Composing Multi-Ported Memories on FPGAs

CHARLES ERIC LAFOREST, University of Toronto
ZIMO LI, University of Toronto
TRISTAN O'ROURKE, University of Toronto
MING G. LIU, University of Toronto
J. GREGORY STEFFAN, University of Toronto

Multi-ported memories are challenging to implement on FPGAs since the block RAMs included in the fabric typically have only two ports. Hence we must construct memories requiring more than two ports either out of logic elements or by combining multiple block RAMs. We present a thorough exploration and evaluation of the design space of FPGA-based soft multi-ported memories for conventional solutions, and also for the recently-proposed Live Value Table (LVT) [LaForest and Steffan 2010] and XOR [LaForest et al. 2012] approaches to unidirectional-port memories, reporting results for both Altera and Xilinx FPGAs. Additionally, we thoroughly evaluate and compare with a recent LVT-based approach to bidirectional-port memories [Choi et al. 2012].

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Style—*Shared Memory*

General Terms: Design Performance Measurement

Additional Key Words and Phrases: FPGA, LVT, memory, multi-port, parallel, XOR

ACM Reference Format:
Charles Eric LaForest, Ming G. Liu, Zimo Li, Tristan O'Rourke, and J. Gregory Steffan. 2012. Composing Multi-Ported Memories on FPGAs. *ACM Trans. Reconfig. Technol. Syst.* 0, 0, Article 0 (0), 22 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Designers increasingly use FPGAs to implement complex systems-on-chip that require frequent communication, sharing, queuing, and synchronization among distributed functional units and compute nodes. These high-contention storage mechanisms often contain *multi-ported memories* that allow multiple simultaneous reads and writes. For example, the register file of an FPGA-based, scalar, in-order MIPS-like *soft processor* requires one write port and two read ports, while processors that issue multiple instructions more require even more ports. However, constructing a multi-ported memory solely out of FPGA logic elements results in an inefficient solution [LaForest and Steffan 2010]. Furthermore, FPGA substrates typically provide block RAMs (BRAMs) with only two ports, hence memories with more than two ports must be "soft", i.e., constructed using logic elements and/or hard BRAMs. The ability to construct efficient soft multi-ported memories remains important as it frees FPGA vendors from having to include in their device fabrics costlier hard BRAMs with more than two ports.

1.1. Conventional Approaches

We can implement a multi-ported memory using only the basic logic elements of an FPGA, as illustrated in Figure 1 which shows a D-location memory with m write ports and n read ports. For now we consider only memories with unidirectional ports; we consider memories with bidirectional

Author's address: Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto

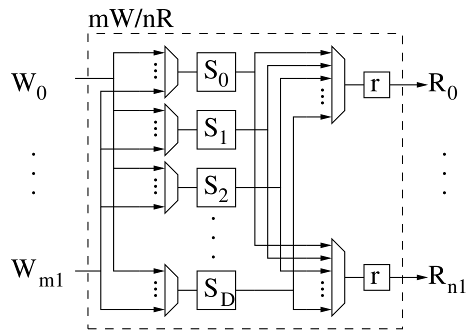


Fig. 1. A multi-ported memory implemented using FPGA logic blocks, having D single-word storage locations (S), m write (W) ports, and n read (R) ports (denoted as mW/nR), and n temporary registers r. We show only read and write data lines (i.e., not address lines).

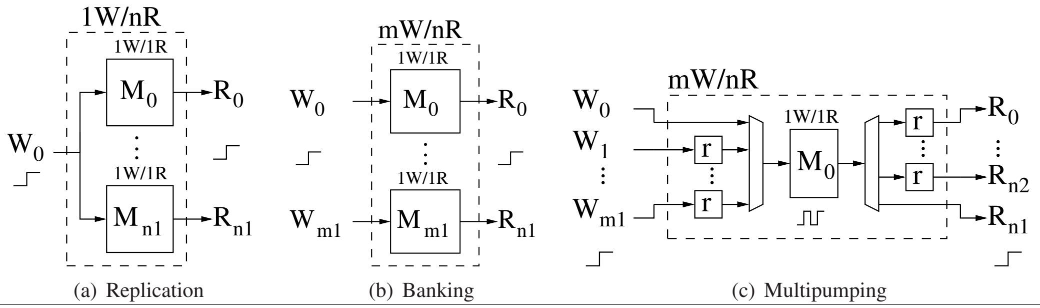


Fig. 2. Three conventional techniques for providing more ports given a 1W/1R memory (we do not depict read and write address values, only data values): (a) *Replication* maintains an extra copy of the memory to support each additional read port, but supports only a single common write port; (b) *Banking* divides data across multiple memories, but each read or write port can only access one specific memory (fracturing access to data); (c) *Multipumping* multiplies the number of read/write ports of a memory by adding internal data and address multiplexers and temporary registers (r), and internally clocking the memory at a multiple of the external clock (that quickly degrades the maximum external operating frequency).

ports later in Section 7. As shown, we require D m-to-one multiplexers to steer writes to the appropriate memory locations, and n D-to-one multiplexers to allow each read to access any memory location. Note also the registered read outputs necessary (r) to implement a synchronous memory that holds the outputs stable between clock edges. This circuit scales very poorly with memory depth: the area and the decoding/multiplexing increase rapidly, severely limiting the maximum operating frequency.

We normally implement memories on FPGAs more efficiently using the provided block RAMs, which can be quite denser than logic registers (e.g., 9, 18, or 36 Kbits) while supporting high operating frequencies (e.g., 550 MHz [Altera 2012; Xilinx 2012]). However, FPGA block RAMs currently provide only two ports for reading and/or writing. Note that Altera's defunct *Mercury* line of Programmable Logic Devices (PLDs) [Altera 2003] previously provided quad-port RAMs to support gigabit telecommunications applications—however, this feature does not exist in any other Altera device, likely due to the high hardware cost.

System designers therefore used one or a combination of three conventional techniques, as shown in Figure 2, to increase the effective number of ports of FPGA block RAMs. The first, **replication**, increases the number of read ports by maintaining a replica of the memory for each additional read port. However, this technique alone cannot support more than one write port since the single common external write port must be routed to each block RAM, using up its second port, to keep it up-to-date. The second technique, **banking**, divides memory locations among multiple block RAMs (banks), allowing each additional bank to provide an additional read and write port. However, with this approach each read or write port can only access its corresponding memory fraction—hence a pure banked design does not truly support sharing across ports.

The third technique, that we call **"multipumping"**, includes any memory design internally clocked at a multiple of the external clock to provide the illusion of a multiple of the actual number of ports and allowing a designer to trade speed for area reduction. For example, running a 1W/1R (one write and one read) memory internally at twice the external clock frequency can give the illusion of a 2W/2R memory. A multipumped design must also include multiplexers and registers to temporarily hold the addresses and data of pending reads and writes, and must carefully define the ordering of reads and writes. While reasonably straight-forward, a multipumped design has the drawback that each increase in the number of ports dramatically reduces the maximum external operating frequency of the memory. LaForest and Steffan [LaForest et al. 2012] present a detailed analysis of the impact of multipumping on multi-ported memory designs which we do not repeat here.

1.2. The Live-Value Table (LVT) Approach

The Live Value Table (LVT) approach [LaForest and Steffan 2010] augments a banked approach with a table that uses output multiplexers to steer reads to the most recently updated bank for each memory address. The LVT approach improves significantly on the area and speed of comparable designs built using only logic elements, and often results in the fastest design. The LVT approach can also support the implementation of multi-ported memories having bidirectional ports [Choi et al. 2012].

1.3. The XOR-Based Approach

The `XOR` operation ($\oplus$) has interesting and useful properties, particularly that $A \oplus B \oplus B = A$. Network coding schemes [Katti et al. 2006] commonly use `XOR` to transmit coded/mixed values together rather than individually, to later decode/unmix them at the receiver. RAID systems [Patterson et al. 1988] also use `XOR` to implement parity and provide data recovery capability should one hard-drive of an array of drives fail.

Similar to the LVT approach, the XOR approach to multi-ported memories [LaForest et al. 2012] internally uses banking and replication. However, the XOR design avoids the need for a Live Value Table to direct reads and thus also avoids the corresponding output multiplexing, instead allowing the logic of each read port to consist solely of an `XOR` of values read from all banks of BRAMs. We demonstrate that XOR designs consume less logic but require more BRAMs than corresponding LVT designs, and that some configurations of XOR designs can run faster and consume less total area than the equivalent LVT designs.

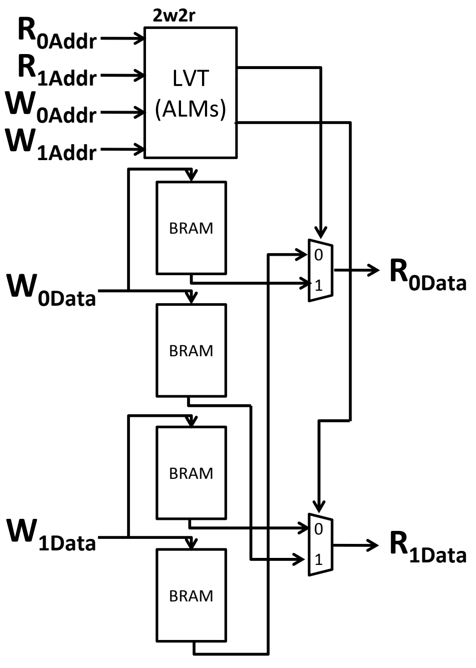
1.4. Contributions

In this work, we extend and refine the results of prior publications [LaForest and Steffan 2010; LaForest et al. 2012]:

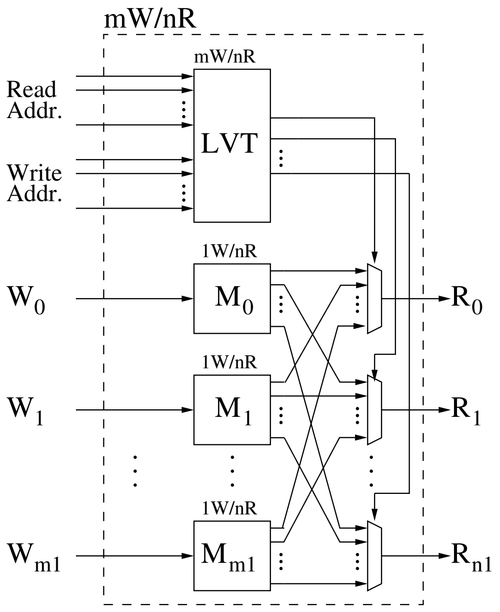
(1) we present the Live Value Table (LVT) design for implementing multi-ported memories;
(2) we present an alternative `XOR`-based design for implementing multi-ported memories;
(3) we thoroughly compare the speed and resource usage of pure-LE, LVT, and XOR approaches.

