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

XXX
XXX@XXX

Abstract

We propose a generalization to a previous XOR memory implementation to allow for any number of full, read-only, and write-only ports. This implementation creates these additional ports by encoding and decoding the data entries with XOR logic. The incoming (written) data is encoded with the data corresponding to the other ports and the outgoing (read) data is decoded by XORing all the entries in the memory banks. Using XOR operators to accomplish this encoding/decoding has advantages over other approaches because the XOR operator is a simple and efficient operation. The result is a high-throughput multi-ported memory that is particularly well-suited for implementation on FPGAs. This paper also presents an efficient architecture for creating a live value table (LVT) memory using an XOR-based scheme by utilizing its bidirectional capabilities. 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 such 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 that FPGA memories 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 can be configured to achieve high throughput 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 of FPGA memories 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 Generalizing 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. Equation 1 follows from:

$$a \oplus 0 = a \tag{2}$$

and

$$a \oplus a = 0 \tag{3}$$

Meaning XOR is the inverse operator of itself. Combined with the commutative and associative property of XOR we can generalize this to:

$$b_0 \oplus \cdots \oplus b_n \oplus (a \oplus b_0 \cdots \oplus b_n) = a \tag{4}$$

This implementation creates these additional ports by encoding and decoding the entries with XOR logic. The incoming (written) data is encoded with the data corresponding to the other ports, and the outgoing (read) data is decoded by XORing all the entries in the memory banks. Using XOR operators to accomplish this encoding/decoding has advantages over other approaches, because the XOR operator is a simple and efficient operation.

We use XOR logic to add bidirectional ports and analyze the performance of distributed memory and block memory versions of this design.

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

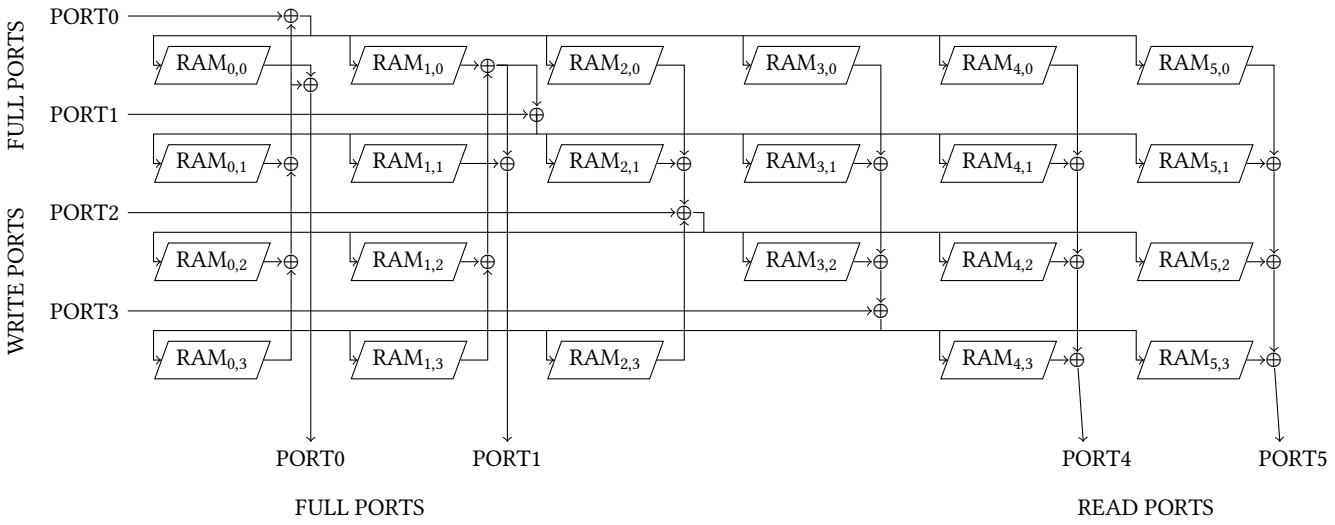


Figure 1: A multi-port memory with 2 full ports, 2 write-only ports and 2 read-only ports.

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 (e.g., 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. 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.

Note 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.

The number of RAMs (N_{RAM}) needed for this memory is:

$$N_{RAM} = (W + F)(W + F + R) - W \tag{5}$$

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:

$$N_{RAM} = W^2 + 2WF + F^2 + WR + FR - W \tag{6}$$

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, they will remain the same as long as the memories are initially the same.

We provide a second example with Figure 2 showing two clock cycles of operation of an XOR memory with 2 full ports. We use 2 ports for simplicity. This example can be extrapolated to more ports.

