapter, we instead reduce area by optimizing the design of the data banks portion. The optimization is embedded into a memory compiler that solves a set cover problem; the optimization can only reduce the area of the data banks and never inflate it. When the set cover problem is solved optimally, the data banks use minimum area. Experimental results on 10 random instances of multi-port RAMs show 18% BRAM reduction on average compared to the best of other approaches. In our experiments, the optimization is always able to find an optimal cover and this results in minimum area for the data banks. Formal proofs for the suggested methods, resources consumption analysis, usage guideline and analytic comparison to other methods are provided. The compiler and a fully parameterized Verilog implementation is released as an open source library. The library has been extensively tested using Altera's EDA tools.

The rest of this chapter is organized as follows. In Section 4.1, RAM port classification is provided. Multi-ported memories with single switched port are proposed in Section 4.2. In Section 4.3, the switched ports approach is generalized to support multi-switched-ports. A multi-switched-ports memory compiler automates the generation of switched ports based on user's requirements is described in detail in Section 4.3. Conclusions are drawn in Section 4.4.

4.1 RAM Port Classification

As described in Figure 4.1 (top), RAM ports can be classified into two categories: fixed ports and switched ports.

A *fixed port* is either a *simple read port* or a *simple write port*; the activity of any write is not switched with any other read. Hence, for a set of fixed ports, all writes and reads in the set can be concurrently active.

A *true dual-port*, or simply a *true port*, is a single port that can perform either read or write under the control of a single read/write line. A true port is often drawn as two distinct data ports (one for read, one for write), but they share address lines in common. It is possible (but uncommon) to do a read at the same time as a write to the same address, but the resulting data are implementation-specific.

A *switched port* is a collection or set of read ports and write ports. The number of read and write ports may be different. The read-address lines are usually distinct from the write-address lines. However, the entire set of ports share a single read/write line that controls whether the write ports are active, or the read ports are

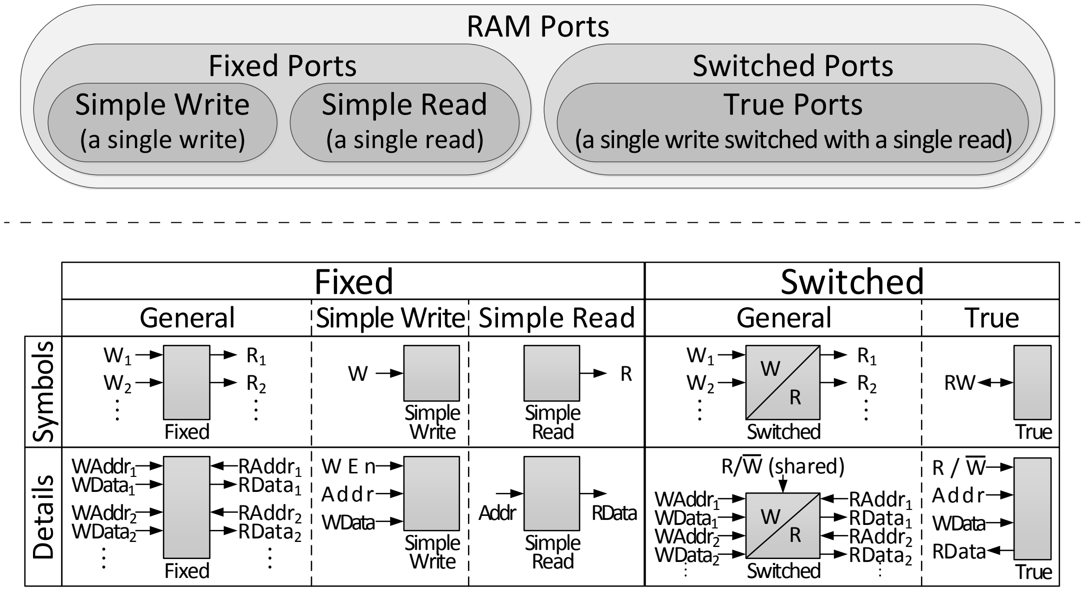


Figure 4.1: RAM port classification: (top) Venn diagram. (bottom) Symbols and port assignment.

active. Reads and writes cannot be simultaneously active. Note that a true port is a special case where a switched port consisting of a single read and a single write, and the address lines are shared. A given application may have multiple switched ports, each with an independent read/write line.

Figure 4.1 (bottom) shows symbols and black box connectivity for these different type of RAM ports. The switched port read/write activity is controlled by a shared $R/\overline{W}$ control signal. Figure 4.2 shows the two modes of a switched port. The first is the write mode where $R/\overline{W} = 0$, writes are active and reads are inactive. The second is the read mode, where $R/\overline{W} = 1$, reads are active and writes are inactive.

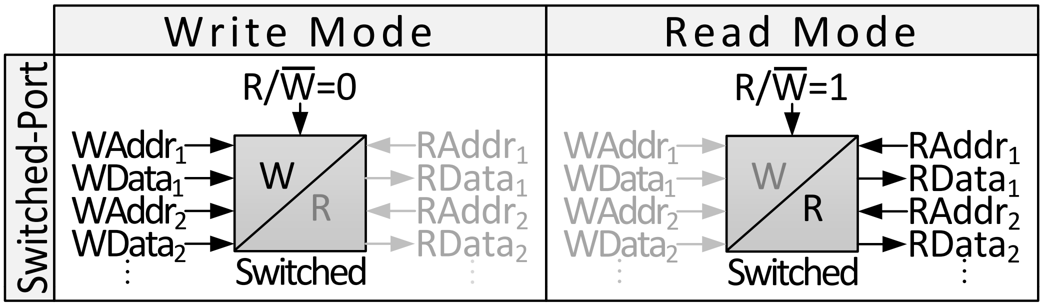


Figure 4.2: Switched-port modes; faint ports are inactive.

4.2 Multi-Ported Memories with Single Switched-Port

In this section, a novel, modular and parametric switched multi-ported RAM is constructed out of basic dual-ported BRAMs while keeping minimal area and performance overhead. The proposed method provides a modular architecture that supports mixed simple/switched port requirements and significantly reduces BRAM consumption and improves performance compared to previous attempts. Despite being less flexible than true RAM ports, switched ports dramatically reduce BRAM consumption if mixed-ports are required. The suggested switched data banks employs an SRAM-based invalidation-live-value-table (I-LVT) [1] to track the latest written data banks for each RAM address, hence this architecture is purely SRAM-based. To verify correctness, the proposed architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II. A large variety of different architectures and parameters, e.g., bypassing, memory depth, width and number of ports are simulated in batch, each with over a million random memory cycles. Stratix V, Altera's high-end performance-oriented

FPGA, is used to implement and compare the proposed approach with previous techniques. In addition to the suggested switched multi-ported RAM architecture, major contributions of this chapter are:

- A bypassing circuitry for both simple (unidirectional) and true (bidirectional) ports. The bypassing circuit is capable of producing new data for Read-After-Write (RAW) and Read-During-Write (RDW) data dependencies.

- A detailed analytic comparison of the proposed and previous methods. A guideline for choosing the most efficient architecture based on design parameters is also provided.

- A fully parameterized Verilog implementation of the suggested methods, together with previous approaches. A flow manager to simulate and synthesize designs with various parameters in batch using Altera's ModelSim and Quartus II is also provided. The Verilog modules and the flow manager are available online [106].

The rest of this section is organized as follows. In Section 4.2.1, multi-ported memories with a single switched port are discussed. Bypassing techniques of these multi-ported memories are discussed in Section 4.2.1.1, whereas Section 4.2.1.2 provides an analysis of BRAM consumption based on port functionality. The experimental framework, including simulation and synthesis results, are discussed in Section 4.2.2.

4.2.1 Single Switched-Port Support

The I-LVT multi-ported RAM, similar to other previous register-based LVT [11] and XOR [12] cancellation methods, offers a fixed number of simple writing and reading ports. However, the vast majority of computation applications use different numbers of reading and writing ports for each computation cycle. On the other hand, Choi *et al.* multi-ported RAM architecture supports bidirectional true-ports only [56]; this excessive port flexibility incurs high BRAM consumption, especially for memories with mixed simple/true-port requirements.

For instance, Figure 4.3 (left) provides an example of a CPU–RAM pairing where the CPU operations are mutual-exclusive; namely, only one operation can be active at a single cycle. The mutual-exclusive operations in Figure 4.3 (left) are: f with 3 operands and 3 results, and g with 6 operands and a single result. when f is active, 3 RAM writes and 3 RAM reads are required, while a single RAM write and 6 RAM reads are required when g is active. At any given cycle, a maximum of 3 writes and 6 reads are required, hence using a multi-ported RAM with fixed ports requires three writing ports ($n_W = 3$) and 6 reading ports ($n_R = 6$). On the other hand, true ports can be configured into writes or reads, hence using true-ports requires the maximum of total writes and reads at any given cycle, namely 7 true-ports ($n_t = 7$). Since RAM ports will not be active at the same cycle, a multi-ported RAM with switched write/read ports as illustrated in Figure 4.3 (right) can be used to reduce SRAM consumption.

The configurability of true dual-ported BRAMs is utilized to construct RAM ports with exchangeable read and write functionality. The proposed switched ports architecture has two sets of ports. The first set is $n_{R,f}$ and $n_{W,f}$ read and write simple (fixed) ports, respectively. As their name suggest, these are fixed simple unidirectional ports. The second set is $n_{R,s}$ and $n_{W,s}$ read and write switched ports, respectively. The functionality of these ports alternates at runtime dynamically in two modes, read and write, as follows. If the write mode is chosen, the $n_{W,s}$ switched write ports are active, while the $n_{R,s}$ read ports are inactive. On the other hand, if the read mode is chosen, the $n_{R,s}$ switched read ports are active, while the $n_{W,s}$ write ports are inactive. The suggested architecture reconfigures dual-ported BRAM write ports into reads when more reads and less writes are required (read mode). As depicted in Figure 4.5 (right), dual-ported BRAMs in switched banks are replicated $n_{R,f}$ times to provide $n_{R,f}$ reads in write mode. Each instance of the $n_{R,f}$ replicas reconfigures its write into a read (in addition to the other read port) in the read mode. Hence, up to $n_{R,f}$ additional switched reads can be generated, namely $n_{R,s} \leq n_{R,f}$.

The key idea behind the SRAM savings is reconfiguring unused writing ports into reading ports. Figure 4.5 illustrates the suggested architecture. Only data banks whose writing ports are unused in read mode are altered, namely $n_{W,s}$ banks. The writing ports of each of these banks are redirected to serve as reading ports in read mode as depicted in Figure 4.5 (lower right). Other banks that will keep writing ports in read mode ($n_{W,f}$ banks) must increase the number of reading ports to $n_{R,f} + n_{R,s}$ to match read ports requirement in read mode as described in Figure 4.5 (upper right). Compared to a fixed ports multi-ported RAM with the maximum number of writing ports $n_W = n_{W,f} + n_{W,s}$ and the maximum number of reading ports $n_R = n_{R,f} + n_{R,s}$, the proposed switched architecture reduces the number of data banks by $n_{W,s} \cdot n_{R,s}$. Hence, this reduces the number of BRAMs by

$$n_{W,s} \cdot n_{R,s} \cdot n_{M20K}(d,w). \tag{4.1}$$

Figure 4.4 describes an example of a switched multi-ported RAM with $(n_{W,f}, n_{R,f}) = (1,3)$ fixed ports and $(n_{W,s}, n_{R,s}) = (2,3)$ switched ports. Therefore, in read mode, there is only one write port and double the fixed read ports. Figure 4.4 (left) shows the write mode configuration, while Figure 4.4 (right) shows the read mode configuration. In this example, the upper multi-read bank keeps the minimal single write operation, while the other banks sacrifice write ports to provide additional read ports. According to Equation 4.1, if the switched RAM in Figure 4.4 has 32-bits in width ($w = 32$) and 32 k-lines in depth ($d = 32k$), 384 BRAM blocks are saved compared to fixed-ports RAM.

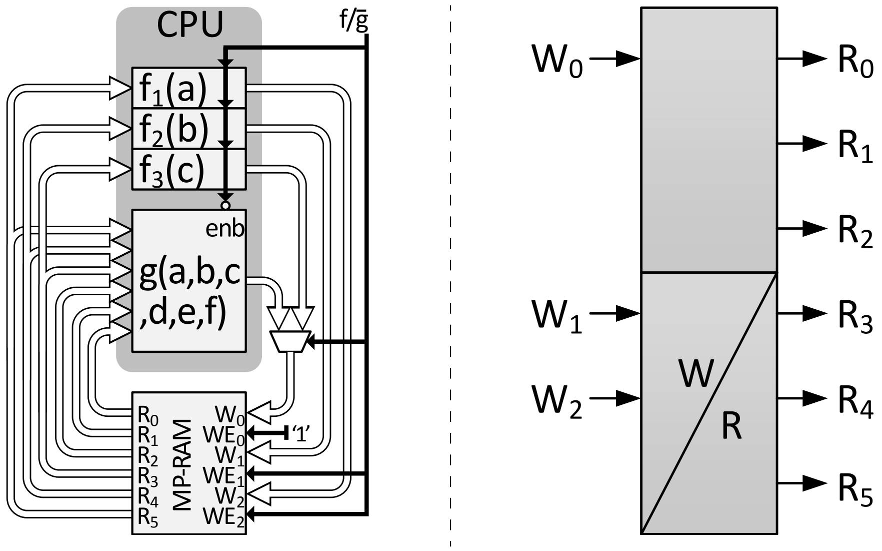


Figure 4.3: (left) Switched ports application example: CPU multi-ported RAM pairing. (right) Symbol of th required switched ports RAM.

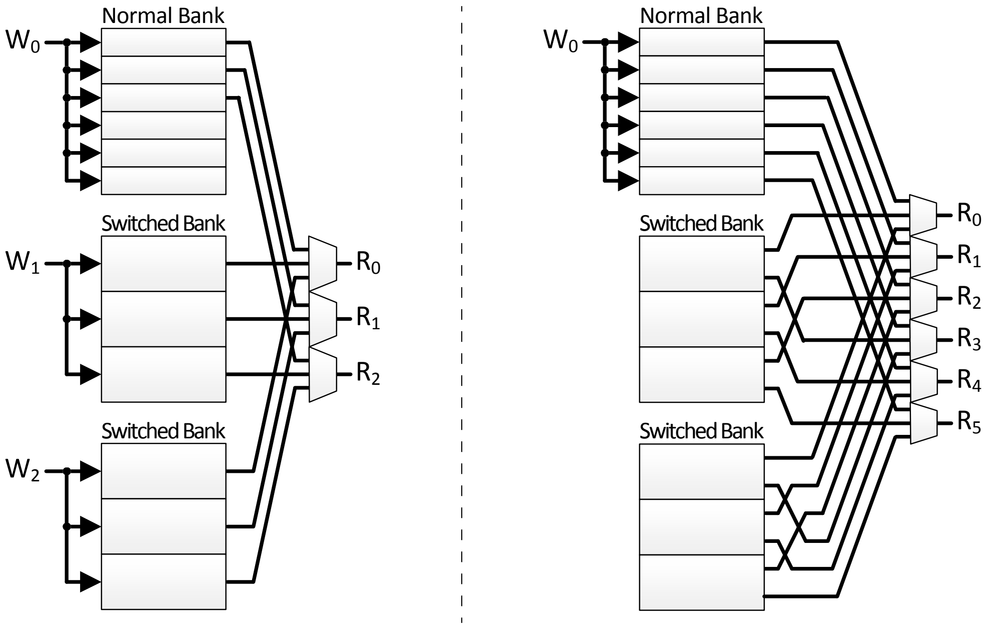


Figure 4.4: switched ports example with $(n_{W,f}, n_{R,f}) = (1,3)$ and $(n_{W,s}, n_{R,s}) = (2,3)$ (left) Write configuration (right) Read configuration.

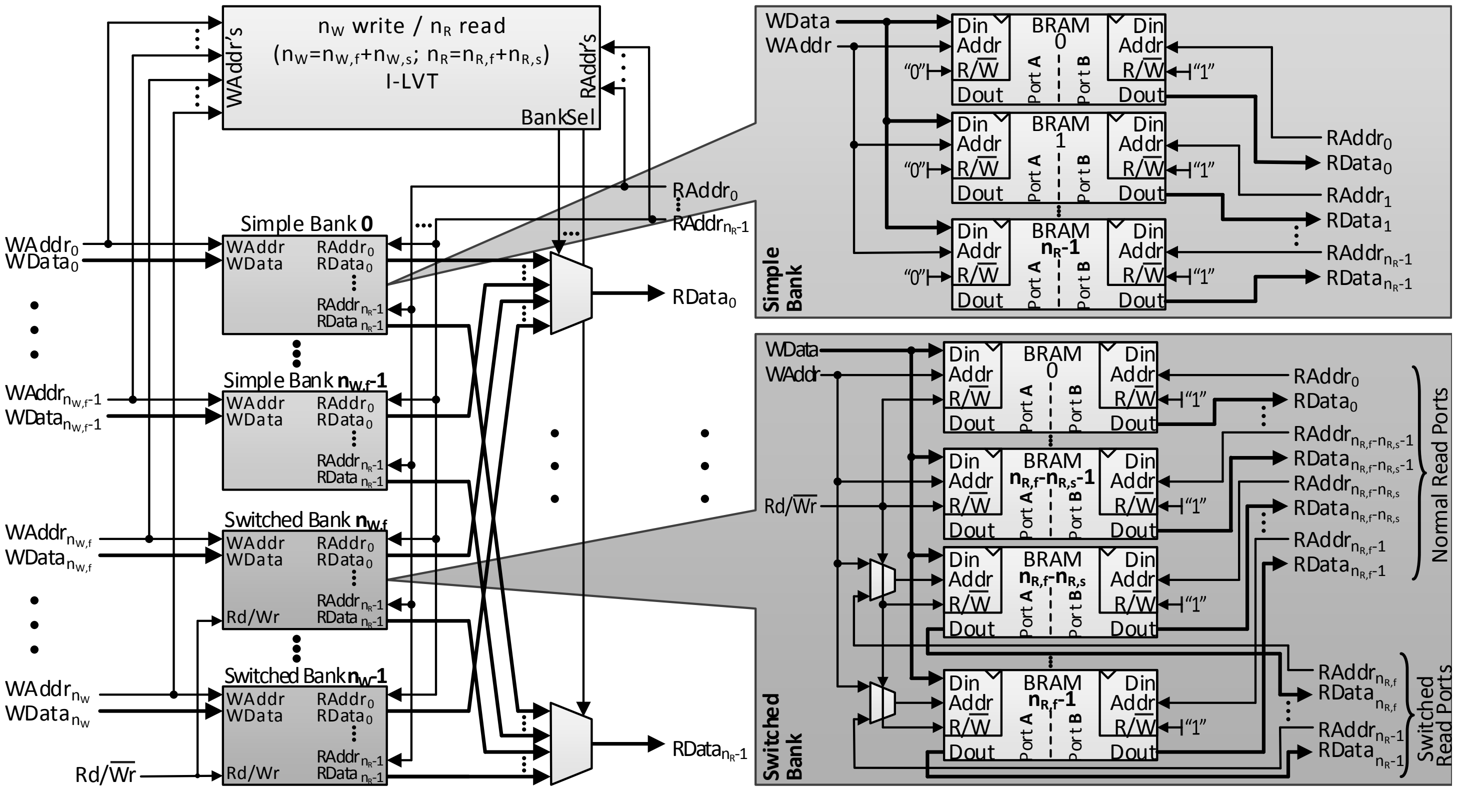


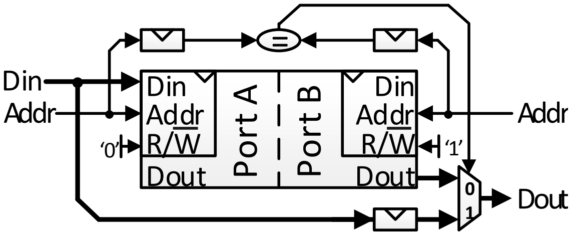
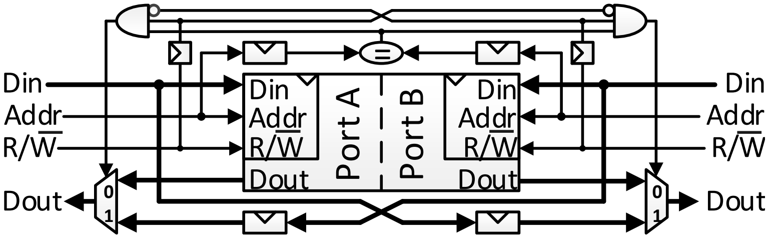
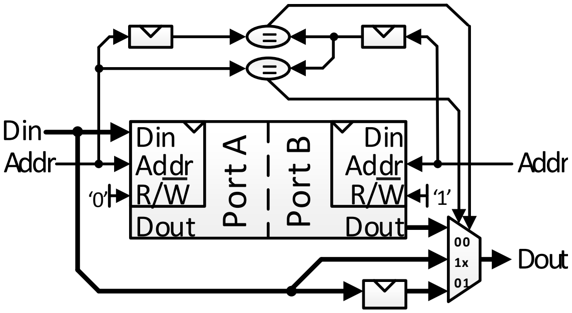
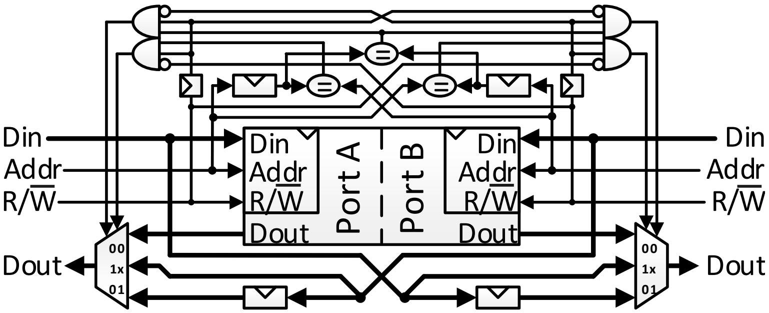
Figure 4.5: (left) Switched ports architecture. Data banks: (upper right) simple, and (lower right) switched.

4.2.1.1 Data Dependencies and Bypassing

The switched multi-ported RAM described in Section 4.2.1 utilizes true-dual-port BRAMs to switch port functionality. However, since writing and reading operations in true-dual-ported RAMs are exchangeable, the bypassing circuitry requires special handling. As described in Table 4.1 (right), the bypass circuit of the bidirectional RAM is mirrored compared to the unidirectional RAM. Thus, it can bypass written data from any direction. However, the control logic that drives the bypassing mux selectors need to be altered to detect the direction of writing. Since the bidirectional bypassing circuit is a mirroring of the unidirectional bypassing circuit, it requires twice the registers used for the unidirectional bypass circuit, hence

$$n_{BReg,bidirectional}(d,w) = 2 \cdot n_{BReg,unidirectional}(d,w). \tag{4.2}$$

Table 4.1: Single-stage and two-stage bypassing for simple and true dual-port RAM.

	Unidirectional (Simple-dual-port)	**Bidirectional (True-dual-port)**
Single-stage	 Din Addr '0' Din Addr R/W Dout Port A Port B Din Addr R/W Dout '1' Addr = 0 1 Dout	 Din Addr R/W Dout Din Addr R/W Dout Port A Port B = 0 1 Din Addr R/W Dout
Two-stage	 Din Addr '0' Din Addr R/W Dout Port A Port B Din Addr R/W Dout '1' Addr = 00 1x 01 Dout	 Din Addr R/W Dout Din Addr R/W Dout Port A Port B = 00 1x 01 Din Addr R/W Dout

4.2.1.2 SRAM Usage based on Port Functionality

The previous analysis provides a guideline for using XOR, LVT, binary-coded or thermometer-coded I-LVT. However, a guideline for using simple, true, or switched port architectures as illustrated in Figure 4.1 and Figure 4.2 is required. A multi-ported RAM with the following mixed port requirements is used for comparison: (1) $n_{W,f}$ write / $n_{R,f}$ read simple (fixed) ports, (2) n_t true-ports, and (3) $n_{W,s}$ write / $n_{R,s}$ read switched ports. To implement the mixed-port multi-ported RAM using different port architectures, the following transformations are required: (1) A true-port can be emulated as two simple ports, a write and a read sharing the same address. (2) A switched port can be emulated as $n_{W,s}$ simple write ports and $n_{R,s}$ simple read ports, or $max(n_{W,s}, n_{R,s})$ true-ports. The M20K count of the mixed-port RAM for each port architecture is provided by:

$$\begin{aligned}
&\text{Simple} &&: \mathrm{n}_{\mathrm{M20K}}(d,w)\Big((n_{R,f}+n_t+n_{R,s})\cdot(n_{W,f}+n_t+n_{W,s})\Big)\\
&\text{True} &&: \mathrm{n}_{\mathrm{M20K}}(d,w)\Big(\tfrac{n_a(n_a-1)}{2}\Big|_{n_a=n_{W,f}+n_{R,f}+n_t+\max(n_{W,s},n_{R,s})}\Big)\\
&\text{Switched} &&: \mathrm{n}_{\mathrm{M20K}}(d,w)\Big((n_{W,f}+n_t+n_{W,s})(n_{R,f}+nt)+(n_{W,f}+n_t)n_{R,s}\Big).
\end{aligned} \tag{4.3}$$

To simplify the comparison, we assume that the number of read ports is twice the number of write ports for simple and switched ports, hence $n_{R,f} = 2n_{W,f}$ and $n_{R,s} = 2n_{W,s}$. Figure 4.6 shows a linear approximation of the BRAM consumption equilibrium. These plots form guidelines for choosing the most area efficient architecture. Figure 4.6 (left) shows that the true-ports architecture is more efficient in BRAM usage only if the number of true-ports is more than $\sqrt{5}$ times the

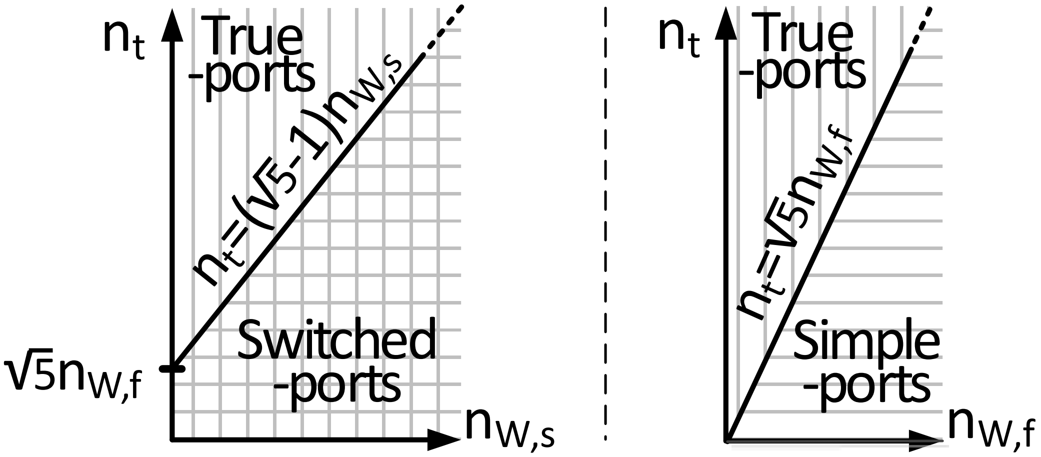


Figure 4.6: Port architecture usage guideline.

simple write ports count. On the other hand, Figure 4.6 (right) shows that the true-ports architecture is more efficient only if the number of true-ports is more than $\sqrt{5}n_{W,f} + (\sqrt{5}-1)n_{W,s}$.