In addition to the above revisions, in this article we also make the following new contributions:

(4) we expand the explored space of memories to include both smaller and greater depths, and with a larger number of ports;
(5) we repeat our evaluations on Xilinx Virtex-6 devices, using the ISE 14.1 CAD tools (in Section 6), *evaluating multi-ported memories for the first time across both major FPGA vendors*;



(a) 2W/2R LVT design



(b) Generalized mW/nR LVT design

Fig. 3. Live Value Table (LVT) designs: (a) a specific 2W/2R design, and (b) a generalized mW/nR design.

(6) for the LVT-based design with bidirectional ports recently proposed by Choi *et al.* [Choi et al. 2012], we thoroughly explore the parameter space and compare with the pure-LE alternative (in Section 7);
(7) we compare the speed and area of unidirectional and bidirectional memories under different port configurations on Altera devices (in Section 8);
(8) we present a single, simple, and accurate predictive area model for all memory designs on Altera devices (in Section 9).

2. THE LIVE VALUE TABLE DESIGN

The Live Value Table (LVT) multi-ported memory design allows for a BRAM-based implementation of a memory with more than one write port instead of having to build such a memory solely from logic elements. Basically, an LVT design augments a banked memory with the ability to connect each read port to the most-recently written bank for a given memory location. In this section, we briefly summarize the construction and operation of the LVT design; see LaForest and Steffan [LaForest and Steffan 2010] for a full treatment.

As a simple example, Figure 3(a) shows a two-write-two-read (2W/2R) LVT-based memory. Each write port requires its own bank of BRAMs, with each bank containing two BRAMs to supply the two read ports. Each read port contains a multiplexer driven by the LVT, which selects the most-recently written (a.k.a. *live*) bank for the given read address. The LVT consists of a 2W/2R memory composed of logic elements (e.g., Altera's ALMs), of the same depth as the BRAM banks, but only as wide as the log_2 of the number of write ports (typically, 1-3 bits)—and hence requires much less area and runs faster than an entire word-wide 2W/2R memory built out of logic elements. Since the LVT is itself a multi-ported memory requiring the same number of read and write ports as the external memory ports, we implement it using logic elements.

Figure 3(b) shows a generalized mW/nR LVT design. Again, each write port has a bank of BRAMs, with each bank internally replicated into a 1W/nR memory for an overall memory with n external read ports. The *replication* of BRAMs within a bank creates a memory with a single write port, common to every BRAM inside, plus a read port from each internal BRAM feeding into one of each of the output multiplexers. In total, an LVT design requires $m \cdot n$ BRAMs plus the logic required to implement the LVT and the multiplexers.

We add forwarding logic around the BRAMs of the LVT design, similar to that shown later in Figure 4(b) for the XOR design, to increase clock frequency at a modest area cost. Forwarding logic bypasses a BRAM such that if a write and a read operation access the same location during the same cycle, the read will return the new write value instead of the old stored value. To remain compatible with the expected behavior of a one cycle read-after-write latency, we register the write addresses and data to delay them by one cycle. Using forwarding increases the maximum operating frequency of the BRAMs from 375 MHz to 550 MHz, as per the Stratix IV FPGA BRAM specifications [Altera 2012].[1]

For LVT-based memories, we have found that the LVT table and the output multiplexers together (i) constitute the critical path, and (ii) can require a significant number of logic elements to implement as the memory deepens. In the next section, we pursue an alternative design that avoids both of these disadvantages.

3. AN XOR-BASED DESIGN

In this section, we overcome the drawbacks of the LVT design by eliminating the narrow but multi-ported LVT and the output multiplexers it controls. As we mentioned in the Introduction, our design aims for each read port to require only the computation of the XOR of values read from BRAMs. We start with a review of some of the properties of XOR and gradually build-up a general XOR-based multi-ported memory design.

3.1. XOR Properties and Basic Use

The bitwise XOR operation ($\oplus$) is commutative, associative, and has the following properties:

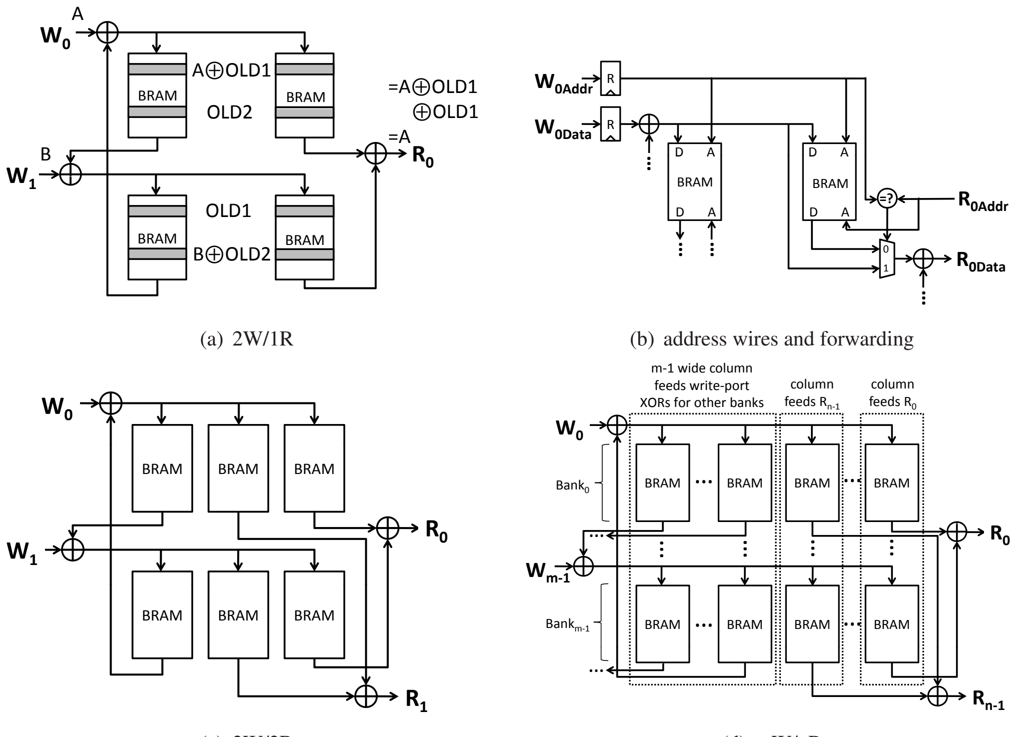
— $A \oplus 0 = A$
— $B \oplus B = 0$
— $A \oplus B \oplus B = A$

The third property, which follows from the first two, implies that we can XOR two values A and B together and later recover A by XORing the result with B. We can exploit this property to allow the XOR of two instances of a memory location to return the most recent version. For example, suppose $M[location_1]$ contains some OLD value, and then we save a new value A in a corresponding $M[location_2]$ by XORing it with the OLD value, i.e., by storing $A \oplus OLD$ in $M[location_2]$. Explicitly: $M[location_2] = A \oplus M[location_1] = A \oplus OLD$. We can then recover A (i.e., read the most recently-written value) by simply returning the XOR of the two locations, without having to select between them. Explicitly: $output = M[location_2] \oplus M[location_1] = (A \oplus OLD) \oplus OLD = A$. This scheme allows two write ports to write to two separate BRAMS (or banks of BRAMs) simultaneously (like the LVT design), while read ports need only XOR the contents of a BRAM location across all banks to return the most recent value (unlike the LVT design, which requires output multiplexing).

3.2. Simple XOR Designs

To begin a detailed example, we use the basic properties of XOR to construct a simple 2W/1R memory out of dual-ported BRAMs, as illustrated in Figure 4(a). Note that the figure shows only

[1]Normally, a BRAM internally performs a read on the clock rising edge and a write on the falling edge. Allowing both to occur on the same edge raises the operating frequency, but corrupts the read data, hence the need for forwarding logic to discard the bogus data coming from the BRAM read lines when the read and write addresses coincide.



(a) 2W/1R (b) address wires and forwarding

(c) 2W/2R (d) mW/nR

Fig. 4. XOR-based multi-ported memory design. Subfigure (a) shows a 2W/1R memory implemented using XOR, with example data values. Note that we only show data wires. W_0 stores the value A XOR'ed with the old contents ($OLD1$) of the other bank. Similarly, W_1 stores the value B XOR'ed with the old contents ($OLD2$) of the other bank. Reading the location containing A computes $(A \oplus OLD1) \oplus OLD1$ which returns A. Subfigure (b) outlines the details of the address wires, registers, and forwarding circuitry used in the XOR design, not shown in other figures for simplicity. Subfigure(c) depicts a 2W/2R memory implemented using XOR. Compared to the 2W/1R memory in Figure 4(a), an additional column of BRAMs (in the center) supplies data to the additional read port. Subfigure (d) describes a generalized mW/nR memory implemented using XOR. Each write port requires a bank (row) of BRAMs, while each read port requires a column of BRAMs. Additionally, the write ports also require extra columns in their banks (*one for each of one less than the total number of write ports*) to provide the old data for the other write ports to XOR with their new data. These extra columns receive the write addresses (as read addresses) without going through the initial registers shown in Subfigure (d).

data wires and their values. Each write port (W_0, W_1) has its own bank of two BRAMs, set in a row. Each write port copies its data to both BRAMs so that corresponding locations in every BRAM in a bank always hold the same value. When the write port W_0 stores the value A to the upper locations (in grey), it first XORs A with the old value ($OLD1$) from the same location in W_1's bank. Similarly, when the write port W_1 stores the value B to the lower locations (also in grey), it first XORs B with the old value ($OLD2$) from the same location in W_0's bank .

We implement the read port circuitry without multiplexing, solely the XOR of BRAM outputs. Reading the upper location computes the XOR of both versions from both the upper and lower banks, which results in isolating the value most recently stored to that location by computing $(A \oplus OLD1) \oplus OLD1$, which returns A.