5 Analysis of XOR Memory Performance

We analyze several configurations of XOR memories in terms of resource usage. Tables 1 through 4 present some sample performance characteristics, including usage of look up tables (LUTs), flip-flops (FFs), block RAMs (BRAMs) and the maximum frequency that can be achieved under variations of number of ports and the width and depth of the memory.

Table 1 shows the varying resources and max frequency of the memory for different numbers of ports. As seen the number of LUTs used for memory is relatively large and particularly large for more ports. We use 2, 4, 8, 16, and 32 ports although there is no limitation for using other port counts (3, 5, 6, 7, etc.).

Table 2 shows the varying resources and max frequency of the memory for different widths. As expected increasing the width of the memory resulted in a roughly linear increase in resource usage and a decrease in timing performance.

Table 3 shows the varying resources and max frequency of the memory for different memory depths. Also as expected, increasing the depth of the memory resulted in a roughly linear increase in resource usage and a decrease in timing performance.

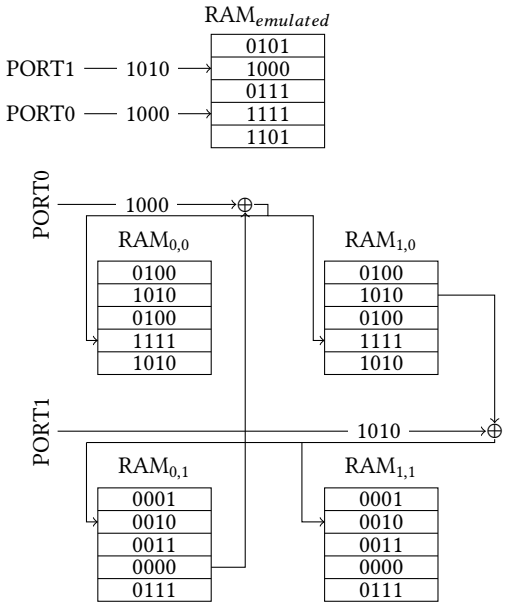
Table 4 shows results from an implementation using block RAMs. This required pipelining and introducing a cycle of write delay. We also changed the block RAM to be write before read to remove the cycle of write delay.

6 Live Value Table Memory

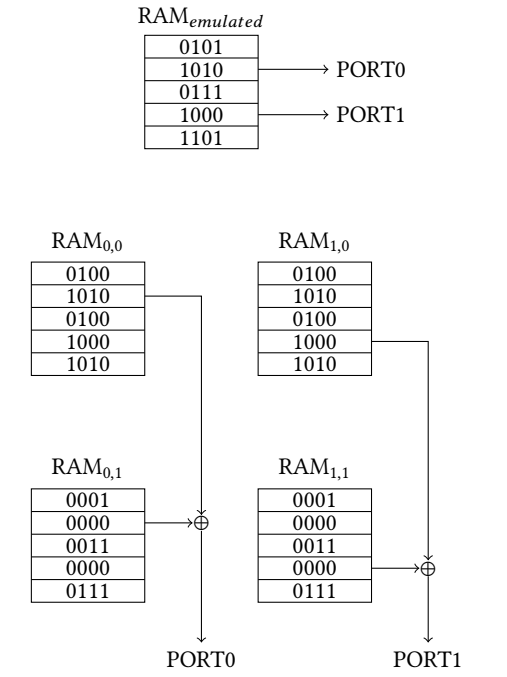
XOR memories can be used by themselves, but combining a live value table (LVT) with XOR may be more efficient. We present a LVT memory that utilizes XOR memory.

Previous work used distributed memory [1]. However, that work did not use bidirectional XOR ports in their implementation. In an alternate implementations[7][4], the live value table was implemented with registers.

We create a LVT memory using the technique described in [4]. This LVT memory is composed of bi-directional dual-port memories. Each port of the LVT memory shares a RAM with another port of the LVT memory. This results in $F(F-1)/2$ RAMs being needed, where F is the number of bi-directional (full) ports. This is less than the F^2 RAMs needed in the XOR implementation. Figure 3 illustrates the implementation for 4 ports.



(a) The memory on clock cycle 0, where 1000_2 is written to address 1 from port 0 and 1010_2 is written to address 3 from port 1.



(b) The memory on clock cycle 1, where 1010_2 is read from address 3 from port 0 and 1000_2 is read from address 1 from port 1.

Figure 2: This shows an XOR memory during 2 clock cycles of operation. The XOR memory has 2 full ports and is constructed from 4 RAMs with 1 read and 1 write port.

Table 1: Synthesis results of XOR memory for different port counts. The memory has a depth of 1024 and width of 32 bits. 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 ports. The 32 port design used more resources than were available on the FPGA therefore we do not have timing for that design.

