Modular Switched Multi-ported SRAM-based Memories

AMEER M.S. ABDELHADI, The University of British Columbia
GUY G.F. LEMIEUX, The University of British Columbia

Multi-ported RAMs are essential for high-performance parallel computation systems. VLIW and vector processors, CGRAs, DSPs, CMPs and other processing systems often rely upon multi-ported memories for parallel access. Although memories with a large number of read and write ports are important, their high implementation cost means they are used sparingly. As a result, FPGA vendors only provide dual-ported block RAMs (BRAM) to handle the majority of usage patterns. Furthermore, 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 paper, 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. 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. Previous live-value-table (LVT) and XOR approaches are merged and optimized into a generalized and modular 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 suffers from excessive storage overhead since wider memories are required to accommodate the XOR-ed data. Two specific I-LVT implementations are proposed and evaluated, binary and thermometer coding. The I-LVT approach is especially suitable for deep memories because the table is implemented only in SRAM cells. The I-LVT method gives higher performance while occupying less BRAMs than earlier approaches: for several configurations, BRAM usage is reduced by over 44% and clock speed is improved by over 76%. The I-LVT can be used with fixed ports, true-ports or the proposed switched ports architectures. Formal proofs for the suggested methods, resources consumption analysis, usage guideline and analytic comparison to other methods are provided. A fully parameterized Verilog implementation is released as an open source library. The library has been extensively tested using Altera's EDA tools.

Categories and Subject Descriptors: B.3.2 [**MEMORY STRUCTURES**]: Design Styles Cache memories, Shared memory; C.1.2 [**PROCESSOR ARCHITECTURES**]: Multiple Data Stream Architectures (Multiprocessors) Interconnection architectures, Parallel processors

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Embedded memory, programmable memory, block RAM, multi-ported memory, shared memory, cache memory, register-file, parallel memory access

ACM Reference Format:
Ameer M.S. Abdelhadi and Guy G.F. Lemieux, 2014. Modular Switched Multi-ported SRAM-based Memories. *ACM Trans. Embedd. Comput. Syst.* 0, 0, Article 0 (November 2014), 25 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

Author's addresses: Ameer M.S. Abdelhadi and Guy G.F. Lemieux, Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, B.C., V6T 1Z4, Canada.

DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Multi-ported memories are the cornerstone of all high-performance CPU designs. They are often used in the register files, but also in other shared-memory structures such as 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 [Tseng and Asanović 2003], VLIW processors [Tseng and Asanović 2003],[Fisher 1983], multi-core processors [Fetzer and Orton 2002],[Bajwa and Chen 2007], vector processors, coarse-grain reconfigurable arrays (CGRAs) [Kwok and Wilton 2005], and digital signal processors (DSPs). For example, the second generation of the Itanium processor employs a 20-port register file constructed from SRAM bit cells with 12 read ports and 8 write ports [Fetzer and Orton 2002]. The key requirement for all of these designs is fast, single-cycle, and concurrent access from multiple requesters for performance reasons.

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. Another way is to alter the basic SRAM bit cell to provide extra access ports, but area growth is quadratic with the number of ports, so this requires a custom design for each unique set of parameters (number of ports, width and depth of RAM). 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. A multi-ported RAM can also be emulated through banking or multi-pumping. Banking uses hashing and arbitration to provide access, but it leads to unpredictable (multi-cycle) access latencies under collisions; this complicates system design and compromises performance. Multi-pumping provides a few extra ports, but it is limited by the amount of overclocking. Hence, a method of composing arbitrary, multi-ported RAMs from simpler RAM blocks is required.

Recently, a few FPGA-based modular simple/true-multi-ported RAM designs have been proposed. Live-value-table (LVT) is used together with multi-banking to create simple-multi-ported RAM [LaForest and Steffan 2010]. While each writing port writes to a different bank, the LVT tracks the latest written bank for each memory address, allowing to read the latest data. However, the LVT is composed of registers, a limited and area-intensive resource. Alternatively, The XOR-based cancellation method retrieves the latest written data by utilizing logical XOR properties [Laforest et al. 2012]. 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. Instead, An SRAM-based LVT is proposed by utilizing invalidation-table data structure [Abdelhadi and Lemieux 2014]. 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. In contrast to register-based LVTs, the I-LVT is composed of BRAMs only, hence it is capable of constructing deep LVTs. Choi *et al.* changed the data bank connectivity in the LVT-based approach to support bidirectional true-ports [Choi et al. 2012]. Their design has two main shortcomings. First, the LVT is still register-based and does not scale well for deep memories. Second, all ports are true bidirectional ports; this flexibility incurs high BRAM consumption, especially for memories with mixed simple/true-port requirements. Hence, a method of composing modular BRAM-based and storage-efficient multi-ported RAM supporting both simple and switched ports is required.

As described in Figure 1 (left), RAM ports can be classified into three categories based on their functionality: (1) simple/unidirectional ports with fixed read/write functionality, (2) true/bidirectional ports, where each port can switch read/write functionality, and (3) switched ports, where a group of write ports can switch with another

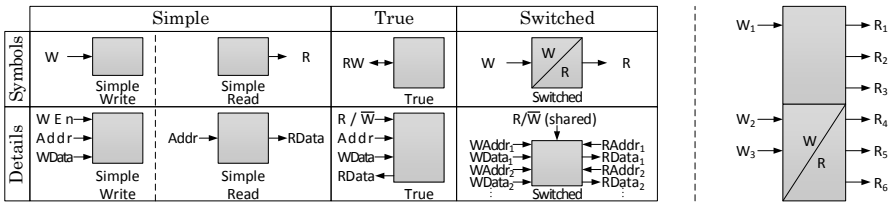

Fig. 1. (left) RAM port classification. (right) Example of mixed simple and switched ports.

group of read ports dynamically, hence altering the number of active ports. Figure 1 (right) provides an example of RAM module with mixed simple and switched port requirements; specifically, one simple write (W_1) and three simple reads ($R_{1..3}$) have fixed functionality, while two write ports ($W_{2..3}$) are switched with three read ports ($R_{4..6}$).

In this paper, 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) [Abdelhadi and Lemieux 2014] 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 paper 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 [Abdelhadi and Lemieux 2015].

Notation and abbreviations used for the rest of the paper are listed in Table I. The rest of this paper is organized as follows. In section 2, conventional RAM multi-porting techniques in embedded systems are reviewed. Previous attempts to provide multi-ported memories are reviewed in section 3. The proposed invalidation-based live-value-table method with switched ports is described in detail and compared to previous methods in section 4. The experimental framework, including simulation and synthesis and results, are discussed in section 5, and conclusions are drawn in section 6.

2. RAM MULTI-PORTING TECHNIQUES IN EMBEDDED SYSTEMS

This section provides a review of current methods of creating multi-ported RAMs in embedded systems. Creating multi-ported access to register-based memories is described in subsection 2.1. Multi-pumping is described in subsection 2.2. Replicating

Table I. List of notations and abbreviations

n_W, n_R, n_t	Write/read/true ports number	WAddr/RAddr	Write/read address
$n_{W,f}$, $n_{R,f}$	Write/read simple (fixed) ports number	WData/RData	Write/read data
$n_{W,s}$, $n_{R,s}$	Write/read switched-ports number	RWData	Bidirectional read/write data
d, w	Memory depth, data width	RBankSel	Read bank selector
n_{M20K}, n_{BReg}	M20K blocks / bypass registers number	LVT	Live-value-table
f_{fb}, f_{out}	LVT feedback/out functions	I-LVT	Invalidation LVT

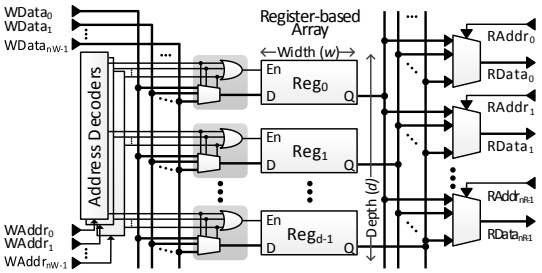

Fig. 2. Register-based multi-ported RAM.