However, the XOR design requires that a read precedes each write: We must store the XOR of the new write value with the old value read from the same location in the other bank. This constraint both increases the number of BRAMs to provide the needed read ports, and forces writes to take two cycles to complete since a read from the other bank must complete first. However, we can keep the

XOR design *black-box-compatible* with previous designs and give the illusion that writes effectively take only one cycle with two additions to the design, as illustrated in Figure 4(b): First, we register the write port address and data to the write ports' BRAMs to delay them by one cycle, during which we use the write address to perform the associated preceding internal read. Second, we instantiate forwarding circuitry that allows the write data to flow directly to the read data port in the event that we read a location in the cycle immediately following a previous write to the same location. For simplicity, we do not show these extra registers and forwarding logic other than in Figure 4(b). Quartus implements forwarding logic automatically when given the appropriate behavioral Verilog description of a BRAM.

We can add another read port to extend this 2W/1R memory to 2W/2R, as shown in Figure 4(c). The 2W/2R memory functions in the same manner as the 2W/1R design, except that another column of BRAMs (in the center) provides data for the additional read port, without needing changes to the write port logic.

To summarize the XOR design, each time we write to a given location in a particular memory bank, we XOR the new data with the old contents from the same location in all the other banks. To read a location we calculate the XOR of the values in that location across all banks, which recovers the latest value written.

3.3. A Generalized XOR Design

In Figure 4(d), we present a generalized mW/nR XOR design. Each write port has its own bank (row) of BRAMs. To write a value to a location in a given write port bank, we XOR the new value with all of the old values from that same location in all the other banks, and then store the result of that XOR in all the BRAMs of the write port bank. Therefore, the XOR design also requires extra column of BRAMs, of number one less than the number of write ports, to provide sufficient internal read ports to support writing without delay. Furthermore, each external read port requires its own column of BRAMs. Overall, an XOR design requires $m * (m - 1 + n)$ BRAMs to provide m writes and n reads.

4. EXPERIMENTAL FRAMEWORK (ALTERA)

In this section, we provide details about block RAMs, the memory designs that we study, our CAD flow, and our method for measuring speed and area. We focus on Altera devices and their CAD flow, but later in Section 6 we consider the same for Xilinx devices.

4.1. BRAM Memory

Modern FPGAs usually implement BRAMs directly as ASICs. These BRAMs typically have two ports that can each function either as a read or a write port. BRAMs use less area and run at a higher frequency than similar memories created from the FPGA's reconfigurable logic, but do so at the expense of having a fixed storage capacity and number of ports.

The Stratix IV FPGAs mostly contain M9K block RAMs[2], which hold 9 kilobits of information at various word widths and depths. At a width of 32 bits, an M9K holds 256 words. When configured as a "simple dual-port" memory with one fixed read port and one fixed write port, an M9K has a published maximum operating frequency of 550 MHz, which may decrease depending on the actual target device. When configured in "true dual-port" mode, where both ports can do either a read or a write each cycle, the M9K maximum published operating frequency drops to 375 MHz [Altera 2012].

[2]Stratix IV FPGAs also contain larger M144K and smaller MLAB memories. There are too few M144Ks to fully explore the design space, and past work demonstrated that using MLABs to construct multi-ported memories scales very poorly [LaForest and Steffan 2010].

4.2. CAD Flow

We implement all the designs in generic Verilog-2001 without any vendor-specific modules. We place our circuits inside a synthesis test harness designed to both: (i) register all inputs and outputs to ensure an accurate timing analysis, and (ii) to reduce the number of I/O pins to a minimum as the larger circuits will not otherwise fit on the FPGA. The test harness also avoids any loss of circuitry caused by I/O synthesis optimizations: Shift registers expand single-pin inputs, while registered AND-reducers compact word-wide signals to a single output pin. We use Altera's Quartus 12.0 to target the Stratix IV `EP4SE530H40C2` FPGA, a device of the highest available speed grade and containing 1280 M9K BRAMs.

We configured the synthesis process to favour speed over area and enabled all relevant optimizations, including circuit transformations such as register retiming. The impact on area of register retiming varies, depending on the logic found beyond the I/O registers, so the absolute results presented here might not appear in a real system. However, comparing our various designs inside a real system would yield proportionally similar results. We tested all designs inside identical test harnesses.

We configured the place and route process to make a standard effort at fitting with only two constraints: (i) to avoid FPGA I/O pin registers to prevent artificially long paths that would affect the clock frequency, and (ii) to set the target clock frequency to 550 MHz: the maximum clock frequency specified for M9K BRAMs. Similarly, for pure-LE design that do not use BRAMs we set the target to 800 MHz, the rated limit of the clock tree feeding the Adaptive Logic Modules. Setting a higher target F_{max} does not improve results, and may in fact worsen them (on average) if a slower, derived clock exists and thus aims towards an unnecessarily high target frequency, causing competition for fast paths. We define all clocks as externally generated and any of their fractions (e.g., half-rate) used in multipumping designs as synchronous to the main system clock (i.e., when generated by a PLL).

We report maximum operating frequency (F_{max}) by averaging the results of ten place and route runs, each starting with a different random seed for initial placement. We select the worst-case F_{max} report (slow corner/high temperature) for the default range of die temperatures of 0 to 85°C. Similarly, we also report an averaged area usage.

4.3. Measuring Area

When comparing designs as a whole, we report area as the *Total Equivalent Area* (TEA), which estimates the actual silicon area of a design point: we calculate the sum of all the Adaptive Logic Modules (ALMs) used partially or completely, plus the area of the BRAMs *counted as their equivalent area in ALMs*. A Stratix IV ALM contains two Adaptive Lookup Tables (ALUTs), each roughly equivalent to a 6-LUT, two adder and carry-chain stages, and two flip-flops. Wong *et al.* [Wong et al. 2011] provide the raw layout area data: one M9K BRAM has an area equivalent to 28.7 ALMs. Previously [LaForest and Steffan 2010], we had to use a derived estimate of equivalent area.

4.4. Design Parameters Considered

For simplicity, we consider only the common case of 32-bit-wide memories. We do not consider one-write-one-read (1W/1R) memories as they directly map to a single FPGA BRAM. Similarly, replication trivially enables 1W/nR memories. The challenge lies in creating *concurrent multiple write ports*. Overall, we consider two design classes: "unidirectional" memories, where some ports always read and some always write, and "bidirectional" memories where all ports can either read or write.

Although the internal implementations vary, we ensure that all designs function as "black-box equivalent" from an outside point of view. Specifically, all ports operate simultaneously within a single external clock cycle, and any latencies between writing and subsequently reading data remain equal across designs. Any one design can substitute for another within a system, with clock

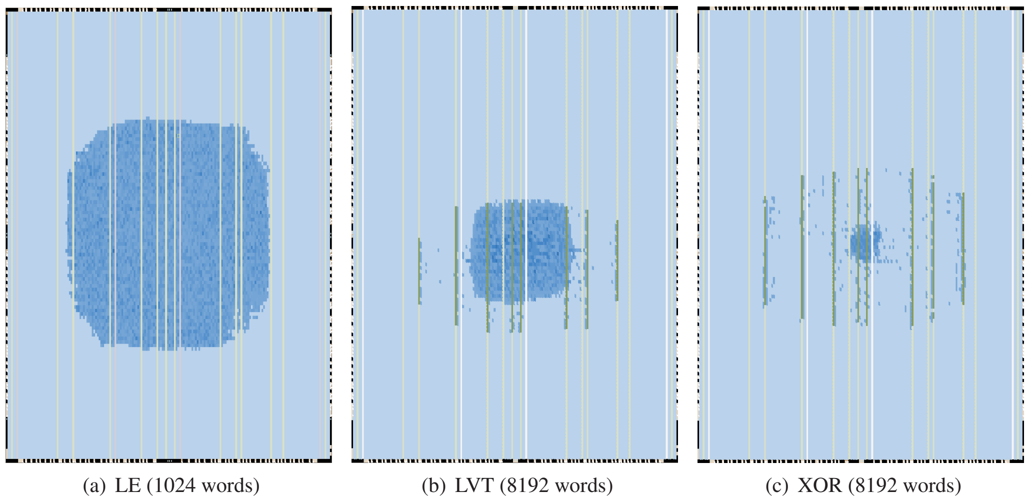


Fig. 5. Resource layout of (a) a 1024-deep 2W/4R pure-LE memory, and 8192-deep 2W/4R memories for (b) LVT, and (c) XOR designs, as rendered by Quartus. The thin columns represent BRAMs or DSPs (darkened indicates in-use), the dots and dot-clouds point out ALMs.

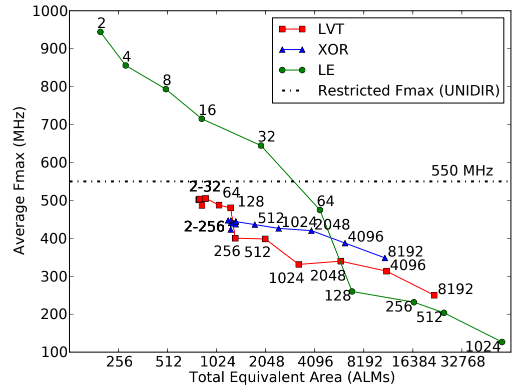
frequency and resource usage as the only differences. We do not consider memories that may stall (e.g., take multiple cycles to perform a read or write if there exists a resource conflict), although such designs suggest compelling future work. Finally, we assume that multiple simultaneous writes to the same address result in undefined behavior and that the enclosing system avoids these conflicts.

We evaluate a representative sample from the range of multi-ported memory configurations having 2 to 8 write ports and 4 to 16 read ports: 2W/4R, 4W/8R, and 8W/16R. We explore these configurations over memory depths of 2 to 8192 words, although some designs with depths exceeding 2048 words begin to consume a major fraction of the targeted device, making them potentially impractical for current-day applications.

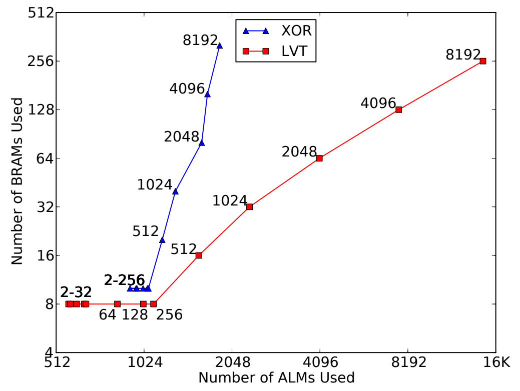
5. COMPARING PURE LE, LVT, AND XOR DESIGNS (ALTERA)

In this section we compare the pure-LE (Logic Elements), LVT, and XOR approaches to implementing multi-ported memories. We begin by illustrating example resource usage and layout of the three design approaches. Next, we compare in detail the speed and resource usage of a broad range of memories of varying depths and numbers of ports. Finally, we summarize the design space by highlighting the designs that minimize delay, ALM usage, or BRAM usage. For now, we focus solely on Altera devices/CAD; later in Section 6 we consider Xilinx devices/CAD.