Table 4.2: Resources consumption for a 4W/8R multi-ported RAM test-case with 8k-entries of 32-bit words and new data RDW bypassing. [1] [2].

		Register-based LVT			**Binary-coded I-LVT**			**Thermometer-coded I-LVT**		
	XOR-based	**Simple**	**Switched**	**% Change from XOR**	**Simple**	**Switched**	**% Change from XOR**	**Simple**	**Switched**	**% Change from XOR**
M20K's	704	512	384	-45.5%	556	428	-39.2%	588	460	-34.7%
Registers	2781	18288	17816	+540.6%	3220	2748	-1.2%	3240	2768	-0.5%
ALM's	2010	61662	61333	+2951.4%	1454	1290	-35.8%	1604	1445	-28.1%
F_{max}(Mhz)	270	213	213	-21.1%	325	338	+25.2%	390	388	+43.7%

Table 4.3: Register consumption for a 4W/8R multi-ported RAM test-case with 8k-entries of 32-bit words [3].

		Register-based LVT			**Binary-coded I-LVT**			**Thermometer-coded I-LVT**		
	XOR-based	**Simple**	**Switched**	**% Change from XOR**	**Simple**	**Switched**	**% Change from XOR**	**Simple**	**Switched**	**% Change from XOR**
No Bypass	184	16400	16400	+8813.0%	56	56	-69.6%	56	56	-69.6%
Allow WAW	892	16400	16400	+1738.6%	404	404	-54.7%	392	392	-56.1%
New Data RAW	2780	16400	16400	+489.9%	1332	1307	-53.0%	1352	1352	-51.4%
New Data RDW	2781	18288	17816	+540.6%	3220	2748	-1.2%	3240	2768	-0.5%

4.2.2 Experimental Results

The switched multi-ported RAM is demonstrated with several design cases and nominal parameters of $d = 32K$ and $w = 32$. The total number of write ports is fixed to 3 ($n_W = n_{W,s} + n_{W,f} = 3$), while the number of switched write ports is a sweep of 1, 2 and 3 ports ($n_{W,s} = 1,2,3$); hence the number of simple write ports is a sweep of 2, 1, and 0 ($n_{W,f} = 2,1,0$), respectively. On the other hand, the number of simple read ports is set to 3 ($n_{R,f} = 3$), while the number of switched ports is a sweep of 1, 2, and 3 ports ($n_{R,s} = 1,2,3$); hence the number of total read ports is a sweep of 4, 5, and 6 ($n_R = n_{R,s} + n_{R,f} = 4,5,6$), respectively. Figure 4.7 (top and middle) is a plot of $\mathrm{F_{max}}$ increase and ALMs overhead percentages, respectively, compared to a simple-ports baseline design (without the switched ports mechanism) and with the maximum available ports, namely n_W writing ports and n_R reading ports. Figure 4.7 (bottom) shows the M20K reduction for the equivalent design with simple-ports only (n_W writing ports and n_R reading ports), and for the equivalent design with true-ports only ($n_{W,f} + n_{R,f} + \max(n_{R,s}, n_{W,s})$ true-ports, as described in Section 4.2.1.2). For the given test case, using switched-ports can save up to 45% of the BRAM consumption compared to the equivalent simple-port or true-port designs. The BRAM consumption can be anticipated from Equation 4.1 and Equation 4.3. The BRAM reduction relieves the routing resources hence an $\mathrm{F_{max}}$ increase is observed as shown in Figure 4.7 (top). The $\mathrm{F_{max}}$ increase is more significant in thermometer-based I-LVT, achieving over 22% improvement. Additional data multiplexing causes an increase of ALMs. However, the increase percentage is lower than 31% in all designs as shown in Figure 4.7 (middle).

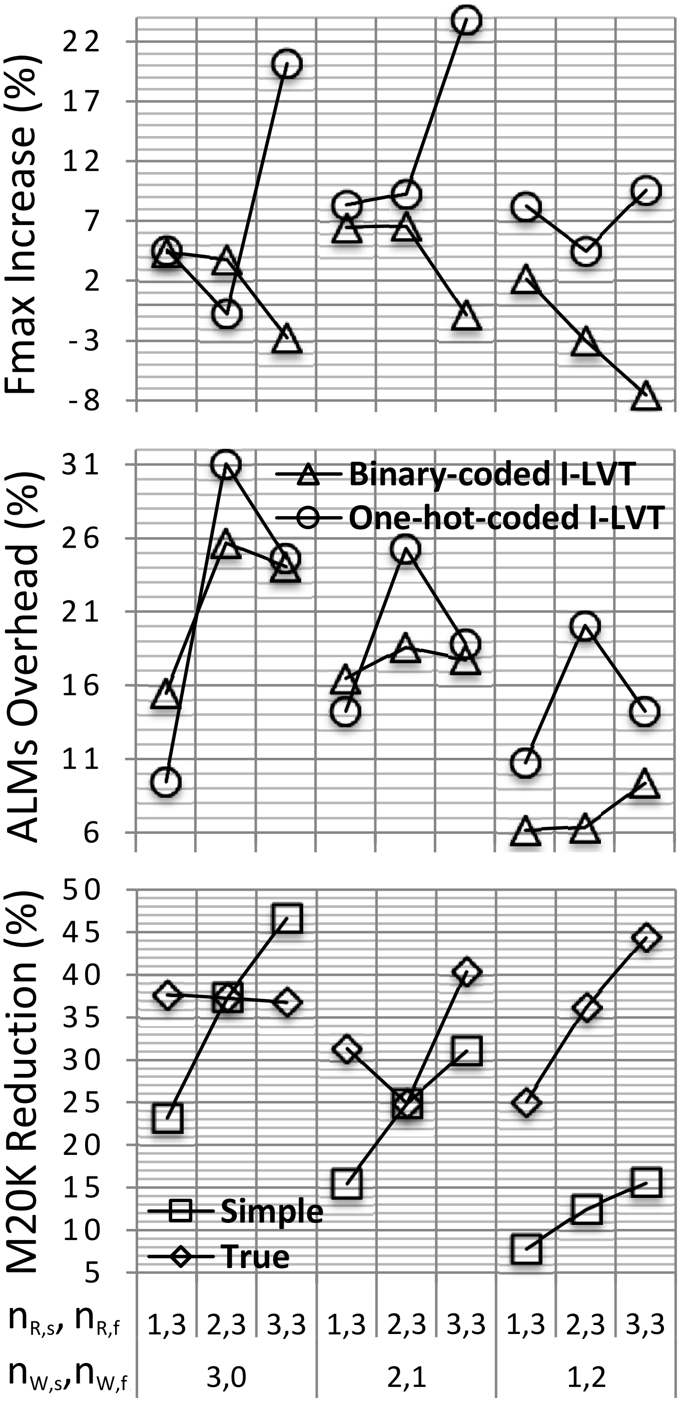


Figure 4.7: Switched-ports compared to simple-ports and true-ports.

4.3 Multi-Switched-Ports

In this section, we demonstrate how to build multi-port RAMs that contain any number of swiched ports. As well, the multi-port RAM may contain any number of simple read ports, simple write ports, and any number of true dual ports (each of the latter will be treated as a switched port). Since a true port is a special case of a switched port, we are effectively generalizing the techniques used in [56] and [2]. Under this new model, we show an optimization method that can minimize the use of block RAMs in the data banks. We also note that for any given problem, there may be multiple solutions. Our solution uses a minimum set cover algorithm to map required read/write dependences into true dual-port block RAMs. In all of our test cases, the set cover problem was solved optimally, leading to the use of minimal block RAMs in the data banks.

This section constructs novel, generic, modular and parametric switched BRAM-based multi-switched-ports RAMs. Compared to previous simple-port and true-port methods, the proposed technique significantly reduces BRAM consumption. The data banks are optimized to support mixed-port configuration by utilizing true-dual-ported BRAMs. The data bank connectivity is converted into a Data Flow Graph (DFG), where vertices represent ports and edges represent data banks, respectively. Each block RAM deployed in the solution will cover one or more DFG edges. An optimal set covering all of the DFG edges results in deploying a minimal number of block RAMs. Our architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II. A large variety of different architectures and parameters, bypassing, depth and width are simulated in batch, each with over a million random memory cycles. Stratix V, Altera's high-end performance-oriented FPGA, is used to implement and compare the proposed approach with previous techniques. A run-in-batch simulation and synthesis flow manager is also provided. The Verilog modules and the flow manager are available online [107].

The rest of this section is organized as follows. The proposed multi-switched-ports synthesis method is described in detail in Section 4.3.1. The experimental framework and results, are discussed in Section 4.3.2.

4.3.1 Multi-Ported RAM with Multiple Switched Ports

In this section, we describe how multiple switched ports may arise, and how to design multi-ported RAMs with them.

4.3.1.1 Motivation and Key Idea

Consider the design of a Processing Element (PE) that has three distinct (non-overlapping) states of operation:

1. Write to shared RAM using 4 write ports
2. Compute value in shared RAM with Arithmetic Logic Unit (ALU) using 2 read ports, 1 write port
3. Read from shared RAM using 4 read ports

In this example, it is necessary to build a shared RAM structure that has multiple read and write ports. There are several approaches to building such a multi-port RAM, including:

A. 4 fixed read ports and 4 fixed write ports

B. 4 true dual-ports

C. two switched ports, with the first having 1 read or 1 write port (ie, a true port), and the second having 3 read or 3 write ports

D. 2 fixed read ports, 1 fixed write port, and one switched port containing 2 read or 3 write ports

In addition, there are many other possible port assignment solutions, where each port assignment may yield a solution requiring a different number of block RAMs. In this dissertation, we do not consider the problem of computing an optimal port assignment; that is left for future work. Instead, we must first be able to compute the block RAM requirements for a given port assignment; that is the purpose of this chapter.

The example above requires port assignment because there are multiple states. Consider simpler problems, such as those in Figure 4.8, where each user of the shared memory is under the control of a single read/write line. In such cases, there exists only one possible port assignment with switched ports.

The one valid port assignment is illustrated on the right-hand side of Figure 4.8 and can broken down as follows. The ALU consists of two mutually-exclusive functional blocks, f and g. First, the control signal $f/\overline{g}$ enables either f or g functional blocks; in either case the R_0 operand must always be read and can be assigned to a fixed port $R_{0,0}$. When f is selected, the R_1 operand can be read from switched read port $R_{2,0}$, but when g is selected we do not need $R_{2,0}$ but instead need switched write ports $W_{2,0}$ and $W_{2,1}$. Thus, $f/\overline{g}$ is also the read/write control signal for the P_2 group of switched ports. Similarly, the bus has a read/write control signal that directly controls the P_1 group of switched ports with $R_{1,0}$ and $W_{1,0}$.

Thus, Figure 4.8 shows two possible implementations of the shared memory: one with all fixed ports on the left, and one with switched ports on the right. This leads to two possible ways to build the shared memory; we will show that the latter way is more efficient in terms of block RAM usage.

The key idea of our methodology is based on constructing a Data Flow Graph (DFG) to describe RAM port dependencies, where vertices represent ports and edges represent data banks. Two types of edges exist: regular (solid) edges for fixed ports, and dashed edges for switchable ports.

The goal is to cover all of these edges, and this cover directly describes an implementation using BRAMs. For example, a single regular edge can be covered by a simple dual-port BRAM with 1 fixed write port and 1 fixed read port (1W/1R). However, true dual-port BRAMs have two terminals on each end (2W/2R), and up to 4 edges between them. Thus, up to 4 edges in the graph can be covered by a true dual-port BRAM. The objective is to cover all edges with minimal BRAMs. This forms a set cover problem (SCP) which can be solved using special subgraph patterns to indicate possible covers.

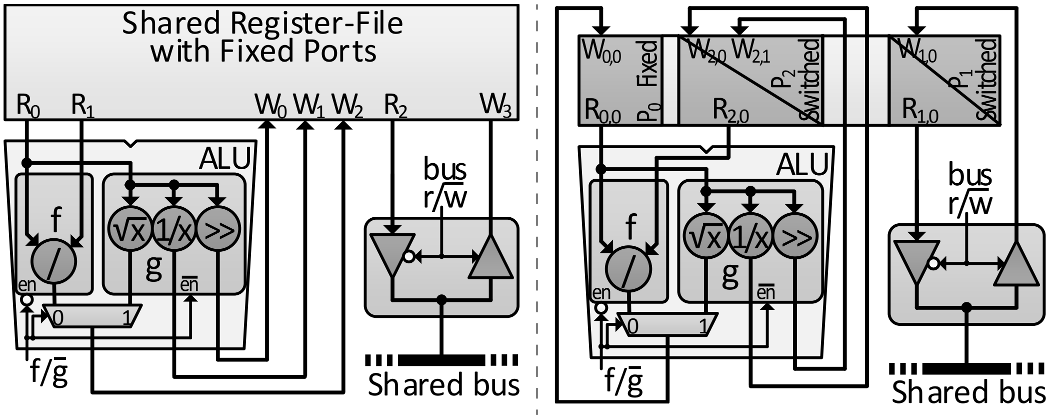


Figure 4.8: Simplified parallel system with shared memory. (left) Multi-ported RAM with fixed ports connection. (right) Multi-switched-ports connection.

4.3.1.2 Port Assignment and Problem Definition

The problem input is a list of port requirements. Unlike switched ports, fixed ports are unrelated and operate individually; hence, they can be aggregated into a single port group (as in Figure 4.8) named P_0. The superset of all port groups, P, includes n_P port groups, were the first port P_0 is a fixed-port and each of the remaining $P_{1\cdots n_{P-1}}$ are switched port groups. Each P_i represents an ordered pair with the number of writes $n_{W,i}$ and the number of reads $n_{R,i}$ for this specific port, namely

$$P = \left\{P_0, P_1, \cdots, P_{n_{P-1}} \mid P_i = \left(n_{W_i}, n_{R_i}\right)\right\}. \tag{4.4}$$

For instance, port requirements for the example in Figure 4.8 is

$$P = \left\{P_0 = (1,1), P_1 = (1,1), P_2 = (2,1)\right\}. \tag{4.5}$$

Writes and reads in this RAM are indexed by two indices, the first index is the port group index ranging $0 \cdots n_P - 1$, while the second index is the write index within a specific port i ranging $0 \cdots n_{W,i} - 1$, or the read index within a specific port i ranging $0 \cdots n_{R,i} - 1$. The writes group W and the reads group R are defined as

$$\begin{aligned} W &= \left\{ W_{i,j} \mid 0 \leq i < n_P;\ 0 \leq j < n_{W,i} \right\} \\ R &= \left\{ R_{i,j} \mid 0 \leq i < n_P;\ 0 \leq j < n_{R,i} \right\} \end{aligned} \tag{4.6}$$

For instance, write and read sets for the example in Figure 4.8 are $W = \left\{ W_{0,0}, W_{1,0}, W_{2,0}, W_{2,1} \right\}$ and $R = \left\{ R_{0,0}, R_{1,0}, R_{2,0} \right\}$, respectively.

The total number of writes and reads is denoted by n_W and n_R (without indices), respectively, namely,

$$\begin{aligned} n_w = |W| &= \sum_{i=0}^{n_P-1} n_{W,i} \\ n_R = |R| &= \sum_{i=0}^{n_P-1} n_{R,i} \end{aligned} \tag{4.7}$$

For instance, the example in Figure 4.8 consists of 4 writing ports and 3 reading ports in total, hence $n_W = 4$ and $n_R = 3$.

Figure 4.9 generalizes the port assignment for the multi-switched-ports RAM. Given these port requirements, and using true-dual-ported BRAMs, the objective of our work is to construct the data banks to satisfy the given fixed and switched ports with the fewest BRAMs.

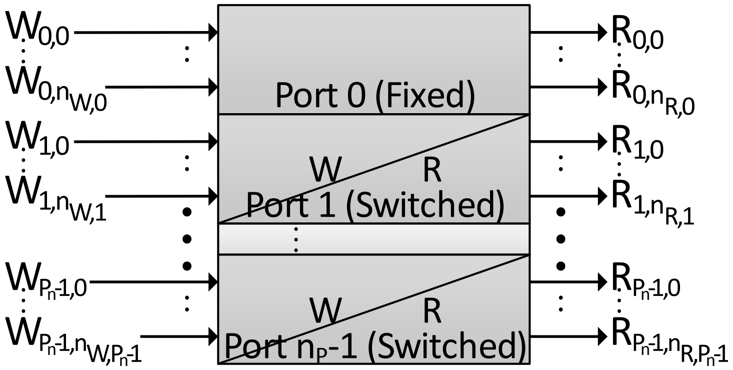


Figure 4.9: Multi-switched-ports RAM port assignment.

4.3.1.3 Modeling Data Banks with Data Flow Graph (DFG)

An LVT-based multi-ported RAM built using only fixed ports requires every write port to write to a dedicated data bank, allowing concurrent writes. Furthermore, every write-specific bank should be accessible by every read port, allowing all read ports to read data written by any write port. This requirement can be modeled as a *complete bipartite graph (complete bigraph)*.

A bigraph is a graph G consisting of two disjoint sets of vertices, say U and V. Each edge in a bigraph connects a vertex from U to another vertex in V. A complete bigraph is a special case where every vertex in U is connected to every vertex in V, in other words

$$G = \Big(U, V, E \mid U \cap V = \emptyset, E = \big\{\{u, v\} \mid u \in U, v \in V\big\}\Big) \tag{4.8}$$

To model data bank connectivity, a bigraph DFG is constructed where source vertices are writing ports $U = W$ and sink vertices are reading ports $V = R$. Graph

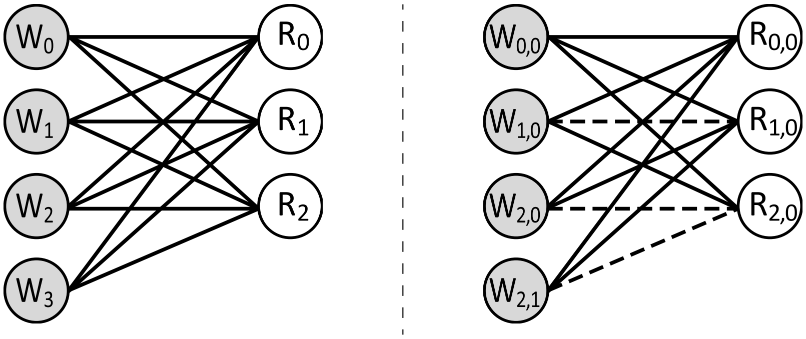


Figure 4.10: Bigraph DFG representing data banks connectivity of the shared memory in Figure 4.8 (left) fixed-ports data banks (right) multi-switched-ports.

edges connect writes to reads, hence they represent 1W/1R simple-dual-port BRAMs. Figure 4.10 (left) shows the bigraph of the fixed-port system in Figure 4.8 (left).

Similarly, Figure 4.10 (right) shows a bigraph of the switched-port system in Figure 4.8 (right). However, the bigraph is slightly different from before; some edges $E_s \subseteq E$ are labeled as switched edges using dashed lines in the figure. Except for port group P_0, which has only fixed ports, the other port groups give rise to switched edges that connect the write vertices to read vertices within the same port group. Formally, the switched edge set is described as follows

$$E_s = \Big\{ \{W_{p,i}, R_{p,j}\} \mid 1 \le p < n_P, 0 \le i < n_{W,p}, 0j < n_{R,p} \Big\}. \tag{4.9}$$

For instance, in Figure 4.10 (right), the switched edges are $E_s = \Big\{ \{W_{1,0}, R_{1,0}\}, \{W_{2,0}, R_{2,0}\}, \{W_{2,0}, R_{2,1}\} \Big\}$.

4.3.1.4 Multi-Switched-Ports DFG Optimization

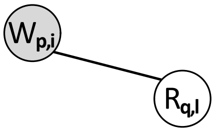
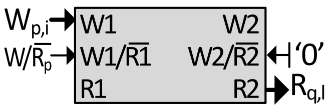
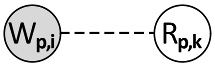
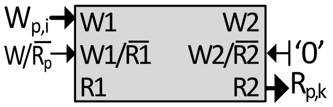
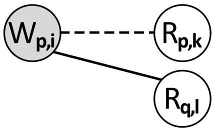
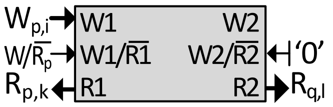
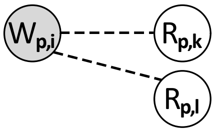
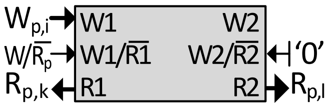
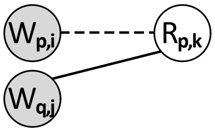
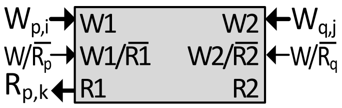
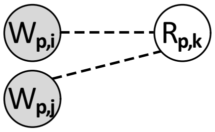
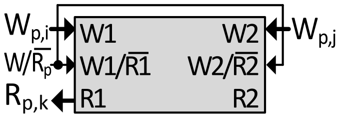
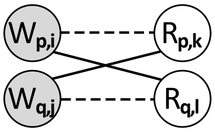
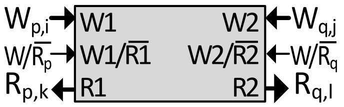
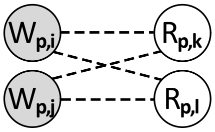
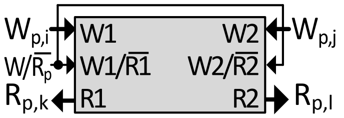
Using a true dual-ported BRAM gives us the ability to cover several possible subgraphs that appear in a bigraph DFG. Since these subgraphs are also complete bigraphs, we call them biclique patterns, or BPs. All different *biclique patterns* are described in Table 4.4.

For each BP in Table 4.4, we can identify several specific instances that it can cover within the biclique DFG from Figure 4.10 (right). Figure 4.11 enumerates all possible BP instances that occur within the original biclique DFG.

Once this full enumeration has occurred, all that is necessary is to select a subset of these BP instances such that all edges in the original biclique DFG are covered. Each BP instance requires a BRAM, so minimizing the number of BP instances in the cover will minimize BRAM usage.

Biclique patterns must cover all the edges in the switched bigraph DFG. However, different BPs may have shared edges. *Efficiently* covering the bigraph DFG edges is not trivial; simply choosing all largest 2W/2R-BPs may result in inefficient results. Covering the edges of the bigraph DFG is actually equivalent to the set cover problem. The set cover problem is an NP-complete problem [108].

Table 4.4: Biclique patterns and their attributes.

Pattern		Name	Biclique	BRAM Connectivity
1W/1R	Fixed	F1W1R		 Wp,i W/R̄p W1 W1/R̄1 R1 W2 W2/R̄2 R2 '0' Rq,l
	Switched	S1W1R		 Wp,i W/R̄p W1 W1/R̄1 R1 W2 W2/R̄2 R2 '0' Rp,k
1W/2R	Fixed	F1W2R		 Wp,i W/R̄p Rp,k W1 W1/R̄1 R1 W2 W2/R̄2 R2 '0' Rq,l
	Switched	S1W2R		 Wp,i W/R̄p Rp,k W1 W1/R̄1 R1 W2 W2/R̄2 R2 '0' Rp,l
2W/1R	Fixed	F2W1R		 Wp,i W/R̄p Rp,k W1 W1/R̄1 R1 W2 W2/R̄2 R2 Wq,j W/R̄q
	Switched	S2W1R		 Wp,i W/R̄p Rp,k W1 W1/R̄1 R1 W2 W2/R̄2 R2 Wp,j
2W/2R	Fixed	F2W2R		 Wp,i W/R̄p Rp,k W1 W1/R̄1 R1 W2 W2/R̄2 R2 Wq,j W/R̄q Rq,l
	Switched	S2W2R		 Wp,i W/R̄p Rp,k W1 W1/R̄1 R1 W2 W2/R̄2 R2 Wp,j Rp,l

4.3.1.5 Solving the Cover Problem

The set cover problem takes a universe set U and another set S of subsets of U whose union covers U ($\bigcup_{s\in S} s = U$), namely,

$$SCP = \left(U, S \mid \forall s \in S : s \subseteq U, \bigcup_{s\in S} s = U \right). \tag{4.10}$$

The objective of the set cover problem is to find a cover of U with the fewest sets from S. Let $T \subseteq S$ be such a subset of S, therefore the set cover problem objective is,

$$\min_{T\subseteq S} \left(|T| \middle| \bigcup_{t\in T} t = U \right) \tag{4.11}$$

The bigraph DFG optimization solves a set cover problem where all DFG edges are the universe and all biclique patterns are the covering subsets, namely,

$$SCP = \left(U, S = \left(\begin{array}{l} F1W1R \cup S1W1R \cup F1W2R \cup S1W2R \cup \\ F2W1R \cup S2W1R \cup F2W2R \cup S2W2R \end{array} \right) \right). \tag{4.12}$$

The set cover problem can be formulated and solved as the following binary linear programming (BLP) problem

$$\begin{aligned} \text{minimize} & \sum_{s\in S} x_s \\ \text{subject to} & \sum_{s:e\in s} x_s \geq 1 \quad \forall e \in U, \\ & x_s \in \{0,1\} \quad \forall s \in S \end{aligned} \tag{4.13}$$

Where x_s is a binary decision variable, indicating whether s is part of the solution or not. Figure 4.11 provides a synthesis example for the bigraph DFG from Figure 4.10 (right). A set cover solution uses the highlighted biclique patterns in Figure 4.11 (left), producing the final synthesized data bank shown in Figure 4.11 (right).