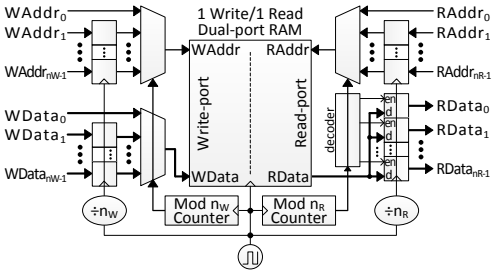

Fig. 3. Multi-pumping: RAM is overclocked, allowing multiple access during one pipeline cycle.

a memory bank to increase the number of read and write ports is described in subsections 2.3 and 2.4, respectively.

2.1. Register-based RAM

Multi-ported RAM arrays can be constructed using basic flip-flop cells and logic. As depicted in Figure 2, 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.

2.2. RAM Multi-pumping

A time-multiplexing approach can be applied to a single dual-ported SRAM block to reuse access ports and share them among several clients, each during a different time slot. As depicted in Figure 3, 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. A number of designs utilize multi-pumping to gain additional access ports while keeping area overhead minimal [Chappell et al. 1996],[Yokota 1990]. The 2.3GHz Wire-Speed POWER processor uses double-pumping to double the writing ports [Ditlow et al. 2011].

2.3. Multi-read RAM: Bank Replication

To provide more reading ports, the whole memory bank is replicated while keeping common write address and data as shown in Figure 4. 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 ports increases; only write delays increase due to fan-out, but this can be hidden by pipelining and bypassing. The bank replication

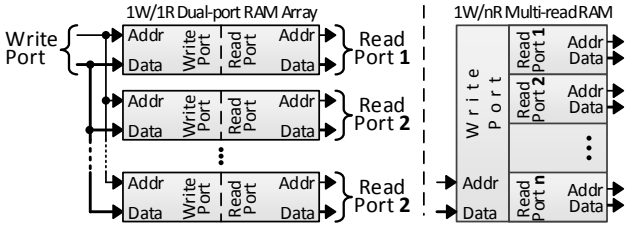

Fig. 4. (left) Replicated dual-ported banks with a common write port. (right) Multi-read RAM symbol used in this paper.

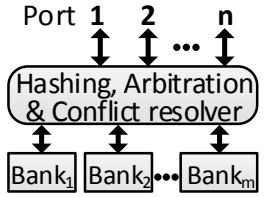

Fig. 5. Multi-banking: RAM capacity is divided into several banks. Ports access with a fixed hashing scheme.

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 [Ditlow et al. 2011]. 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 [Kessler 1999].

2.4. 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 5, 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 access 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 data in 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 [Tseng and Asanović 2003],[Mattausch 1997],[Ji et al. 2007],[Zuo et al. 2008]. 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 [Alpert and Avnon 1993].

3. MULTI-PORTED SRAM-BASED MEMORIES: PREVIOUS WORK

In this section a review of two previous multi-ported SRAM-based memories are provided. The first approach is based on multi-banking with a live-value-table (LVT) [LaForest and Steffan 2010], [Laforest et al. 2014] and is described in subsection 3.1. The second approach retrieves the latest written data by utilizing logical XOR properties [Laforest et al. 2012], [Laforest et al. 2014] and is described in subsection 3.2.

3.1. LVT-based Multi-ported RAM

For each RAM address, the LVT stores the ID of the bank replica that holds the latest data. As depicted in Figure 6 (left), an LVT-based multi-ported RAM uses a different bank replica for each writing port, while each bank 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 RAM with the same depth and number of writing ports as the implemented multi-ported memory. However, since the LVT stores only bank IDs, the data (line) width of the LVT table is only $\lceil \log_2 \rceil$ of the number of

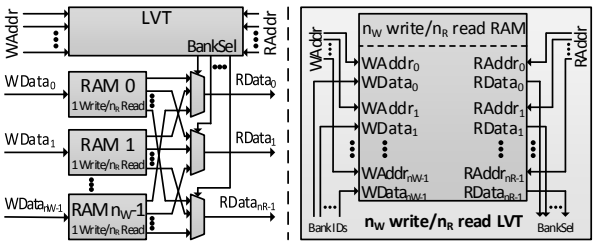

Fig. 6. (left) LVT-based multi-ported RAM. (right) An LVT implemented using multi-ported RAM.

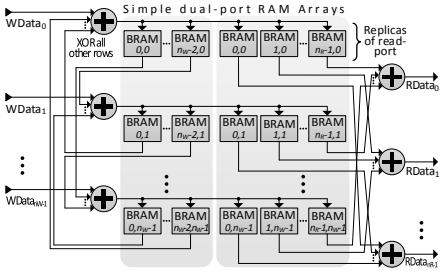

Fig. 7. XOR-based multi-ported RAM.

banks, which is equal to the number of writing ports. As described in Figure 6 (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 registered-based, multi-ported RAM. As explained in subsection 2.1, 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 this paper, to reduce this area overhead, two methods of constructing SRAM-based LVTs are described. The methodology of constructing SRAM-based LVTs is also generalized. To the authors' best knowledge this is the first attempt to build an LVT out of SRAM blocks only.

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

$$d \cdot \lceil \log_2 n_W \rceil. \tag{1}$$

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.

A register-based LVT with SRAM banks requires n_W multi-read banks for each write port. Each multi-read bank supports n_R reading ports, allowing the LVT to select the required data block. The total number of SRAM cells is

$$d \cdot w \cdot n_W \cdot n_R. \tag{2}$$

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

$$n_{M20K}(d, w) \cdot n_W \cdot n_R. \tag{3}$$

Where $n_{M20K}(d, w)$ is the number of M20K Blocks required to construct a RAM with a specific depth and data width. This value, described by Equation (4), is derived from Figure 8, which shows how Altera's M20K blocks can be configured into several RAM depth and data width configurations [Altera Corp. 2013]. 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 as follows

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

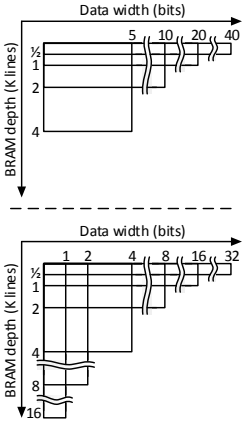

Fig. 8. Altera M20K configuration (top) 20Kb (bottom) 16Kb.

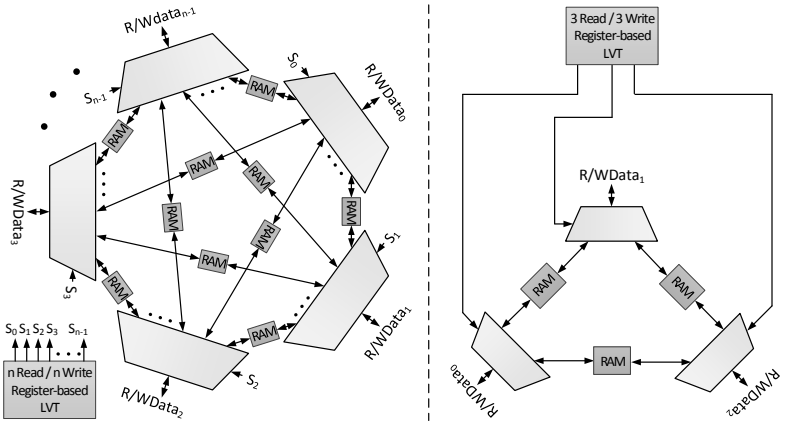

Fig. 9. Data banks with bidirectional true-ports support. Each port share a single BRAM with any other port.(left) Generalized approach. (right) A 3 true-ports example.

