

Yet Another Multi-Port Memory for FPGAs Using XOR and LVT Methods

XXX
XXX@XXX

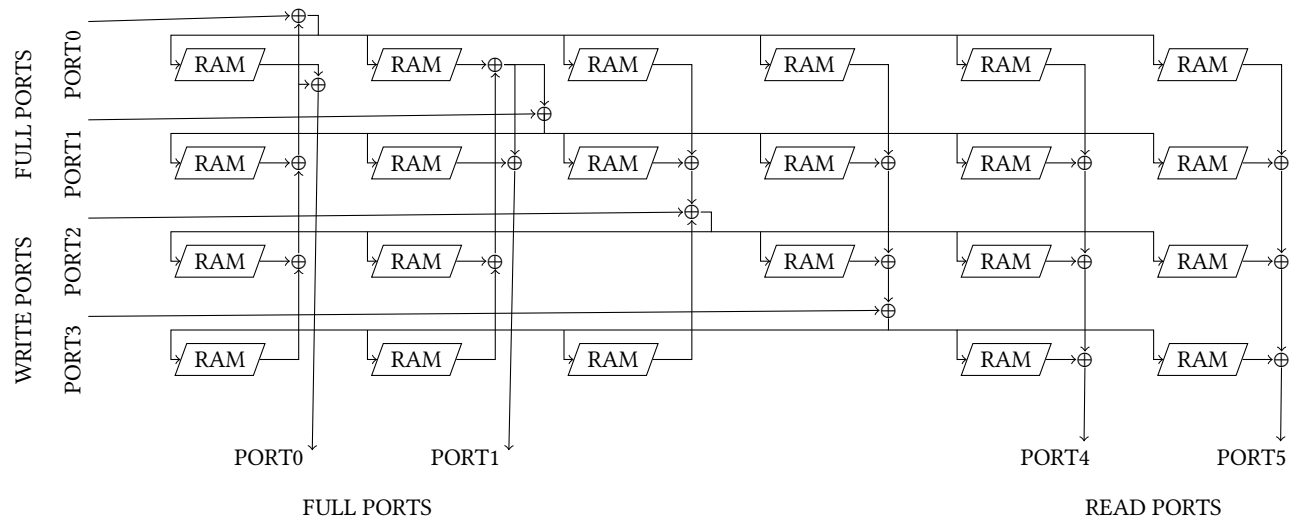


Figure 1: A multi-port memory with 2 full ports, 2 write-only ports and 2 read-only ports.

Abstract

We propose a simple extension to XOR memories presented in previous work. In this paper we generalize the XOR memory allowing for any number of full, read-only and write-only ports. FPGAs. This paper also presents a novel and efficient architecture for creating live value table memory using an XOR-based scheme, emphasizing its bidirectional capabilities.

This is achieved by exploiting the properties of XOR, allowing any data entry to be reconstructed by XORing the corresponding entries from all memory banks. The result is a high-throughput, coherent multi-ported memory that is particularly well-suited for implementation on FPGAs.

We use an XOR memory with full ports to implement a live value table (LVT) design. We evaluate the architecture's performance and resource utilization, showing that it uses significantly less logic and can achieve higher frequencies for deep memory configurations compared to LVT-based designs. This makes the XOR-based bidirectional live value table a compelling alternative for applications requiring high-performance, flexible memory access.

ACM Reference Format:

XXX. 2025. Yet Another Multi-Port Memory for FPGAs Using XOR and LVT Methods. In *Proceedings of International Symposium on Field Programmable Gate Arrays (FPGA '26)*. ACM, New York, NY, USA, 3 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Motivation

As computation needs keep increasing, one way to keep up with Moore's Law has been specialized architectures. FPGAs provide a way to implement architectures without taping out an ASIC. However, the limitations of FPGA resources means some creativity is needed to map designs to FPGAs. This paper explores the limitation of FPGAs in the fact that FPGA memories have a limited number of ports. Specifically we look at creating memories with more than 2 ports.

The 3 major FPGA vendors[3, 6, 7] implement distributed memory (small memories) and block memory (large memories) differently, however they share some characteristics.

All 3 vendors support distributed memory configurations with 1 full port and between 1 to 3 read ports.

All 3 vendors support block memory with 2 full ports. None of the vendors support memories with more than 2 full ports.