As a comparison, Table 4.5 compares solutions for the purely fixed-ports and multi-switched-ports implementations of Figure 4.8 For this specific example, the fixed-ports method consumes 12 BRAMs to construct the data banks while the multi-switched-ports method consumes only 8 BRAMs, yielding a 25% BRAM reduction.

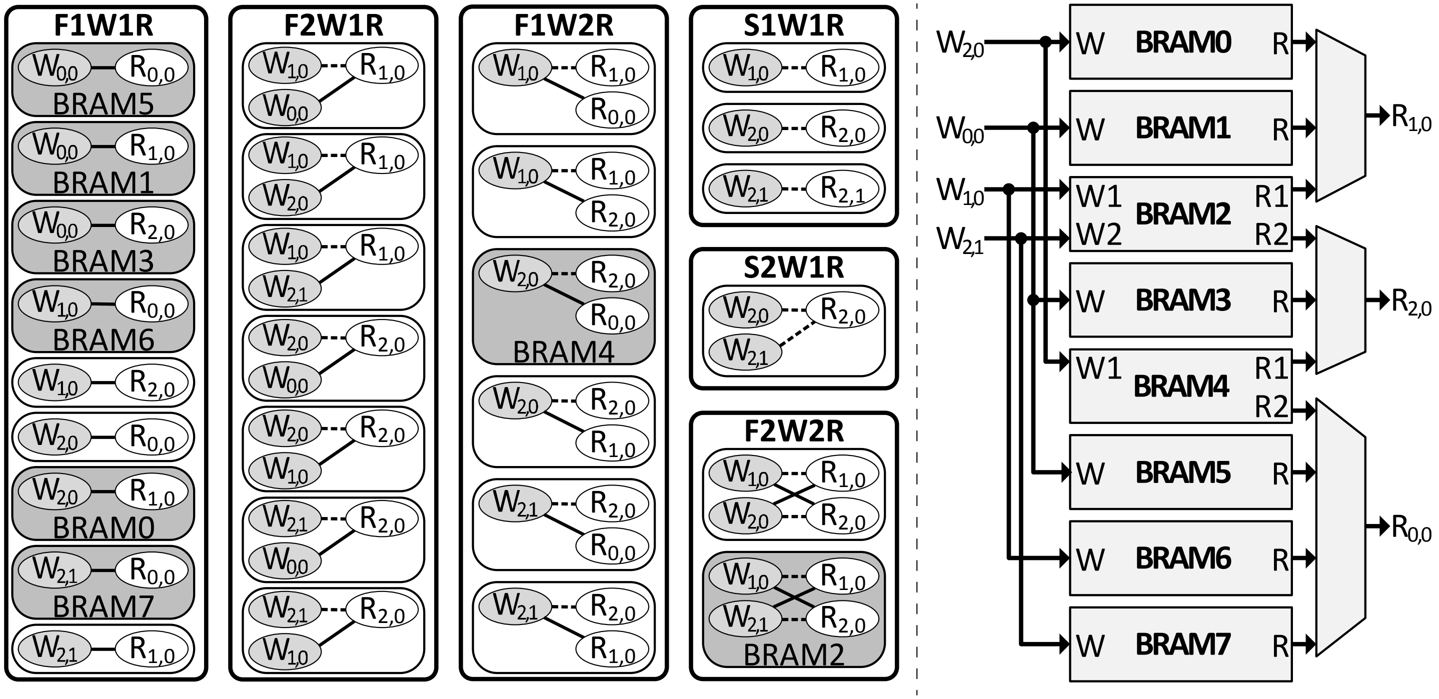

F1W1R
$W_{0,0}$ $R_{0,0}$ BRAM5
$W_{0,0}$ $R_{1,0}$ BRAM1
$W_{0,0}$ $R_{2,0}$ BRAM3
$W_{1,0}$ $R_{0,0}$ BRAM6
$W_{1,0}$ $R_{2,0}$
$W_{2,0}$ $R_{0,0}$
$W_{2,0}$ $R_{1,0}$ BRAM0
$W_{2,1}$ $R_{0,0}$ BRAM7
$W_{2,1}$ $R_{1,0}$
F2W1R
$W_{1,0}$ $W_{0,0}$ $R_{1,0}$
$W_{1,0}$ $W_{2,0}$ $R_{1,0}$
$W_{1,0}$ $W_{2,1}$ $R_{1,0}$
$W_{2,0}$ $W_{0,0}$ $R_{2,0}$
$W_{2,0}$ $W_{1,0}$ $R_{2,0}$
$W_{2,1}$ $W_{0,0}$ $R_{2,0}$
$W_{2,1}$ $W_{1,0}$ $R_{2,0}$
F1W2R
$W_{1,0}$ $R_{1,0}$ $R_{0,0}$
$W_{1,0}$ $R_{1,0}$ $R_{2,0}$
$W_{2,0}$ $R_{2,0}$ $R_{0,0}$ BRAM4
$W_{2,0}$ $R_{2,0}$ $R_{1,0}$
$W_{2,1}$ $R_{2,0}$ $R_{0,0}$
$W_{2,1}$ $R_{2,0}$ $R_{1,0}$
S1W1R
$W_{1,0}$ $R_{1,0}$
$W_{2,0}$ $R_{2,0}$
$W_{2,1}$ $R_{2,1}$
S2W1R
$W_{2,0}$ $W_{2,1}$ $R_{2,0}$
F2W2R
$W_{1,0}$ $W_{2,0}$ $R_{1,0}$ $R_{2,0}$
$W_{1,0}$ $W_{2,1}$ $R_{1,0}$ $R_{2,0}$ BRAM2
$W_{2,0}$
$W_{0,0}$
$W_{1,0}$
$W_{2,1}$
W BRAM0 R
W BRAM1 R
W1 W2 BRAM2 R1 R2
W BRAM3 R
W1 BRAM4 R1 R2
W BRAM5 R
W BRAM6 R
W BRAM7 R
$R_{1,0}$
$R_{2,0}$
$R_{0,0}$

Figure 4.11: Synthesis example of the DFG from Figure 4.8 (left) All possible biclique patterns; optimal BP's are highlighted. (right) Synthesized data banks.

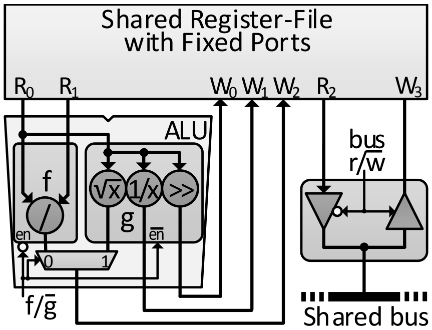
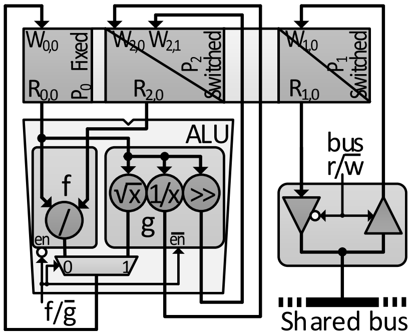
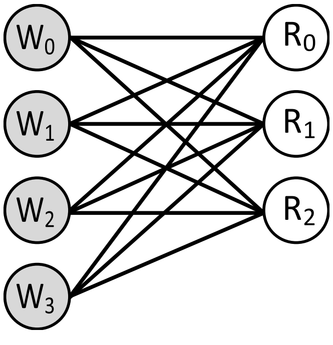
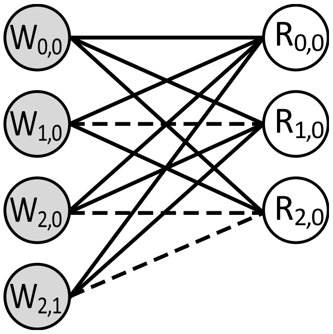
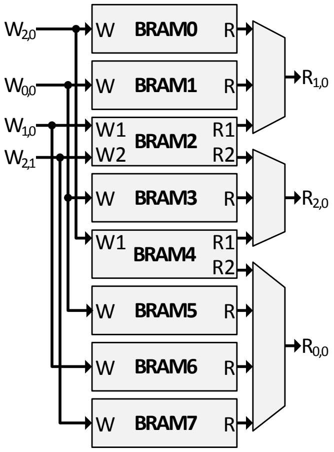
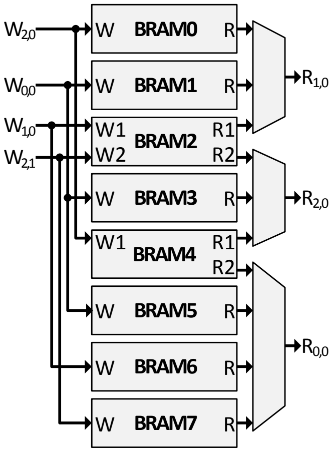
4.3.1.6 Data Dependencies and Bypassing

Due to the pipelined nature of building a full multi-switched-port RAM, data dependencies due to internal latencies arise naturally. This requires internal forwarding and bypassing to solve these hazards. The full multi-switched-port RAM design consists of the data banks and the I-LVT. The I-LVT itself consists of feedback banks and output extraction banks. For each of these three structures, Table 4.6 summarizes the type of bypassing required to produce a correct design that can tolerate certain hazards. Further detail is provided below.

The I-LVT-based structure [1] is used to steer the read data out of the multi-switched-ports data banks, however the I-LVT incurs data dependencies due to the feedback functions and the latency of reading the I-LVT to decide about the last written bank [1]. Data dependencies can be handled by employing bypassing, also known as forwarding. Bypassing is necessary since dual-port BRAMs cannot internally forward new data when one port reads and the other port writes the same address on the same clock edge, constituting a Read-During-Write (RDW) hazard.

Table 4.7 shows two types of bypassing based on write data and address pipelining. Both bypassing techniques are functionally equivalent, allowing reading of the data that is being written on the same clock edge, similar to single register functionality. However, the fully-pipelined two-stage bypassing shown in Table 4.7 (bottom) can overcome an additional cycle latency, namely an additional pipe stage on writing data and address (not shown in the figures). This capability is required if a BRAM has pipelined inputs (e.g., cascaded from another BRAM) that need to be bypassed.

Table 4.5: Comparison of purely fixed-ports versus multi-switched-ports implementations of the example in Figure 4.8.

	Fixed-ports	Multi-switched-ports
Port Assignment	 Shared Register-File with Fixed Ports R_0 R_1 W_0 W_1 W_2 R_2 W_3 ALU f / √x 1/x >> g en $\overline{en}$ 0 1 $f/\overline{g}$ bus r/w Shared bus	 $W_{0,0}$ $R_{0,0}$ P_0 Fixed $W_{2,0}$ $W_{2,1}$ $R_{2,0}$ P_2 Switched $W_{1,0}$ $R_{1,0}$ P_1 Switched ALU f / √x 1/x >> g en $\overline{en}$ 0 1 $f/\overline{g}$ bus r/w Shared bus
Biclique DFG		 $W_{0,0}$ $W_{1,0}$ $W_{2,0}$ $W_{2,1}$ $R_{0,0}$ $R_{1,0}$ $R_{2,0}$
Synthesized Data Banks	 $W_{2,0}$ $W_{0,0}$ $W_{1,0}$ $W_{2,1}$ W BRAM0 R W BRAM1 R W1 W2 BRAM2 R1 R2 W BRAM3 R W1 BRAM4 R1 R2 W BRAM5 R W BRAM6 R W BRAM7 R $R_{1,0}$ $R_{2,0}$ $R_{0,0}$	 $W_{2,0}$ $W_{0,0}$ $W_{1,0}$ $W_{2,1}$ W BRAM0 R W BRAM1 R W1 W2 BRAM2 R1 R2 W BRAM3 R W1 BRAM4 R1 R2 W BRAM5 R W BRAM6 R W BRAM7 R $R_{1,0}$ $R_{2,0}$ $R_{0,0}$

The proposed multi-switched-ports RAM utilizes true-dual-port BRAMs to provide switched port functionality. However, since writing and reading operations in true-dual-ported RAMs are exchangeable, the bypassing circuitry requires special handling. As described in Table 4.7 (right), the bypass circuit of the true/true RAM configuration is mirrored compared to the true/simple RAM configuration. Thus, it can bypass written data from any direction. However, the control logic that drives the bypassing mux selectors need to be altered to detect the direction of writing.

The most severe data dependency that I-LVT design [1] suffers from is Write-After-Write (WAW), namely, writing to the same address that has been written in the previous cycle. This dependency occurs because of the feedback reading and writing latency. A single-stage bypassing for the feedback banks solves this dependency.

Two types of reading hazards are also introduced by the I-LVT design, Read-After-Write (RAW) and Read-During-Write (RDW). RAW occurs when the same data that was written in the previous clock edge are read in the current clock edge. RDW occurs when the same data are written and read on the same clock edge.

Due to the latency of the I-LVT, reading from the same address on the next clock edge after writing (RAW) will provide the old data. To read the new data instead, the output extraction banks of the I-LVT should be bypassed by a single-stage bypass to overcome the I-LVT latency.

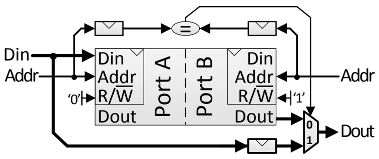
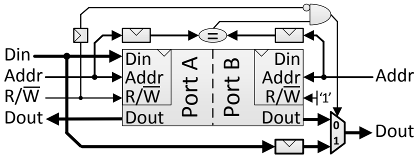
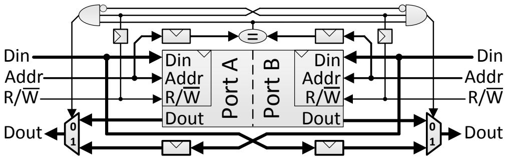
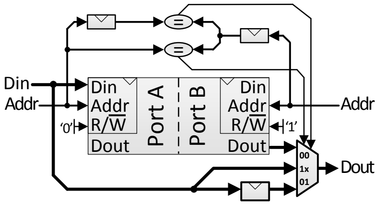
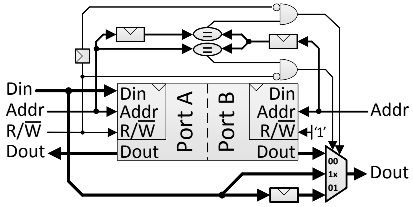
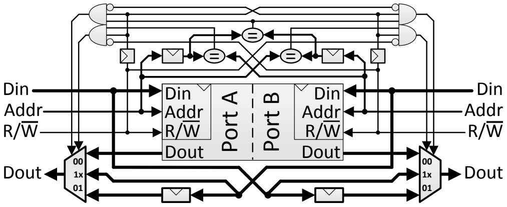
The deepest bypassing stage is reading new data on the same writing clock edge (RDW), which is similar to a single register stage latency. This can be achieved by

Table 4.6: Bypassing of multi-switched-ports.

	Data Banks	**I-LVT Banks**	
		Feedback	**Output Extract**
Allow WAW	None	1-stage	None
New Data RAW	None	1-stage	1-stage
New Data RDW	1-stage	1-stage	2-stage

2-stage bypass on the output extraction banks of the I-LVT to allow reading on the same clock edge. The data banks, which are working in parallel with the I-LVT should be bypassed by a single-stage to provide new data.

Table 4.7: Single-stage and two-stage BRAM bypassing.

	Simple/Simple Configuration	True/Simple Configuration	True/True Configuration
Single-stage	 Din Addr '0' Din Addr R/W Dout Port A Port B Din Addr R/W Dout '1' Addr 0 1 Dout	 Din Addr R/W Dout Din Addr R/W Dout Port A Port B Din Addr R/W Dout '1' Addr 0 1 Dout	 Din Addr R/W Dout 0 1 Din Addr R/W Dout Port A Port B Din Addr R/W Dout Din Addr R/W 0 1 Dout
Two-stage	 Din Addr '0' Din Addr R/W Dout Port A Port B Din Addr R/W Dout '1' Addr 00 1x 01 Dout	 Din Addr R/W Dout Din Addr R/W Dout Port A Port B Din Addr R/W Dout '1' Addr 00 1x 01 Dout	 Din Addr R/W Dout 00 1x 01 Din Addr R/W Dout Port A Port B Din Addr R/W Dout Din Addr R/W 00 1x 01 Dout

4.3.2 Experimental Results

4.3.2.1 Experimental Framework

The proposed multi-switched-port RAM approach, complete with bypassing, has been fully implemented in parameterized Verilog. For a given design instance, we developed a memory compiler to convert the RAM port assignment into a biclique DFG, enumerate all of the biclique pattern instances in the DFG, and use these to describe a set cover problem instance. The set cover problem is formulated as a Binary Linear Programming (BLP) problem using AMPL (A Mathematical Programming Language) [109], which is an algebraic modeling language used to describe large-scale mathematical optimization problems. The BLP optimization problem is solved using GLPK (GNU Linear Programming Kit) [110], an open source large-scale linear programming solver. Finally, the selected biclique patterns (covers) are used to automatically construct the data banks as described in Table 4.4 and shown for example in Figure 4.11.

A run-in-batch flow manager has also been developed to simulate and synthesize these designs with various parameters in batch using Altera's ModelSim and Quartus II. The Verilog modules, the algorithmic scripts and the flow manager are available online as an open source contribution [107].

To verify correctness, each design instance is simulated using Altera's ModelSim. A large variety instances with of different RAM port assignments and design parameters, e.g., bypassing, RAM depth and data width, are swept and simulated in batch, each with over a million random cycles. These multi-switched-port RAM design modules were then compiled with Altera's Quartus II into Altera's Stratix V `5SGXEA7N1F45C1` device [55]. This is a speed grade 1 device with 234k ALMs and 2560 M20K blocks.

4.3.2.2 Methodology

The proposed multi-switched-port RAM design process is generic and can support any number of fixed and heterogeneous switched ports. This means all previous multiport-RAM methods are actually special cases of this new proposed method. For benchmarking purposes, we take a number of design instances, and for each one we find a multi-switched port RAM using our new method. We then need to generate comparative results using multiport RAM designed with fixed ports [1], with true ports [56], and with a single switched port [2].

Then, we can compare results against the older fixed-port method [1] using the same tooling. By treating switched ports as fixed ports, we thereby place all read and write ports into the first port group P_0 (the fixed port). This generates a fixed-port solution that satisfies the same design instance requirements.

To compare against using the true-ports method [56], we must avoid using the fixed port group P_0. Instead, as described in Table 4.8, each fixed port must be mapped into a true port (with a fixed $R\overline{W}$ control), hence $n_{W,0}+n_{R,0}$ true ports are required to implement the fixed ports. In addition, every read and write pair in a switched port can be mapped into a single true port, hence, $\max(n_{W,i},n_{R,i})$ true ports are required to implement a switched port group Pi. In total, the number of the required true ports is

$$n_t = n_{W,0} + n_{R,0} + \sum_{i=1}^{n_P - 1} \max(n_{W,i}, n_{R,i}) \tag{4.14}$$

For example, the system in Figure 4.8 can be implemented using $n_t = 5$ true ports.

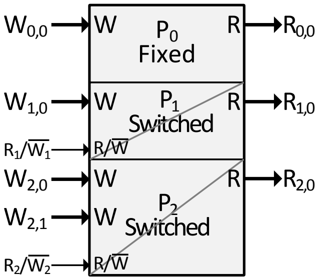
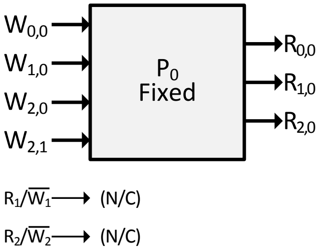
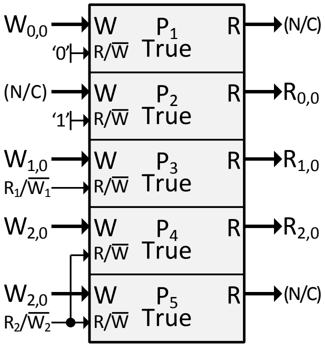
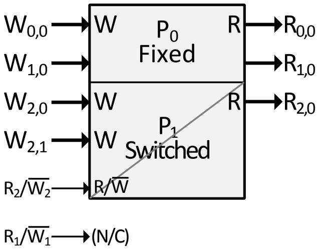
To compare against the single-switched-port method [2], the largest switched port (say P_m) is chosen to be implemented as the single-switched port, and all the other ports are implemented using fixed ports. This becomes

$$P = \left\{ P_0 = \left(n_W - n_{W,m}, n_R - n_{R,m} \right), P_1 = \left(n_{W,m}, n_{R,m} \right) \right\} \tag{4.15}$$

where m is the index of the switched-port with the maximum writes and reads,

$$1 \leq m < n_P \middle| n_{W,m} + n_{R,m} = \max_{1 \leq i < n_P} \left(n_{W,i} + n_{R,i} \right) \tag{4.16}$$

Table 4.8: Multi-switched-ports conversion (example from Figure 4.8).

	Multi-switched	Fixed [1, 11]	True [56]	Single-switched [2]
Port Assignment	 $W_{0,0}$ W P_0 Fixed R $R_{0,0}$ $W_{1,0}$ W P_1 Switched R $R_{1,0}$ $R_1/\overline{W_1}$ $R/\overline{W}$ $W_{2,0}$ W R $R_{2,0}$ $W_{2,1}$ W P_2 Switched $R_2/\overline{W_2}$ $R/\overline{W}$	 $W_{0,0}$ $W_{1,0}$ $W_{2,0}$ $W_{2,1}$ P_0 Fixed $R_{0,0}$ $R_{1,0}$ $R_{2,0}$ $R_1/\overline{W_1}$ (N/C) $R_2/\overline{W_2}$ (N/C)	 $W_{0,0}$ W P_1 True R (N/C) '0' $R/\overline{W}$ (N/C) W P_2 True R $R_{0,0}$ '1' $R/\overline{W}$ $W_{1,0}$ W P_3 True R $R_{1,0}$ $R_1/\overline{W_1}$ $R/\overline{W}$ $W_{2,0}$ W P_4 True R $R_{2,0}$ $R/\overline{W}$ $W_{2,0}$ W P_5 True R (N/C) $R_2/\overline{W_2}$ $R/\overline{W}$	 $W_{0,0}$ W P_0 Fixed R $R_{0,0}$ $W_{1,0}$ $R_{1,0}$ $W_{2,0}$ W R $R_{2,0}$ $W_{2,1}$ W P_1 Switched $R_2/\overline{W_2}$ $R/\overline{W}$ $R_1/\overline{W_1}$ (N/C)

4.3.2.3 Test Cases

A number of random multi-switched-ports test-cases are listed in Table 4.9. Ten random test cases, TC1 to TC10, have been generated for illustrative purposes; real cases are difficult to extract from applications without a precise understanding of their use of multi-port RAMs and how their FSMs specify (possibly mutually exclusive cases of) read/write behaviour. While our method can synthesize any number ports, the test-cases are limited to 8 switched ports to avoid accidentally exceeding device resources. These random test-cases use 4 to 8 switched ports, where each switched port has 1 to 4 writes or reads. For each test case, we specify a multi-port RAM that the multi-switched ports approach proposed in this chapter. In addition, we show other multi-port RAM designs with fixed ports [1], true ports [56], and a single switched-port [2] that address the same requirements.

4.3.2.4 Results

Table 4.10 lists the experimental results of the ten random test-cases defined in Table 4.9, implemented using the four design styles. All synthesized test-cases have one byte of data width and 8k-lines in depth, new-data-RAW bypassing and use a binary-coded I-LVT [1]. BRAM consumption, ALMs and F_{max} are given directly in the table. Table 4.11 lists the change percentage in these parameters compared to our proposed method. Compared to the single switched-port [2] and the fixed-port [1] methods, ALM consumption and F_{max} are similar while BRAM consumption reduced by 18% on average. On the other hand, comparing our proposed method to the true-ports method shows a 42% BRAM reduction, 53% fewer ALMs, and

15% higher F_{max}.

Table 4.9: Heterogeneous multi-ported RAM testcases.