Two papers [Choi et al. 2012], [Laforest et al. 2014] introduce a modification to LVT's data bank to support bidirectional (true) ports. This is achieved by utilizing the bidirectional functionality of true-dual-port BRAMs. Figure 9 (left) illustrates a generalization of this method. Each port either writes to a set of data banks, or reads from them. *Every port 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 9 (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

$$\frac{1}{2} \cdot n_t \cdot (n_t - 1) \tag{5}$$

data copies are required, where n_t is the number of bidirectional (true) ports. Nevertheless, this true-port RAM architecture is still based on a register-based LVT, hence it suffers from the same shortcomings.

3.2. XOR-based Multi-ported RAM

While the LVT-based multi-port RAM just shown implements its LVT as a register-based multi-ported RAM, the XOR-based multi-ported RAM is implemented using SRAM blocks [Laforest et al. 2012], [Laforest et al. 2014]. 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 to retain the latest written data for each write port. 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 $a \oplus a = 0$.

As illustrated in Figure 7, each write port has a bank with multi-read and a single write. Part of the read ports are used as a feedback to generate new data and rewrite a specific bank, while the other read ports generate the data outputs. To perform a write, the new data is XOR'ed together with all the old data from the other banks; the result is rewritten to the corresponding bank. Hence if an address A is written through write port i with data $WData_i$, $Bank_i$ will be rewritten with

$$Bank_i[A] \leftarrow Bank_0[A] \oplus Bank_1[A] \oplus \cdots \oplus WData_i \oplus \cdots \oplus Bank_{nW-1}[A]. \tag{6}$$

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

$$RData_i[A] \leftarrow Bank_0[A] \oplus Bank_1[A] \cdots \oplus Bank_{n_W-1}[A]. \quad (7)$$

Substituting $Bank_i[A]$ from (6) into (7) 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 RAM requires n_W multi-read banks for each write port. Each multi-read bank supports $n_W - 1$ read ports to feedback the other ports via XORs, 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 is

$$d \cdot w \cdot n_W \cdot (n_R + n_W - 1). \quad (8)$$

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

$$n_M 20K(d, w) \cdot n_W \cdot (n_R + n_W - 1). \quad (9)$$

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

$$n_W \cdot (w + \lceil \log_2 d \rceil + n_W). \quad (10)$$

4. INVALIDATION TABLE

As described in the previous section, the XOR-based multi-ported memories requires $n_W \cdot (n_R + n_W - 1)$ manipulated 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 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 described in Figure 10, 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-LTV 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 paper, 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 mutual-exclusive conditions to invalidate table entries and generate one-hot-coded bank selectors. The two methods are described in subsections 4.1 and 4.2, respectively.

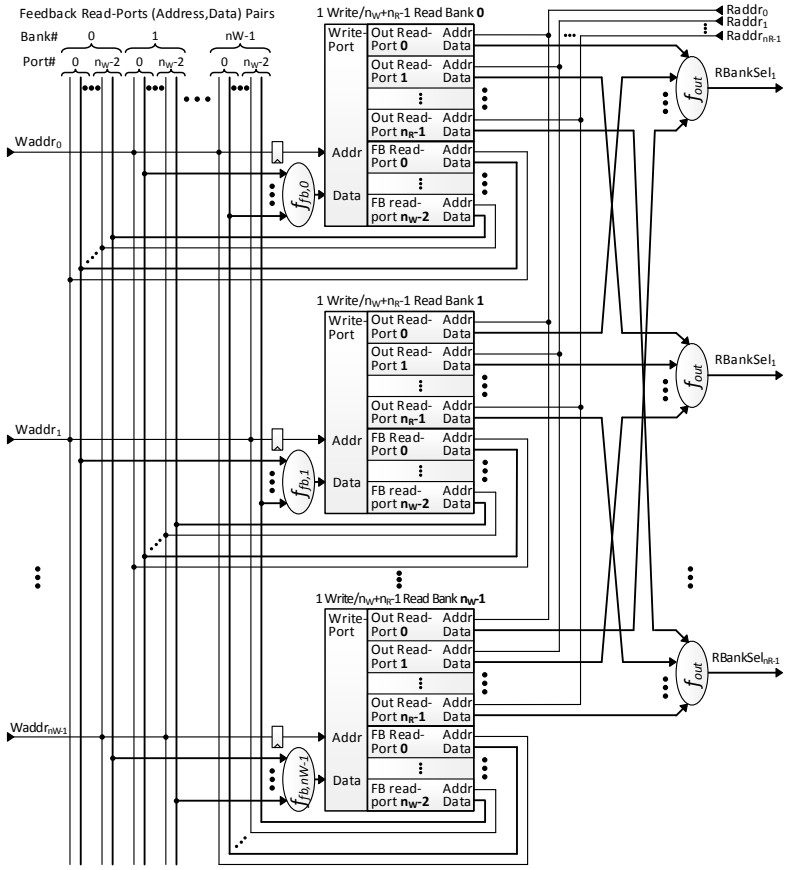

Fig. 10. Generalized approach for building the I-LVT.

4.1. 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,

$$f_{fb,k} = k \bigoplus_{0 \le i < n_W ; i \neq k} Bank_i[WAddr_k], \tag{11}$$

where k is the ID of the currently written bank.

Similar to the XOR-based method described in subsection 3.2, 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

$$f_{out,k} = \bigoplus_{0 \le i < n_W} Bank_i[RAddr_k]. \tag{12}$$

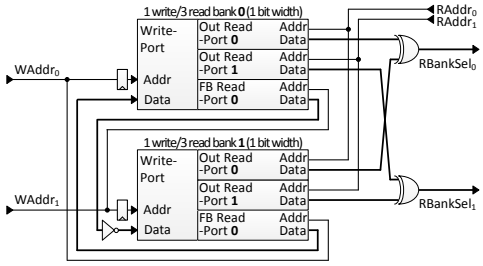

Fig. 11. A 2W/2R SRAM-based I-LVT; identical for binary-coded or thermometer-coded bank IDs.

Without loss of generality, if address A in bank k is written with the feedback function from Equation (11), then

$$Bank_k[A] = k \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A]. \tag{13}$$

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 (12), hence

$$RBankSel_r = \bigoplus_{0 \le i < n_W} Bank_i[A]. \tag{14}$$

Due to XOR operation associativity, $RBankSel_r$ from (14) can be expressed as

$$RBankSel_r = Bank_k[A] \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A], \tag{15}$$

Substituting $Bank_k[A]$ from (13) into (15) provides

$$RBankSel_r = k \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A] \bigoplus_{0 \le i < n_W; i \ne k} Bank_i[A]. \tag{16}$$

The last two series in (16) can be reduced revealing that $RBankSel_r = k$, the ID of the latest writing bank into address A, as required.

Figure 11 provides an example of 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 12 (left) shows a 3W/2R binary-coded I-LVT.

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

$$d \cdot \lceil \log_2 n_W \rceil \cdot n_W \cdot (n_W + n_R - 1). \tag{17}$$

Respectively, the number of required M20k block RAMs is

$$n_{M20K}(d, \lceil \log_2 n_W \rceil) \cdot n_W \cdot (n_W + n_R - 1). \tag{18}$$

Similarly, the number of registers required for retiming is

$$n_W \cdot (\lceil \log_2 d \rceil + 1). \tag{19}$$

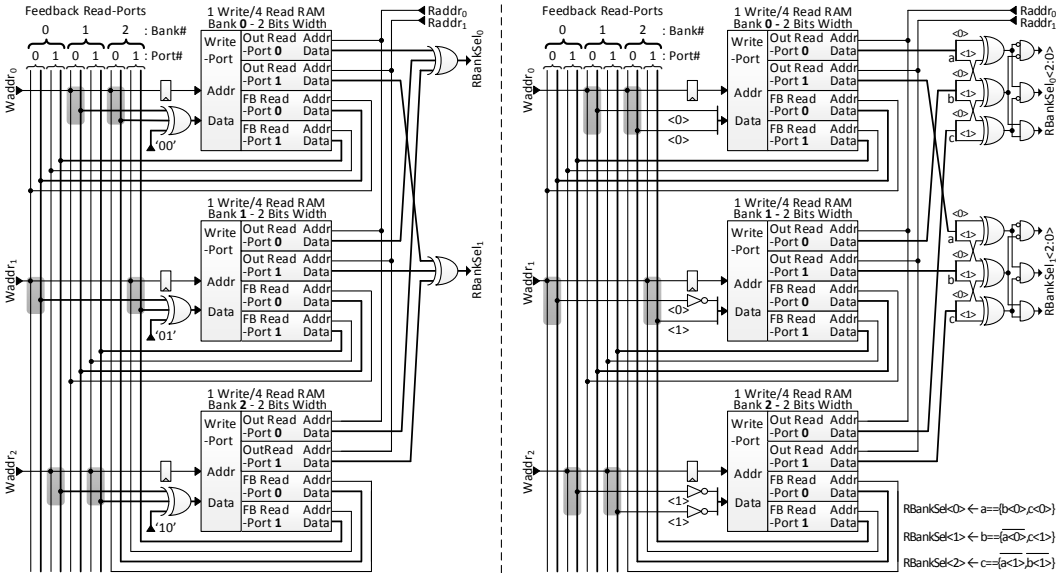

Fig. 12. A 3W/2R SRAM-based I-LVT with (left) binary-coded bank IDs, and (right) thermometer-coded bank IDs.

