Multi-Port Memory with Bi-directional Ports for FPGAs Using XOR and LVT Methods

XXX
XXX@XXX

Abstract

We propose an extension to XOR memory implementation on FPGAs. We generalize the previous XOR memory implementation to allow for any number of full, read-only, and write-only ports. This paper also presents an efficient architecture for creating live value table (LVT) memory using an XOR-based scheme by utilizing its bidirectional capabilities. This implementation exploits the properties of XOR to allow any data entry to be reconstructed by XORing the corresponding entries from all memory banks. The result is a high-throughput multi-ported memory that is particularly well-suited for implementation on FPGAs. We use an XOR memory with full ports to implement a bi-directional live value table design. We evaluate the architecture's performance and resource utilization, and show that the XOR-based bidirectional live value table is a compelling alternative for applications requiring high-performance, flexible memory access.

ACM Reference Format:
XXX. 2025. Multi-Port Memory with Bi-directional Ports for FPGAs Using XOR and LVT Methods. In *Proceedings of International Symposium on Field Programable Gate Arrays (FPGA '26).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Motivation

As computation needs keep increasing, one way to keep up has been specialized architectures. FPGAs provide a way to implement architectures without taping out an ASIC. However, the limitations of FPGA resources requires some creativity to map designs to FPGAs. This paper explores how to overcome the limitation of FPGAs that have a limited number of ports. Specifically we propose a method to create memories with more than 2 ports.

The major FPGA vendors (AMD[13], Intel[5], Lattice[9], Microchip[10], and Achronix[2]) implement distributed memory (small memories) and block memory (large memories) differently. However they all share some characteristics. All vendors support distributed memory configurations with 1 full or write port and between 1 to 3 read ports. All vendors support block memory with 2 full ports. AMD has a limited offering of multiport memories, but otherwise none of the vendors support memories with more than 2 full ports. Although this limitation is problematic for designs requiring multiple ports (and particularly write or full ports), we show that these resources make it possible to achieve high throughput quad and octal full port memories.

FPGA '26, Seaside, CA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

2 Source Code

We provide all of the source code used in implementation and testing our design at https://anonymous.4open.science/r/mpm-7666/. We tested our design with Verilator[12] and synthesized the design with Vivado[3] targeting an AMD Virtex UltraScale+ HBM VU47P-3 FPGA.

3 Related Work

Several previous solutions to the port limit on FPGAs exist including multi-pumping, banking and replication[8]. Multi-pumping is the process of reducing the clock speed to increase the number of ports. For example, a 300Mhz single port memory can handle two 150Mhz ports. Banking[11] requires stalls and routing logic due to the segmented memory. Our design is most similar to replication. Replication involves tying the write ports of multiple memories together to create additional read ports.

4 XOR memory

We propose a simple generalization to XOR memories presented in previous work[6]. XOR memories work by using the following property:

$$a \oplus b \oplus b = a \tag{1}$$

Where $\oplus$ is the bitwise XOR operator. Combined with the commutative and associative property of XOR we can generalize this property to:

$$b_0 \oplus \cdots \oplus b_n(a \oplus b_0 \cdots \oplus b_n) = a \tag{2}$$

We add bidirectional ports and analyze the perforamance of distributed memory and block memory versions of this design. We also present applications for these memories.

The number of RAMs (NRAM) needed is:

$$NRAM = (W + F)(W + F + R) - W \tag{3}$$

Where W is the number of write ports, R is the number of read ports and F is the number of full ports. This expands to:

$$NRAM = W^2 + 2WF + F^2 + WR + FR - W \tag{4}$$

Figure 1 demonstrates how to create an XOR memory with 2 read ports, 2 write ports and 2 full ports, which requires 22 RAMs. To read data from an XOR memory all of the data from the RAMs in one column is XORed from the same address. For example, say address x has values A, B, C and D, the data read would be $A \oplus B \oplus C \oplus D$.

Writing to the memory involves reading from all memories except the current row (say row/port 2 in figure 1) and XORing the incoming data E (in the example this results in $A \oplus B \oplus D \oplus E$) and storing that value in all the RAMs in that row.