Testcase#	Writes and reads for each port; $\left(\mathbf{n_{W,i}}, \mathbf{n_{R,i}}\right)$ from Equation 4.4													
	Multi-switched								Fixed	True			Single-switched	
	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_0	P_0	$P_0 \cdots P_{n_t}$	n_t	P_0	P_1
TC1	(1,1)	(1,1)	(2,1)	(2,1)	(1,2)	(1,2)	(2,3)	(2,3)	(11,13)	(0,0)	(1,1)	15	(9,10)	(2,3)
TC2	(1,1)	(1,1)	(2,1)	(1,2)	(2,2)	(2,3)	(3,2)	(3,3)	(13,14)	(0,0)	(1,1)	15	(10,11)	(3,3)
TC3	(1,2)	(1,1)	(1,2)	(2,1)	(2,2)	(2,3)	(3,2)	-	(12,12)	(0,0)	(1,1)	15	(9,10)	(3,2)
TC4	(2,1)	(1,1)	(1,1)	(1,2)	(1,3)	(2,3)	(3,3)	-	(9,15)	(0,0)	(1,1)	15	(6,12)	(3,3)
TC5	(1,1)	(1,3)	(2,1)	(2,1)	(2,2)	(2,3)	-	-	(10,10)	(0,0)	(1,1)	13	(8,7)	(2,3)
TC6	(1,3)	(1,1)	(1,1)	(1,3)	(2,2)	(3,3)	-	-	(8,13)	(0,0)	(1,1)	13	(5,10)	(3,3)
TC7	(2,2)	(1,1)	(2,4)	(2,1)	(1,3)	-	-	-	(8,11)	(0,0)	(1,1)	14	(6,7)	(2,4)
TC8	(2,1)	(1,1)	(1,4)	(2,1)	(3,1)	-	-	-	(8,10)	(0,0)	(1,1)	14	(7,6)	(1,4)
TC9	(2,3)	(2,1)	(1,4)	(2,4)	-	-	-	-	(7,12)	(0,0)	(1,1)	15	(5,8)	(2,4)
TC10	(3,1)	(1,2)	(2,4)	(3,4)	-	-	-	-	(9,11)	(0,0)	(1,1)	14	(6,7)	(3,4)

Table 4.10: Experimental results.

Testcase#	Multi-switched			Single-switched [2]			True [56]			Fixed [1]		
	BRAMs	ALMs	F_{max}(MHz)	BRAMs	ALMs	F_{max}(MHz)	BRAMs	ALMs	F_{max}(MHz)	BRAMs	ALMs	F_{max}(MHz)
TC1	826	3417	247.4	1054	3349	242.25	1290	6222	215.84	1034	3271	235.29
TC2	1044	4781	224.16	1316	4828	228.1	1290	6222	215.84	1352	4840	225.33
TC3	904	3793	232.56	1104	3717	237.3	1290	6222	215.84	1080	3634	241.9
TC4	726	2732	250.75	882	2653	253.87	1290	6222	215.84	918	2617	245.04
TC5	628	2434	248.32	756	2441	244.38	962	4503	235.68	780	2414	253.94
TC6	568	2031	258	700	1948	260.42	962	4503	235.68	704	1916	246.37
TC7	540	1801	257.67	608	1746	265.32	1120	5483	214.13	608	1703	260.96
TC8	524	1675	265.82	576	1629	270.42	1120	5483	214.13	592	1614	266.03
TC9	504	1499	279.17	556	1502	270.27	1290	6222	215.84	560	1444	275.71
TC10	586	2321	252.33	690	2205	257.33	1120	5483	214.13	702	2154	247.46
Avg.	685	2648.4	251.62	824.2	2601.8	253	1173.4	5656.5	219.3	833	2560.7	249.8

Table 4.11: Results comparison.

Testcase#	Reduction compared to:			ALM Reduction Compared to:			F_{max} Increase Compared to:		
	Single-switched [2]	True [56]	Fixed [1]	Single-switched [2]	True [56]	Fixed [1]	Single-switched [2]	True [56]	Fixed [1]
TC1	22%	36%	20%	-2%	45%	-4%	2%	15%	5%
TC2	21%	19%	23%	1%	23%	1%	-2%	4%	-1%
TC3	18%	30%	16%	-2%	39%	-4%	-2%	8%	-4%
TC4	18%	44%	21%	-3%	56%	-4%	-1%	16%	2%
TC5	17%	35%	19%	0%	46%	-1%	2%	5%	-2%
TC6	19%	41%	19%	-4%	55%	-6%	-1%	9%	5%
TC7	11%	52%	11%	-3%	67%	-6%	-3%	20%	-1%
TC8	9%	53%	11%	-3%	69%	-4%	-2%	24%	0%
TC9	9%	61%	10%	0%	76%	-4%	3%	29%	1%
TC10	15%	48%	17%	-5%	58%	-8%	-2%	18%	2%
Avg.	17%	42%	18%	-2%	53%	-3%	-1%	15%	1%

4.4 Conclusions

In this chapter, we have proposed a novel, modular, BRAM-based switched-multi-ported RAM architecture. In addition to unidirectional ports with fixed read/write, this switched architecture allows a group of write ports to switch with another group of read ports dynamically. The proposed switched-ports architecture is less flexible than a true-multi-ported RAM where each port is switched individually. Nevertheless, switched memories can dramatically reduce BRAM consumption compared to true or simple ports for systems with alternating port requirements.

In addition, we propose a new idea of having multiple switched ports in multi-ported RAM design. This method requires a memory compiler to create a specific design instance, and solving a set cover problem to optimize its implementation. Our Computer-Aided Design (CAD) approach always finds a minimal implementation for all of our test cases, but there is opportunity for further CAD research to improve run-time while still being optimal. On average out of 10 random test-cases, the suggested multi-switched-ports method reduces BRAM use by 18% compared to the best of previous methods, while maintaining ALM count and F_{max}. Future research may address the RAM port assignment problem to more complex cases where there are more than two states governing memory port usage.

The fully parameterized and generic Verilog modules, the algorithmic scripts, the multi-switched-ports memory compiler, and the flow manager are available online as an open source contribution [106, 107].

Chapter 5

2-Dimensional Hierarchical Search BCAMs (2D-HS-BCAMs)

In this chapter, a novel and efficient technique for constructing BCAMs out of standard SRAM blocks in FPGAs is proposed. The new technique divides a CAM into wide rows or sets, instead of just being a single pattern wide. Using Altera's Stratix V device, the traditional design method achieves up to a 64K-entry BCAM while the proposed technique achieves up to 4M entries. For the 64K-entry test-case, the traditional method consumes 43 times more ALMs, 18 times longer mapping runtime, and achieves only one-third of the F_{max} of the proposed method. A fully parameterized Verilog implementation is being released as an open source hardware library. The library has been extensively tested using ModelSim and Altera's Quartus tools.

5.1 Introduction

In this section, a modular SRAM-based BCAM suitable for many storage entries is proposed. The CAM storage is divided into equal-sizes sets. CAM lookup depends upon two steps. First, a RAM structure stores hit or miss information for each set, where a set stores several patterns. Second, the specific set is searched in parallel for a match. The set data (the patterns themselves) are stored in a second RAM structure. To achieve a write and a match of the same pattern in the same cycle, a CAM bypassing mechanism is also provided.

The proposed method is device-independent; hence, it can be applied to any FPGA device containing standard dual-ported BRAMs. The proposed approach dramatically improves CAM storage efficiency and operation frequency compared to conventional methods. In contrast to other attempts that require several cycles to write or match [111–113], the proposed approach is high-throughput and can perform a pattern read (match) every cycle and a pattern write every two cycles.

Major contributions of this work are:

- A novel BCAM architecture capable of producing millions of CAM entries. Compared to other conventional BCAM approaches, the proposed technique exhibits the highest storage efficiency while providing improved overall performance. To the authors' best knowledge, research and patent literature do not have similar BCAM techniques.
- A CAM bypassing mechanism is also provided to write and match the same pattern in the same cycle.

- A parameterized Verilog implementation of the suggested methods, together with previous standard approaches. A flow manager to simulate and synthesize various designs with various parameters in batch using Altera's ModelSim and Quartus II is also provided. The Verilog modules and the flow manager are available online [114].

To verify correctness, the proposed BCAM architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II [57]. A large variety of BCAM architectures and parameters, e.g., BCAM depth, pattern width, bypassing, and pipelining are simulated in batch, each with over one million random BCAM write and match cycles. Stratix V, Altera's high-end performance-oriented FPGA [55], is used to implement and compare the proposed architecture with previous approaches.

The rest of this chapter is organized as follows. The motivation and method of the proposed 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) approach to generate deep BCAMs is described in detail in Section 5.2. BCAM bypassing techniques are described in Section 5.3. Discussion of the suggested method and comparison to previous techniques are provided in Section 5.4. The experimental framework, simulation and synthesis results, are discussed in Section 5.5. Conclusions are drawn in Section 5.6.

5.2 The 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) Approach

As shown in Figure 5.1, pattern lookup works in two stages. First, a RAM structure, called the Set Transposed Indicators RAM (STIRAM), is indexed by the pattern; for each pattern, it stores match information about where the pattern is present in each possible set. This can be used by a priority encoder to identify the address or ID of a set containing the pattern; the set ID forms the upper bits of the match address. The set ID also indexes into a second RAM structure, called the Sets RAM (SetRAM), to produce the set content. Second, the wide set is searched in parallel for a match to the pattern; the location of the match within the set produces the lower bits of the match address.

While pattern-cascadable BCAMs require indicators from every address location at every stage, this requirement can be alleviated if the BCAM will not be cascaded in pattern width. Instead of storing pattern indicators for each address location separately, an indicator is generated for a group of addresses, indicating if the pattern exists at any of these addresses. An address set of width S_W is a group of successive S_W addresses, hence an address range of depth S_W. For a C_D-entry BCAM, and a S_W set width, $\left\lceil \frac{C_D}{S_W} \right\rceil$ sets exist, and can be enumerated as

$$S = \left\{ 0, 1, \cdots, \left\lceil \frac{C_D}{S_W} \right\rceil - 1 \right\}. \tag{5.1}$$

A set indicator $I_{p,s}$ indicates if any of the addresses in set s, hence addresses

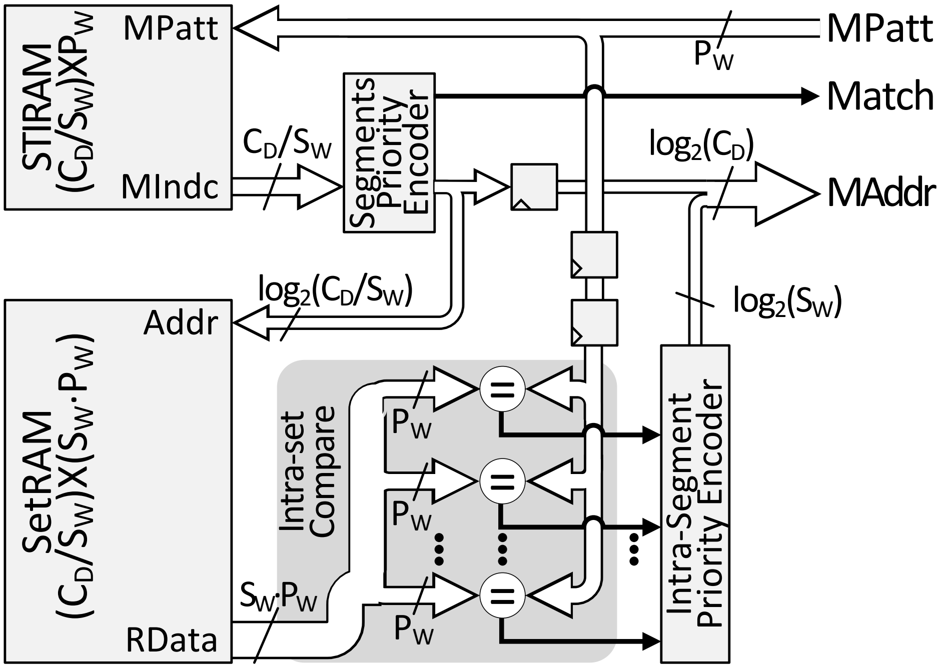


Figure 5.1: The match portion of the 2D-HS-BCAM system.

$S_W \cdot s$ upto $S_W \cdot (s+1) - 1$ contains the pattern p, namely,

$$\forall s \in S, p \in P : I_{P,S} = \bigvee_{a=S_W\cdot s}^{a=S_W\cdot(s+1)-1} \big(RAM\,[a] \;\; EQ \;\; p\big). \tag{5.2}$$

Where P is the patterns set. The STIRAM is therefore

$$STIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \cdots & I_{0,|S|-1} \\ I_{1,0} & I_{1,1} & \cdots & I_{1,|S|-1} \\ \vdots & \vdots & \ddots & \vdots \\ I_{|P|-1,0} & I_{|P|-1,1} & \cdots & I_{|P|-1,|S|-1} \end{bmatrix}. \tag{5.3}$$

STIRAM provides information for matched patterns within a set, not the exact location. To detect the exact pattern location, an auxiliary RAM stores the patterns associated with each set, with all patterns for the set in one RAM address, hence it is called the Sets RAM (SetRAM). If a match is found in a specific set, this set will be fetched from the SetRAM and patterns within this set are compared concurrently to the match pattern.

Figure 5.2 illustrates the complete 2D-HS-BCAM system, whereas an example of an 8-entry, 2-bit BCAM with $S_W = 2$ is given in Figure 5.3. Two RAM structures are required, the first is STIRAM, a transposed RAM, which is addressed by patterns and stores set indicators. The second is SetRAM, which stores the data patterns for each set in one RAM line. The reference RAM in Figure 5.3 describes the content of the BCAM, but is not required for the 2D-HS-BCAM implementation. Instead, BCAM content is stored in the SetRAM as sets of data patterns.

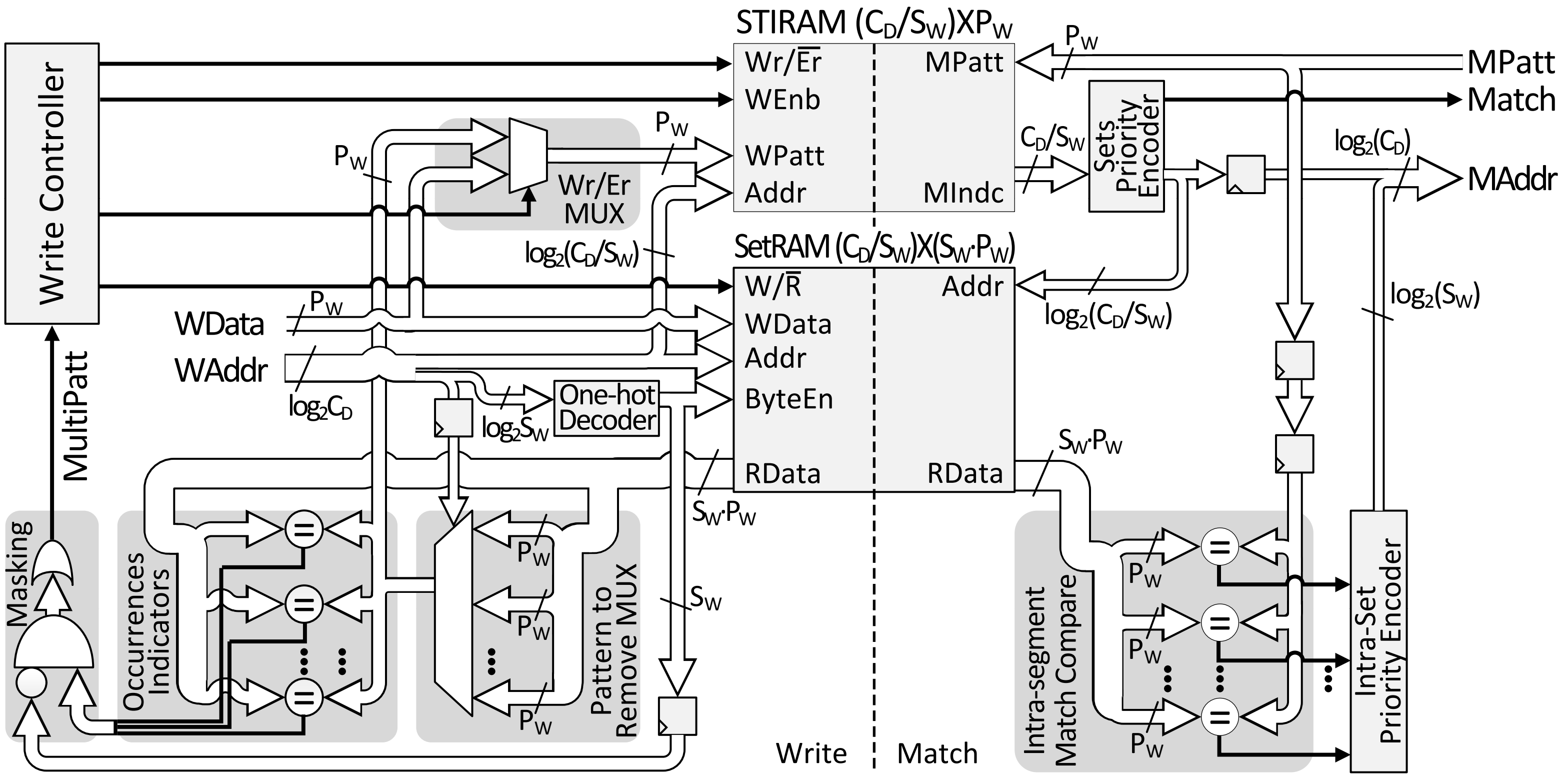


Figure 5.2: The complete 2D-HS-BCAM system.

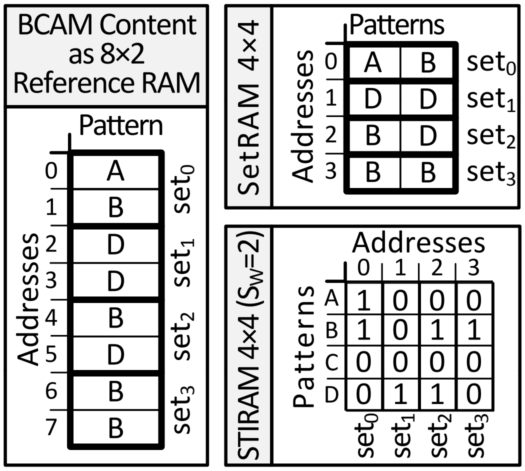


Figure 5.3: 8×2; $S_W = 2$ example of the 2D-HS-BCAM.

The match operation checks STIRAM for a match within a set, detects the first matching set using a priority encoder, then fetches the corresponding set patterns (with a match) from SetRAM, then compares all patterns with the match pattern in parallel to detect the exact match location. The match operation is described in details as follows.

1. Detect match among sets:

   1.1. MPatt is provided to STIRAM for search.

   1.2. STIRAM detects which sets contains MPatt, outputting a match indicator for each set.

   1.3. Sets priority-encoder (PE) generates the binary address for the first set with a matching pattern. This address composes the higher part of MAddr.

1.4. Sets PE also generates the binary Match signal, which indicates that a match was found.

2. Detect match exact location within the set:

   2.1. The address of the first matched set (step 1.3) is provided to the SetRAM to fetch the entire set.

   2.2. Each pattern within the set is compared to MPatt

   2.3. A priority-encoder (intra-set PE) detects the first matching pattern. The address of the first matching pattern produces the lower part of MAddr.

While detecting a match is completed by reading the STIRAM in one cycle, computing the exact match address requires another cycle to read the SetRAM. Hence, the match operation latency is two cycles. The match operation throughput is a single cycle since both STIRAM and SetRAM are read concurrently.

Similar to the Brute-Force Transposed Indicators (BF-TI) writing mechanism (see Appendix A), writing to the 2D-HS-BCAM requires two cycles, one cycle for new pattern insertion, and a second cycle for old pattern deletion. However, before clearing the old data indicator in the STIRAM, the set should be checked to detect other occurrences of the old data. If another occurrence of the deleted pattern is found in the same set, the set indicator in the STIRAM should not be cleared. Figure 5.2 illustrates the additional circuitry required for writing the BCAM. In detail, the 2D-HS-BCAM writing is performed as follows.

1. Cycle 1: STIRAM write; SetRAM read

1.1. Write STIRAM with WPatt (also called WData) and the higher $\left\lceil \log_2 \frac{C_D}{S_W} \right\rceil$ bits of WAddr to set the corresponding set indicator.

1.2. Read the entire corresponding set for SetRAM (addressed by the higher $\left\lceil \log_2 \frac{C_D}{S_W} \right\rceil$ bits of WAddr)

  1.2.1. The Pattern to remove MUX selects the pattern that is being rewritten from the corresponding set. The selector is the lower $\lceil \log_2 S_W \rceil$ bits of WAddr.

  1.2.2. The Occurrences Indicators are a compare of the currently rewritten pattern with all the other patterns in the set to detect other occurrences.

  1.2.3. The final masking stage masks the indicator of the currently rewritten pattern, since only other occurrences should be detected. All the indicators are OR'ed to detect any other occurrence

2. Cycle 2: SetRAM write; STIRAM conditional erase

  2.1. The SetRAM is written with WPatt. Byte-enable is used to write only the corresponding pattern in the set.

  2.2. If no other occurrences of the currently rewritten pattern are detected (stage 1.2.3 above), the STIRAM indicator for the replaced pattern and the current address is cleared.

The read and write operations described above rely upon an efficient implementation of a very wide priority encoder. The design we use is described in

Appendix B.

To implement a BCAM with C_D entries and P_W pattern width, namely a $C_D \times P_W$ BCAM, the 2D-HS-BCAM approach requires $\left\lceil \frac{C_D}{S_W} \right\rceil \times P_W \cdot S_W$ SRAM cells for the SetRAM and $2^{P_W} \times \left\lceil \frac{C_D}{S_W} \right\rceil$ SRAM cells for the STIRAM, a total of

$$\left\lceil \frac{C_D}{S_W} \right\rceil \times P_W \cdot S_W + 2^{P_W} \times \left\lceil \frac{C_D}{S_W} \right\rceil . \tag{5.4}$$

Assuming a wide RAM of $R_{W,max}$, an upper bound estimate for the BRAMs needed to construct the STIRAM is

$$\left\lceil \frac{2^{P_W}}{R_{D,min}} \right\rceil \cdot \left\lceil \frac{\left\lceil \frac{C_W}{S_W} \right\rceil}{R_{W,max}} \right\rceil . \tag{5.5}$$

The SetRAM is a true-dual-port RAM. $R_{W,byte}$ describes the width of data controlled by a single byte-enable. A single BCAM accommodates $\frac{R_{W,max}}{R_{W,byte}}$ individual byte-enable controlled data, and each pattern requires $\left\lceil \frac{P_W}{R_{W,byte}} \right\rceil$ of them. Finally, S_W lines of this structure are required. Therefore, the number of BCAMs needed to construct the SetRAM is

$$\frac{R_{W,max}}{R_{W,byte}} \cdot \left\lceil \frac{P_W}{R_{W,byte}} \right\rceil \cdot \left\lceil \frac{S_W}{R_{D,min}} \right\rceil . \tag{5.6}$$

5.3 BCAM Bypassing

Writing a new pattern to a register-based BCAM is immediate on the triggering clock edge and it is ready to be matched immediately. In contrast, BRAM-based BCAM methods, namely BF-TI and 2D-HS-BCAM, require two cycles for writing. Hence, matching a pattern that is being written in the same cycle will match old indicators, introducing a read-after-write hazard. However, some applications, e.g., caches and TLBs, require immediate matching of the recently written patterns, hence, pattern bypassing is required.

Figure 5.4 shows the bypassing circuitry for both BF-TI and 2D-HS-BCAM. In both, the writing address (WAddr) is one-hot encoded; the insertion mask (bitwise OR) forces '1' into the matched indicators (MIndc) in location MAddr. Similarly, the removal mask (bitwise AND) forces '0' into the matched indicators (MIndc) in location MAddr. In the TIRAM approach, if the matched pattern (MPatt) is equivalent to the written pattern (WPatt), the insertion mask output is passed to the matching indicators output (MIndc); otherwise the removal mask output is passed. In the 2D-HS-BCAM approach, there is another case. The removal mask output is allowed to be passed only if MPatt equals to the removed pattern (RmPatt), and there are no other occurrences of the removed pattern in the same set (negated MultiPatt).

5.4 Comparison and Discussion

The BCAM storage efficiency μ_s is first introduced in this dissertation, and is defined as the SRAM cell utilization for a specific BCAM implementation. In

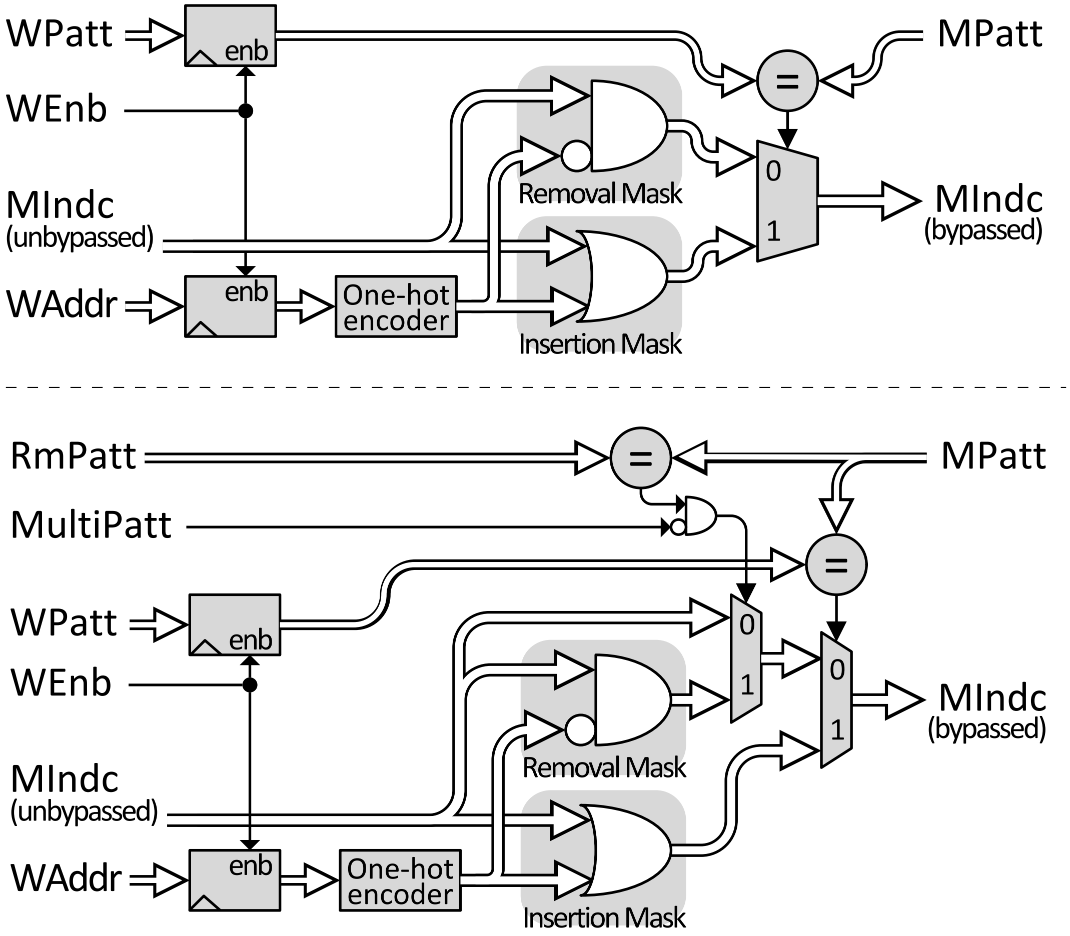


Figure 5.4: Bypassing logic for (top) Brute-Force Transposed Indicators (BF-TI) approach, and (bottom) the proposed 2D-HS-BCAM approach.

other words, μ_s is the ratio between the total BCAM bits and the total SRAM cells used to implement this BCAM. Using Equation 2.11 and Equation 2.13, μ_s for the

Brute-Force Transposed Indicators (BF-TI) is estimated as follows.

$$\mu_s\left(BFTI\right) \approx \begin{cases} \frac{1}{1+\frac{2^{P_W}}{P_W}} & uncascaded \\ \frac{1}{1+\frac{R_{D,min}}{\log_2\left(R_{D,min}\right)}} & pattern-cascaded \end{cases} \tag{5.7}$$

Equation 5.7 shows that the storage efficiency of the uncascaded BF-TI implementation is inversely proportional to the exponent of P_W, hence, decays rapidly with P_W increase as shown in Figure 5.5.