4.2. Mutual-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.

Mutual-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 mutual-exclusive condition for each bank, only the latest written bank will hold the valid data.

Equations (20) and (21) describe mutual-exclusive feedback functions for $n_W = 2, 3$, respectively. 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 $bank_2$ when $n_W = 3$ will write $Bank_1\langle 0\rangle \leftarrow \overline{Bank_0\langle 0\rangle}$ which will invalidate bank 0, and $Bank_1\langle 1\rangle \leftarrow \overline{Bank_2\langle 1\rangle}$ which will invalidate $bank_2$.

$$n_W = 2 : \begin{cases} f_{fb,0} : Bank_0\langle 0\rangle \leftarrow \overline{Bank_1\langle 0\rangle} \\ f_{fb,1} : Bank_1\langle 0\rangle \leftarrow \overline{Bank_0\langle 0\rangle} \end{cases} \tag{20}$$

$$n_W = 3 : \begin{cases} f_{fb,0} : Bank_0\langle 1:0\rangle \leftarrow \langle Bank_2\langle 0\rangle, \overline{Bank_1\langle 0\rangle}\rangle \\ f_{fb,1} : Bank_1\langle 1:0\rangle \leftarrow \langle \overline{Bank_2\langle 1\rangle}, \overline{Bank_0\langle 0\rangle}\rangle \\ f_{fb,2} : Bank_2\langle 1:0\rangle \leftarrow \langle \overline{Bank_1\langle 1\rangle}, \overline{Bank_0\langle 1\rangle}\rangle \end{cases} \tag{21}$$

Equation (22) generalizes the feedback function to

$$f_{fb,k}\langle i\rangle|_{0\leq i<n_W-1} : Bank_k[WAddr_k]\langle i\rangle \leftarrow \begin{cases} \overline{Bank_i[WAddr_k]\langle k-1\rangle} & i<k \\ Bank_{i+1}[WAddr_k]\langle k\rangle & \text{otherwise} \end{cases} \quad (22)$$

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 (22) 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 (22) $Bank_{k_1}\langle 1\rangle \leftarrow \overline{Bank_{k_0}\langle i_0\rangle}$ 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 mutual-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 11, while a 3W/2R thermometer-coded I-LVT is depicted in Figure 12 (right).

The thermometer-coded I-LVT requires n_W-1 SRAM bits to save the mutually exclusive cases. However, the feedback read ports requires 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 cases rewriting. Hence, the number of required SRAM cells is

$$d\cdot(n_W-1)\cdot n_W\cdot n_R + d\cdot n_W\cdot(n_W-1). \quad (23)$$

Respectively, the number of required M20k block RAMs is

$$n_{M20K}(d, n_W-1)\cdot n_W\cdot n_R + B_{M20K}(d,1)\cdot n_W\cdot(n_W-1). \quad (24)$$

Similarly, the number of registers required for retiming is equal to the binary-coded case and is described by (19).

4.3. Switched-ports Support

The I-LVT multi-ported RAM, similar to other previous register-based LVT [LaForest and Steffan 2010] and XOR [Laforest et al. 2012] 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 [Choi et al. 2012]; this excessive port flexibility incurs high BRAM consumption, especially for memories with mixed simple/true-port requirements. For instance, Figure 13 (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 13 (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 13 (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 archi-

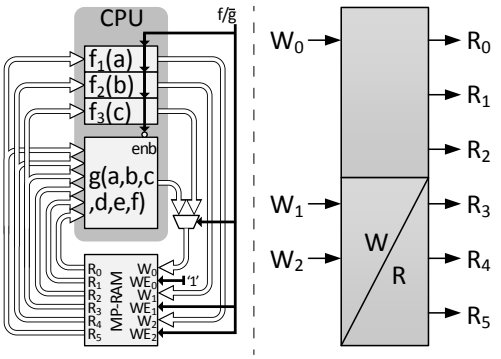

Fig. 13. (left) Switched ports application example: CPU multi-ported RAM pairing. (right) Symbol of th required switched ports RAM.

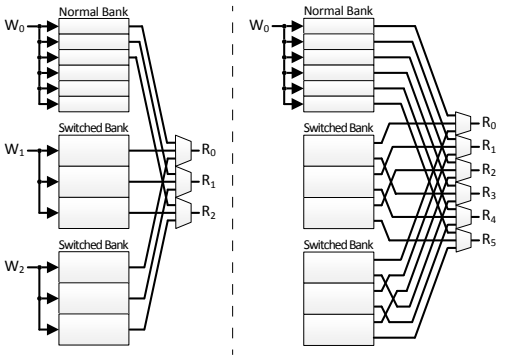

Fig. 14. switched ports example with $(n_{W,f}, n_{R,f}) = (1, 3)$ and $(n_{W,s}, n_{R,s}) = (2, 3)$ (left) Write configuration (right) Read configuration.

tecture 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 15 (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. Figures 15 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 15 (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 15 (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

$$n_{W,s} \cdot n_{R,s} \cdot n_{M20K}(d, w). \tag{25}$$

Figure 14 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 14 (left) shows the write mode configuration, while Figure 14 (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 25, if the switched RAM in Figure 14 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.

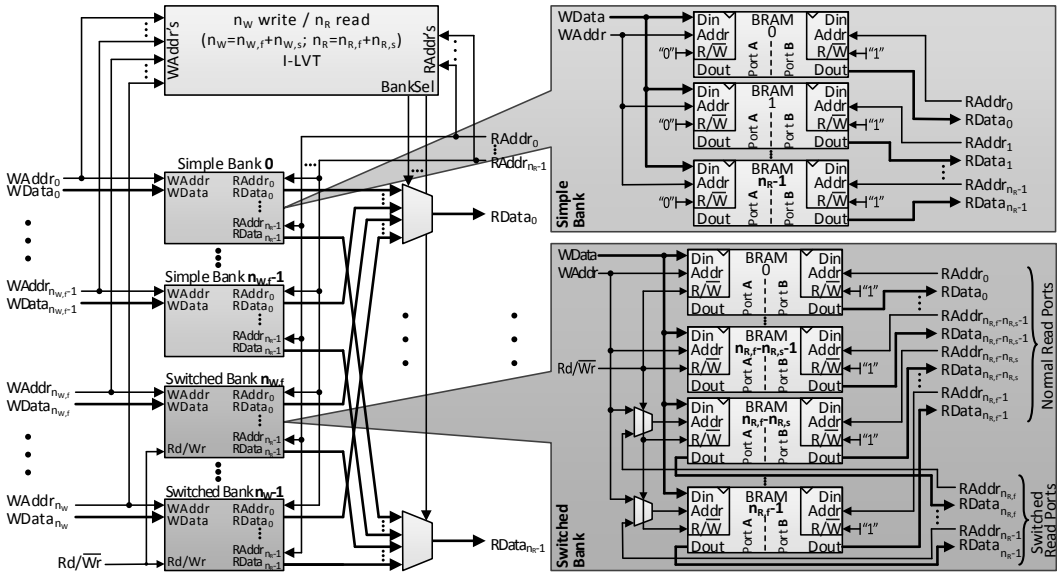

Fig. 15. (left) Switched ports architecture. Data banks: (upper right) simple, and (lower right) switched.