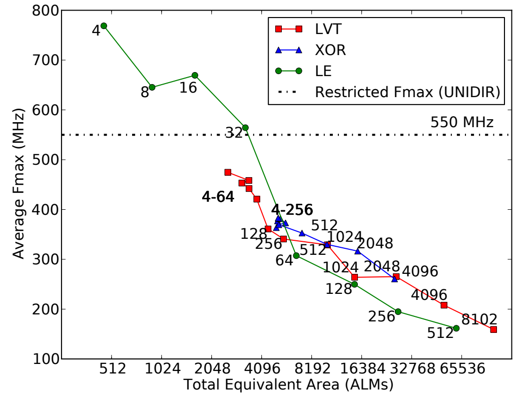
To illustrate the resource diversity of the pure-LE, LVT, and XOR approaches, we present in Figure 5 the resource layout of a 1024-deep 2W/4R pure-LE memory alongside those of 8192-deep 2W/4R LVT and XOR memories, as rendered by Quartus. The thin columns represent BRAMs or DSPs, while the dots and dot-clouds represent ALMs. Darkened areas represent in-use resources. We chose these memory depths because of the large resulting designs relative to the total capacity of the chip, and to emphasize the differences between the designs. All three designs have the same memory width (32 bits), and consume resources in a centered, somewhat circular pattern due to Quartus's efforts to minimize delay and resource consumption. While the pure-LE design consumes no BRAMs, the XOR design consumes more BRAMs than the LVT design (and hence has taller columns of in-use BRAMs). Conversely, the LVT design consumes more ALMs than the XOR design, and the pure-LE design consumes the most by far. Since a multi-ported memory normally exists within a larger design, one can see how the XOR memory would integrate better with an



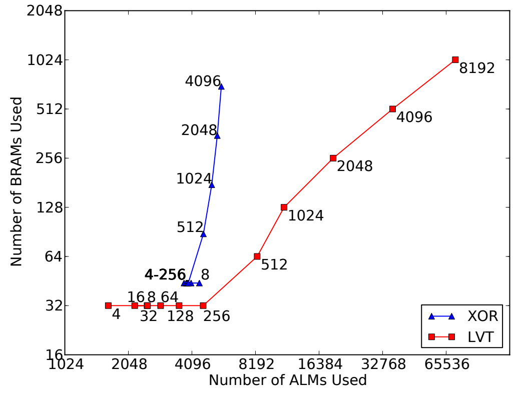
(a) 2W/4R: F_{max} vs. Total Equivalent Area



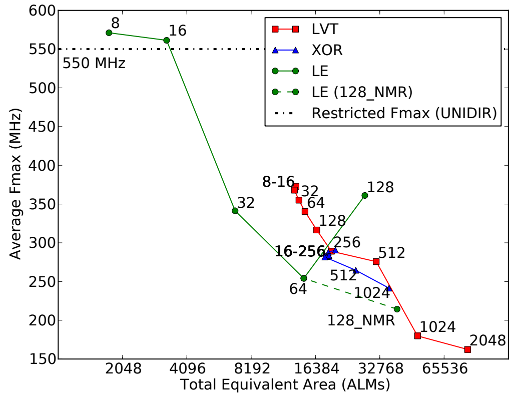
(b) 2W/4R: BRAMs used vs. ALMs used



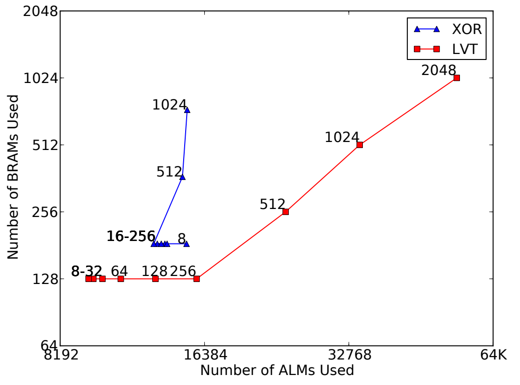
(c) 4W/8R: F_{max} vs. Total Equivalent Area



(d) 4W/8R:BRAMs used vs. ALMs used



(e) 8W/16R: F_{max} vs. Total Equivalent Area



(f) 8W/16R: BRAMs used vs. ALMs used

Fig. 6. Altera results: speed and area for LE, LVT, and XOR implementations of 2W/4R, 4W/8R, and 8W/16R memories of increasing depth (as will fit on the FPGA device). For each chart, we show the average unrestricted F_{max} vs average Total Equivalent Area (on the left), as well as BRAM usage vs averaged ALM usage (on the right). The dotted line shows the 550 MHz restricted F_{max} for unidirectional designs using BRAM running in simple dual-port mode.

enclosing design that consumes many ALMs, while the LVT and pure-LE designs would integrate better with an enclosing design that demands many BRAMs.

Depth	Design that minimizes:		
	Delay	ALMs	BRAMs
2	LE	LE	LVT
4	LE	LE	LVT
8	LE	LE	LVT
16	LE	LVT	LVT
32	LE	LVT	LVT
64	LE/LVT	LVT	LVT
128	LVT	LVT/XOR	LVT
256	XOR	LVT/XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT
2K	XOR	XOR	LVT
4K	XOR	XOR	LVT
8K	XOR	XOR	LVT

(a) 2W/4R

Depth	Design that minimizes:		
	Delay	ALMs	BRAMs
4	LE	LE	LVT
8	LE	LE	LVT
16	LE	LE	LVT
32	LE	LVT	LVT
64	LVT	LVT	LVT
128	LVT/XOR	LVT	LVT
256	LVT/XOR	XOR	LVT
512	LVT/XOR	XOR	LVT
1K	XOR	XOR	LVT
2K	XOR	XOR	LVT
4K	XOR	XOR	LVT

(b) 4W/8R

Depth	Design that minimizes:		
	Delay	ALMs	BRAMs
8	LE	LE	LVT
16	LE	LE	LVT
32	LE/LVT	LE	LVT
64	LVT	LVT	LVT
128	LVT	LVT	LVT
256	LVT/XOR	XOR	LVT
512	LVT/XOR	XOR	LVT
1K	XOR	XOR	LVT

(c) 8W/16R

Fig. 7. Altera design space navigation: for each memory depth, we list the design that minimizes either delay (i.e., has the highest F_{max}), the number of ALMs used, or the number of BRAMs used, for (a) 2W/4R, (b) 4W/8R, and (c) 8W/16R memories. We consider results within 5% as "equal" and list multiple designs then.

In Figure 6, we compare pure-LE, LVT, and XOR implementations of 2W/4R, 4W/8R, and 8W/16R memories, with depths varying from 2 entries up to 8192, as can fit on the FPGA device—XOR and LVT memories with more ports exhaust the available BRAMs more quickly, and pure-LE memories rapidly consume all available ALMs. In the leftmost figures, we plot the average *unrestricted* maximum operating frequency (F_{max}) versus the average area. (A minimum clock pulse width requirement for the BRAMs restricts the actual F_{max} to 550 MHz on Stratix IV devices, despite having a lower actual propagation delay.) We report the *Total Equivalent Area* (TEA) in terms of ALMs, that accounts for both the ALM and BRAM usage in a single measure, as described in Section 4. *Note the logarithmic x-axis for TEA.*

5.1. Fmax vs Area

Using the results for the 2W/4R memories (Figure 6(a)) as an example, we observe that the pure-LE designs show a clear F_{max} advantage for depths of up to 32 entries, losing ground to LVT at 64 entries, and slowing down below LVT and XOR at 128 entries and beyond. The TEA of LE designs grows rapidly: both LVT and XOR use less area, in all cases, for depths greater than 16 entries. For depths up to 128 entries, the LVT designs have better F_{max} and TEA than XOR designs but then lose on both counts past 256 entries, having a somewhat lower F_{max} and about twice the TEA of XOR designs at greater depths due to multiplexing overhead and the increasing area of the Live Value Table itself. Overall, the pure-LE approach benefits the smallest designs, the LVT approach works best (or at least as well as XOR) for moderate design sizes, while deeper designs save more area and gain more speed from the XOR approach.

5.2. BRAMs vs ALMs

The rightmost figures of Figure 6 explode our TEA metric into the actual numbers of BRAMs and ALMs consumed by the LVT and XOR designs—*note both logarithmic axes*. Using Figure 6(b) as an example, the two designs clearly exhibit resource diversity, with XOR designs consuming far fewer ALMs but more BRAMs than the corresponding LVT designs. *Hence the relative availability of ALMs or BRAMs in a given encompassing design plays a large role in the selection of the best choice of multi-ported memory implementation.* However, the number of BRAMS used remains constant, for both designs, over depths of 2 to 256 entries, reflecting the native capacity of each BRAM. The number of ALMs used by XOR memories grows very slowly as memory depth increases, since ALMs implement the `XOR` operations, which do not widen as memory depth increases[3], and forwarding logic, which grows linearly with the number of BRAMs, rather than depth. In contrast, the number of ALMs used by LVT memories grows quickly since ALMs implement the Live Value Table (itself a pure-LE memory, which scales poorly with depth) and the read port multiplexers, whose number of inputs increases linearly with the number of BRAMs used. Finally, XOR designs consume more BRAMS than the corresponding LVT designs due to the extra replicated memories required to support the write port `XOR` operations. For example, at 8192 entries the XOR design consumes 25% more BRAMs than the LVT design.

5.3. Increasing Ports

Figures 6(c)-(f) plot the same results previously discussed, but for 4W/8R and 8W/16R memories, that show similar overall trends as the 2W/4R memories. For memories having more than 256 entries, the XOR designs consume less TEA with the relative savings increasing with the number of entries: e.g., for 2W/4R 8192-entry memories the XOR design is 51% of the area of the LVT design. This TEA difference persists in 4W/8R and 8W/16R memories. As the number of ports increases, XOR designs become comparatively slower than LVT designs at shallower depths, due to the initial overhead of write port `XOR` operations.