The pattern width cascaded Brute-Force Transposed Indicators (BF-TI) is not related to P_W, hence the efficiency is not affected when P_W increases. However, the efficiency is affected by an intrinsic BRAM characteristic, namely, the minimal depth $R_{D,min}$ (associated with the maximum width). The efficiency is inversely proportional to $R_{D,min}$, hence, shallow and wide BRAMs with smaller $R_{D,min}$ (and larger $R_{W,max}$) will exhibit higher storage efficiency.

Using Equation 5.4, μ_s for the 2D-HS-BCAM is estimated as follows.

$$\mu_s\left(2DHS\right) \approx \frac{1}{1+\frac{2^{P_W}}{S_W \cdot P_W}} \tag{5.8}$$

Similar to the uncascaded BF-TI, the efficiency of the proposed 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) approach is inversely proportional to the exponent of P_W. However, the exponential relation is mitigated by the set width S_W, an external and user-driven parameter.

For example, with $S_W = 4K$, $R_{D,min} = 512$ (as in Altera's M20K), and a pattern width of $P_W = 12$, the storage efficiency of the proposed 2D-HS-BCAM is $\mu_s = 0.923$ while using the pattern width cascaded BF-TI approach provides $\mu_s = 0.017$. The 2D-HS-BCAM approach provides the highest efficiency up to $P_W = 22$, compared to the BF-TI. To generalize, 2D-HS-BCAM is superior to the BF-TI method up to

$$P_W = -\frac{W_{-1}\left(-\frac{\ln 2}{a}\right)}{\ln 2},\ a = S_W \cdot \frac{R_{D,min}}{\log_2\left(R_{D,min}\right)}, \tag{5.9}$$

Where W_{-1} is the lower branch of the Lambert Product Logarithm function (also called Omega function).

Practically, the set width S_W has a limitation due to the minimal width of the Block-RAM used in the SetRAM. The SetRAM depth is $\left\lceil \frac{C_D}{S_W} \right\rceil$ and is limited by $R_{D,min}$, hence, to achieve maximum storage efficiency, S_W is bounded by

$$S_W \leq \frac{C_D}{R_{D,min}}. \tag{5.10}$$

For deep memories, S_W can be set to higher values, allowing higher storage efficiency. Furthermore, providing shallow and wide BRAM, namely a lower $R_{D,min}$, will allow higher S_W values, hence a higher storage efficiency.

To reduce ALM consumption it is recommended to divide the priority-encoders in the 2D-HS-BCAM design equally, hence, $S_W \approx \sqrt{C_D}$. Each priority encoder will be of width $\sqrt{C_D}$. Furthermore, dividing the priority-encoders equally will balance the priority-encoders depth, hence increasing $\mathrm{F}_{\mathrm{max}}$ and reducing the maximum

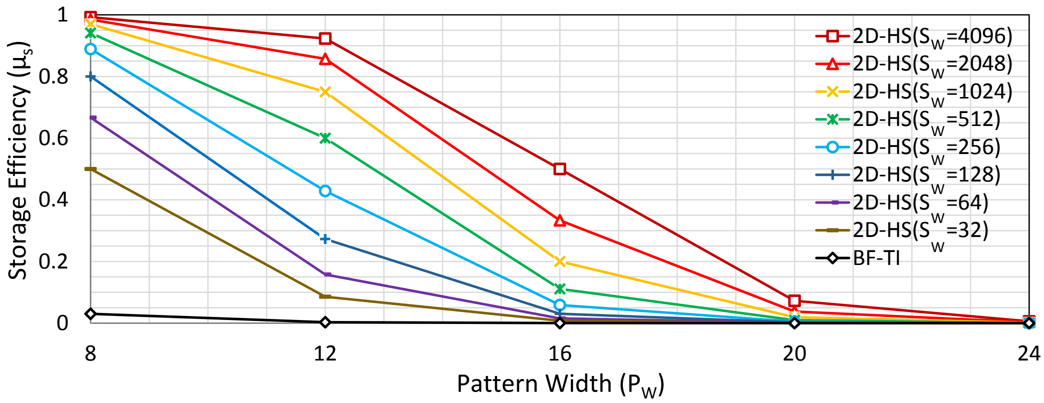


Figure 5.5: Storage efficiency (μ_s) as function of pattern width (P_W) for the uncascaded BF-TI and 2D-HS-BCAM.

pipe stages in the pipelined design.

5.5 Experimental Results

To verify and simulate the suggested approach and compare to standard techniques, fully parameterized Verilog modules have been developed. Register-based, pattern width cascaded and uncascaded brute-force TIRAM, and the proposed 2D-HS-BCAM methods have been implemented. To simulate and synthesize these designs with various parameters in batch using Altera's ModelSim and Quartus II, a run-in-batch flow manager has also been developed. The Verilog modules and the flow manager are available online [114].

To verify correctness, the proposed architecture is simulated using Altera's ModelSim. A large variety of different BCAM architectures and parameters, e.g., bypassing, depth, pattern width, and set width, are swept and simulated in batch, each with over one million random cycles. All different BCAM design modules were implemented using Altera's Quartus II on Altera's Stratix V 5SGXMA7N1F45C1 device [55]. This is a high-performance device with 235k ALMs and 2560 M20Ks.

Figure 5.6 and Figure 5.7 plot feasible BCAM depth and pattern width sweeps implemented on Altera's Stratix V device. Within the device limitation, the proposed 2D-HS-BCAM approach is able to reach 4M entries of CAM, while the BF-TI and the register-based BCAMs cannot exceed 64K and 32K entries, respectively. The number of Altera's M20K blocks used to implement each BCAM configuration is plotted in Figure 5.6 (bottom). Even for shallow memories of 64K and 32K, the proposed approach demonstrates lower BRAM consumption compared to the other methods. The columns in Figure 5.6 show the BRAM consumption of the two RAM structures that compose the 2D-HS-BCAM; the SetRAM and the STIRAM. The SetRAM stores the data patterns; hence, if the SetRAM BRAM consumption is dominating the STIRAM consumption, the storage efficiency will be higher.

The proposed 2D-HS-BCAM method exhibits significantly lower ALM count and higher F_{max} due to splitting the priority-encoder as shown in Figure 5.6 (middle and top). The register-based BCAM consumes the highest ALMs due to massive register usage. To achieve high F_{max}, all testcases are fully pipelined. Pipelining and BRAM access latency increase the overall system latency in both traditional and proposed approaches. The latency overhead is reported in Figure 5.7 (bottom). Brute-force TIRAM approach has a lower latency by maximum one cycle in some shallow testcases. The latency of both approaches in these shallow testcases is 9 or

10 cycles. The growth of latency as depth increases is logarithmic, since PE depth is logarithmic. The latency of the deepest 4M-entry 2D-HS-BCAM is 13 cycles.

Figure 5.7 (lower middle) plots the optimal set width. As the depth increases, optimal set width is larger. Similar with pattern width; if pattern width increases, optimal set width increases to overcome the increase in STIRAM BRAM consumption to wide patterns.

The storage efficiency is plotted in Figure 5.7 (upper middle). The 2D-HS-BCAM overcomes the brute-force TIRAM in 32K and 64K CAMs. Furthermore, storage efficiency is inversely related to pattern width as expected from 2D-HS-BCAM. As 2D-HS-BCAM goes deeper, the efficiency increases. The 4M $\times$ 9 BCAM test case demonstrates a high storage efficiency of 0.9.

Figure 5.7 (top) plots the full Quartus II flow runtime. 2D-HS-BCAM synthesis is faster than register-based or BF-TI BCAMs. 2D-HS-BCAM synthesis runs up to 3 hours in its largest 4M testcase.

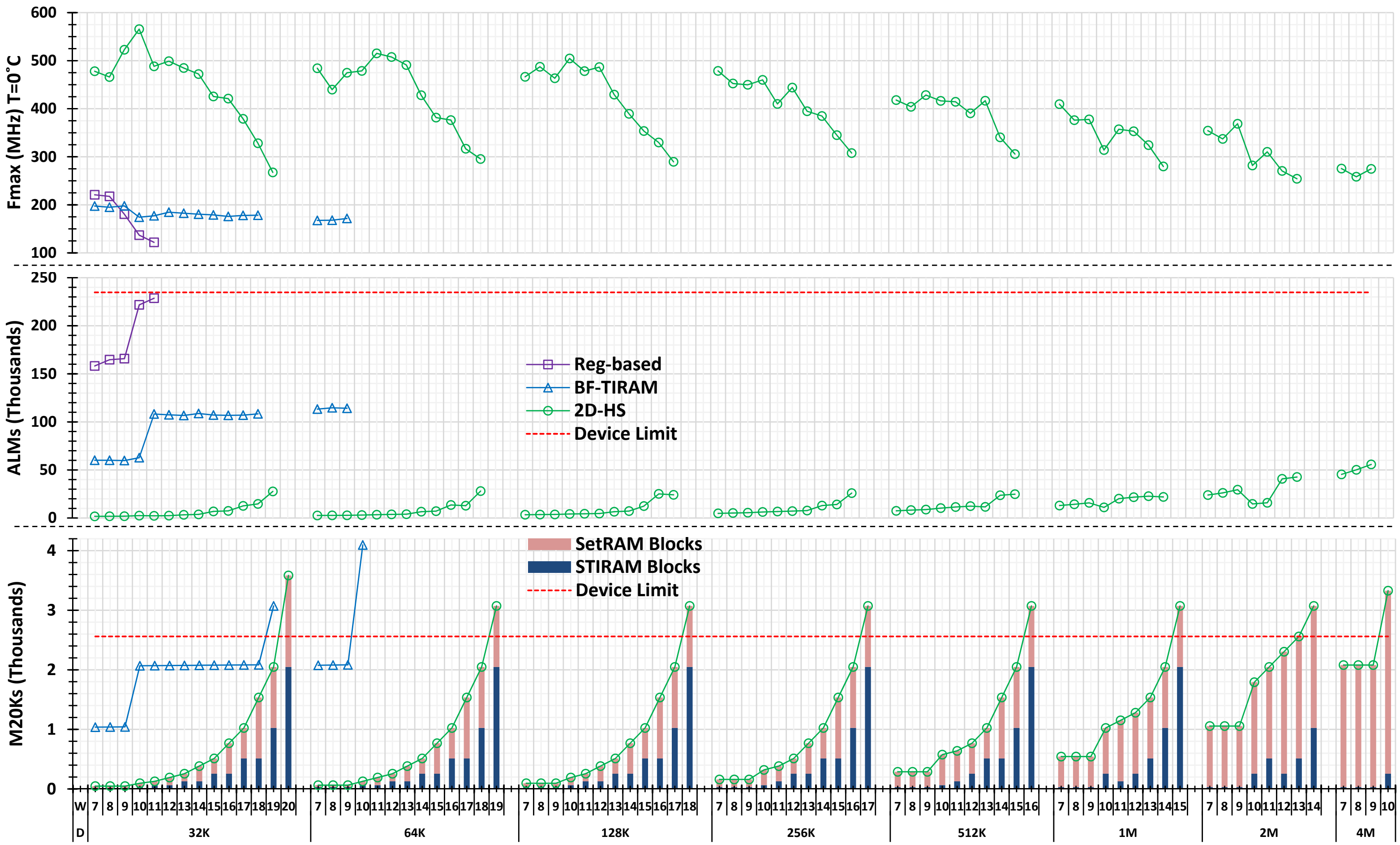


Figure 5.6: Results for several BCAM depth and pattern width sweeps (top) F_{max} (middle) ALMs count (bottom) M20K count.

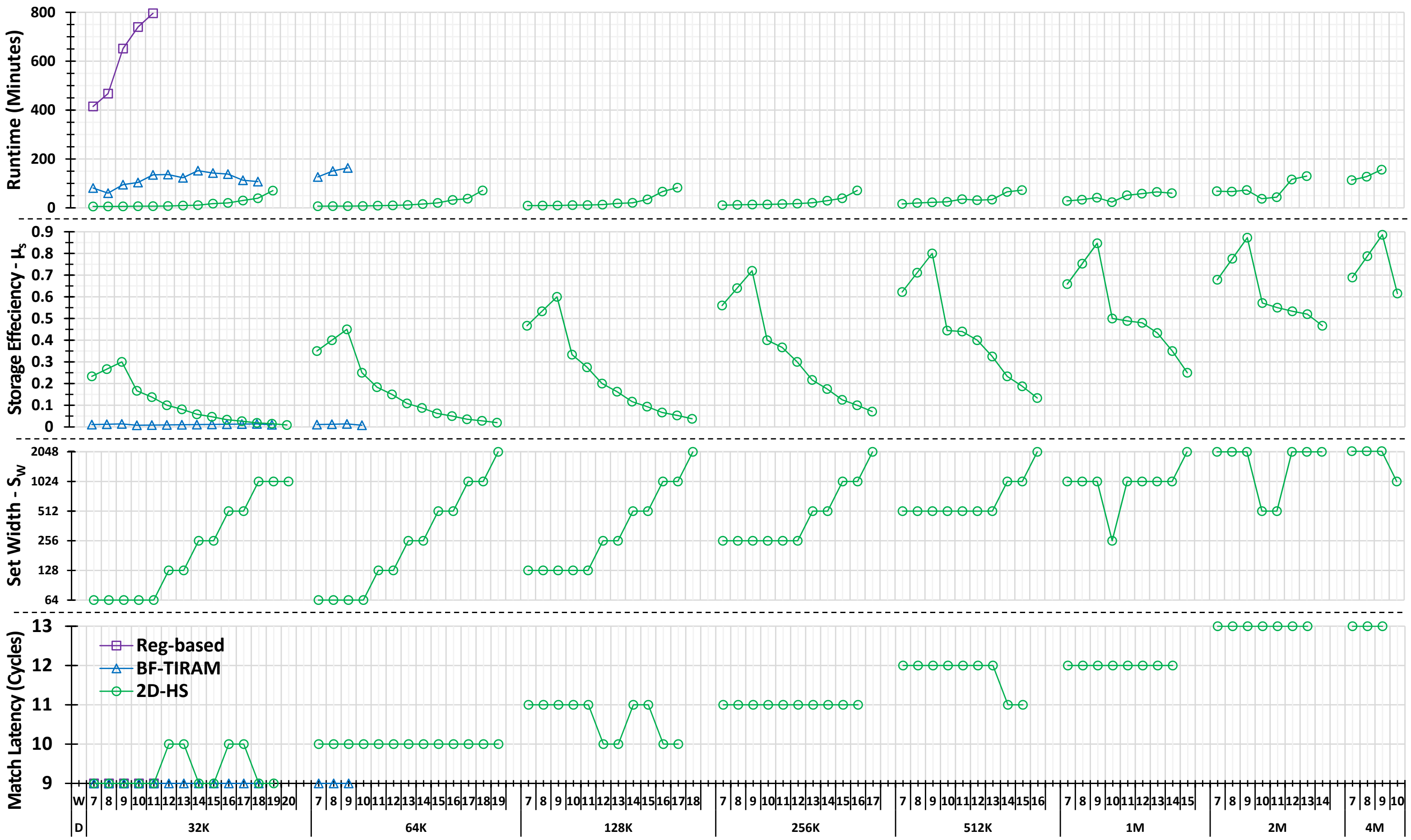


Figure 5.7: Additional results for several BCAM depth and pattern width sweeps (top) Runtime (upper middle) Storage efficiency (lower middle) Set width (bottom) Match latency.

5.6 Conclusions

In this chapter, a novel BCAM architecture for FPGAs is proposed. The approach is fully BRAM-based and employs hierarchical search to reduce BRAM consumption. While traditional brute-force approach have a pattern match indicator for each single address, the proposed approach maintains a single pattern match indicator for each address set. The suggested method is capable of implementing a 4M-entry CAM and significantly improves area and performance. In contrast, traditional methods cannot exceed 64K in depth. For the 64K-entry test-case, the traditional method consumes 43 times more ALMs, 18 times longer mapping runtime, and achieves only one-third of the $\mathrm{F_{max}}$ of the proposed method. The suggested 2D-HS-BCAM design *completely dominates* all past designs in BRAM and ALM consumption, $\mathrm{F_{max}}$ and runtime.

A fully parameterized and Verilog implementation of the suggested methods is provided as open source hardware [114].

Chapter 6

Indirectly Indexed Hierarchical Search BCAMs (II-HS-BCAMs)

In this chapter, a novel, efficient and modular technique for constructing BCAMs out of standard SRAM blocks in FPGAs is proposed. Hierarchical search is employed to achieve high storage efficiency. While pattern width cascading requires producing the match indicator of every single address in every cascaded stage, the previous 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) approach in Chapter 5 provides a single matching address only. Hence, the 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) approach cannot be cascaded in pattern width; this incurs an exponential increase of RAM consumption as pattern width increases. The Indirectly Indexed Hierarchical Search BCAM (II-HS-BCAM) approach, however, efficiently regenerates a match indicator for every single address by storing indirect indices for address match indicators. Hence, the proposed method can be cascaded in pattern width and exponential growth is alleviated into linear. Our method exhibits high storage efficiency and is capable of implementing up to nine times wider BCAMs compared to the previous 2D-HS-BCAM approach. Compared to brute-force techniques, this method requires a maximum of 18% the RAM storage, while enhancing clock speed by 45% on average. A fully parameterized Verilog implementation is being released as an open source library. The library has been extensively tested using Altera's Quartus and ModelSim.

6.1 Introduction

In this chapter, a modular SRAM-based BCAM is proposed. Similar to 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM), our approach arranges the memory into two-dimensional data sets. Our approach, however, is superior to the 2D-HS-BCAM approach since it efficiently regenerates match indicators for every single address by storing indirect indices for address match indicators. Thus, unlike hierarchical search, the proposed method can support wide patterns by pattern width cascading; this transforms exponential RAM growth into linear.

The proposed method is device-independent; hence, it can be applied to any FPGA device containing standard dual-ported BRAMs. The proposed approach dramatically improves CAM area efficiency compared to conventional methods. In contrast to algorithmic approaches (e.g., hashes and tries) or other BCAM techniques that require several nondeterministic cycles to write or match [111–113], our approach is high-throughput and can perform a pattern read (match) every cycle and a pattern write every two cycles.

Major contributions of this chapter are:

- A novel highly efficient BCAM architecture. Compared to other BCAM approaches, the proposed technique provides up to nine times wider BCAMs. To the authors' best knowledge, research and patent literature do not have similar BCAM techniques.

- A parameterized Verilog implementation of our method, together with other approaches. A flow manager to simulate and synthesize designs with various parameters in batch using Altera's ModelSim and Quartus II is also provided. The Verilog modules and the flow manager are available online [115].

To verify correctness, the proposed BCAM architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II [57]. A large variety of BCAM architectures and parameters, e.g., BCAM depth and pattern width are simulated in batch, each with over one million random BCAM write and match cycles. Stratix V, Altera's high-end FPGA, is used to implement and compare the proposed architecture with previous approaches.

The rest of this chapter describes in detail the 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) approach and is organized as follows. The motivation and key idea for this work are explained in Section 6.2. The design method is described in Section 6.3. Section 6.4 describes a device-specific instance for Altera's Stratix device family. Discussion of the suggested method and comparison to previous techniques are provided in Section 6.5. The experimental framework, simulation and synthesis results, are discussed in Section 6.6. Conclusions are drawn in Section 6.7.

6.2 Motivation and Key Idea

As shown in Chapter 5, the previous 2-Dimensional Hierarchical Search BCAMs (2D-HS-BCAMs) cannot be cascaded in pattern width. Hence, the required memory size grows exponentially with pattern width. This limitation is crucial since the vast majority of applications require wide patterns. To support BCAM pattern width cascading, all match indicators for every single BCAM address shall be generated, as in the brute-force approach (Section 2.3.2). However, storing all match indicators requires wide RAM and incurs high memory overhead.

Our proposed Indirectly Indexed Hierarchical Search BCAM (II-HS-BCAM) is based on 2-Dimensional Hierarchical Search BCAMs (2D-HS-BCAMs) and utilizes the sparsity of the Set Transposed Indicators RAM (STIRAM) to store only the required match indicators. It is able to regenerate all match indicators for every BCAM address, allowing it to be cascaded in pattern width. The following theorem is the basic keystone of our technique.

Lemma 6.2.1 (TIRAM column match indicators bound)**.** *The number of binary '1' (matches) in each RAM column (CAM location) of the TIRAM matrix from Equation 2.10, is exactly 1, namely*

$$\forall a \in A : \sum_{p \in P} I_{p,a} = 1. \tag{6.1}$$

Proof. Similar to RAM, each address of a BCAM contains one and only one valid pattern. A pattern p' located at address a' means $I_{p',a'} = 1$ and $I_{p,a'} = 0$ for every $p \neq p'$. Hence, the corresponding column of address a' in the TIRAM matrix has a binary '1' only in $I_{p',a'}$ and zeros for all the other patterns. □

Theorem 6.2.2 (STIRAM column match indicators bound)**.** *The number of binary '1' (matches) in each column (set) of the STIRAM matrix from Equation 5.3 is limited to the set width S_W, namely,*

$$\forall s \in S : \sum_{p \in P} I_{p,s}^{S_W} \leq S_W. \tag{6.2}$$

Proof. A match indicator of a set is defined in Equation 5.2 and Equation 5.3 and indicates if any of the addresses in the set has a match. Given a set s' of S_W addresses, Equation 5.2 and Equation 5.3 provides $I_{p,s'}^{S_W} = \bigvee_{a=S_W \cdot s'}^{a=S_W \cdot (s'+1)-1} I_{p,a}$. Lemma 6.2.1 ensures that each address $a \in \left\{S_W \cdot s', \cdots, S_W \cdot \left(s'+1\right) - 1\right\}$ has one and only one p such that $I_{p,a} = 1$. Since the set s' has S_W addresses, exactly S_W different $I_{p,a} = 1$ exist. Hence, in the entire set, a maximum of S_W patterns have a match (less than S_W in the case where two addresses or more have the same pattern). □

The significance of Theorem 6.2.2 lies in the measurement it provides for the STIRAM matrix sparsity. Namely, it provides an upper bound of the number of binary '1' (matches) for a set (a column in STIRAM). Instead of storing match indicators for every address and pattern pair as in the brute-force approach (Figure 6.1 (left)), or set indicators as in the hierarchical search approach (Figure 6.1 (middle)), we store all address indicators only for sets with a match, as they are

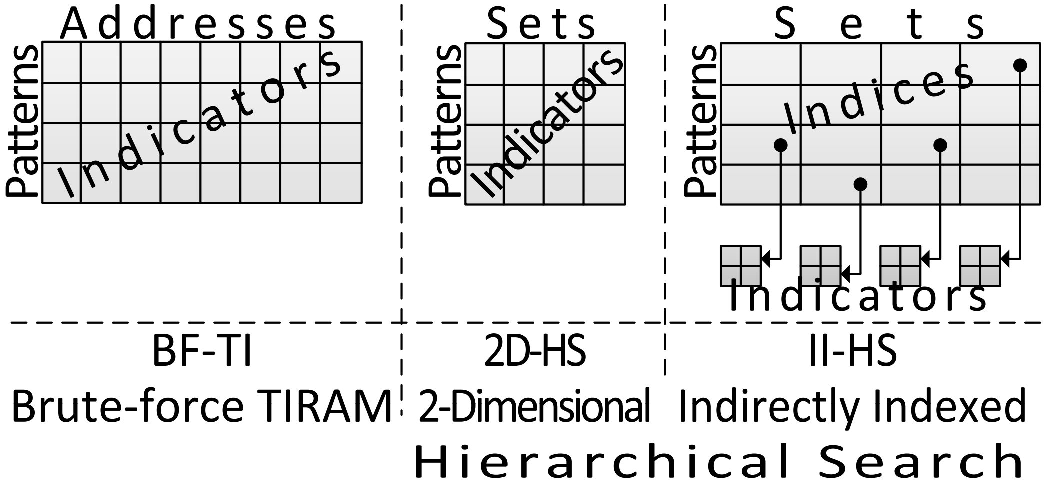


Figure 6.1: Indicators arrangement for three different approaches.

limited to S_W. To reduce memory consumption, address match indicators are saved in another auxiliary structure, while the original STIRAM will hold indices to the auxiliary structure (Figure 6.1 (right)).