4.4. Data Dependencies and Bypassing

The new I-LVT structure and the previous XOR-based multi-ported RAMs incur 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 16 illustrates two types of bypassing based on write data and address pipelining. Bypassing is necessary because Altera's dual-port block RAMs 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-stages bypassing shown in Figure 16 (lower left) 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 block RAM has pipelined inputs (*e.g.*, cascaded from another block RAM) that need to be bypassed.

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

$$n_{BReg,unidirectional}(d, w) = w + 2\lceil \log_2 d \rceil + 1. \tag{26}$$

The switched multi-ported RAM described in section 4.3 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 Figure 16 (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

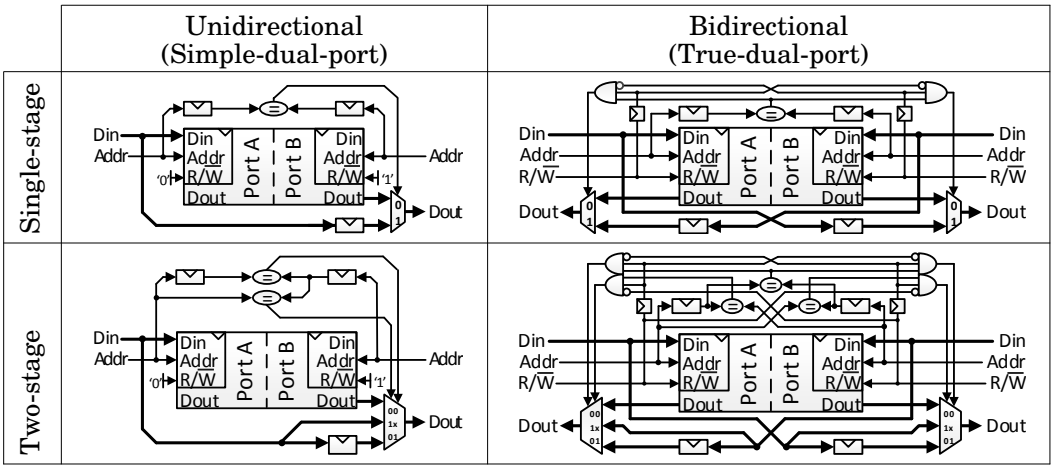

Fig. 16. Single-stage and two-stage bypassing for simple and true dual-port RAM.

used for the unidirectional bypass circuit, hence

$$n_{BReg,bidirectional}(d,w) = 2 \cdot n_{BReg,unidirectional}(d,w). \tag{27}$$

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) provides the old data. To read the new data, the output banks of the I-LVT are 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 2-stage bypass 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, are bypassed by a single-stage to provide new data. Table II 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.

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

$$n_W \cdot (n_W + n_R - 1) \cdot n_{BReg}(d,w). \tag{28}$$

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

$$n_W \cdot (n_W + n_R - 1) \cdot n_{BReg}(d, \lceil \log_2 n_W \rceil). \tag{29}$$

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,

Table II. 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

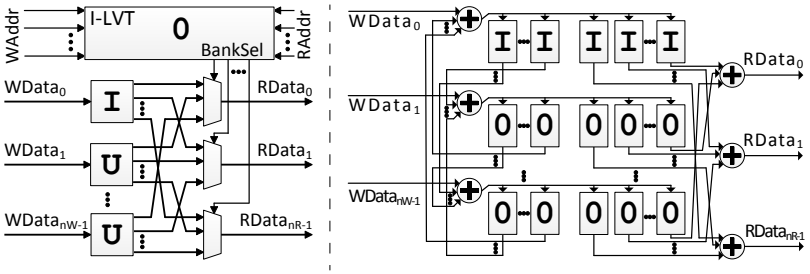

Fig. 17. Initial value for (left) I-LVT-based (right) XOR-based. (0: zeros, I: initial content, U: uninitialized)

hence a total registers of

$$n_W \cdot (n_W - 1) \cdot n_{BReg}(d, 1) + n_W \cdot n_R \cdot n_{BReg}(d, n_W - 1). \tag{30}$$

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 (29) and (30)

$$n_W \cdot n_R \cdot n_{BReg}(d, w). \tag{31}$$

4.5. Initializing Multi-ported RAM Content

Due to the special structure of the proposed I-LVT-based multi-ported memories and the previously proposed XOR-based method, RAM data may have replicas in several banks. Hence, initializing the multi-ported RAM with a specific content requires special handling. For the XOR-based multi-ported RAM, the first multi-read bank should be initialized to the required initial content; all the other multi-read banks should be initialized to zero. On the other hand, the binary/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 17.

4.6. 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 5.

4.6.1. 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.

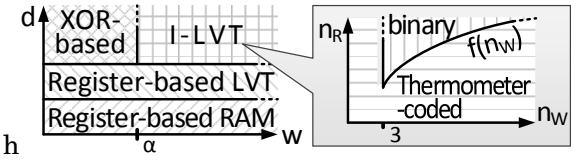

Fig. 18. Multi-ported RAM usage guideline. α is determined by the minimum of Equations (32) and (33), while the trend $f(n_W)$ is determined by Equation (34).

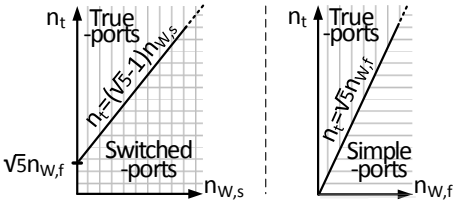

Fig. 19. Port architecture usage guideline.

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

$$w < n_R + 1. \tag{32}$$

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 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

$$w < \left.\frac{\lceil \log_2 n_W \rceil \cdot (n_W + n_R - 1)}{n_W - 1}\right|_{n_W > 1}. \tag{33}$$

Both (32) and (33) 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 III shows that the thermometer-coded I-LVT consumes fewer SRAM cells than the binary-coded I-LVT if

$$1 < n_W \leq 3 \;\text{ OR }\; n_R < \left.\frac{(n_W - 1) \cdot (\lceil \log_2 n_W \rceil - 1)}{(n_W - 1) - \lceil \log_2 n_W \rceil}\right|_{n_W > 3}. \tag{34}$$

Figure 18 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 Equations (32) and (33). Choosing binary-coded or thermometer-coded I-LVT is based on n_W and n_R only and is determined by Equation (34).

4.6.2. Register Usage based on RAM Architecture. Table V 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

$$w < \lceil \log_2 n_W \rceil. \tag{35}$$

Equation (35) 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

$$w < \left. \frac{1 + n_R}{1 + \frac{n_R}{n_W - 1}} \right|_{n_W > 1}. \tag{36}$$

In a typical compute-oriented designs, $n_R = 2 \cdot n_W$. 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 (36), 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 (31). In this case, XOR-based design consumes fewer registers than the binary-coded I-LVT design only if

$$w < \lceil \log_2 n_W \rceil \cdot \left(1 + \frac{n_R}{n_W - 1}\right). \tag{37}$$

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

$$w < n_R + 1. \tag{38}$$

Similar to (32), which is equal to (38), 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.