Although this limitation is problematic for designs requiring multiple ports particularly write or full ports, we show that these resources make it possible to achieve high throughput quad and octal full port memories.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FPGA '26, Seaside, CA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
<https://doi.org/XXXXXXX.XXXXXXX>

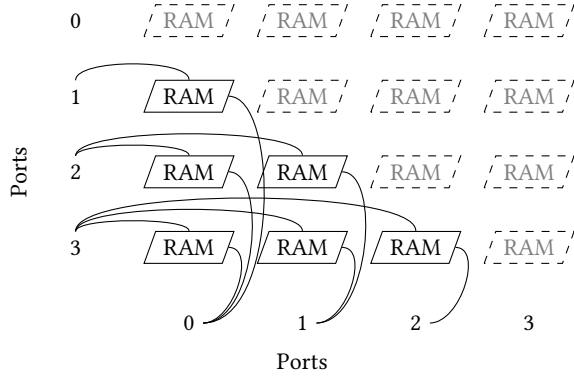


Figure 2: Multiport memory created with dual-port memories.

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 and implemented the design with Vivado (Xilinx/AMD).

3 XOR memory

We propose a simple generalization to XOR memories presented in previous work[4]. XOR memories work by using the $a \oplus b \oplus b = a$ property. We add bidirectional ports and analyze the performance of distributed memory and block memory versions of this design. We also present applications for these memories.

The number of RAMs needed is: $(W + F)(W + F + R) - W$, which expands to: $W^2 + 2WF + F^2 + WR + FR - W$.

Reading data from an XOR memory simply involves XORing all of the data from the RAMs in one column 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 the example) 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 .

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

4 Live Value Table Memory

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

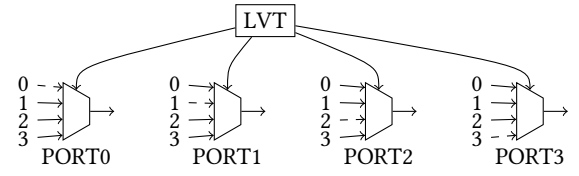


Figure 3: LVT Multiport memory created with dual-port memories.

Table 1: Synthesis results for different port counts

Ports	LUTS	FF	BRAM	Max Frequency
2	127	0	1	455Mhz
4	640	0	6	364Mhz
8	3,012	0	28	278Mhz
16 ¹	16,544	0	120	146Mhz
32 ¹	85,728	0	496	67Mhz

We present a LVT memory that utilises distributed memory xor live value table.

In [5] the live value table was implemented with registers. Previous work used distributed memory [1]. However they did not use bidirectional xor ports in their implementation.

We create a LVT memory using the technique described in [2]. This live value memory is composed of 2-(full)port memories. Each port shares a RAM with another port. This results in $N(N - 1)/2$ RAMs being needed. See figure (todo cite figure).

The memory gets its name because of a multiport 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 efficiently this way.

Instead of using a register based live-value table as in [2] we use a xor memory similar to [1].

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

5 Implementation

We explore designs with 4 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. N^2 for XOR and $N(N-1)/2$ for LVT. However we were able to synthesize a 32 port memory.

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

6 Conclusion

Several solutions to the port limit on FPGAs other than what is presented here. For example multi-pumping and banking. 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 2 150Mhz ports. Banking requires stalls and routing logic

⁰To 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.

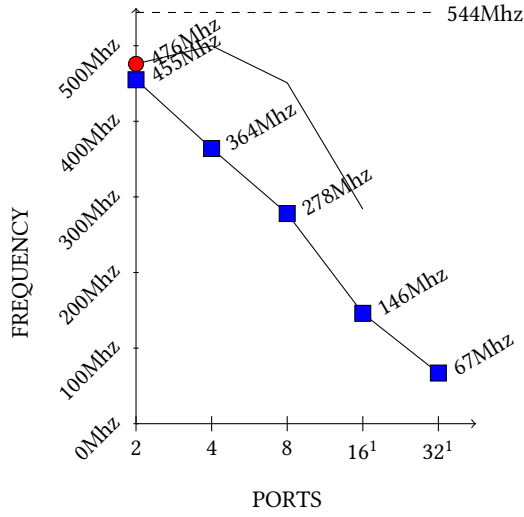


Figure 4: Frequency of design.

Table 2: Synthesis results for different port counts for pipelined design

Ports	LUTS	FF	BRAM	Max Frequency
2	111	26	1	476Mhz
4	708	64	6	500Mhz
8	3,092	152	28	451Mhz
16 ¹	16,800	448	120	284Mhz
32 ¹	86,608	1,502	496	XMhz

due to the segmented memory. Our design uses replication and some creativity XOR and LVT to create multiple ports.

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] 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
- [3] 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.
- [4] 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
- [5] 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
- [6] 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.
- [7] Xilinx. 2020. *7 Series FPGAs Data Sheet: Overview*. Xilinx. <https://docs.amd.com/api/khuf/documents/2LByHkO-nSZXcei2D55fTg/content> v2.6.1.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009