6.3 Design and Functionality

As depicted in Figure 6.2, a single-stage of the proposed II-HS-BCAM consists of three parts. First, the *match* RAM where the set indices and match indicators are stored; this is the most memory consuming structure. Second, the *status RAM* where the system status is stored and feeds the control logic with system status. Finally, the *control and steering logic*, which generates control signals to control the entire structure (based on system status), feeds the match RAM with indicators and indices, and updates the status RAM.

1. *Match RAM*: As described in the previous subsection, instead of storing

set match indicators in the STIRAM, we store only indirect indices for an auxiliary RAM, which holds the match indictors for all the addresses in the set, hence it is called the indicators RAM (IndRAM). Theorem 6.2.2 shows that a maximum of S_W different patterns can have a match in a set; hence, the depth of IndRAM is S_W at most. Each set has S_W addresses, therefore IndRAM should have S_W address indicators for each set and its width should also be S_W. The address space consists of $\left\lceil \frac{C_D}{S_W} \right\rceil$ sets; hence, $\left\lceil \frac{C_D}{S_W} \right\rceil$ IndRAM $S_W \times S_W$ blocks are required.

The STIRAM holds indices for all pattern and set pairs. To represent all patterns the required depth is 2^{P_W}. For each of the $\left\lceil \frac{C_D}{S_W} \right\rceil$ sets, $\lceil \log_2 S_W \rceil$ bits are required for each index. In total, the STIRAM width is $\left\lceil \frac{C_D}{S_W} \right\rceil \cdot \lceil \log_2 S_W \rceil$ bits.

When a match operation is performed, all indices associated with the match pattern are read from the STIRAM (a single STIRAM row). Next, these set indices will address the IndRAM to fetch the complete address indicators for every CAM location. Unlike STIRAM, every set in the IndRAM is stored in a separate RAM structure. Hence, rows with different indices can be fetched from each set, depending on the corresponding index from the STIRAM.

Figure 6.2 (bottom) shows an example where pattern ‘4’ is matched. The STIRAM reveals one index in set_1 pointing to address ‘2’ of the IndRAM (shaded field in the STIRAM). Reading address ‘2’ from the IndRAM in set_1 (shaded field in the IndRAM) shows a single match in offset address ‘2’ of *set*$_1$ (circled indicator in the IndRAM).

2. *Status RAM*: The status RAM is updated at each write to reflect the system status and consists of three RAM structures as follows.

   (a) *Sets RAM (SetRAM)*: Similar to the hierarchical search method, this RAM holds all CAM patterns, with each set's patterns packed in one row that can be fetched in a single cycle. Its size is therefore $\left\lceil \frac{C_D}{S_W} \right\rceil \times (P_W \cdot S_W)$. At each write, the SetRAM will be updated with the new pattern.

   (b) *Indices RAM (IdxRAM)*: The Indices RAM stores the index of each pattern in the BCAM, arranged similar to the SetRAM, namely each set's indices in one row. Its size is $\left\lceil \frac{C_D}{S_W} \right\rceil \times \left(\left\lceil \log_2 S_W \right\rceil \cdot S_W \right)$ At each write, the IdxRAM will be updated with the index that have been assigned to the new pattern. For instance, Figure 6.2 (bottom) shows an example where pattern '4' is located in the SetRAM in *set*$_1$, the corresponding field in IdxRAM shows index '2'; the corresponding index in STIRAM.

       The IdxRAM is mainly used when two identical patterns are located in the same set. In this case, the same index should be assigned these two identical patterns. Hence, the index of the old pattern will be read from the IdxRAM and assign back to the new (and identical) pattern.

   (c) *Vacancy RAM (VacRAM)*: The Vacancy RAM indicates for each row of the IndRAM whether it is vacant or holds valid indicators. The

IndRAM consists of $\left\lceil \frac{C_D}{S_W} \right\rceil$ RAM blocks (for each set), each with S_W rows. The Vacancy RAM hold the status of each IndRAM block in one row, hence its depth is $\left\lceil \frac{C_D}{S_W} \right\rceil$ and marks the validity of all the S_W IndRAM rows, hence its width is S_W.

At each write, if a new index needs to be assigned to the new pattern, the VacRAM will be used to detect the first available row of the corresponding set in IndRAM. The VacRAM will be updated to indicate that this IndRAM row has been taken and is not vacant anymore.

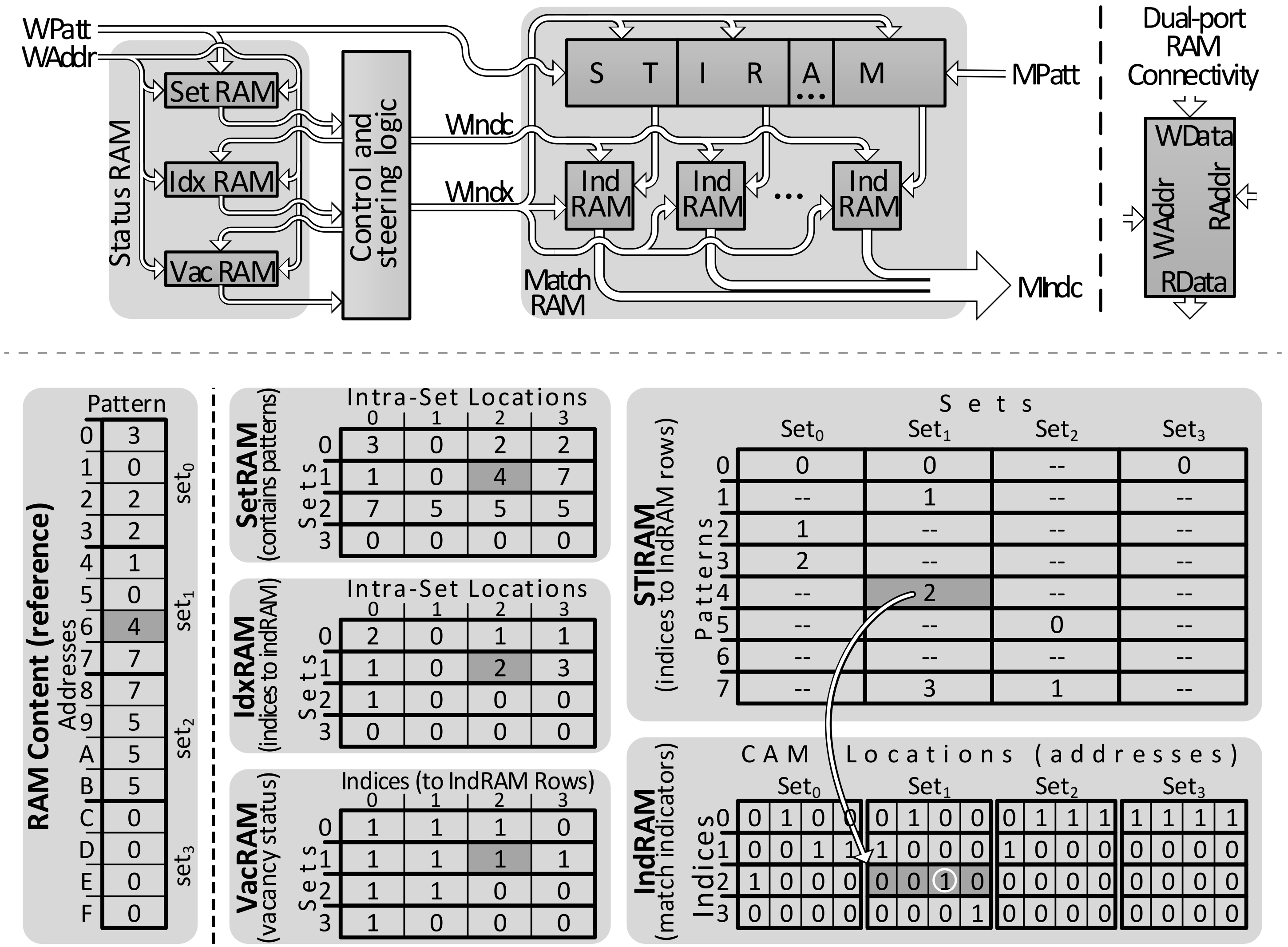
For instance, the example in Figure 6.2 (bottom) shows that the corresponding field in VacRAM of the recently written pattern '4' hold a binary '1' (shaded field in VacRAM) to indicate that the corresponding row in IndRAM has been taken.

3. *Control and steering logic*: Based on current system status and the required write pattern and write address, it generates write indicators to IndRAM and indices to IndRAM and STIRAM. Furthermore, it updates the IdxRAM and VacRAM status.

Table 6.1: BRAM usage of a single[a] stage II2D-BCAM.

Structure	**Blocks#**	**Depth**	**Width**
STIRAM	1	2^{P_W}	$\left\lceil \frac{C_D}{S_W} \right\rceil \cdot \lceil \log_2 S_W \rceil$
IndRAM	$\left\lceil \frac{C_D}{S_W} \right\rceil$	S_W	S_W
SetRAM	1	$\left\lceil \frac{C_D}{S_W} \right\rceil$	$P_W \cdot S_W$
IdxRAM	1	$\left\lceil \frac{C_D}{S_W} \right\rceil$	$\lceil \log_2 S_W \rceil \cdot S_W$
VacRAM	1	$\left\lceil \frac{C_D}{S_W} \right\rceil$	S_W

[a] For pattern width cascaded II2D-BCAM, cascading method from Section 2.3.3. is employed; $P_W = P_{W,opt} = \lfloor \log_2 R_{D,min} \rfloor$ is used for each stage, while $n_c = \left\lceil \frac{P_{w,total}}{P_{w,opt}} \right\rceil$ stages are required.



RAM Content (reference)

Addresses	Pattern	
0	3	set$_0$
1	0	
2	2	
3	2	
4	1	set$_1$
5	0	
6	4	
7	7	
8	7	set$_2$
9	5	
A	5	
B	5	
C	0	set$_3$
D	0	
E	0	
F	0	

SetRAM (contains patterns)

	Intra-Set Locations			
Sets	0	1	2	3
0	3	0	2	2
1	1	0	4	7
2	7	5	5	5
3	0	0	0	0

IdxRAM (indices to indRAM)

	Intra-Set Locations			
Sets	0	1	2	3
0	2	0	1	1
1	1	0	2	3
2	1	0	0	0
3	0	0	0	0

VacRAM (vacancy status)

	Indices (to IndRAM Rows)			
Sets	0	1	2	3
0	1	1	1	0
1	1	1	1	1
2	1	1	0	0
3	1	0	0	0

STIRAM (indices to IndRAM rows)

	Sets			
Patterns	Set$_0$	Set$_1$	Set$_2$	Set$_3$
0	0	0	--	0
1	--	1	--	--
2	1	--	--	--
3	2	--	--	--
4	--	2	--	--
5	--	--	0	--
6	--	--	--	--
7	--	3	1	--

IndRAM (match indicators)

	CAM Locations (addresses)															
	Set$_0$				Set$_1$				Set$_2$				Set$_3$			
Indices 0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 6.2: II2D-BCAM single-stage (top) high-level architecture (bottom) 8×3; $S_W = 4$ example; pattern '4' is highlighted with all related RAM content.

6.4 Feasibility on Altera's Stratix Devices

An area efficient implementation of the proposed II2D-BCAMs requires heterogeneous BRAMs on the FPGA. The relatively small BRAMs will be used to construct IndRAM and store match indicators. A perfect candidate is the LUT configuration RAM, known as LUT RAM, and is supported by both Altera and Xilinx devices. On the other hand, the relatively large BRAMs will be used to construct other structures; M20K BRAMs will be used for this purpose.

Altera's Stratix V memory architecture provides a 640-bit LUT RAM memory called MLAB (Memory Logic Array Block) as well as 20Kb BRAMs called M20K.

Both MLABs and M20Ks are used in their shallowest//widest configuration modes. Hence, the MLABs are 32×20, and the M20Ks are 512×40. Since the depth of MLABs is 32, and they are used to implement the IndRAM, we set $S_W = 32$. With an index width of $\lceil \log_2 S_W \rceil = 5$, a single M20K line can store up to 8 indices. To write a single 5-bit index in the STIRAM, M20K's mixed-width port is used to write a single 5-bit field [55].

6.5 Comparison and Discussion

Table 6.2 shows storage efficiency estimation for different BCAM architectures and is derived from Equation 2.11, Equation 2.13, Equation 5.4 and Table 6.1. The storage efficiency of the uncascaded (in pattern width) Brute-Force Transposed Indicators (BF-TI) implementation is inversely proportional to the exponent of P_W, hence, decays rapidly with P_W increase, as shown in Figure 5.5.

The efficiency of BF-TI when cascading the pattern width is independent

Table 6.2: Storage efficiency μ_s (inversed).

	Uncascaded	**Pattern Width Cascaded**
Brute-force TIRAM (BF-TI)	$\mu_s(BFTI)^{-1} \approx 1 + \frac{2^{P_W}}{P_W}$	$\mu_s(BFTI)^{-1} \approx 1 + \frac{R_{D,min}}{\log_2 R_{D,min}}$
Hierarchical Search (HS)	$\mu_s(2DHS)^{-1} \approx 1 + \frac{2^{P_W}}{P_W \cdot S_W}$	$\mu_s(IIHS)^{-1} \approx 1 + \frac{1+S_W+\log_2 S_W \cdot \left(1+\frac{R_{D,min}}{S_W}\right)}{\log_2 R_{D,min}}$

of the value of P_W. However, the efficiency is affected by an intrinsic BRAM characteristic, the minimal depth $R_{D,min}$ (associated with the maximum width). As shown in Figure 6.3 (top), the efficiency is inversely proportional to $R_{D,min}$, hence, shallow and wide BRAMs with smaller $R_{D,min}$ (and larger $R_{W,max}$) will exhibit higher storage efficiency.

Similar to the uncascaded BF-TI, the efficiency of the 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) is inversely proportional to the exponent of P_W. However, the exponential relation is mitigated by the set width S_W, an external and user-driven parameter. On the other hand, the efficiency of our II-HS-BCAM approach is not related to the pattern width P_W, but is augmented by S_W.

The 2D-HS-BCAM is more efficient than the II-HS-BCAM for narrow patterns, namely, for patterns narrower than a specific pattern width threshold $P_{W,th}$ as follows

$$\mu_s(2DHS) \geq \mu_s(IIHS) \Longleftrightarrow P_W \leq P_{W,th}. \qquad (6.3)$$

To solve the left side inequality, storage efficiency of the 2D-HS-BCAM and the II-HS-BCAM in Table 6.2 are used, hence,

$$
\begin{aligned}
& \mu_s(2DHS) && \geq && \mu_s(IIHS) \\
\iff & \mu_s(2DHS)^{-1} && \leq && \mu_s(IIHS)^{-1} \\
& \text{(substitute storage efficiencies from Table 6.2, where 2D-HS-BCAM has a} \\
& \text{set width of } S_{W,2D} \text{ and II-HS-BCAM has a set width of } S_{W,II}\text{)} \\
\iff & 1+\frac{2^{P_W}}{P_W \cdot S_{W,2D}} && \leq && 1+\frac{1+S_{W,II}+\log_2 S_{W,II}\cdot\left(1+\frac{R_{D,min}}{S_{W,II}}\right)}{\log_2 R_{D,min}} \quad \text{(subtract 1)} \\
\iff & \frac{2^{P_W}}{P_W \cdot S_{W,2D}} && \leq && \frac{1+S_{W,II}+\log_2 S_{W,II}\cdot\left(1+\frac{R_{D,min}}{S_{W,II}}\right)}{\log_2 R_{D,min}} \quad \text{(multiply by } P_W \cdot S_{W,2D}\text{)} \\
\iff & 2^{P_W} \leq P_W \cdot S_{W,2D} \cdot \frac{1+S_{W,II}+\log_2 S_{W,II}\cdot\left(1+\frac{R_{D,min}}{S_{W,II}}\right)}{\log_2 R_{D,min}} \quad \text{(isolate constants)} \\
\iff & 2^{P_W} \leq a \cdot P_W,\ a = S_{W,2D} \cdot \frac{1+S_{W,II}+\log_2 S_{W,II}\cdot\left(1+\frac{R_{D,min}}{S_{W,II}}\right)}{\log_2 R_{D,min}} \\
& \text{(exponential inequality of the form } b^x \leq ax \text{ can be solved with the} \\
& \text{Lambert Product Logarithm function (Omega function), where } W_{-1} \\
& \text{denotes the lower branch of the Lambert Product Logarithm function)} \\
\iff & P_W \leq -\frac{W_{-1}\left(-\frac{\ln 2}{a}\right)}{\ln 2},\ a = S_{W,2D} \cdot \frac{1+S_{W,II}+\log_2 S_{W,II}\cdot\left(1+\frac{R_{D,min}}{S_{W,II}}\right)}{\log_2 R_{D,min}}.
\end{aligned}
\tag{6.4}
$$

From Equation 6.4 and the right side of Equation 6.3, the 2D-HS-BCAM is more efficient than the II-HS-BCAM for pattern narrower than the threshold

$$
P_{W,th} = -\frac{W_{-1}\left(-\frac{\ln 2}{a}\right)}{\ln 2},\ a = S_{W,2D} \cdot \frac{1+S_{W,II}+\log_2 S_{W,II}\cdot\left(1+\frac{R_{D,min}}{S_{W,II}}\right)}{\log_2\left(R_{D,min}\right)}. \tag{6.5}
$$

For the 2D-HS-BCAM approach, the set width S_W has a limitation due to the minimal width of the SetRAM it is stored in. The SetRAM depth is $\lceil \frac{C_D}{S_W} \rceil$ and is limited by $R_{D,min}$. Hence, to achieve maximum efficiency, S_W is bounded by

$$S_W \leq \frac{C_D}{R_{D,min}}. \tag{6.6}$$

On the other hand, the set width S_W for our II-HS-BCAM is bounded by internal RAM parameters. Specifically, the depth of the MLAB (LUTRAM) and the BRAM allowable write port width in mixed-width mode. As described in Section 6.4, Altera's Stratix V MLAB depth (for the widest configuration) is 32, and the mixed-width mode of the M20K supports a native width of$\log_2(32) = 5$ for write data, so $S_W = 32$ is a suitable set width. This restriction is in agreement with Figure 6.3 (bottom) where the storage efficiency μ_s is given as function of the set width S_W. Figure 6.3 (bottom) shows that the storage efficiency decreases for narrow sets since the STIRAM portion of the II-HS-BCAM is not efficiently compressed using narrow sets. On the other hand, storage efficiency decreases for wide sets due to the increase of the IndRAM portion.

Applying typical parameters of Altera's Stratix V M20K BRAM and MLAB LUTRAM, i.e., $R_{D,min} = 512$, $S_{W,II} = 32$, and $S_{W,2D} = 4k$, into Equation 6.5 shows that the previous 2D-HS-BCAM is more efficient than the II-HS-BCAM for up to $P_W = 20$ bits only. For these parameter settings, the storage efficiency of the II-HS-BCAM is $\mu_s = 0.08$. A sweep of different values is shown in Figure 6.3.

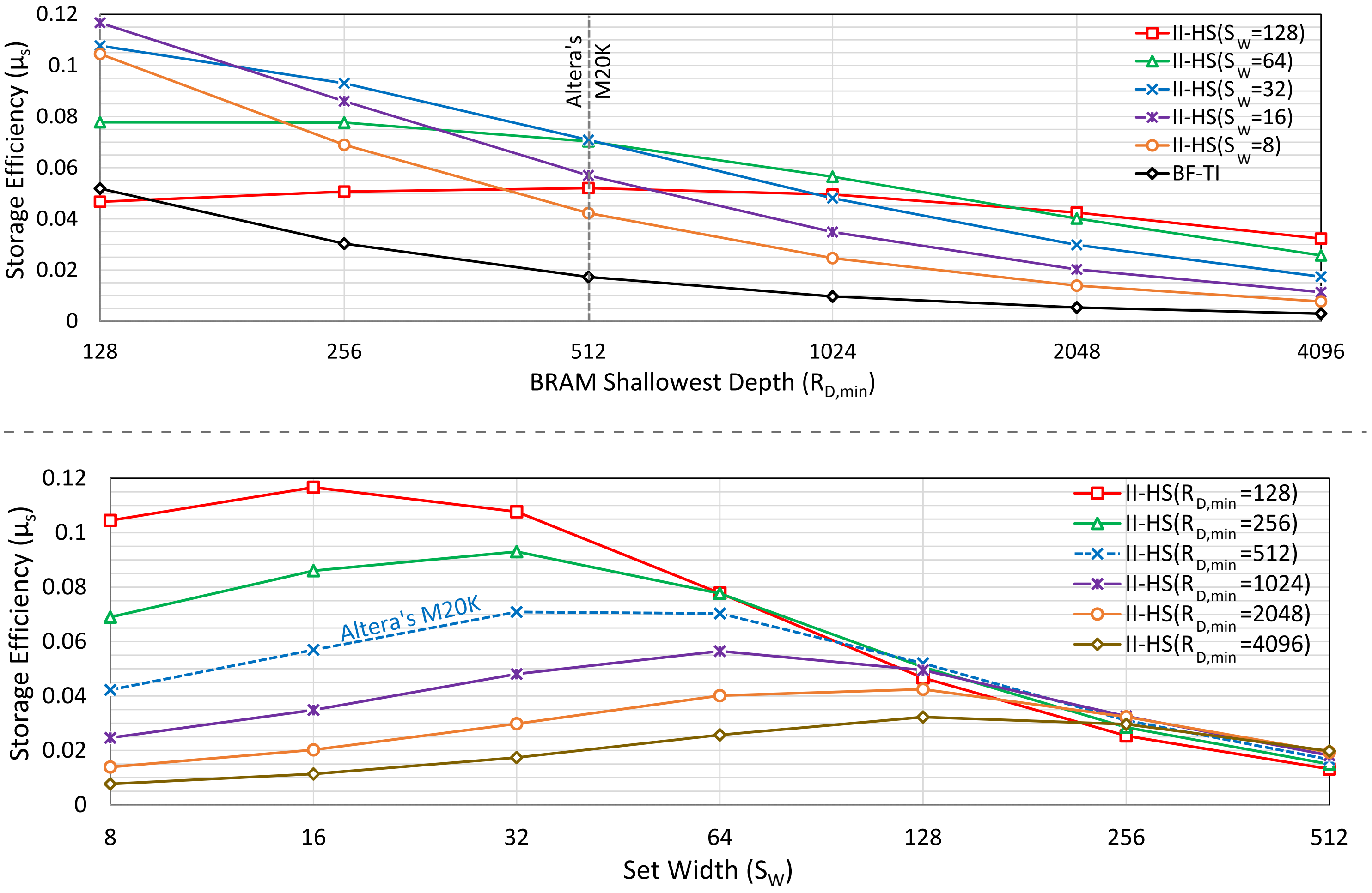


Figure 6.3: Storage Efficiency μ_s as function of (top) BRAM shallowest depth ($R_{D,min}$), and (bottom) set width (S_W) for 2D-HS-BCAM and the pattern width cascaded BF-TI.

6.6 Experimental Results

To verify and simulate the suggested II-HS-BCAM approach and compare to standard and previous techniques, fully parameterized Verilog modules have been developed. Register-based, pattern width cascaded and uncascaded Brute-Force Transposed Indicators (BF-TI), Hierarchical search BCAM, and the proposed II-HS-BCAM methods have been implemented. A run-in-batch flow manager has also been developed to simulate and synthesize these designs with various parameters in batch using Altera's ModelSim and Quartus II. The Verilog modules and the flow manager are available online [115].

To verify correctness, the proposed architecture is simulated using Altera's ModelSim. A large variety of different BCAM architectures and parameters, e.g., BCAM depth and pattern width, are swept and simulated in batch, each with over a million random cycles. All different BCAM design modules were implemented using Altera's Quartus II on Altera's Stratix V `5SGXMABN1F45C2` device. This is a speed grade 2 device with 360k ALMs and 2640 M20Ks. Half of the ALM's can be used to construct MLABs, while a single MLAB consists of 10 ALMs.

Figure 6.4 plots feasible BCAM depth and pattern width sweeps (both P_W and C_D are the x-axis). Within the device limits, the proposed II-HS-BCAM approach is able to reach a 153-bit pattern width, while other approaches cannot exceed 45-bits for 16K entries. The number of Altera's M20K blocks used to implement each BCAM configuration is plotted in Figure 6.4 (bottom). The BF-TI and our II-HS-BCAM exhibit a linear growth of M20K consumption as pattern width increases since both methods are pattern width cascaded. However, the II-HS-BCAM growth rate is lower since addresses are grouped as sets. On the other hand, the 2D-HS-BCAM suffers from exponential growth, hence it cannot exceed a very narrow pattern width of 20 bits.

As shown in Figure 6.4 (middle), the proposed II-HS-BCAM method and the BF-TI also exhibit linear ALM count growth as pattern width increases. The priority-encoder in the 2D-HS-BCAM approach is split in two, hence the ALM count is lower. Furthermore, our II-HS-BCAM approach uses ALMs as MLABs, hence it has higher ALM consumption. The register-based BCAM consumes the most ALMs due to massive register usage.

Figure 6.4 (top) plots the $\mathrm{F_{max}}$ of all BCAM architectures. The 2D-HS-BCAM exhibits the highest $\mathrm{F_{max}}$ for very narrow patterns, where it is feasible. However, $\mathrm{F_{max}}$ drops dramatically as the pattern width increases due to a massive increase of M20K usage. On the other hand, II-HS-BCAM performs better than BF-TI and register-based BCAM for shallow memory. As can be seen, $\mathrm{F_{max}}$ decreases mildly as pattern width increases due to pattern width cascading.