4.6.3. 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 1 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:

$$\begin{array}{ll} \text{Simple} & : \mathrm{n_{M20K}}(d,w)\big((n_{R,f} + n_t + n_{R,s}) \cdot (n_{W,f} + n_t + n_{W,s}) \big) \\ \text{True} & : \mathrm{n_{M20K}}(d,w)\big(\frac{n_a(n_a-1)}{2}\big|_{n_a = n_{W,f} + n_{R,f} + n_t + \max(n_{W,s}, n_{R,s})}\big) \\ \text{Switcheded} & : \mathrm{n_{M20K}}(d,w)\big((n_{W,f} + n_t + n_{W,s})(n_{R,f} + nt) + (n_{W,f} + n_t) n_{R,s}\big). \end{array} \tag{39}$$

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 19 shows a linear approximation of the BRAM consumption equilibrium. These plots form guidelines for choosing the most area efficient architecture. Figure 19 (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 simple write ports count. On the other hand, Figure 19 (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 III. Summary of SRAM bits usage

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

Table IV. Summary of M20K blocks usage

	Data banks	LVT feedback banks	LVT output banks
Register-based LVT	$n_{M20K}(d,w)\ n_W\ \ n_R$	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	$n_{M20K}(d,w)\ n_W\ \ n_R$	$n_{M20K}(d,\lceil \log_2 n_W \rceil)\ n_W\ (n_W-1)$	$n_{M20K}(d,\lceil \log_2 n_W \rceil)\ n_W\ n_R$
Thermometer-coded I-LVT	$n_{M20K}(d,w)\ n_W\ \ n_R$	$n_{M20K}(d,\ \ \ \ \ )\ n_W\ (n_W-1)$	$n_{M20K}(d,n_W-1\ \ \ )\ n_W\ n_R$

Note: $B_{M20K}(d, w)$ is the number of Altera's M20Ks required to construct a d deep by w wide RAM as described in (4).

Table V. Summary of register usage

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

Note: $n_{BReg}(d, w)$ is the number of additional registers required to bypass a d deep by w wide RAM as described in (26) and (28); use $n_{BReg,bidirectional}$ if the proposed switched architecture is used, otherwise use $n_{BReg,unidirectional}$.

Table VI. Resources consumption for a 4W/8R multi-ported RAM test-case with 8k-entries of 32-bit words

		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%

Note a: This test-case consists of 2 fixed and 2 switched write ports $(n_{W,f}, n_{W,s}) = (2, 2)$, 4 fixed and 4 switched read ports $(n_{R,f}, n_{R,s}) = (4, 4)$ and new data RDW bypassing.

Note b: A register-based multi-ported RAM with the same parameters does not fit in the same device.

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

		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%

Note a: This test-case consists of 2 fixed and 2 switched write ports $(n_{W,f}, n_{W,s}) = (2, 2)$ with 4 fixed and 4 switched read ports $(n_{R,f}, n_{R,s}) = (4, 4)$.

5. 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 [Abdelhadi and Lemieux 2015]. 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 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: $n_W = 2..4$, $n_R = 3..6$, $d = 16k, 32k$, $w = 8, 16, 32$, bypassing: None, RAW and RDW. Following this, we analyze the full set of results. In this paper, we omit many of the in-between settings because they behaved as one might expect to see via interpolation of the endpoints.

Figure 20 plots the maximum frequency derived from Altera's Quartus II STA at 0.9V and temperature of 0 °C. The results show a higher Fmax for binary/thermometer-coded I-LVT compared to the XOR-based approach for all design cases. With 3 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 Fmax by 38% on average for all design configurations, while the maximum Fmax improvement is 76%.

Figure 21 (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 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 21 (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 22 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 ALM's 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 23. 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 III, IV and V and not from experimental results. Hence, the register-based LVT trend was added as a reference baseline to Figure 21 and 23 only.

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 24 (top and middle) is a plot of Fmax 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 24 (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 4.6.3). 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 Equations 25 and 39. The BRAM reduction relieves the routing resources hence an Fmax increase is observed as shown in Figure 24 (top). The Fmax 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 24 (middle).

6. CONCLUSIONS AND FURTHER DIRECTIONS

In this paper, 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 dramatically reduce BRAM consumption compared to true or simple ports for systems with alternating port requirements.

An invalidation-based live-value-table (I-LVT) is used to determine the latest written data banks for our suggested multi-ported RAM. 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. While the original LVT demand use of an infeasible number of registers, the I-LVT register usage is not directly proportional to memory depth; hence it requires orders of magnitudes fewer registers. Furthermore, the proposed I-LVT can reduce BRAM consumption up to 44% and improve Fmax by up to 76% compared to previous approaches. The thermometer-coded I-LVT method exhibits the highest Fmax, while keeping BRAM consumption within 6% of the minimal required BRAM count.

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.

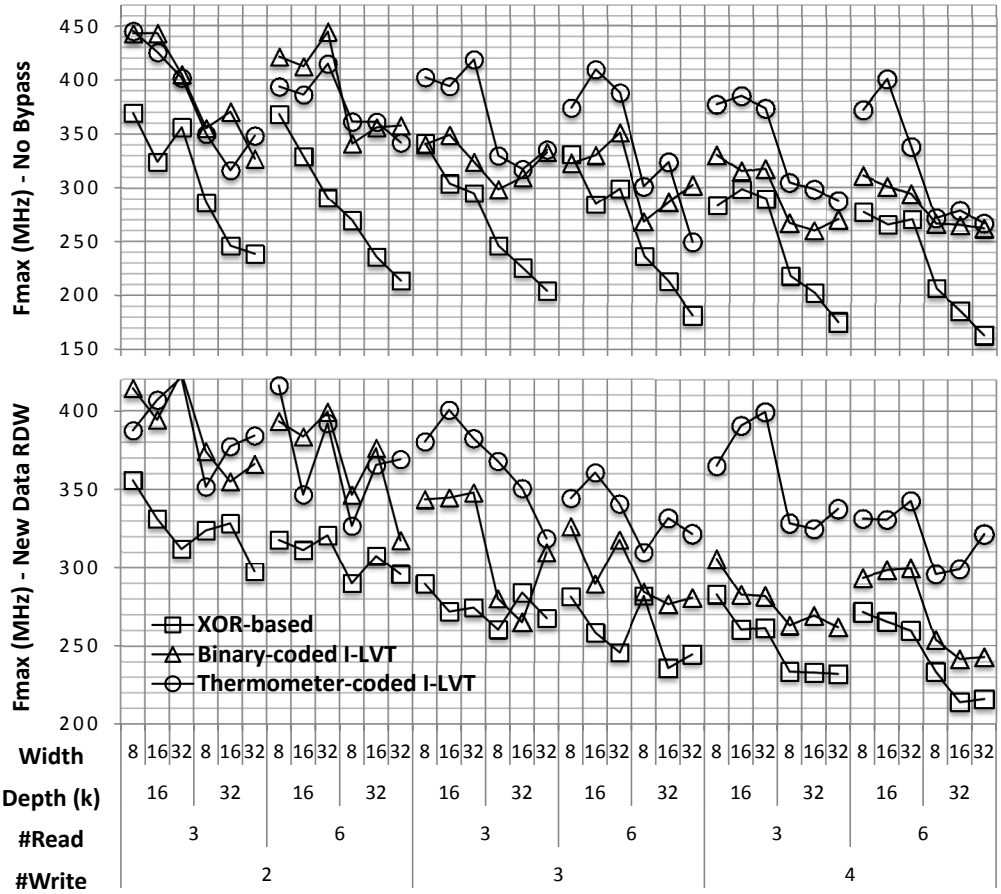

Fig. 20. Fmax (MHz) T=0C (top) No bypass (bottom) new data RDW bypass.

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 [Abdelhadi and Lemieux 2015].