The characteristics of pure-LE designs become increasingly irregular as the number of ports increases. For example, at 8W/16R ports, the F_{max} of the 128-deep pure-LE design unexpectedly increases due to the impact of multiplexer restructuring by the CAD tools. Disabling that optimization for this case yields the alternate *128_NMR* data point (on the dotted line), which follows the expected curve. However, also at 8W/16R ports, the TEA of the shallowest (2 to 32 entries) pure-LE designs becomes considerably smaller than the equivalent LVT and XOR designs. *Contrary to designs that use block RAMs, pure-LE designs do not increase the amount of internal data replication as the number of ports increases, but only add more input/output multiplexing and registering.*

Finally, Figure 6(f) shows that although the ALM and BRAM usage of the LVT and XOR designs increases in proportion to the product of the number of read and write ports (e.g., going from 2W/4R to 4W/8R quadruples the resources required for a memory of the same depth), the relative proportions and trends remain unchanged.

5.4. Navigating the Design Space

A system designer would ask "*Which memory design should I use given my constraints?*". To summarize the design space, we list in Figure 7 the design that, for each memory depth, minimizes delay (i.e., has the highest F_{max}), the number of ALMs used, or the number of BRAMs used, displayed for (a) 2W/4R, (b) 4W/8R, and (c) 8W/16R memories. We consider results within 5% of each other as effectively equal, due to normal CAD variations, and label them as such by listing all equal designs.

Overall, pure-LE designs have the highest speed for memories of less than 64 entries, regardless of port count. LVT designs initially (e.g., at 2W/4R ports) fill a narrow depth range for best speed between the pure-LE and XOR designs, but this range grows as the number of port increases. The trend reverses for area: as the port count increases pure-LE memories gradually use relatively fewer ALMs (compared to LVT designs) for shallow memories since they do not replicate memory. In all cases, the LVT designs use the least number of BRAMs. The properties of XOR memories change little with the number of ports; however, running out of BRAMs at shallower depths limits our analysis.

6. COMPARING PURE LE, LVT, AND XOR DESIGNS (XILINX)

In this section, we repeat our evaluation of pure-LE, LVT, and XOR designs in the Xilinx environment. We target Virtex-6 FPGAs as the most similar to the Altera Stratix IV devices measured in the previous section. We describe the Virtex-6 BRAMs, our CAD flow, our method of measurement,

[3]We include, but do not analyze, the area contribution of the multiplexers automatically created by the CAD tool to unite multiple BRAMs into a single deeper/wider memory bank.

and the results. However, we cannot chart the Total Equivalent Area (TEA) since we do not have any data about the area of the Virtex-6 BRAMs.

6.1. Virtex-6 Slices and BRAMs

We use Xilinx ISE 14.1 to target the Virtex-6 `XC6VHX380T` FPGA, a device of the highest available speed grade and containing 768 36-Kb BRAMs, and 59,760 slices. The *slice* is the Virtex-6 standard logic element, analogous to Altera's ALM. A Virtex-6 slice contains four six-input look-up tables and eight registers, as well as multiplexers and carry logic.

Virtex-6 BRAMs have a capacity of 36 Kbits configurable to various word-widths and depths, and can also function as two independent 18 Kbit BRAMs. For example, at a width of 32 bits (which we assume throughout), a BRAM holds 1024 words when configured in "simple dual-port" mode (one fixed read port and one fixed write port). To ensure that Xilinx implementations remain "black-box equivalent" to the Altera implementations, we must configure the BRAMs into *read first* mode, where a read returns the old value of a location even if there is a write to that same location in that same cycle, and then ensure the CAD tools automatically generate forwarding logic between the write and read ports. In read first mode, BRAMs have a published maximum operating frequency of 525 MHz for Virtex-6 devices of the highest speed grade [Xilinx 2012].

6.2. CAD Flow

We configured the synthesis process to favor speed over area. We measured all designs within a test harnesses identical to that described in Section 4. We enabled all relevant optimizations except for those allowing optimization across hierarchical boundaries (e.g., register retiming), since this adjustment yielded better results. We report the worst-case F_{max} (slow corner/high temperature) for the default range of die temperatures of 0 to 85°C.

When measuring the F_{max} of a design in ISE, we must choose the target frequency carefully: too-high a target results in a significantly slower design than a target closer to the actual achievable F_{max}; furthermore, ISE will not surpass a significantly lower target frequency. Hence, for each design, we perform a process similar to a bisection search for the best achievable frequency: (i) the initial run uses a target frequency equal to that of the limiting component (525 MHz for BRAM-based designs, 800 MHz for pure-LE designs); (ii) the following run targets the average of the previous target frequency and the F_{max} of the previous run; (iii) we repeat this process until the resulting F_{max} lies within 10% of (or just surpasses) the target frequency. We then use this final frequency target for the 10 seed runs used to compute the reported average speed and area of the design.

6.3. Results

Figure 8 shows, in tabular form, the CAD results for Xilinx FPGAs for all three implementations of multi-ported memories (pure-LE, LVT, and XOR) with varying numbers of ports (2W/4R, 4W/8R, 8W/16R) and memory depths (2 to 8129, as fits on the device). For each design point we report F_{max} (measured in MHz), the number of 36Kbit BRAMs used, and the number of slices used. We had to omit some of the deeper memories (e.g., 2K and 4K 4W/8R LVT, and 128-deep 4W/8R pure-LE) that conceivably should have fit, due to anomalously large place and route times that made their exploration impractical.

Based on these CAD results, Figure 9 lists, for each port configuration and depth, the design that minimizes either delay (i.e., has the highest F_{max}), the number of slices used, or the number of BRAMs used. Similar to the Altera results, and as expected, the LVT design minimizes the number of BRAMs used for all port and depth configurations. The XOR design shows a remarkably stable area (and therefore speed) as depth increases, regardless of port count.

For 2W/4R memories, the minimum delay crosses over from pure-LE to LVT at depth 32, and to XOR at depth 1024. All three designs have very close initial (depths 2 to 4) slice counts, but the area of the pure-LE design expands rapidly as depth increases. The 64-deep pure-LE design shows anomalously high speed and low area, contrary to its expected 250 MHz speed and 6000

Depth	CAD Result:	
	Fmax	Slices
2	800	48
4	797	67
8	753	202
16	473	1177
32	326	3379
64	454	1768
128	197	8590
256	159	12709
512	127	25938

(a) LE 2W/4R

Depth	CAD Result:		
	Fmax	Slices	BRAMs
2	410	43	4
4	408	45	4
8	404	48	4
16	406	64	4
32	403	71	4
64	404	104	4
128	395	149	4
256	401	201	4
512	398	524	4
1K	294	1655	8
2K	250	2913	16
4K	164	7127	32
8K	134	16984	64

(b) LVT 2W/4R

Depth	CAD Result:		
	Fmax	Slices	BRAMs
2	308	51	5
4	310	50	5
8	308	49	5
16	311	49	5
32	309	51	5
64	308	51	5
128	305	48	5
256	307	51	5
512	305	47	5
1K	299	60	10
2K	292	67	20
4K	290	76	40
8K	255	88	80

(c) XOR 2W/4R

Depth	CAD Result:	
	Fmax	Slices
4	452	67
8	444	202
16	367	1177
32	256	3379
64	194	1768

(d) LE 4W/8R

Depth	CAD Result:		
	Fmax	Slices	BRAMs
4	387	399	16
8	380	421	16
16	386	438	16
32	370	536	16
64	308	740	16
128	359	902	16
256	308	1190	16
512	228	2484	16
1K	213	5644	32

(e) LVT 4W/8R

Depth	CAD Result:		
	Fmax	Slices	BRAMs
4	252	264	22
8	245	272	22
16	255	275	22
32	248	270	22
64	249	267	22
128	254	278	22
256	256	267	22
512	250	276	22
1K	238	289	44
2K	212	338	88
4K	193	359	176
8K	167	382	352

(f) XOR 4W/8R

Depth	CAD Result:	
	Fmax	Slices
8	398	2452
16	304	4345
32	214	7574
64	142	14608
128	94	26038

(g) LE 8W/16R

Depth	CAD Result:		
	Fmax	Slices	BRAMs
8	306	1596	64
16	283	1319	64
32	288	1992	64
64	271	2462	64
128	260	2637	64
256	174	3644	64
512	120	7225	64
1K	104	20882	128
2K	89	33081	256

(h) LVT 8W/16R

Depth	CAD Result:		
	Fmax	Slices	BRAMs
8	172	477	92
16	182	477	92
32	186	475	92
64	173	486	92
128	190	479	92
256	188	478	92
512	185	482	92
1K	136	501	184
2K	151	531	368
4K	109	559	736

(i) XOR 8W/16R

Fig. 8. CAD results on a Xilinx FPGA for multi-ported memories of varying implementation (pure-LE, LVT, and XOR) over varying numbers of ports (2W/4R, 4W/8R, 8W/16R) and memory depths. For each design point we report the average F_{max} (measured in MHz), the number of 36Kbit BRAMs used, and the average number of slices used.

slices area, when approximated from the neighboring 32 and 128-deep results. This anomaly either originates from the CAD tools, or from an unexpected interaction between the pure-LE design and the architecture of the underlying FPGA device.