Similar to BF-TI and 2D-HS-BCAM, II-HS-BCAM is capable of matching a pattern every cycle and writing a pattern every two. Pipelining is employed to increase $\mathrm{F_{max}}$ and this adversely increases latency. The longest combinational path goes through the output priority-encoder, and is pipelined every stage. For the device-specific PE described in Appendix B, $\left\lceil \log_4 \left\lceil \frac{C_D}{S_W} \right\rceil \right\rceil$ pipe stages are required. 2D-HS-BCAM benefits from higher S_W, hence it can exhibit lower latency. On the other hand, sets are not used in BF-TI, hence its match latency is $\lceil \log_4 C_D \rceil$.

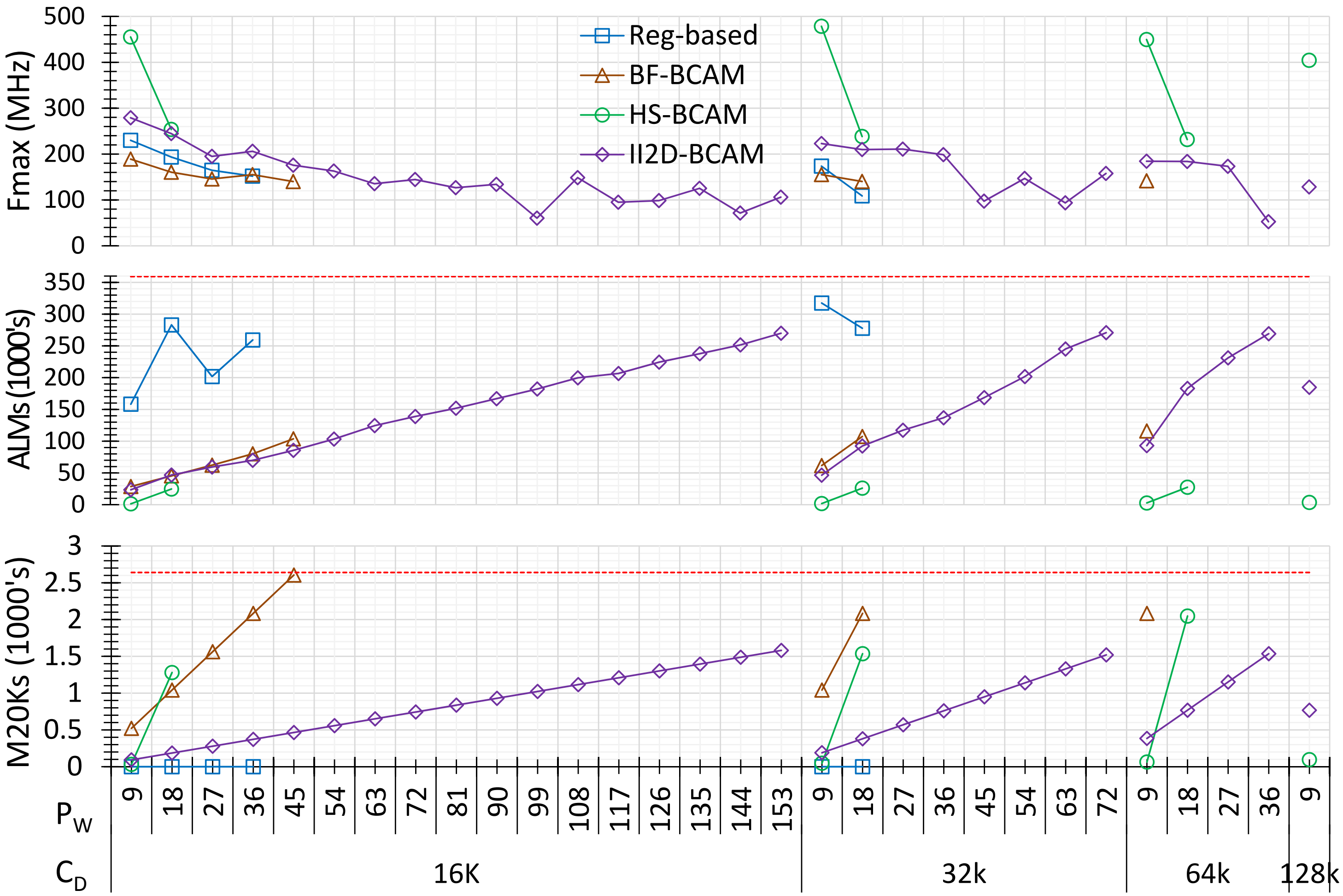


Figure 6.4: Results for several BCAM depth and pattern width sweeps (bottom) M20K count (middle) ALMs count (top) F_{max} at T=0°C.

6.7 Conclusions

In this chapter, a novel BCAM architecture for FPGAs is proposed. The approach is fully BRAM-based and employs hierarchical search to reduce BRAM consumption. While traditional brute-force approaches have a pattern match indicator for every single address, the proposed approach groups addresses into sets and maintains a single pattern match indicator for every set. The hierarchical search BCAMs from Chapter 5 cannot be cascaded in pattern width since they provide a single matching address; this incurs an exponential increase of RAM consumption as pattern width increases. On the other hand, the approach in this chapter efficiently regenerates a match indicator for every single address by storing indirect indices for address match indicators. Hence, the proposed method can be cascaded in pattern width and the exponential growth is alleviated into linear. The storage efficiency of our approach is five times the storage efficiency of the brute-force approach. Furthermore, our technique supports up to nine times wider patterns compared to the 2D-HS-BCAM approach in Chapter 5. Compared to brute-force techniques, this method requires a maximum of 18% the RAM storage, while enhancing clock speed by 45% on average.

A fully parameterized Verilog implementation of the suggested methods is provided as open source hardware [115].

Chapter 7

Conclusions

In this chapter, we summarize the results of this dissertation, which show that the proposed I-LVT dominates and should replace existing LVT and XOR methods for MPRAM. Also, key results and contributions for CAMs are discussed. We conclude with suggestions for future work.

7.1 Dissertation Summary

This dissertation provides an attempt to resolve the memory bottleneck of massively parallel reconfigurable systems by providing efficient, parallel and customizable embedded memory structures. Although concurrent multi-ported memories are important, their high implementation cost means they are used sparingly. As a result, FPGA vendors only provide standard dual-ported memories to handle the majority of usage patterns. This dissertation describes a novel, efficient and modular approach to construct multi-ported memories out of basic dual-ported

Block-RAMs.

Another massively parallel memory structure that is investigated in this dissertation is the content-addressable memory (CAM), a hardware implementation of associative arrays. Despite their importance, FPGAs lack an area-efficient CAM implementation. This dissertation proposes new methods to construct SRAM-based, efficient and modular binary CAMs (BCAMs).

In Chapter 3, an invalidation-based live-value-table, or I-LVT, is used to build modular SRAM-based multi-ported memories. The invalidation-based live-value-table (I-LVT) determines the latest written data bank. The I-LVT generalizes and replaces two prior techniques, the LVT and XOR-based approaches. A general I-LVT is described, along with two specific implementations: binary-coded and thermometer-coded. Both methods are purely SRAM based, so they scale well with memory depth. The original LVT approach can use an infeasible number of registers. In contrast, the I-LVT register usage is not directly proportional to memory depth; hence it requires orders of magnitude fewer registers. Furthermore, the proposed I-LVT method can reduce BRAM consumption up to 44% and improve F_{max} by up to 76% compared to the previous XOR-based approach. The thermometer-coded I-LVT method exhibits the highest F_{max}, while keeping BRAM consumption within 6% of the minimal required BRAM count. Meanwhile, the binary-coded I-LVT uses fewer BRAMs than the thermometer-coded when there are more than 3 write ports. Based on our results, past approaches of XOR and LVT are only recommended for narrow data widths or shallow depths, respectively. In all other cases, the new I-LVT approaches are superior.

In Chapter 4, we have proposed a novel, modular, BRAM-based and switched-multi-ported RAM architecture. In addition to unidirectional ports with fixed read/write, this switched architecture allows a group of write ports to switch with another group of read ports dynamically. The proposed switched-ports architecture is less flexible than a true-multi-ported RAM where each port is switched individually. Nevertheless, switched memories can reduce BRAM consumption compared to true or simple ports for systems with alternating port requirements.

When multiple switched ports exist in a multi-ported RAM design, a memory compiler can be used to create a specific design instance. The compiler must solve a set cover problem to optimize the implementation. Our CAD approach always finds a minimal implementation for all of our test cases, but there is opportunity for further CAD research to improve run-time while still being optimal. On average out of 10 random test-cases, the suggested multi-switched-ports method reduces BRAM use by 18% compared to the best of previous methods, while maintaining ALM count and F_{max}. Future research may address the RAM port assignment problem to more complex cases where there are more than two states governing memory port usage.

In Chapter 5, a novel BCAM architecture for FPGAs is proposed. The approach is fully BRAM-based and employs hierarchical search to reduce BRAM consumption. While the traditional brute-force approach has a pattern match indicator for each single address, the proposed approach maintains a single pattern match indicator for a set of addresses. The suggested method is capable of implementing a 4M-line CAM and significantly improves area and performance. In contrast, traditional methods cannot exceed 64K in depth. The suggested 2-Dimensional Hierarchical Search BCAM (2D-HS-BCAM) design *completely dominates* all past designs in BRAM and ALM consumption, $\mathrm{F}_{\mathrm{max}}$ and runtime. However, this hierarchical search BCAM cannot be cascaded in pattern width since it provides a single matching address; this incurs an exponential increase of RAM consumption as pattern width increases. Hence, it only supports narrow pattern widths.

In Chapter 6, the narrow pattern width restriction is addressed. While traditional brute-force approaches have a pattern match indicator for every single address, the hierarchical approach groups addresses into sets and maintains a single pattern match indicator for every set. The new approach here efficiently regenerates a match indicator for every single address by storing indirect indices for address match indicators. Hence, the proposed method can be cascaded in pattern width and the exponential growth is alleviated into linear. Our technique supports up to four times wider patterns compared to the brute-force BCAM and up to nine times wider patterns compared to the hierarchical search BCAM.

7.2 Future Directions

Future directions of the parallel memory structures described in this dissertation are explored in this section.

7.2.1 Invalidation-Table Multi-Ported Memories

The suggested multi-ported memories can be tested with various other FPGA vendors' tools and devices. Furthermore, these methods can also be tested for ASIC implementation using dual-ported RAMs as building blocks, and compared against memory compiler results. Also, to improve $\mathrm{F_{max}}$, time-borrowing techniques can be utilized. The goal would be to recover the frequency drop due to the multi-ported RAM additional logic, feedback and bank selection logic. One possible approach uses shifted clocks to provide more reading and writing time [116]. However, adapting this method to multi-ported memories is not trivial due to internal timing paths across the I-LVT.

The true-multi-ported RAM proposed by others [56, 73] can utilize our I-LVT method to implement BRAM-based LVT instead of register-based LVT, which eliminates the need of register-based memories and allows higher RAM capacities.

There is opportunity for further CAD research to improve run-time of the multi-switched-ports RAM compiler while still being optimal. Future research may address the RAM port assignment problem to more complex cases where there are more than two states governing memory port usage.

7.2.2 BRAM-Based Content-Addressable Memories

The suggested BCAMs can be tested with other FPGA vendors' tools and devices. Furthermore, these methods can be tested for ASIC implementation using dual-ported RAMs as building blocks, and compared against custom-designed BCAMs.

Increased pipelining and time-borrowing techniques can be used to improve $\mathrm{F_{max}}$. The goal would be to recover the frequency drop due to the comparators and priority-encoder. One possible approach uses shifted clocks to provide more reading and writing time [116]. However, adapting this method to BCAMs is not trivial due to internal timing paths across the BCAM.

To fit some applications that require single-cycle writing, e.g., caches and TLBs, the proposed technique can be enhanced to support single cycle write by using multi-write RAM [1].

7.2.3 Parallel Reconfigurable Computing with Customizable Concurrent Memories

Since FPGAs must fix their RAM block designs for generic designs, it is too costly to provide highly specialized RAMs with a large number of ports. Due to this restriction, state-of-the-art parallel reconfigurable systems support reconfigurable distributed dual-ported memory blocks only. As future research, reconfigurable systems that can take advantage of the multi-ported memories and content-addressable memories proposed in this dissertation can be investigated. Future research could also perform a design space exploration for new parallel reconfigurable architectures with customizable concurrent memories.

Bibliography

[1] A. M. S. Abdelhadi and G. G. F. Lemieux. Modular Multi-Ported SRAM-Based Memories. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 35–44, February 2014. → pages iv, v, 10, 81, 109, 111, 115, 117, 118, 121, 122, 170

[2] Ameer M.S. Abdelhadi and Guy G.F. Lemieux. Modular Switched Multi-Ported SRAM-Based Memories. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 9(3):22:1–22:26, July 2016. → pages iv, vi, 11, 96, 115, 116, 117, 118, 121, 122

[3] A. M. S. Abdelhadi and G. G. F. Lemieux. A Multi-Ported Memory Compiler Utilizing True Dual-Port BRAMs. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 200–207, May 2016. → pages v, vi

[4] A. M. S. Abdelhadi and G. G. F. Lemieux. Deep and Narrow Binary Content-Addressable Memories Using FPGA-Based BRAMs. In *International Conference on Field-Programmable Technology (FPT)*, pages 318–321, December 2014. → pages v, vi, 12

[5] A. M. S. Abdelhadi and G. G. F. Lemieux. Modular SRAM-Based Binary Content-Addressable Memories. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 207–214, May 2015. → pages iv, v, vi, 13

[6] J. H. Tseng and K. Asanović. Banked Multiported Register Files for High-Frequency Superscalar Microprocessors. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 62–71, June 2003. → pages 3, 22

[7] J. A. Fisher. Very Long Instruction Word Architectures and the ELI-512. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 140–150, June 1983. → pages 3

[8] E. S. Fetzer and J. T. Orton. A Fully-Bypassed 6-Issue Integer Datapath and Register File on an Itanium Microprocessor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 420–478, February 2002. → pages 3

[9] H. Bajwa and X. Chen. Low-Power High-Performance and Dynamically Configured Multi-Port Cache Memory Architecture. In *International Conference on Electrical Engineering (ICEE)*, pages 1–6, April 2007. → pages 3

[10] Z. Kwok and S. J. E. Wilton. Register File Architecture Optimization in a Coarse-Grained Reconfigurable Architecture. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 35–44, April 2005. → pages 3

[11] C. E. LaForest and J. G. Steffan. Efficient Multi-Ported Memories for FPGAs. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 41–50, February 2010. → pages 4, 10, 23, 26, 83, 117

[12] C. E. Laforest, M. G. Liu, E. R. Rapati, and J. G. Steffan. Multi-Ported Memories for FPGAs via XOR. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 209–218, February 2012. → pages 4, 10, 23, 26, 83

[13] Sateh M. Jalaleddine. Associative Memories and Processors: The Exact Match Paradigm. *Journal of King Saud University Computer and Information Sciences*, 11:45–67, January 1999. → pages 5

[14] M. Peng and S. Azgomi. Content-Addressable Memory (CAM) and its Network Applications. In *International IC–Taipei Conference*, pages 1–3, May 2001. → pages 5

[15] Qutaiba Ali. A Flexible Design of Network Devices Using Reconfigurable Content Addressable Memory. *International Arab Journal of Information Technology (IAJIT)*, 8(3):235–243, July 2011. → pages

[16] H. Chen, Y. Chen, and D. H. Summerville. A Survey on the Application of FPGAs for Network Infrastructure Security. *IEEE Communications Surveys Tutorials*, 13(4):541–561, April 2011. → pages

[17] K. McLaughlin, N. O'Connor, and S. Sezer. Exploring CAM Design For Network Processing Using FPGA Technology. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW)*, pages 84–84, February 2006. → pages 5

[18] I. Y.-L. Hsiao and C.-W. Jen. A New Hardware Design and FPGA Implementation for Internet Routing Towards IP over WDM and Terabit Routers. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 387–390 vol.1, May 2000. → pages 5, 40

[19] W. Jiang, Q. Wang, and V. K. Prasanna. Beyond TCAMs: An SRAM-Based Parallel Multi-Pipeline Architecture for Terabit IP Lookup. In *IEEE Conference on Computer Communications (INFOCOM)*, April 2008. → pages

[20] H. Le, W. Jiang, and V. K. Prasanna. A SRAM-Based Architecture for Trie-based IP Lookup Using FPGA. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 33–42, April 2008. → pages

[21] H. Le, W. Jiang, and V. K. Prasanna. Scalable High-Throughput SRAM-Based Architecture for IP-Lookup Using FPGA. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 137–142, September 2008. → pages

[22] H. Le and V. K. Prasanna. Scalable High Throughput and Power Efficient IP-Lookup on FPGA. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 167–174, April 2009. → pages 40

[23] D. Unnikrishnan, R. Vadlamani, Y. Liao, A. Dwaraki, J. Crenne, L. Gao, and R. Tessier. Scalable Network Virtualization Using FPGAs. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 219–228, February 2010. → pages

[24] N. Mudaliar. Design and Implementation of MobilityFirst Router on the NetFPGA Platform. Master's thesis, Dept. of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, New Brunswick, NJ, May 2013. → pages 5

[25] L. Bu and J. A. Chandy. FPGA Based Network Intrusion Detection Using Content Addressable Memories. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 316–317, April 2004. → pages 5

[26] B.-K. Kim, Y.-J. Heo, and J.-T. Oh. High-Performance Intrusion Detection in FPGA-Based Reconfiguring Hardware. In *Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 563–573, September 2005. → pages

[27] A. Kaleel Rahuman and G. Athisha. Reconfigurable Hardware Architecture for Network Intrusion Detection System. *American Journal of Applied Sciences*, 9(10):1618–1624, October 2012. → pages

[28] H. Song and J. W. Lockwood. Efficient Packet Classification for Network Intrusion Detection Using FPGA. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 238–245, February 2005. → pages

[29] F. Yu. *High Speed Deep Packet Inspection with Hardware Support*. Phd thesis, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA, November 2006. → pages 5

[30] V. Puš and J. Korenek. Fast and Scalable Packet Classification Using Perfect Hash Functions. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 229–236, February 2009. → pages 5, 40

[31] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, and V. Prasanna. Multi-Dimensional Packet classification on FPGA: 100 Gbps and Beyond. In *International Conference on Field-Programmable Technology (FPT)*, pages 241–248, December 2010. → pages

[32] R. Wei, Y. Xu, and H. Chao. Block Permutations in Boolean Space to Minimize TCAM for Packet Classification. In *IEEE Conference on*

Computer Communications (INFOCOM), pages 2561–2565, March 2012. → pages 5

[33] U. Dhawan and A. DeHon. Area-Efficient Near-Associative Memories on FPGAs. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 191–200, February 2013. → pages 5, 40

[34] H. Wong, V. Betz, and J. Rose. Comparing FPGA vs. Custom CMOS and the Impact on Processor Microarchitecture. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 5–14, February 2011. → pages 5

[35] H. Wong, V. Betz, and J. Rose. Quantifying the Gap Between FPGA and Custom CMOS to Aid Microarchitectural Design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2067–2080, October 2014. → pages 5

[36] H. Wong, V. Betz, and J. Rose. Efficient Methods for Out-of-Order Load/Store Execution For High-Performance Soft Processors. In *International Conference on Field-Programmable Technology (FPT)*, pages 442–445, December 2013. → pages 5

[37] Pedro Alcocer and Colin Phillips. Using Relational Syntactic Constraints in Content-Addressable Memory Architectures for Sentence Parsing. *Special Issue of Topics in Cognitive Science on Computational Psycholinguistics*, April 2012. → pages 5

[38] K. Suman N. Manonmani and C.Udhayakumar. Dynamic Based Reconfigurable Content Addressable Memory for Fast String Matching. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, 3(1):429–435, January 2014. → pages

[39] G. Nilsen. A Variable Word-Width Content Addressable Memory (CAM) for Fast String Matching. Master's thesis, Dept. of Informatics, University of Oslo, Oslo, Norway, May 2013. → pages 5

[40] R.-Y. Yang and C.-Y. Lee. High-Throughput Data Compressor Designs Using Content Addressable Memory. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 147–150 vol.4, May 1994. → pages 5

[41] J. V. Oldfield, R. D. Williams, N. E. Wiseman, and M. R. Brule. Content-Addressable Memories for Quadtree-Based Images. In *Eurographics Conference on Advances in Computer Graphics Hardware (EGGH)*, pages 67–84, September 1988. → pages 5

[42] Y. C. Shin, R. Sridhar, V. Demjanenko, P. W. Palumbo, and S. N. Srihari. A Special-Purpose Content Addressable Memory Chip for Real-Time Image Processing. *IEEE Journal of Solid-State Circuits (JSSC)*, 27(5):737–744, May 1992. → pages 5

[43] D. Agrawal and A. E. Abbadi. Hardware Acceleration for Database Systems Using Content Addressable Memories. In *International Workshop on Data Management on New Hardware (DaMoN)*, June 2005. → pages 5

[44] A. Goel and P. Gupta. Small Subset Queries and Bloom Filters Using Ternary Associative Memories, with Applications. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 143–154, June 2010. → pages 5

[45] S. A. Guccione and E. Keller. Gene Matching Using JBits. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 1168–1171, September 2002. → pages 5

[46] R. V. Satya, A. Mukherjee, and U. Ranga. A Pattern Matching Algorithm for Codon Optimization and CpG Motif-Engineering in DNA Expression Vectors. In *IEEE Computer Society Conference on Bioinformatics (CSB)*, pages 294–305, August 2003. → pages 5

[47] S. Ahmad and R. Mahapatra. TCAM Enabled On-chip Logic Minimization. In *Design Automation Conference (DAC)*, pages 678–683, June 2005. → pages 5

[48] Y. Tatsumi and H. J. Mattausch. Fast Quadratic Increase of Multiport-Storage-Cell Area with Port Number. *Electronics Letters*, 35(25): 2185–2187, December 1999. → pages 7

[49] K. Pagiamtzis and A. Sheikholeslami. Content-Addressable Memory (CAM) Circuits and Architectures: a Tutorial and Survey. *IEEE Journal of Solid-State Circuits (JSSC)*, 41(3):712–727, March 2006. → pages 8, 29

[50] J. P. Wade and C. G. Sodini. A Ternary Content Addressable Search Engine. *IEEE Journal of Solid-State Circuits (JSSC)*, 24(4):1003–1013, August 1989. → pages

[51] A. J. McAuley and C. J. Cotton. A Reconfigurable Content Addressable Memory. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 24.1/1–24.1/4, May 1990. → pages

[52] K. J. Schultz and P. G. Gulak. Fully Parallel Integrated CAM/RAM Using Preclassification to Enable Large Capacities. *IEEE Journal of Solid-State Circuits (JSSC)*, 31(5):689–699, May 1996. → pages

[53] K. J. Schultz and P. G. Gulak. Fully-Parallel Multi-Megabit Integrated CAM/RAM Design. In *IEEE International Workshop on Memory Technology, Design and Testing (MTDT)*, pages 46–51, August 1994. → pages 8, 29

[54] Altera Corp. *APEX 20K Programmable Logic Device Family Data Sheet*. San Jose, CA, USA, March 2004. Version 5.1. → pages 8, 29, 41

[55] Altera Corp. *Stratix V Device Handbook*. San Jose, CA, USA, May 2013. → pages 9, 14, 15, 30, 32, 38, 58, 115, 126, 140, 156, 188

[56] J. Choi, K. Nam, A. Canis, J. Anderson, S. Brown, and T. Czajkowski. Impact of Cache Architecture and Interface on Performance and Area of FPGA-Based Processor/Parallel-Accelerator Systems. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 17–24, April 2012. → pages 11, 25, 83, 96, 115, 117, 118, 121, 122, 169

[57] Altera Corp. *Quartus II Handbook*. San Jose, CA, USA, November 2013. Version 13.1. → pages 14, 126, 147

[58] A. M. S. Abdelhadi. GitHub Repository, 2014. URL https://github.com/AmeerAbdelhadi. Accessed September 2016. → pages 14

[59] G. A. Malazgirt, H. E. Yantir, A. Yurdakul, and S. Niar. Application Specific Multi-Port Memory Customization in FPGAs. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 1–4, September 2014. → pages 19

[60] H. E. Yantir and A. Yurdakul. An Efficient Heterogeneous Register File Implementation for FPGAs. In *International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*, pages 293–298, May 2014. → pages

[61] H. E. Yantir. A Systematic Approach for Register File Design in FPGAs. Master's thesis, Dept. of Computer Engineering, Boğaziçi University, Istanbul, Turkey, January 2014. → pages

[62] H. E. Yantir, S. Bayar, and A. Yurdakul. Efficient Implementations of Multi-pumped Multi-port Register Files in FPGAs. In *Euromicro Conference on Digital System Design (DSD)*, pages 185–192, September 2013. → pages 19

[63] A. Muddebihal. Area Efficient Multi-Ported Memories with Write Conflict Resolution. Master's thesis, Dept. of Electrical Engineering and Computing Systems, University of Cincinnati, Cincinnati, OH, April 2014. → pages 19

[64] B. A. Chappell, T. I. Chappell, M. K. Ebcioglu, and S. E. Schuster. Virtual Multi-Port RAM Employing Multiple Accesses During Single Machine Cycle, July 1996. US Patent 5,542,067. → pages 19

[65] H. Yokota. Multiport Memory System, May 1990. US Patent 4,930,066. → pages 19

[66] G. S. Ditlow, R. K. Montoye, S. N. Storino, S. M. Dance, S. Ehrenreich, B. M. Fleischer, et al. A 4R2W Register File for a 2.3GHz Wire-Speed POWER$^{\text{TM}}$ Processor with Double-Pumped Write Operation. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 256–258, February 2011. → pages 19, 21

[67] R. E. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro*, 19(2):24–36, March 1999. → pages 21

[68] K. R. Townsend, O. G. Attia, P. H. Jones, and J. Zambreno. A Scalable Unsegmented Multiport Memory for FPGA-Based Systems. *International Journal of Reconfigurable Computing*, December 2015. → pages 22

[69] H. J. Mattausch. Hierarchical N-Port Memory Architecture Based on 1-Port Memory Cells. In *European Solid-State Circuits Conference (ESSCIRC)*, pages 348–351, September 1997. → pages