Ports	LUTs	LUTs configured as memory	FFs	BRAMs	Max Frequency
2	2,706	2,304	0	0	625Mhz
4	11,556	9,728	0	0	500Mhz
8	48,328	39,936	0	0	357Mhz
16	196,640	161,792	0	0	150Mhz
32	794,688	651,264	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	FFs	BRAMs	Max Frequency
1	2,032	1,920	0	0	526Mhz
2	4,040	3,840	0	0	500Mhz
4	8,797	7,680	0	0	400Mhz
8	12,140	9,984	0	0	385Mhz
16	24,201	19,968	0	0	370Mhz
32	48,328	39,936	0	0	357Mhz

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	FFs	BRAMs	Max Frequency
32	2,016	1,248	0	0	588Mhz
64	3,264	2,496	0	0	556Mhz
128	6,483	4,992	0	0	526Mhz
256	12,565	9,984	0	0	476Mhz
512	24,621	19,968	0	0	455Mhz
1,024	48,328	39,936	0	0	357Mhz

The memory gets its name because it uses multi-port memory to track the most recent stored value (i.e., 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 efficiently this way, since the live value table is just storing the port numbers. (e.g., In Figure 3 the live value table memory has a width of 2 since there are 4 ports.)

7 Analysis of LVT Memory

We explore LVT designs with 2 to 32 ports. Although 16 and 32 port designs can fit on large FPGAs, we believe smaller 4 and 8

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	FFs	BRAMs	Max Frequency
2	128	0	150	4	625Mhz
4	256	0	300	16	476Mhz
8	768	0	600	64	370Mhz
16[1]	3,072	0	1,200	256	263Mhz
32[1]	10,240	0	2,400	1024	123Mhz

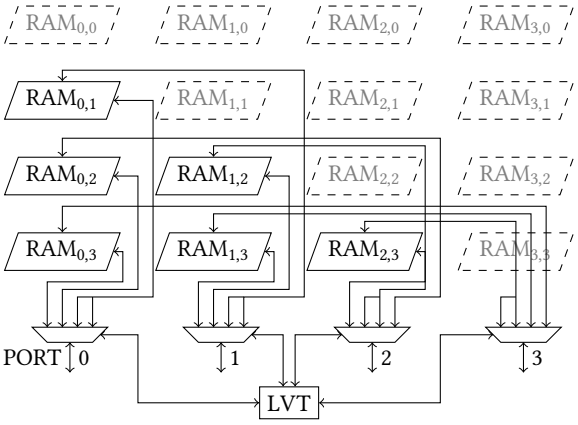


Figure 3: Multi-port memory created with bi-directional dual-port memories. Note, a live value table is needed to determine which memory has the most recent value.

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).

Table 5 shows the varying resources and max frequency of the memory for different numbers of ports.

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 and not just current clock cycles. However, we achieve better timing performance with this memory (see Table 6 and Figure 4).

Table 6 shows the varying resources and max frequency of the memory for different numbers of ports.

Without write delay an 8 port memory runs at 455mhz (63% of max frequency of the block RAM). With write delay and pipelining the design runs at 625mhz (87% of max frequency of the block RAM).

8 Conclusion

A combination of XOR and LVT techniques can efficiently create FPGA memories with multiple ports, but at the cost of using more memory space. Depending on the application, committing these memories may be the best option when creating a multi-port memory.

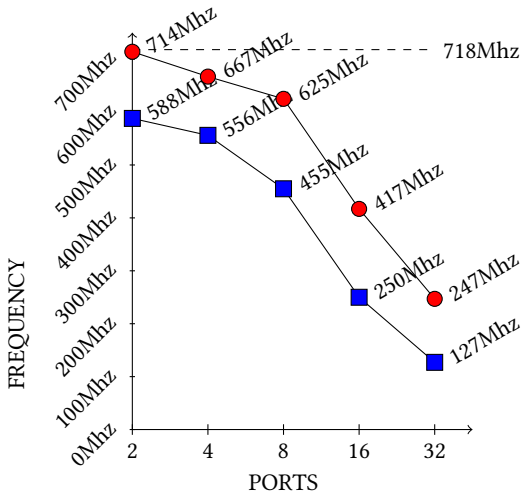


Figure 4: Frequency of LVT design. The ■s are the (non-pipelined) LVT memory designs and the ●s are the pipelined LVT memory designs.

Table 5: Synthesis results of LVT design for different port counts. 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 32ports.

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

Table 6: Synthesis results of LVT design for different port counts for pipelined design. 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 32ports.

Ports	LUTs	LUTs configured as memory	FFs	BRAMs	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	31,328	24,576	688	120	417Mhz
32	162,960	131,072	2,480	496	247Mhz

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.

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