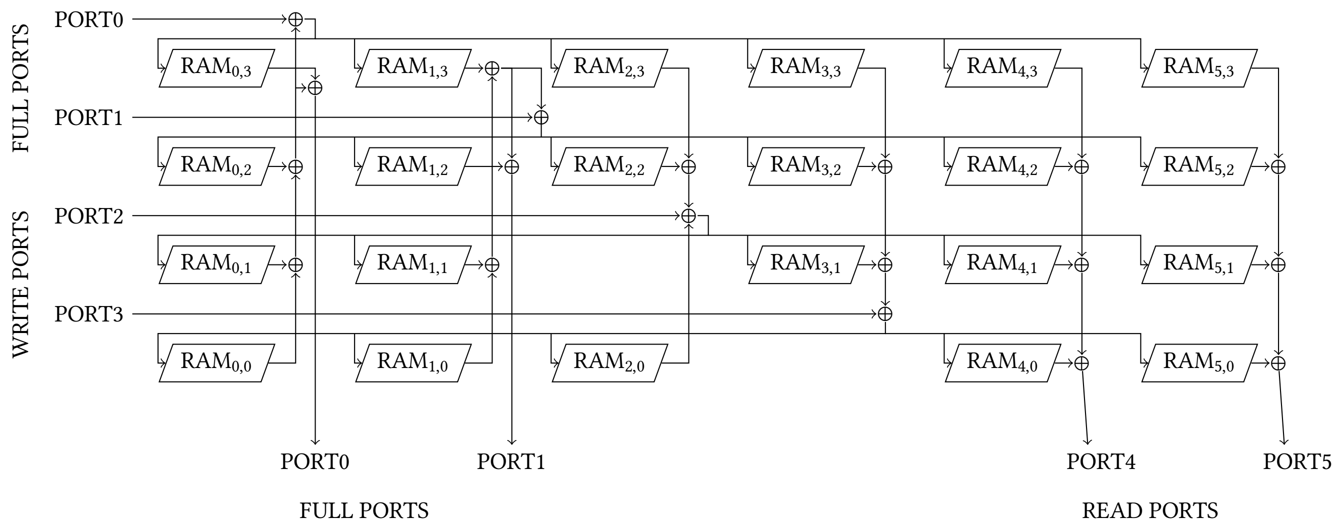


Figure 1: A multi-port memory with 2 full ports, 2 write-only ports and 2 read-only ports.

The next time that data is read the result will be $A \oplus B \oplus (A \oplus B \oplus D \oplus E) \oplus D$, which equals E.

You may notice that writing to a port involves XORing all but one stored value and reading involves XORing all values. This enables full ports to be created just by adding one RAM to what would otherwise be just a write port.

Note that this memory requires that all of the RAMs in a row have the same data. Initializing the rams to the same data (e.g., all 0s) is required for the memory to operate properly. This is not an issue in FPGAs since the memory can be initialized to 0. since rows are written to at the same time, the same they will remain the same as long as the memories are initially.

We provide a second example with Figure 2 showing a clock cycle of an XOR memory with 2 full ports.

5 Analysis of XOR memory

We analyze several configurations of XOR memories. Particularly we vary the number of ports and width of the memory.

Table 1 we show the varying resources and frequency of the memory for different numbers of ports. As seen the number of LUTs used for memory is relatively large particularly for more ports.

As expected increasing the width of the memory resulted in a roughly linear increase in resource usage and a decrease in timing perforamance (see table 2).

Also as expected increasing the depth of the memory resulted in a roughly linear increase in resource usage and a decrease in timing performance (see table 3).

In table 4 we show an implementation using Block RAMs. This required pipelining and introducing a cycle of write delay. One could change the block ram to be write before read to remove the cycle of write delay.

Table 1: Synthesis results of XOR memory for different port counts. The memory has a depth of 1024 and width of 32bits.