[70] W. Ji, F. Shi, B. Qiao, and H. Song. Multi-Port Memory Design Methodology Based on Block Read and Write. In *IEEE International Conference on Control and Automation (ICCA)*, pages 256–259, May 2007. → pages

[71] W. Zuo, Q. Zuo, and J. Li. An Intelligent Multi-Port Memory. In *International Symposium on Intelligent Information Technology Application Workshops (IITAW)*, pages 251–254, December 2008. → pages 22

[72] D. Alpert and D. Avnon. Architecture of the Pentium Microprocessor. *IEEE Micro*, 13(3):11–21, May 1993. → pages 22

[73] C.E. LaForest, Z. Li, T. O'Rourke, M.G. Liu, and J.G. Steffan. Composing Multi-Ported Memories on FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(3):16:1–16:23, September 2014. → pages 23, 25, 26, 169

[74] V. R. K. Naresh, D. J. Palframan, and M. H. Lipasti. CRAM: Coded Registers for Amplified Multiporting. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 196–205, December 2011. → pages 27

[75] K. Locke. Parameterizable Content-Addressable Memory, 2011. Application Note XAPP1151. → pages 34, 36, 41

[76] J.-L. Brelet. Using Block RAM for High Performance Read/Write CAMs, 2000. Application Note XAPP204. → pages 41

[77] J.-L. Brelet and L. Gopalakrishnan. Using Virtex-II Block RAM for High Performance Read/Write CAMs, 2002. Application Note XAPP260. → pages

[78] J. L. Brelet. Methods for Implementing CAM Functions Using Dual-Port RAM, March 2002. US Patent 6,353,332. → pages 34, 36, 41

[79] Altera Corp. Implementing High-Speed Search Applications with Altera CAM, July 2001. Application Note 119, Version 2.1. → pages 34, 36

[80] C. A. Zerbini and J. M. Finochietto. Performance Evaluation of Packet Classification on FPGA-Based TCAM Emulation Architectures. In *IEEE Global Communications Conference (GLOBECOM)*, pages 2766–2771, December 2012. → pages 34

[81] W. Jiang. Scalable Ternary Content Addressable Memory Implementation Using FPGAs. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 71–82, October 2013. → pages 35

[82] Z. Ullah, K. Ilgon, and S. Baeg. Hybrid Partitioned SRAM-Based Ternary Content Addressable Memory. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(12):2969–2979, December 2012. → pages 35

[83] Z. Ullah, M. K. Jaiswal, Y. C. Chan, and R. C. C. Cheung. FPGA Implementation of SRAM-Based Ternary Content Addressable Memory. In *International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*, pages 383–389, May 2012. → pages

[84] Z. Ullah, M. K. Jaiswal, and Cheung R. C. C. Design Space Explorations of Hybrid-Partitioned TCAM (HP-TCAM). In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 1–4, September 2013. → pages

[85] Z. Ullah, M. K. Jaiswal, and R. C. C. Cheung. Z-TCAM: An SRAM-Based Architecture for TCAM. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(2):402–406, February 2015. → pages 35

[86] S. Guccione, D. Levi, and D. Downs. A Reconfigurable Content Addressable Memory. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 882–889, May 2000. → pages 38

[87] S. A. Guccione, D. Levi, and D. J. Downs. Content-Addressable Memory Implemented Using Programmable Logic, February 2002. US Patent 6,351,143. → pages

[88] J. Ditmar, K. Torkelsson, and A. Jantsch. A Dynamically Reconfigurable FPGA-Based Content Addressable Memory for Internet Protocol Characterization. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 19–28, August 2000. → pages

[89] Z. Baruch and C. Savin. Reconfigurable Content-Addressable Memory. In *IEEE International Conference on Intelligent Engineering Systems (INES)*, pages 459–463, September 2004. → pages

[90] P. B. James-Roxby and D. J. Downs. An Efficient Content-Addressable Memory Implementation Using Dynamic Routing. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 81–90, March 2001. → pages

[91] A. Jamadarakhani and S. K. Ranchi. Implementation and Design of High Speed FPGA-Based Content Addressable Memory. *International Journal for Scientific Research and Development (IJSRD)*, 1(9):1835–1842, December 2013. → pages

[92] Q. Ibrahim. Design and Implementation of High Speed Network Devices Using SRL16 Reconfigurable Content Addressable Memory (RCAM). *International Arab Journal of e-Technology (IAJeT)*, 2(2):72–81, June 2011. → pages 38

[93] S. Guccione, D. Levi, and P. Sundararajan. JBits: Java Based Interface for Reconfigurable Computing. In *International Conference on Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD)*, September 1999. → pages 38

[94] H. Qin, T. Sasao, and J. T. Butler. *Reconfigurable Computing: Architectures and Applications*, chapter Implementation of LPM Address Generators on FPGAs, pages 170–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-36863-2. → pages 39

[95] H. Qin, T. Sasao, and J. T. Butler. On the Design of LPM Address Generators Using Multiple LUT Cascades on FPGAs. *International Journal of Electronics*, 94(5):451–467, May 2007. → pages

[96] T. Sasao and J. T. Butler. Implementation of Multiple-Valued CAM Functions by LUT Cascades. In *IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, pages 11–11, May 2006. → pages

[97] H. Nakahara, T. Sasao, and M. Matsuura. A CAM Emulator Using Look-Up Table Cascades. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, March 2007. → pages 39

[98] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 3rd edition, 2009. ISBN 978-0-262-03384-8. → pages 40

[99] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509, 2003. → pages 40

[100] Y. Qiao, T. Li, and S. Chen. Fast Bloom Filters and Their Generalization. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 25(1): 93–103, January 2014. → pages 40

[101] J.-L. Brelet. An Overview of Multiple CAM Designs in Virtex Family Devices, 1999. Application Note XAPP201. → pages 41

[102] J.-L. Brelet and B. New. Designing Flexible, Fast CAMs With Virtex Family FPGAs, 1999. Application Note XAPP203. → pages 41

[103] Lattice Semiconductor Corp. Content Addressable Memory (CAM) Applications for ispXPLD Devices, 2002. Application Note AN8071. → pages 41

[104] Actel Corp. Content-Addressable Memory (CAM) in Actel Devices, 2002. Application Note AC194. → pages 41

[105] A. M. S. Abdelhadi and G. G. F. Lemieux. Multi-Ported RAM Verilog Source Code, 2015. URL https://github.com/AmeerAbdelhadi/Multiported-RAM. Accessed September 2016. → pages 44, 68, 76

[106] A. M. S. Abdelhadi and G. G. F. Lemieux. Switched Multi-Ported RAM Verilog Source Code, 2015. URL https://github.com/AmeerAbdelhadi/Switched-Multiported-RAM. Accessed September 2016. → pages 82, 123

[107] A. M. S. Abdelhadi and G. G. F. Lemieux. Multi-Ported Memory Compiler Verilog Source Code, 2016. URL https://github.com/AmeerAbdelhadi/Multiported-RAM-Compiler. Accessed September 2016. → pages 97, 114, 123

[108] R. M. Karp. *Complexity of Computer Computations*, chapter Reducibility among Combinatorial Problems, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-468-42001-2. → pages 104

[109] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press series. Duxbury Press,

Pacific Grove, CA, 2nd edition, 2003. ISBN 978-0-534-38809-6. → pages 114

[110] GLPK (GNU Linear Programming Kit, 2012. URL https://www.gnu.org/software/glpk/. Accessed September 2016. → pages 114

[111] S. J. E. Wilton, C. W. Jones, and J. Lamoureux. An Embedded Flexible Content-Addressable Memory Core for Inclusion in a Field-Programmable Gate Array. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages II–885–8 Vol.2, May 2004. → pages 125, 146

[112] C. W. Jones and S. J. E. Wilton. Content-Addressable Memory with Cascaded Match, Read and Write Logic in a Programmable Logic Device, September 2003. US Patent 6,622,204. → pages

[113] G. R. Schlacter. Emulation of Content-Addressable Memories, June 2004. US Patent 6,754,766. → pages 125, 146

[114] A. M. S. Abdelhadi and G. G. F. Lemieux. 2D Binary Content-Addressable Memory (BCAM) Verilog Source Code, 2015. URL https://github.com/AmeerAbdelhadi/2D-Binary-Content-Addressable-Memory-BCAM. Accessed September 2016. → pages 126, 139, 144

[115] A. M. S. Abdelhadi and G. G. F. Lemieux. Indirectly Indexed 2D Binary Content-Addressable Memory (BCAM) Verilog Source Code, 2016. URL https://github.com/AmeerAbdelhadi/Indirectly-Indexed-2D-Binary-Content-Addressable-Memory-BCAM. Accessed September 2016. → pages 147, 161, 164

[116] A. Brant, A. Abdelhadi, A. Severance, and G. G. F. Lemieux. Pipeline Frequency Boosting: Hiding Dual-Ported Block RAM Latency Using Intentional Clock Skew. In *International Conference on Field-Programmable Technology (FPT)*, pages 235–238, December 2012. → pages 169

Appendix A

Brute-Force Transposed Indicators (BF-TI) BCAM Writing Mechanism

Writing to the Brute-Force Transposed Indicators (BF-TI) structure requires setting the new indicator and clearing the old indicator. As shown in Figure 2.11, a RefRAM is used in parallel to the Transposed Indicators RAM (TIRAM) in order to track the BCAM content and provide for a given address what pattern is already stored and should be removed. This is useful for the BCAM writing operation where the old indicator should be cleared; RefRAM will provide the old pattern in the current written address. BCAM writing will consume two cycles as follows.

1. Set cycle:

   1.1. Set (write '1') a new pattern indicator to TIRAM

   1.2. Read old data (pattern) from RefRAM

2. Clear cycle:

   2.1. Clear (write '0') the old indicator form the TIRAM (location is already provided by step 1.2)

   2.2. Write new data (pattern) to RefRAM

Writing to the TIRAM structure requires writing to a single bit in the TIRAM to set or clear a pattern indicator. As described in Figure A.1 (top), this can be achieved by employing the BRAM mixed-width capability in simple dual-ported mode supported by most FPGA vendors. The writing port width is set to a single bit, while the reading port is set to the maximum available width. The byte-enable functionality can also be utilized to write part of the data line as described in Figure A.1 (middle); however, usually byte-enables do not control fine-grained parts of the data, which make it impractical for TIRAM implementation. Both mixed-width and byte-enable methods can be combined as shown in Figure A.1 (bottom).

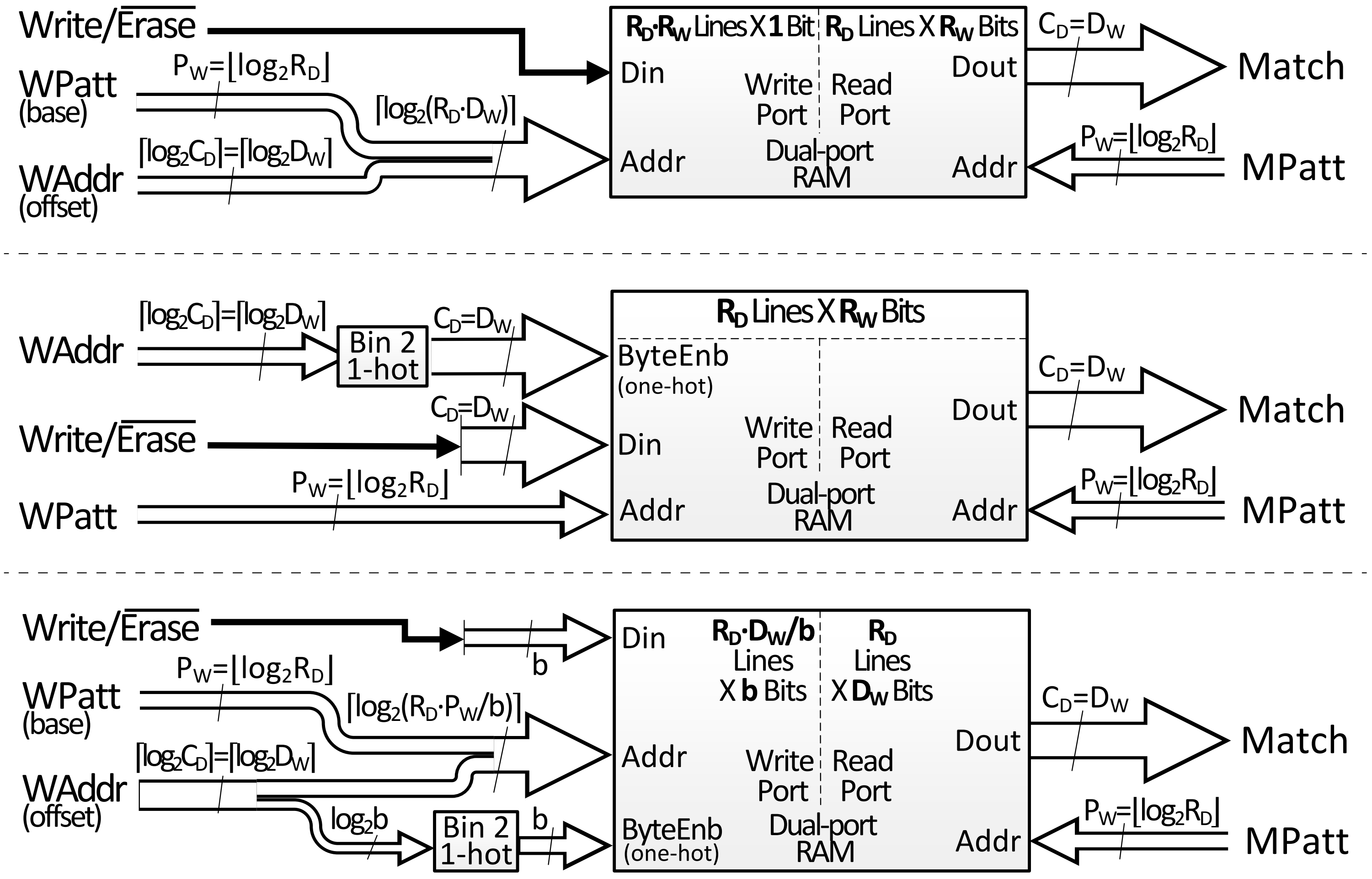


Figure A.1: Transposed Indicators RAM (TIRAM) implementation using (top) Mixed-width BRAM (middle) Byte-enable (bottom) Combined methods.

Appendix B

Wide Priority Encoders in FPGAs

The proposed 2D-HS-BCAM technique successfully reduces the width of the C_D wide priority-encoder used by the brute-force approach and splits it into two narrower priority-encoders. However, priority-encoder delay still affects overall performance. Hence, a fast priority-encoder design is essential.

Our proposed BCAM architecture consists of a $\left\lceil \frac{C_D}{S_W} \right\rceil$ wide priority-encoder and another S_W width priority-encoder. For deep BCAMs, the priority-encoder delay considerably affects the overall performance; hence, a fast priority-encoder design is essential.

A priority-encoder, also called Leading Zero Detector (LZD) or Leading Zero Counter (LZC), receives an n-bit input vector and detects the index of the first binary '1' in the input vector. A valid signal indicates if any binary '1' was detected in the input vector, hence the index is valid.

As depicted in Figure B.1, the suggested priority-encoder is recursively con-

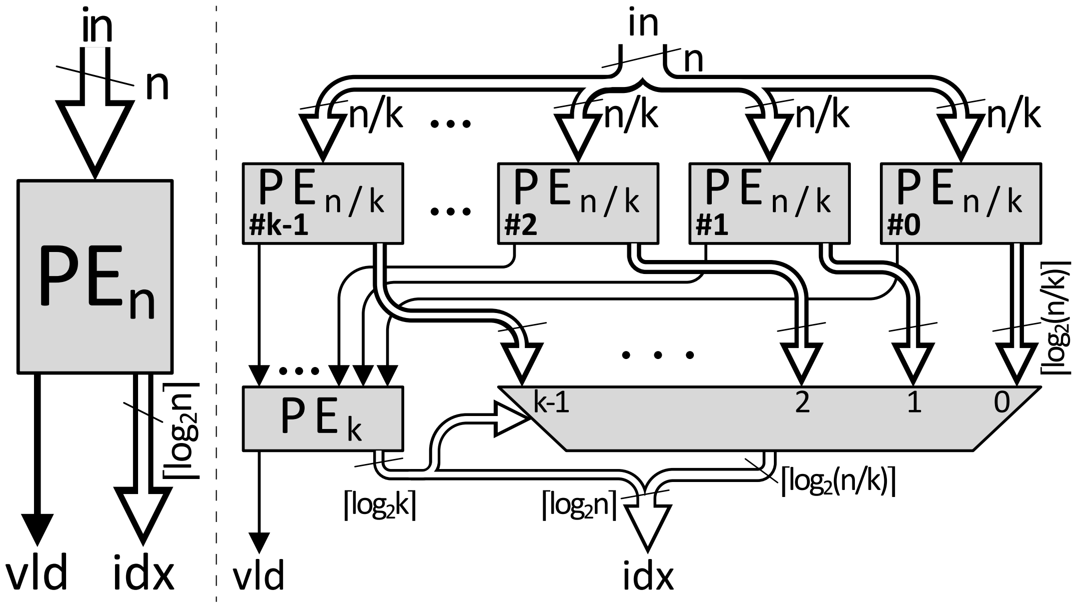


Figure B.1: Priority-encoder (left) symbol (right) recursive definition.

structed. The input vector is split into k equal fragments with $\frac{n}{k}$ bits. A priority encoder PE$_{\frac{n}{k}}$ with a narrower width of $\frac{n}{k}$ is applied for each fragment. The valid bit of each of the k PE$_{\frac{n}{k}}$'s goes to a k bit PE$_k$ to detect the first valid fragment. The location of this fragment is the higher part of the overall index, and steers the exact location within the fragment itself to produce the lower part of the overall index.

The depth of the proposed structure is $\lceil \log_k n \rceil$, while the hardware area complexity is $O(n)$. If Altera's Stratix V [55] or equivalent device is used, $k = 4$ is recommended to achieve higher performance and area compression, since the mux can be implemented using 6-LUT, hence an entire ALM.

Appendix C

Verilog IPs User Guide

This appendix is a user guide for the Switched Multi-ported RAM (SMPRAM) module Verilog package provided with this dissertation. The SMPRAM module is compatible with Verilog-2001. The provided Verilog is generic, however, it has been tested using Altera's ModelSim (version 10.0d) only. The SMPRAM module, including interface signals and configuration parameters is described in Figure C.1. Table C.1 lists all interface ports while Table C.2 lists all configuration parameters for the SMPRAM module. The code in Listing 1 describes an SMPRAM module instantiation. Furthermore, to instantiate the SMPRAM module, all *.v & *.vh files in this package should present in your work directory. The following command-line clones the package from a GitHub repository.

```
git clone https://github.com/AmeerAbdelhadi/Switched-Multiported-RAM.git
```

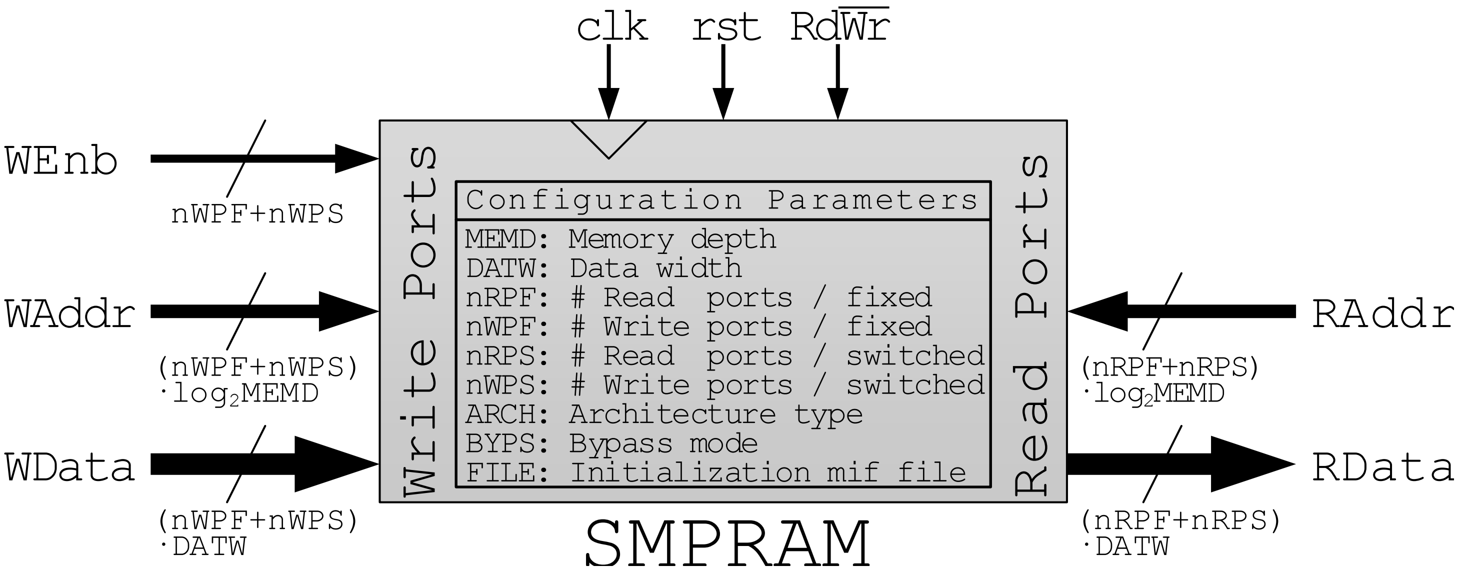


Figure C.1: Switched Multi-ported RAM (SMPRAM) module block.

Table C.1: List of SMPRAM module interface ports

Port	I/O	width	Description
clk	Input	1	Global clock.
rst	Input	1	Global synchronous reset.
rdWr	Input	1	If high, enables switched read ports and disables switched write ports; vice versa otherwise.
WEnb	Input	$\mathrm{nWPF}+\mathrm{nWPS}$	Write enable for nWPF (LSB) fixed write ports and nWPS switched write ports.
WAddr	Input	$(\mathrm{nWPF}+\mathrm{nWPS})\cdot\log_2(\mathrm{MEMD})$	Write addresses: packed from nWPF (LSB) fixed write ports and nWPS switched write ports; $\log_2(\mathrm{MEMD})$ bits each.
WData	Input	$(\mathrm{nWPF}+\mathrm{nWPS})\cdot\mathrm{DATW}$	Write data: packed from nWPF (LSB) fixed write ports and nWPS switched write ports; DATW bits each.
RAddr	Input	$(\mathrm{nRPF}+\mathrm{nRPS})\cdot\log_2(\mathrm{MEMD})$	Read addresses: packed from nRPF (LSB) fixed read ports and nRPS switched read ports; $\log_2(\mathrm{MEMD})$ bits each.
RData	Output	$(\mathrm{nRPF}+\mathrm{nRPS})\cdot\mathrm{DATW}$	Read data: packed from nRPF (LSB) fixed read ports and nRPS switched read ports; DATW bits each.

Table C.2: List of SMPRAM module parameters

Parameter	Type	Default value	Value range	Description
MEMD	Integer	N/A	Power of 2	Memory depth.
DATW	Integer	N/A	$1 \leq \mathrm{DATW}$	Data width.
nRPF	Integer	N/A	$1 \leq \mathrm{nRPF}$	Number of fixed read ports.
nWPF	Integer	N/A	$0 \leq \mathrm{nWPF}$	Number of fixed write ports.
nRPS	Integer	0	$0 \leq \mathrm{nRPS} \leq \mathrm{nRPF}$	Number of switched read ports.
nWPS	Integer	0	$0 \leq \mathrm{nWPS}$ $1 \leq \mathrm{nWPS} + \mathrm{nWPF}$	Number of switched write ports.
ARCH	String	”AUTO”	”AUTO”, ”REG”, ”XOR”, ”LVTREG”, ”LVTBIN”, or ”LVTTHR”	Multi-port RAM architecture: use ”AUTO” to choose automatically, ”REG” for register-based RAM, ”XOR” for XOR-based, ”LVTREG” for register-based LVT, ”LVTBIN” for binary-coded I-LVT-based, or ”LVTTHR” for thermometer-coded I-LVT-based.
BYPS	String	”RAW”	”NON”, ”WAW”, ”RAW”, or ”RDW”	Bypassing type: use ”NON” to prevent additional bypassing circuit, ”WAW” to allow Write-After-Write, ”RAW” to read new data when Read-After-Write, or ”RDW” to read new data when Read-During-Write.
FILE	String	Not initialized	”mif” file name / without extension	Initialization file in ”mif” format, optional.

Listing C.1: Switched Multi-ported RAM (SMPRAM) module instantiation

```
// instantiate a multiported-RAM
smpram #(
        .MEMD (MEMD ), // integer: memory depth
        .DATW (DATW ), // integer: data width
        .nRPF (nRPF ), // integer: # fixed    read  ports
        .nWPF (nWPF ), // integer: # fixed    write ports
        .nRPS (nRPS ), // integer: # switched read  ports
        .nWPS (nWPS ), // integer: # switched write ports
        .ARCH (ARCH ), // string : multi-port RAM architecture
        .BYPS (BYPS ), // string : bypass mode
        .FILE (""   )  // string : Initialization file, optional
) smpram_inst (
        .clk  (clk  ), // global clock
        .rst  (rst  ), // global reset
        .rdWr (rdWr ), // enables read or write switched ports
        .WEnb (WEnb ), // write enables   [(nWPF+nWPS)-1:0             ]
        .WAddr(WAddr), // write addresses [(nWPF+nWPS)*log2(MEMD)-1:0]
        .WData(WData), // write data      [(nWPF+nWPS)*DATW      -1:0]
        .RAddr(RAddr), // read  addresses [(nRPF+nRPS)*log2(MEMD)-1:0]
        .RData(RData)  // read  data      [(nRPF+nRPS)*DATW      -1:0]
);
```