As future work, the new 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 Fmax, 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 [Brant et al. 2012]. However, adapting this method to multi-ported memories is not trivial due to internal timing paths across the I-LVT. The true-multi-pored RAM proposed by others [Choi et al. 2012], [Laforest et al. 2014] can utilize our I-LVT method to implement BRAM-based LVT instead of register-based LVT, which eliminated the need of register-based memories and allows higher RAM capacities.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

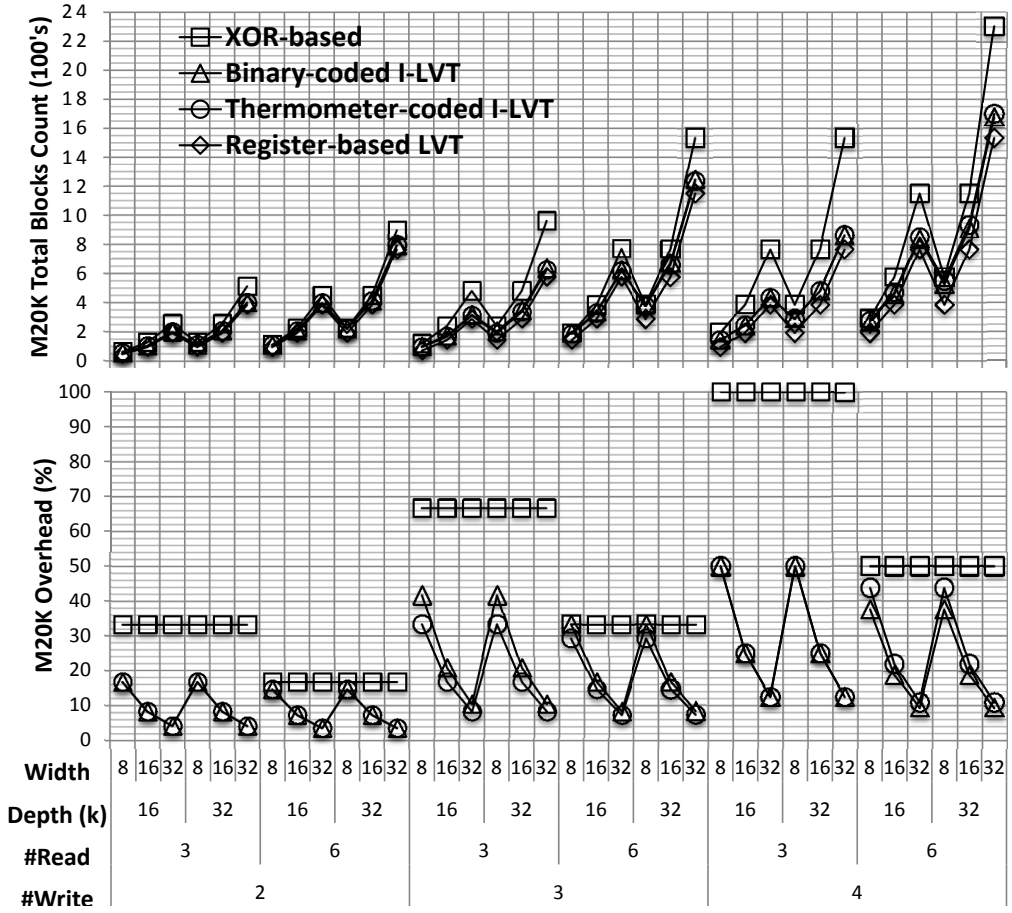

Fig. 21. M20K blocks (top) total count (bottom) overhead percentage relative to register-based LVT.

ACKNOWLEDGMENTS

The authors would like to thank Altera for donations of hardware and tools, as well as NSERC for funding.

REFERENCES

Ameer M.S. Abdelhadi and Guy G.F. Lemieux. 2014. Modular Multi-ported SRAM-based Memories. In *Proc. of the 2014 ACM/SIGDA Internat. Symp. on Field-programmable Gate Arrays (FPGA '14)*. 35–44.

Ameer M.S. Abdelhadi and Guy G.F. Lemieux. 2015. Switched Multi-ported RAM Verilog source code. (2015). https://github.com/AmeerAbdelhadi/Switched-Multiported-RAM

D. Alpert and D. Avnon. 1993. Architecture of the Pentium Microprocessor. *IEEE Micro* 13, 3 (1993), 11–21.

Altera Corp. 2013. Stratix V Device Handbook. (May 2013).

H. Bajwa and X. Chen. 2007. Low-Power High-Performance and Dynamically Configured Multi-Port Cache Memory Architecture. In *Electrical Engineering, 2007. ICEE '07. Internat. Conference on*. 1–6.

A. Brant, A. Abdelhadi, A. Severance, and G.G.F. Lemieux. 2012. Pipeline frequency boosting: Hiding dual-ported block RAM latency using intentional clock skew. In *Field-Programmable Technology (FPT), 2012 Internat. Conference on*. 235–238.

B.A. Chappell, T.I. Chappell, M.K. Ebcioglu, and S.E. Schuster. 1996. Virtual multi-port RAM employing multiple accesses during single machine cycle. (July 30 1996). US Patent 5,542,067.

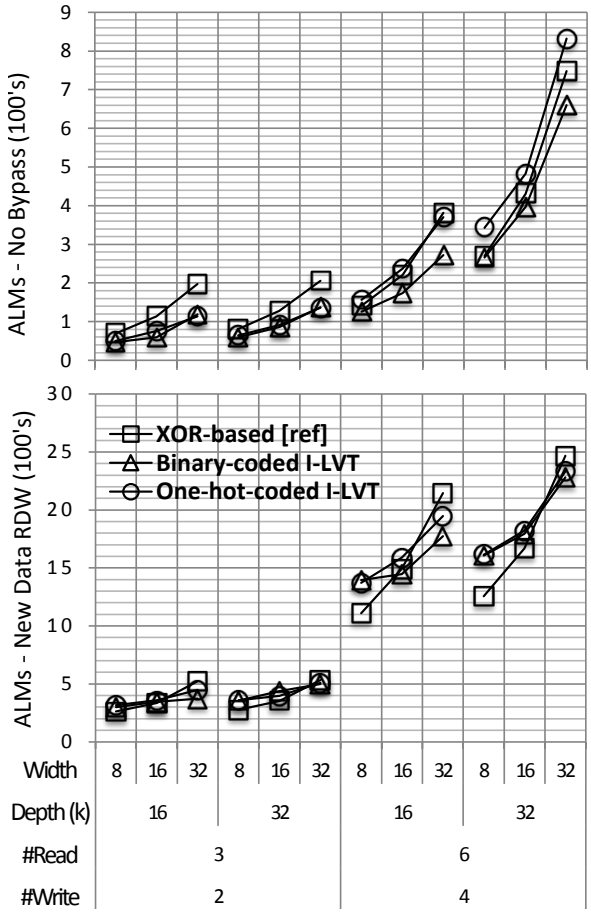

Fig. 22. ALMs (top) no-bypass (bottom) new data RDW bypass.

J. Choi, K. Nam, A. Canis, J. Anderson, S. Brown, and T. Czajkowski. 2012. Impact of Cache Architecture and Interface on Performance and Area of FPGA-Based Processor/Parallel-Accelerator Systems. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Internat. Symp. on*. 17–24.

G.S. Ditlow, R.K. Montoye, S.N. Storino, S.M. Dance, S. Ehrenreich, B.M. Fleischer, T.W. Fox, K.M. Holmes, J. Mihara, Y. Nakamura, S. Onishi, R. Shearer, D. Wendel, and L. Chang. 2011. A 4R2W register file for a 2.3GHz wire-speed POWER^{TM} processor with double-pumped write operation. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE Internat.* 256–258.

E.S. Fetzer and J.T. Orton. 2002. A fully-bypassed 6-issue integer datapath and register file on an Itanium microprocessor. In *Solid-State Circuits Conference, 2002. ISSCC. 2002 IEEE Internat.*, Vol. 1. 420–478.

Joseph A. Fisher. 1983. Very Long Instruction Word Architectures and the ELI-512. In *Proc. of the 10th Annual Internat. Symp. on Computer Architecture (ISCA '83)*. ACM, NY, USA, 140–150.

Weixing Ji, Feng Shi, Baojun Qiao, and Hong Song. 2007. Multi-port Memory Design Methodology Based on Block Read and Write. In *Control and Automation. ICCA '07. IEEE Internat. Conference on*. 256–259.

R. E. Kessler. 1999. The Alpha 21264 Microprocessor. *IEEE Micro* 19, 2 (March 1999), 24–36.

Z. Kwok and S.J.E. Wilton. 2005. Register file architecture optimization in a coarse-grained reconfigurable architecture. In *Field-Programmable Custom Comp. Machines. 13th Annual IEEE Symp. on*. 35–44.