Ports	LUTS	LUTS configured as memory	FF	BRAM	Max Frequency
2	1,492	1,216	0	0	370Mhz
4	6,312	5,248	0	0	317Mhz
8	26,576	21,760	0	0	241Mhz
16[1]	107,424	88,576	0	0	XMhz
32[2]	435,008	357,376	0	0	N/A

Table 2: Synthesis results of XOR memory for different widths. The memory has a depth of 1024 and 8 ports.

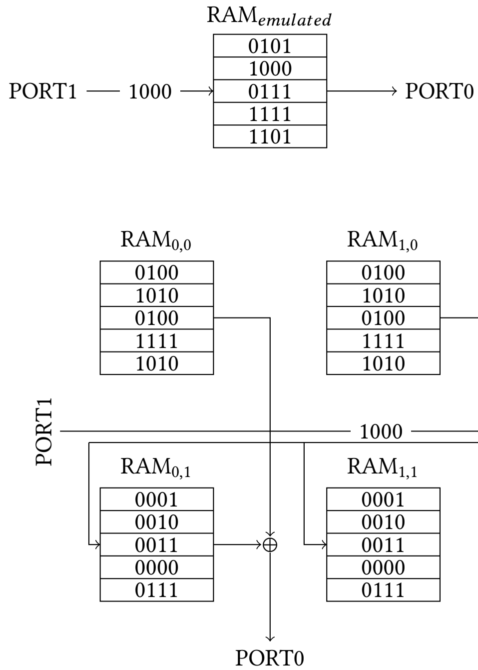
Width	LUTS	LUTS configured as memory	FF	BRAM	Max Frequency
1	1,096	960	0	0	XMhz
2	2,144	1,920	0	0	290Mhz
4	3,616	2,944	0	0	260Mhz
8	7,152	5,888	0	0	235Mhz
16	13,328	10,880	0	0	227Mhz
32	26,576	21,760	0	0	217Mhz

6 Live Value Table Memory

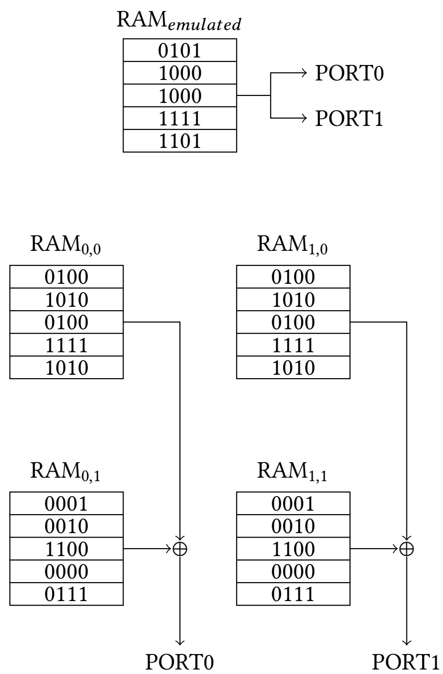
XOR memories can be used by themselves, however a live value table (LVT) may be more efficient.

We present a LVT memory that utilizes XOR memory.

Previous work used distributed memory [1]. However this work did not use bidirectional XOR ports in their implementation. In an elternate implementation[7], the live value table was implemented with registers.



(a) The memory on clock cycle 0, where 1000_2 is written to address 2 from port 1 and 0111_2 is read from address 2 from port 0.



(b) Clock cycle 1.

Figure 2: XOR Multi-port memory.

Table 3: Synthesis results of XOR memory for different depths. The memory has a width of 32 bits and 8 ports.

Depth	LUTS	LUTS configured as memory	FF	BRAM	Max Frequency
32	X	X	X	X	X
64	X	X	X	X	X
128	X	X	X	X	X
256	X	X	X	X	X
512	X	X	X	X	X
1,024	X	X	X	X	X

Table 4: Synthesis results of XOR memory with block RAMs for different port counts. The memory has a depth of 1024 and width of 32 bits.