For 4W/8R memories, the minimum delay crossovers move down to half their depths at 2W/4R: 16 for LE-to-LVT, and 512 for LVT-to-XOR. However, contrary to the 2W/4R case, the pure-LE design initially (depths 4 to 8) has the lowest slice count instead of LVT since the pure-LE design does not replicate storage as the number of ports increases, contrary to the LVT and XOR designs. However, the pure-LE's multiplexing overhead rapidly undoes this advantage as depth increases. The multiplexing overhead of the pure-LE design also shows up again when we increase the ports to 8W/16R: although the LVT-to-XOR delay crossover moves down as expected from depth 512 to

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
2	LE	LVT	LVT
4	LE	LVT	LVT
8	LE	LVT/XOR	LVT
16	LE	XOR	LVT
32	LVT	XOR	LVT
64	LE	XOR	LVT
128	LVT	XOR	LVT
256	LVT	XOR	LVT
512	LVT	XOR	LVT
1K	LVT/XOR	XOR	LVT
2K	XOR	XOR	LVT
4K	XOR	XOR	LVT
8K	XOR	XOR	LVT

(a) 2W/4R

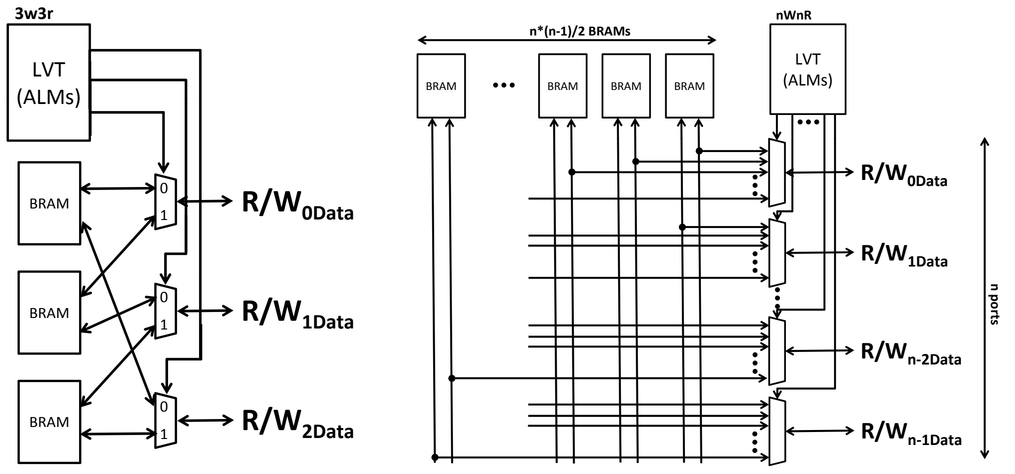
Depth	Design that minimizes:		
	Delay	Slices	BRAMs
4	LE	LE	LVT
8	LE	LE	LVT
16	LE/LVT	XOR	LVT
32	LVT	XOR	LVT
64	LVT	XOR	LVT
128	LVT	XOR	LVT
256	LVT	XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT

(b) 4W/8R

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
8	LE	XOR	LVT
16	LE	XOR	LVT
32	LVT	XOR	LVT
64	LVT	XOR	LVT
128	LVT	XOR	LVT
256	XOR	XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT
2K	XOR	XOR	LVT

(c) 8W/16R

Fig. 9. Xilinx design space navigation: for each memory depth, we list the design that minimizes delay (i.e., has the highest F_{max}), the number of Slices used, or the number of BRAMs used, for (a) 2W/4R, (b) 4W/8R, and (c) 8W/16R memories. We consider results within 5% as "equal", and list multiple designs in such cases.



(a) 3 ports (b) n ports

Fig. 10. LVT-based bidirectional multi-ported memory designs: (a) a 3-port design, and (b) a generalized n-port design. Note that these diagrams show only the data lines, without address wires.

256, the LE-to-LVT delay crossover returns to 32 due to the high initial slice count of the pure-LE design.

7. MEMORIES WITH BIDIRECTIONAL PORTS (ALTERA)

All of the memory designs considered so far implement unidirectional ports, where we configure each port at design-time for either only reading or writing. In this section, we describe and evaluate multi-ported memories with *bidirectional* ports, where each port can perform a read or a write dynamically selected each cycle.

Similar to the unidirectional memories we presented in the previous sections, we can build bidirectional memories using only logic elements (ALMs) and also compose larger memories more effectively using BRAMs. The initial work by Choi *et al.* presents a memory design having bidirectional ports based on an LVT approach [Choi et al. 2012]. For convenience, we provide a brief description of that design here.

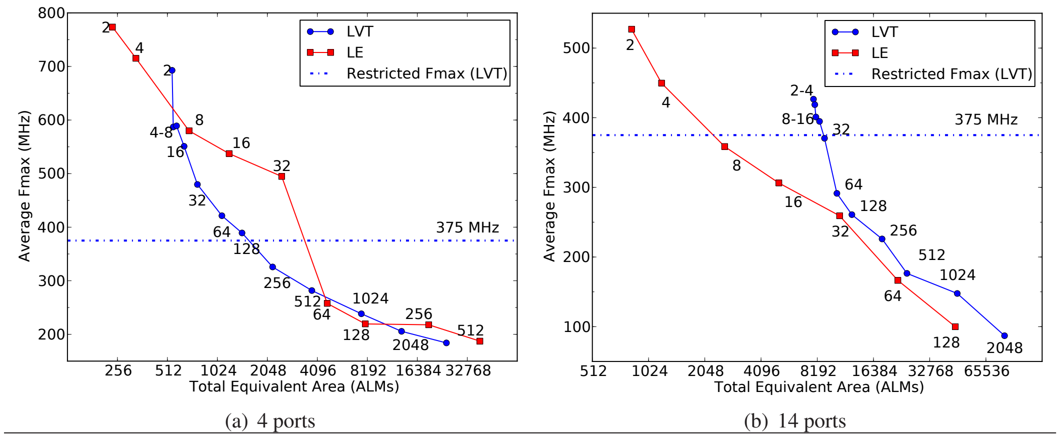


Fig. 11. Speed and area for LVT-based and pure-LE implementations of bidirectional memories of increasing depth for designs with (a) 4 ports and (b) 14 ports. The the dotted line indicates the 375 MHz restricted F_{max} for BRAM running in true dual-port mode on Stratix IV devices.

Figure 10(a) shows a 3-ported bidirectional LVT-based memory. Basically, the design has each possible pair of ports share a BRAM. Since there are three ports, there are three possible port pairs, and hence the design requires three BRAMs. We configure each BRAM in *true dual-port mode*, such that each port can dynamically switch from reading to writing each cycle. Note that a minimum clock pulse width requirement for BRAMs running in true dual-port mode restricts the actual F_{max} to 375 MHz on Stratix IV devices. For each port, a Live Value Table (LVT) composed of logic elements (ALMs) tracks which of its associated BRAMs holds the most recent value for a given location. As before, we must implement the LVT as a multi-ported memory, in this case with three read ports and three write ports.

Figure 10(b) shows a generalized bidirectional memory supporting n ports. The design requires $n(n-1)/2$ BRAMs, enough such that one BRAM connects together one of all possible port pairs, and also requires an LVT with n read and write ports, itself driving n output multiplexers each having $n-1$ inputs.

7.1. Results

In Figure 11, we compare the F_{max} and TEA for both pure-LE and LVT-based designs having a range of depths and supporting two contrasting numbers of ports: (a) 4 ports and (b) 14 ports. For 4-port memories, the pure-LE designs have better F_{max} and TEA for depths of 2 and 4. Beyond 4-deep, the LVT-based designs have better F_{max} and TEA. For 14-port memories, pure-LE designs have the lowest TEA for up to 16-deep memories, beyond which the LVT-based designs prevail.

In Figure 12, we plot the F_{max} and TEA for LVT-based bidirectional memories with 3 to 16 ports and depths ranging from 2 up to the maximum depth that fits the target Altera device (specified in Section 4.2). We label each series as $n$ports_mK to indicate the number (n) of ports and the maximum (m) depth in kilo-words (K) for that design. For example, the 3-port design can fit up to 4K entries, while the 16-port design can fit only up to 1K entries. The resulting frequency/area result space shows a reasonably regular and predictable trend, with F_{max} decreasing and TEA increasing with memory depth for each design. The incremental cost of adding each additional port diminishes with the total number of ports. At the greatest depths, the F_{max} of most designs converges to around 150 MHz.

8. COMPARING UNIDIRECTIONAL AND BIDIRECTIONAL DESIGNS (ALTERA)

In this section, we compare the speed and area of unidirectional and bidirectional designs on Altera devices. We compare the designs in two ways: (i) memories with the same *total* number

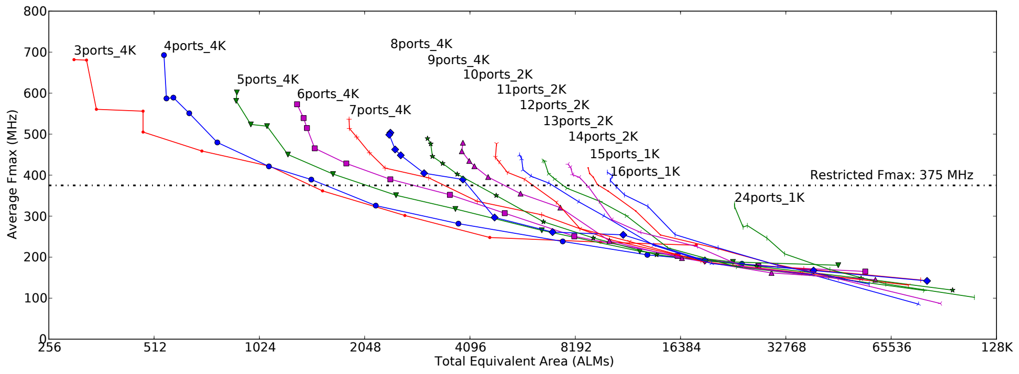


Fig. 12. Speed and area results for LVT-based bidirectional memories with 3 to 16 ports and depths ranging from 2 up to the maximum depth that fits the target device. We label each series as $n$ports_mK to indicate the number (n) of ports and the maximum (m) depth in kilo-words (K) for that design. Note that a minimum clock pulse width requirement for BRAMs running in true dual-port mode on Stratix IV devices restricts the actual F_{max} to 375 MHz.

of ports, under the assumption that a unidirectional memory implements a specialization of a bidirectional memory based on actual memory access patterns, and (ii) memories with the same *minimum equivalent read/write capacity*, where a unidirectional *nWnR* memory can support the same read/write access patterns as a *n-port* bidirectional memory in the worst case, as well as a greater number of simultaneous reads and writes in the best case.

Figure 13 shows speed and area comparisons of: (i) bidirectional LVT memories against unidirectional LVT and XOR memories with the same total number of ports: (a) 6 ports vs. 2w4r, (c) 12 ports vs. 4w8r, and (e) 24 ports vs. 8w16r, and (ii) between bidirectional and unidirectional LVT memories with the same minimum equivalent read/write capacity: (b) 4 ports vs. 4w4r, (d) 8 ports vs. 8w8r, (f) 12 ports vs. 12w12r. The labels denote the depth of each memory design point, and we excluded pure-LE designs for clarity. We report the *unrestricted* F_{max} of all designs, but frequency is actually restricted by the minimum clock pulse widths accepted by different BRAMs configurations: 550 MHz for simple dual-port BRAMs used in unidirectional memories, and 375 MHz for true dual-port BRAMs used in bidirectional memories.