C.E. Laforest, Z. Li, T. O'rourke, M.G. Liu, and J.G. Steffan. 2014. Composing Multi-Ported Memories on FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 7, 3 (Sept. 2014).

C.E. Laforest, M.G. Liu, E.R. Rapati, and J.G. Steffan. 2012. Multi-ported Memories for FPGAs via XOR. In *Proc. of the ACM/SIGDA Internat. Symp. on Field Programmable Gate Arrays (FPGA '12)*. 209–218.

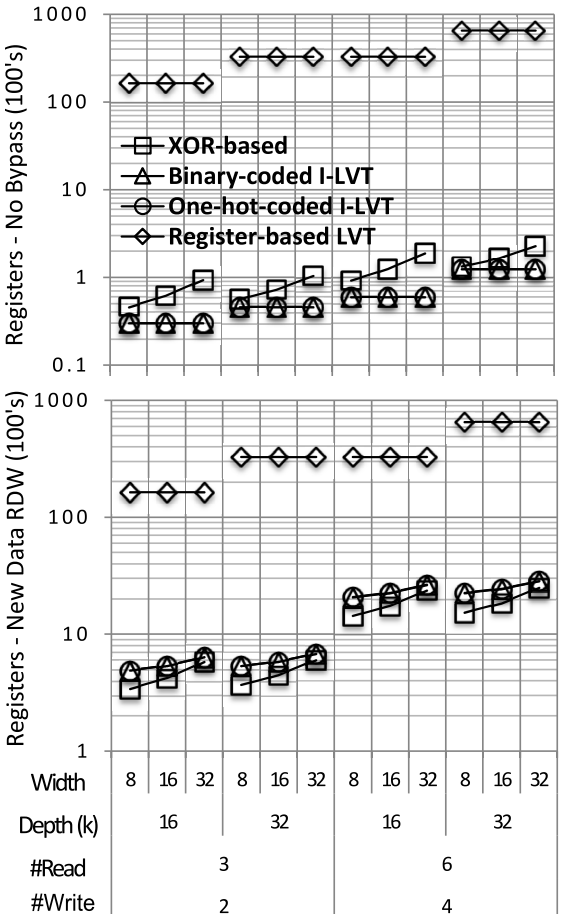

Fig. 23. Registers count (top) no-bypass (bottom) new data RDW bypass.

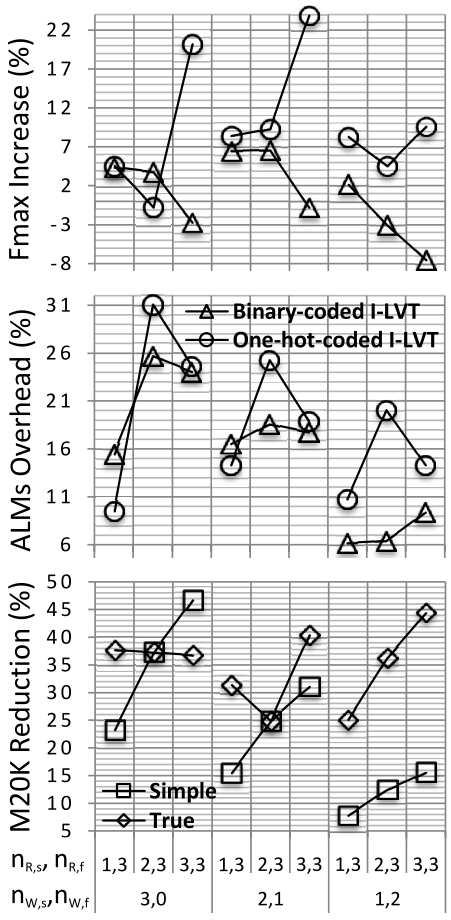

Fig. 24. Switched-ports compared to simple-ports and true-ports.

C.E. LaForest and J.G. Steffan. 2010. Efficient Multi-ported Memories for FPGAs. In *Proc. of the 18th Annual ACM/SIGDA Internat. Symp. on Field Programmable Gate Arrays (FPGA '10)*. 41–50.

H.J. Mattausch. 1997. Hierarchical N-port memory architecture based on 1-port memory cells. In *Solid-State Circuits Conference, 1997. ESSCIRC '97. Proc. of the 23rd European*. 348–351.

J.H. Tseng and K. Asanović. 2003. Banked Multiported Register Files for High-frequency Superscalar Microprocessors. In *Proc. of the 30th Annual Internat. Symp. on Comp. Architecture (ISCA '03)*. 62–71.

H. Yokota. 1990. Multiport memory system. (May 29 1990). US Patent 4,930,066.

Wang Zuo, Wang Zuo, and Li Jiaxing. 2008. An Intelligent Multi-Port Memory. In *Intelligent Information Technology Application Workshops, 2008. IITAW '08. Internat. Symp. on*. 251–254.

Received November 2014; revised March 2015; accepted June 2015

Online Appendix to:
Modular Switched Multi-ported SRAM-based Memories

AMEER M.S. ABDELHADI, The University of British Columbia
GUY G.F. LEMIEUX, The University of British Columbia

This appendix is a user guide for the Switched Multi-ported RAM (SMPRAM) module Verilog package provided with this paper. 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 25. Table VIII lists all interface ports while Table IX 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 commandline clones the package from a GitHub repository.

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

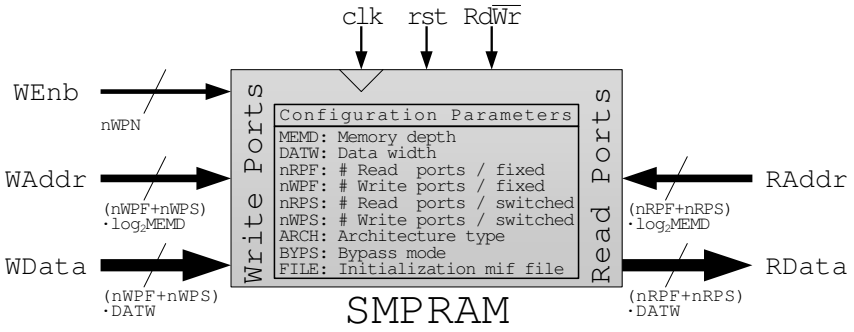

Fig. 25. Switched Multi-ported RAM (SMPRAM) module block.

Table VIII. 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	nWPF + nWPS	Write enable for nWPF (LSB) fixed write ports and nWPS switched write ports.
WAddr	Input	$(\text{nWPF} + \text{nWPS}) \cdot \log_2(\text{MEMD})$	Write addresses: packed from nWPF (LSB) fixed write ports and nWPS switched write ports; $\log_2(\text{MEMD})$ bits each.
WData	Input	$(\text{nWPF} + \text{nWPS}) \cdot \text{DATW}$	Write data: packed from nWPF (LSB) fixed write ports and nWPS switched write ports; DATW bits each.
RAddr	Input	$(\text{nRPF} + \text{nRPS}) \cdot \log_2(\text{MEMD})$	Read addresses: packed from nRPF (LSB) fixed read ports and nRPS switched read ports; $\log_2(\text{MEMD})$ bits each.
RData	Output	$(\text{nRPF} + \text{nRPS}) \cdot \text{DATW}$	Read data: packed from nRPF (LSB) fixed read ports and nRPS switched read ports; DATW bits each.

DOI: http://dx.doi.org/10.1145/0000000.0000000

Table IX. 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	$1 \leq$ DATW	Data width.
nRPF	Integer	N/A	$1 \leq$ nRPF	Number of fixed read ports.
nWPF	Integer	N/A	$0 \leq$ nWPF	Number of fixed write ports.
nRPS	Integer	0	$0 \leq$ nRPS $\leq$ nRPF	Number of switched read ports.
nWPS	Integer	0	$0 \leq$ nWPS $1 \leq$ nWPS + 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 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]
);
```