Ports	LUTS	LUTS configured as memory	FF	BRAM	Max Frequency
2	128	0	150	4	278Mhz
4	256	0	300	16	269Mhz
8	768	0	600	64	197Mhz
16[1]	3,072	0	1,200	256	XMhz
32[1]	10,240	0	2,400	1024	XMhz

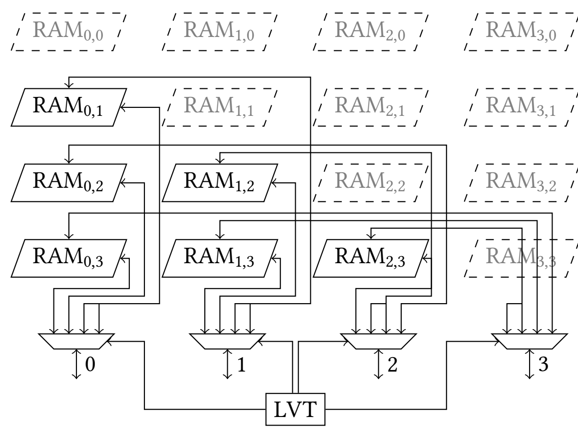


Figure 3: Multi-port memory created with bi-directional dual-port memories. Note, a live value table is needed to determine which memory has the most recent value.

We create a LVT memory using the technique described in [4]. This live value memory is composed of 2-(full)port memories. Each port shares a RAM with another port. This results in $F(F-1)/2$ RAMs being needed, where F is the number of bi-directional (full) ports. See figure 3.

The memory gets its name because of a multi-port memory that tracks the most recent stored value (aka a live value table). The point of a multi-port memory that requires a multi-port memory is that wide (e.g. 32 bit data) can be stored more effeciently this way.

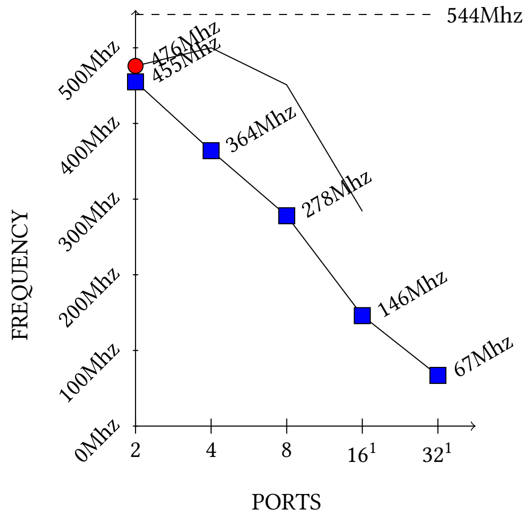


Figure 4: Frequency of LVT design.

Table 5: Synthesis results of LVT design for different port counts.

Ports	LUTS	LUTS configured as memory	FF	BRAM	Max Frequency
2	191	96	0	1	588Mhz
4	1,144	896	0	6	556Mhz
8	5,428	3,968	0	28	455Mhz
16[1]	31,075	24,064	0	120	250Mhz
32[1]	161,216	129,536	0	496	127Mhz

Table 6: Synthesis results of LVT design for different port counts for pipelined design

Ports	LUTS	LUTS configured as memory	FF	BRAM	Max Frequency
2	119	64	26	1	714Mhz
4	1,268	1,024	96	6	667Mhz
8	5,588	4,096	160	28	625Mhz
16[1]	31,328	24,576	688	120	417Mhz
32[1]	162,960	131,072	2,480	496	247Mhz

Instead of using a register based live-value table as in [4] we use a xor memory similar to [1].

We show we utilize x% less resources than LVT and I-LVT.

7 Analysis of LVT Memory

We explore LVT designs with 2 to 32 ports. Although 16 and 32 port designs fit on large FPGAs, we believe smaller 4 and 8 port designs are more practical. We say more practical because of the high resource usage of XOR and LVT memories at high port counts. F^2 for XOR and $F(F-1)/2$ for LVT. However we were able to synthesize a 32 port memory. Higher port counts also had worse timing performance (see table 5 and figure 4).

To get better timing performance we created a pipelined version of the memory. This memory has major drawbacks: In addition to read delay the memory has write delay. The write delay means write conflicts occur on adjacent clock cycles not just current clock cycles. However we achieve better timing performance with this memory (see table 6 and figure 4).