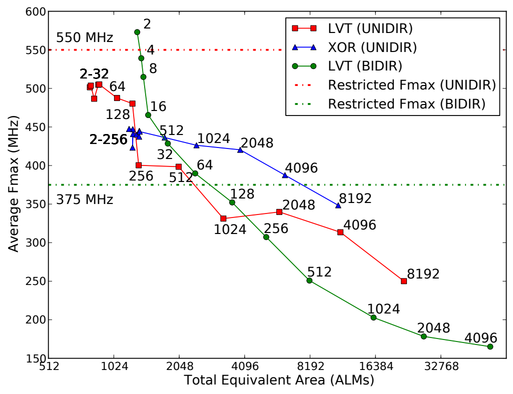
8.1. Same Total Number of Ports

When looking at Figures 13(a),(c), and (e), for depths exceeding a single BRAM (256 words), bidirectional designs run about 100 to 150 MHz slower than unidirectional LVT designs of the same depth. *Unfortunately, the 375 MHz clock pulse width restrictions on true dual-port BRAMs prevent the high speeds at depths of 32 words or less, which appear to exceed that of unidirectional XOR memories, from ever appearing in actual use.*

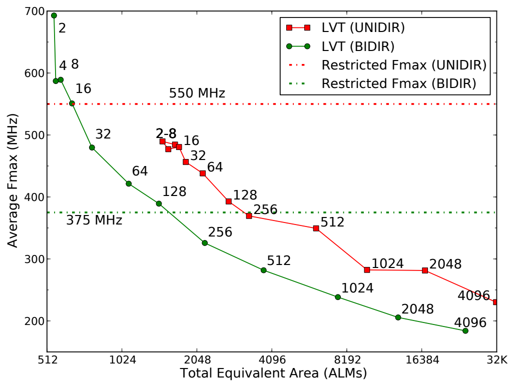
Regardless of any speed increase, replacing a bidirectional memory with an equivalent unidirectional one reduces area to a much greater degree. For example, when replacing a 4096-word 6-port bidirectional memory with a unidirectional 2w4r LVT or XOR memory, the area reduces 4.9x and 8.8x respectively. A designer could actually over-provision the number of unidirectional ports and still save area: compared to the same 4096-word 6-port bidirectional memory, unidirectional 4w8r LVT and XOR memories reduce area by 7.9% and 2.1x respectively, while still exceeding the F_{max} of the original bidirectional memory.

8.2. Same Minimum Equivalent Read/Write Capacity

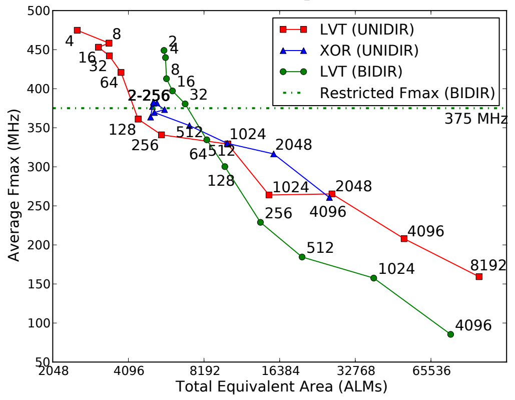
In this experiment we over-provision unidirectional memories to have n read and n write ports, to compare with an n-port bidirectional. In Figures 13(b),(d), and (f), we observe that for memories of depth 64 and shallower, the area of the BRAMs dominates, and thus bidirectional memories are



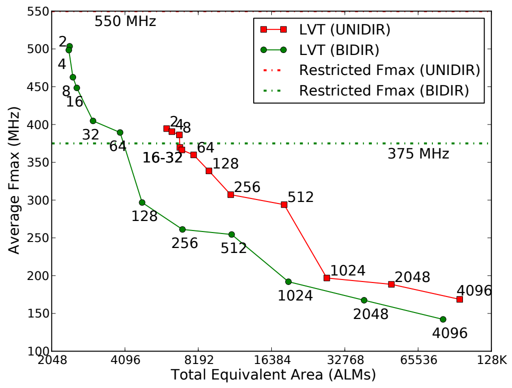
(a) 2w4r unidirectional vs 6 ports bidirectional



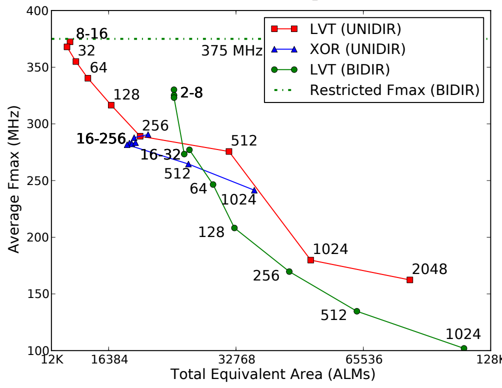
(b) 4w4r unidirectional vs 4 ports bidirectional



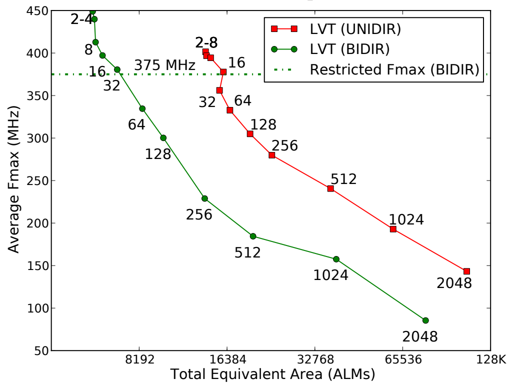
(c) 4w8r unidirectional vs 12 ports bidirectional



(d) 8w8r unidirectional vs 8 ports bidirectional

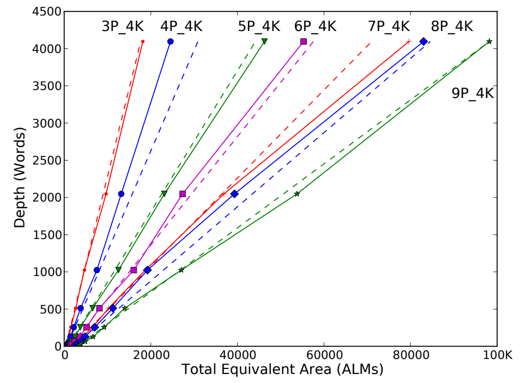


(e) 8w16r unidirectional vs 24 ports bidirectional

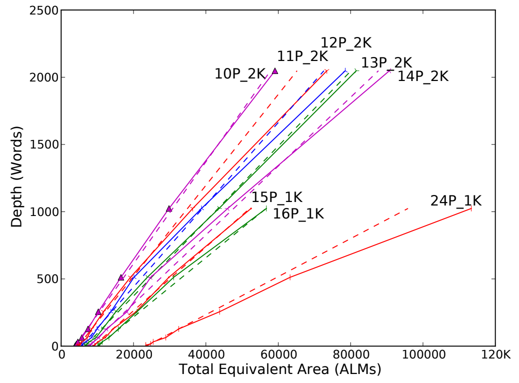


(f) 12w12r unidirectional vs 12 ports bidirectional

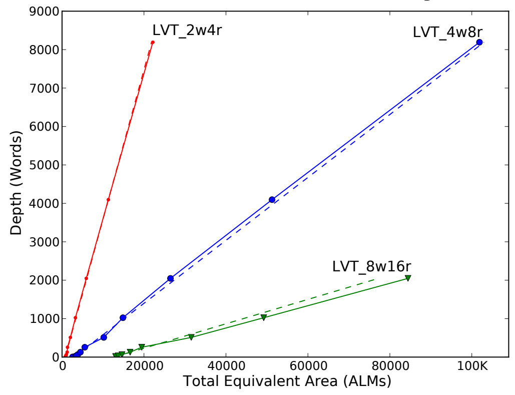
Fig. 13. Speed and area comparison of bidirectional LVT memories against unidirectional LVT and XOR memories with the same total number of ports: (a) 6 ports vs. 2w4r, (c) 12 ports vs. 4w8r, and (e) 24 ports vs. 8w16r, and between bidirectional and unidirectional LVT memories with the same minimum equivalent read/write capacity: (b) 4 ports vs. 4w4r, (d) 8 ports vs. 8w8r, (f) 12 ports vs. 12w12r. The labels denote the depth of each memory. We report the *unrestricted* F_{max} of all designs, but the minimum clock pulse widths accepted by different BRAM configurations (indicated by dotted lines) restrict the actual operating frequencies: 550 MHz for simple dual-port BRAMs used in unidirectional memories, and 375 MHz for true dual-port BRAMs used in bidirectional memories.



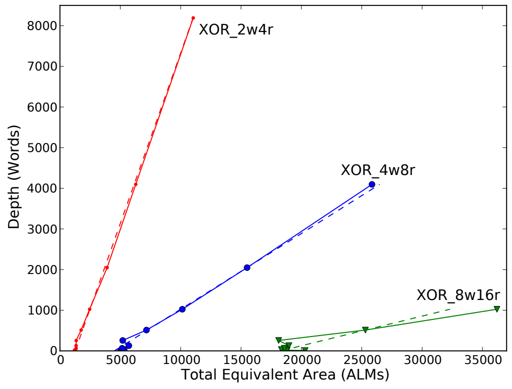
(a) Bidirectional LVT, 4096 words, 3 to 9 ports



(b) Bidirectional LVT, 1024/2048 words, 10-16 and 24 ports



(c) Unidirectional LVT, 2 to 8192 words



(d) Unidirectional XOR, 2 to 8192 words

Fig. 14. Illustration of the Total Equivalent Area (TEA) usage model for unidirectional and bidirectional multi-ported memory designs. Solid lines denote the original data from our experiments, while the dotted lines show the corresponding output of the area model. Most predicted points lie within 10% of the original data.

Design	Coefficients				
	a	b	c	d	e
LVT (UNIDIR)	479.685	-157.517	-7.073	57.605	2.411
XOR (UNIDIR)	442.033	-97.324	-2.980	75.754	1.043
LVT (BIDIR)	128.456	-62.333	-5.264	43.067	3.166

Fig. 15. Coefficients of the TEA usage model $ALM = a + b * ports + c * depth + d * ports^2 + e * ports * depth$, fitted to the lowest total absolute error.

smaller due to their lower BRAM count: $n(n-1)/2$ vs. n^2 for n ports. Since bidirectional memories are actually limited to a frequency of 375 MHz (due to for their use of true dual-port BRAMs), the corresponding LVT designs have a frequency advantage but a TEA disadvantage compared to bidirectional.