Without write delay an 8 port memory runs at Xmhz (x% of max). With write delay and pipelining the design runs at Xmhz (x% of max).

8 Conclusion

XOR and LVT techniques can efficiently create memories with multiple ports. Although at the cost of using more memory space. Depending on the application, these memories may be the best option from the many available when creating a multi-port memory.

References

[1] Ameer M. S. Abdelhadi and Guy G. F. Lemieux. 2016. Modular Switched Multiported SRAM-Based Memories. *ACM Trans. Reconfigurable Technol. Syst.* 9, 3, Article 22 (July 2016), 26 pages. doi:10.1145/2851506

[2] Achronix Semiconductor Corporation 2022. *Speedster7t FPGA Datasheet (DS015).* Achronix Semiconductor Corporation. https://www.achronix.com/sites/default/files/docs/Speedster7t_FPGA_Datasheet_DS015_8.pdf This document contains preliminary information and is subject to change without notice..

[3] AMD. 2025. *Vivado Design Suite User Guide: Using the Vivado IDE* (2025.1 english ed.). AMD, San Jose, CA, USA. https://docs.amd.com/r/en-US/ug893-vivado-ide

[4] Jongsok Choi, Kevin Nam, Andrew Canis, Jason Anderson, Stephen Brown, and Tomasz Czajkowski. 2012. Impact of Cache Architecture and Interface on Performance and Area of FPGA-Based Processor/Parallel-Accelerator Systems. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. 17–24. doi:10.1109/FCCM.2012.13

[5] Intel Corporation 2024. *Intel Agilex 7 FPGA and SoC FPGA Datasheet.* Intel Corporation. https://www.intel.com/content/www/us/en/programmable/support/literature/lit-agilex.html Version 2024.09.03.

[6] Charles Eric Laforest, Zimo Li, Tristan O'rourke, Ming G. Liu, and J. Gregory Steffan. 2014. Composing Multi-Ported Memories on FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 7, 3, Article 16 (Sept. 2014), 23 pages. doi:10.1145/2629629

[7] Charles Eric Laforest, Ming G. Liu, Emma Rae Rapati, and J. Gregory Steffan. 2012. Multi-ported memories for FPGAs via XOR. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '12)*. Association for Computing Machinery, New York, NY, USA, 209–218. doi:10.1145/2145694.2145730

[8] 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* (Monterey, California, USA) *(FPGA '10)*. Association for Computing Machinery, New York, NY, USA, 41–50. doi:10.1145/1723112.1723122

[9] Lattice Semiconductor 2021. *FPGA-DS-02000 – Lattice iCE40 Family Data Sheet.* Lattice Semiconductor. https://www.latticesemi.com/view_document?document_id=52424 Accessed on 2025-09-08.

[10] Microchip Technology Inc. 2025. *PolarFire and PolarFire SoC FPGA Fabric User Guide.* Microchip Technology Inc. https://www.microchip.com/content/dam/mchp/documents/FPGA/ProductDocuments/UserGuides/PolarFire_PolarFire_SoC_FPGA_Fabric_User_Guide_VB.pdf Document Number: UG0680.

[11] Kevin R. Townsend, Osama G. Attia, Phillip H. Jones, and Joseph Zambreno. 2015. A Scalable Unsegmented Multiport Memory for FPGA-Based Systems. *International Journal of Reconfigurable Computing* 2015, 1 (2015), 826283. doi:10.1155/2015/826283 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1155/2015/826283

[12] Veripool, Inc. 2024. *Verilator Reference Manual.* Veripool, Inc. https://verilator.org/guide/latest/ Version 5.041 (or latest as of your date of use). Available at https://verilator.org/guide/latest/.

[13] Xilinx. 2020. *7 Series FPGAs Data Sheet: Overview.* Xilinx. https://docs.amd.com/api/khub/documents/2LByHkO~nSZXcei2D55fTg/content v2.6.1.

[0]To reduce the number of IO ports and fit the design on the FPGA we used a wrapper for the multi-port memory for designs with 16 and 32 ports.

Received 1 Octomber 2025; revised XX XXX XXXX; accepted XX XXX XXXX