9. MODELING AREA USAGE (ALTERA)

In this section, we present a predictive model for the Total Equivalent Area (TEA) of our multi-ported memories on the Altera platform. The same model, using different coefficients, accurately predicts the TEA of unidirectional and bidirectional LVT memories, as well as unidirectional

XOR memories. We found the equation and its coefficients by applying the `pyeq2` curve-fitting library [Phillips 2012] to our actual area data, resulting in the following equation:

$$TEA = a + b * ports + c * depth + d * ports^2 + e * ports * depth \quad (1)$$

We fitted the curve to the lowest total absolute error, weighing all points equally. We use port count and memory depth as our independent variables, as the port count affects area quadratically and multiplies the impact of depth. For bidirectional designs, the port count refers to the total number of bidirectional ports. For unidirectional designs, the port count refers to the number of *read* ports. Thus the given coefficients only apply to unidirectional memories with a 2:1 ratio of read ports to write ports. *We manually removed the quadratic depth term since area does not have a quadratic relationship to depth: A near-zero coefficient introduces insignificant error even at large memory depths.*

Figure 15 lists the resulting coefficients for each memory design, as applied to Equation 1, while Figure 14 compares the predicted TEA to the original data. Solid lines denote the original data from our experiments, while the dotted lines show the corresponding output of the area model. Most predicted points lie within 10% of the original data. *The negative linear depth and port coefficients act as corrections to the dominating* $ports^2$ *and* $ports * depth$ *terms.* Since this area model views the CAD tool as a black box, we expect this model to only remain valid for Quartus 12.0, but serves as evidence that such a model should be buildable for a given version.

10. RELATED WORK

Most prior work on multi-ported memories for FPGAs focuses on register files for soft-processors. Simple replication provides the 1W/2R register file required to support a three-operand ISA [Yiannacouras et al. 2006; Carli 2008; Altera 2011; Fort et al. 2006; Moussali et al. 2007]. Jones *et al.* [Jones et al. 2005] implement a VLIW soft processor with a multi-ported register file, implemented entirely in logic elements, that constituted the operating frequency critical path. Saghir *et al.* [Saghir and Naous 2007; Saghir et al. 2006] also implement a multi-ported register file for a VLIW soft-processor, but use only replication and banking of BRAMs; however, their compiler must schedule register accesses to avoid conflicting reads and writes. Manjikian aggressively multipumps memories by performing reads and writes on consecutive rising and falling clock edges within a processor cycle [Manjikian 2003]—unfortunately, this design forces a multiple-phase clock on the entire system.

More recently, Anjam *et al.* [Anjam et al. 2010a] successfully use a LVT-based register file for their reconfigurable VLIW soft-processor and add one more internal addressing bit to enable splitting a 4W/8R register file into two independent 2W/4R instances. Later work by Anjam *et al.* [Anjam et al. 2010b] removes the need for the LVT by avoiding write bank conflicts via compile-time register renaming, but this solution requires more registers than are architecturally visible.

Hayenga et al. [Hayenga and Lipasti 2011] recently proposed adding an `XOR`-coded bank between two regular banks of an ASIC register file design, such that the `XOR`-coded bank augments the number of reads and writes supported. This approach cannot translate directly to an FPGA as each bank has 2W/4R ports. Our XOR approach differs in that *all* banks use `XOR`-coding. Naresh et al. [Naresh et al. 2011] also recently proposed using `XOR` coding to implement a crossbar that performs packet arbitration within a network-on-chip router.

11. CONCLUSIONS

In this article we described and evaluated, on both Altera and Xilinx FPGAs, several approaches to implementing unidirectional multi-ported memories: pure-LE, Live Value Table (LVT) [LaForest and Steffan 2010], and XOR [LaForest et al. 2012], along with an LVT approach to bidirectional memories [Choi et al. 2012]. We found that:

— pure-LE designs use no BRAMs, while the XOR designs use far less logic but more BRAMs than the LVT designs, demonstrating a resource diversity between the three designs that makes them each desirable for different use-cases;
— for the shallowest designs, building memories using pure-LEs (ALMs or slices) often results in the highest F_{max} or best area usage;
— the LVT approach always uses the least BRAMs;
— for moderate-depth designs the LVT approach is generally the fastest;
— for Xilinx FPGAs the XOR design almost always uses the fewest slices, while for Altera FPGAs the pure-LEs, LVT, and XOR designs each use the fewest ALMs for shallow, moderate, and high-depth designs respectively;
— unidirectional LVT memories that are over-provisioned to have the same read/write capability are faster but larger than corresponding bidirectional memories;
— it is possible to build an accurate predictive area model for unidirectional and bidirectional LVT memories.

To summarize, the pure-LE, LVT, and XOR approaches are valuable and useful in different situations, depending on the constraints and resource utilization (BRAMs vs logic) of the enclosing design. Designers can use the results of this work as a guide when choosing an appropriate multi-ported memory implementation.

12. FURTHER WORK

Although we revisited and expanded our work on multi-ported memories, we still have many directions not yet explored:

Bidirectional XOR Memories Although Choi *et al.* [Choi et al. 2012] extended unidirectional LVT memories to bidirectional ports in a fairly straightforward manner, we have not yet determined how to implement *bidirectional* XOR memories. Given the different logic and BRAM usage of XOR and LVT memories, bidirectional XOR memories might provide another useful range of design trade-offs.

Multi-Pumped LVT and XOR Memory Banks We had previously explored multipumped unidirectional LVT memories [LaForest and Steffan 2010], but we have not yet explored the idea of multipumping the BRAMs by themselves ("Pure Multi-Pumping") to increase their apparent number of ports, and then using these BRAMs to build LVT and XOR) multi-ported memories. Using Pure Multi-Pumped BRAMs would reduce the number of memory banks inside LVT and XOR memories, trading-off speed for a reduction in area.

Different Ratios of Read/Write Ports In all our work to date, we focus on unidirectional memories with twice as many read ports than write ports. These configurations fit computational applications, such as processor register files, where logic uses two operands to produce one result. We should study memories with other read/write port ratios, inspired by different applications.

Stalling Designs We also focus, to date, on memories that do not stall: all reads and writes complete in a single cycle. Lifting this restriction to trade-off on area and speed, for example by having reads that may have to wait for an available memory bank, would also be interesting to investigate, and would match some applications such as blocking L1 memory caches.

ACKNOWLEDGMENTS

The authors thank Altera, NSERC, and the Walter C. Sumner Memorial Fellowship for financial support. We thank Vaughn Betz, Jonathan Rose, and Ketan Padalia for help with tuning Quartus and our test harness, and the anonymous reviewers for constructive feedback. We also thank Jongsok Choi, Jason Anderson, and co-authors [Choi et al. 2012] for inventing and providing the bidirectional-ports design.

REFERENCES

Altera. 2003. Mercury Programmable Logic Device Family Data Sheet. http://www.altera.com/ds/archives/dsmercury.pdf. (Jan 2003). Version 2.2, Accessed Nov. 2012.

Altera. 2011. Nios II Processor Reference Handbook. http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf. (May 2011). Version 11.0.0, Accessed Nov. 2012.

Altera. 2012. DC and Switching Characteristics for Stratix IV Devices. http://www.altera.com/literature/hb/stratix-iv/stx4_siv54001.pdf. (July 2012). Version 5.3, Accessed Nov. 2012.

F. Anjam, M. Nadeem, and S. Wong. 2010a. A VLIW softcore processor with dynamically adjustable issue-slots. In *International Conference on Field-Programmable Technology (FPT)*. 393–398.

F. Anjam, S. Wong, and F. Nadeem. 2010b. A multiported register file with register renaming for configurable softcore VLIW processors. In *International Conference on Field-Programmable Technology (FPT)*. 403–408.

Roberto Carli. 2008. *Flexible MIPS Soft Processor Architecture*. Technical Report. Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory. http://hdl.handle.net/1721.1/41874

J. Choi, K. Nam, A. Canis, J.H. Anderson, S. Brown, and T. Czajkowski. 2012. Impact of cache architecture on speed and area of FPGA-based processor/parallel-accelerator systems. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 17–24.

B. Fort, D. Capalija, Z.G. Vranesic, and S.D. Brown. 2006. A Multithreaded Soft Processor for SoPC Area Reduction. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 131–142.

Mitchell Hayenga and Mikko Lipasti. 2011. The NoX router. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44 '11)*. 36–46.

Alex K. Jones, Raymond Hoare, Dara Kusic, Joshua Fazekas, and John Foster. 2005. An FPGA-based VLIW processor with custom hardware execution. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*. 107–117.

Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. 2006. XORs in the Air: Practical Wireless Network Coding. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 497–510.

Charles Eric LaForest, Ming Gang Liu, Emma Rapati, and J. Gregory Steffan. 2012. Multi-ported Memories for FPGAs via XOR. In *Proceedings of the 20th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. 209–218.

Charles Eric LaForest and J. Gregory Steffan. 2010. Efficient Multi-ported Memories for FPGAs. In *Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. 41–50.

N. Manjikian. 2003. Design Issues for Prototype Implementation of a Pipelined Superscalar Processor in Programmable Logic. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Vol. 1. 155–158.

Roger Moussali, Nabil Ghanem, and Mazen A. R. Saghir. 2007. Supporting multithreading in configurable soft processor cores. In *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. 155–159.

Vignyan Reddy Kothinti Naresh, David J. Palframan, and Mikko H. Lipasti. 2011. CRAM: Coded Registers for Amplified Multiporting. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 196–205.

David A. Patterson, Garth Gibson, and Randy H. Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*. 109–116.

James R. Phillips. 2012. py2eq curve-fitting library. http://code.google.com/p/pyeq2/. (2012). Accessed Aug. 2013.

Mazen Saghir and Rawan Naous. 2007. A Configurable Multi-ported Register File Architecture for Soft Processor Cores. In *Proceedings of the 2007 International Workshop on Applied Reconfigurable Computing (ARC)*. Springer-Verlag, 14–25.

Mazen A. R. Saghir, Mohamad El-Majzoub, and Patrick Akl. 2006. Datapath and ISA Customization for Soft VLIW Processors. In *IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 1–10.

Henry Wong, Vaughn Betz, and Jonathan Rose. 2011. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. 5–14.

Xilinx. 2012. Virtex-6 FPGA Data Sheet: DC and Switching Characteristics. http://www.xilinx.com/support/documentation/data_sheets/ds152.pdf. (January 2012). Version 3.4, Accessed Dec. 2012.

Peter Yiannacouras, J. Gregory Steffan, and Jonathan Rose. 2006. Application-specific customization of soft processor microarchitecture. In *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays (FPGA)*. 201–210.

Received December 2012; revised December 2013; accepted February 2014