This article has been accepted for publication in IEEE Embedded Systems Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LES.2023.3299202

NvMISC: Towards an FPGA-based Emulation Platform for RISC-V and Non-volatile Memories

Yuankang Zhao, Salim Ullah, Siva Satyendra Sahoo, and Akash Kumar, *Senior Member, IEEE*

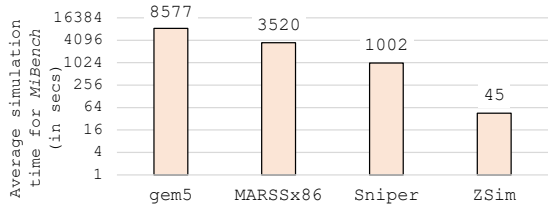
***Abstract*—The emerging Non-volatile Memories (NVMs), such as Spin Transfer Torque Random Access Memory (STT-RAM) and Racetrack Memory (RTM), offer a promising solution to satisfy the memory and performance requirements of modern applications. Compared to the commonly utilized volatile Static Random-access Memories (SRAMs), the NVMs provide better capacity and energy efficiency. However, many of these NVMs are still in the development phases and require proper evaluation in order to evaluate the impact of their use at the system level. Therefore, there is a need to design functional- and cycle-accurate simulators/emulators to evaluate the performance of these memory technologies. To this end, this work focuses on implementing a RISC-V-based emulation platform for evaluating NVMs. The proposed framework provides interfaces to integrate various types of NVMs, with RTMs and STT-RAMs used as test cases. The efficacy of the framework is evaluated by executing benchmark applications.**

***Index Terms*— RISC-V, Non-volatile Memories (NVMs), RTMs, STT-RAMs, Emulation Platform, FPGAs**

I. Introduction

During the past few decades of technology scaling, the memory systems could not achieve the performance pace of Central Processing Units (CPUs). The memory hierarchy's high access latency, resource requirements, and energy consumption have degraded the overall performance of computing systems [1]. To this end, emerging Non-volatile Memories (NVMs), such as Spin Transfer Torque Random Access Memory (STT-RAM) and Racetrack Memory (RTM), offer a promising solution to overcome the bottlenecks (capacity, latency, and energy) of the commonly utilized Dynamic Random-access Memory (DRAM)- and Static Random-access Memory (SRAM)-based solutions [2]–[5]. The NVMs offer better density and energy efficiency than the SRAM- and DRAM-based technologies [6]. For instance, compared to SRAM, RTM technology has a much smaller cell size ($\leq$ 2 F^2) and lower leakage power. SRAM technology, on the other hand, has a larger cell size (120 $-$ 200 F^2) and higher leakage power.

The NVMs, however, have some limitations, such as variable access latency and costly write operations, which result in limited memory performance improvement. These challenges of NVMs open a research space for exploring various hardware and software architectures to overcome these challenges of NVMs and enable hybrid technologies-based memory hierarchies to improve memory systems' performance. Considering the relatively contemporary and exploratory phase of NVMs, limited simulation and emulation frameworks for analyzing the system-level performance impact of NVMs are available in the literature. For example, the authors of [7] and [8] provide open-source frameworks that can be integrated with system-level simulators to explore the performance of NVMs. However, as shown in Figure 1, the commonly utilized cycle-accurate system-level simulators, such as *gem5* and *Sniper*, have a large execution time to evaluate the impact of a cache configuration on the overall system performance [9], [10].



Emulator	Focus	Latency	NVM	Variable Capacity
[11]	Main Memory	Crude	PCM	No
[12]	Main Memory	Crude	PCM	No
ERMES [13]	Scratchpad	Accurate	RTM	Yes
NvMISC	L1 Cache	Accurate	SRAM/RTM/ STT-RAM	Yes

Figure 1: Related simulation and emulation frameworks

Some works have also proposed various field-programmable gate array (FPGA)-based emulation platforms to provide fast emulation of NVMs-based memories. The table in Figure 1 compares some of the recently proposed emulation platforms for NVMs. However, these platforms do not support the emulation of different types of NVMs. The platforms presented in [11] and [12] assume uniform access latencies for NVMs and therefore do not provide accurate execution and latency information for various operations. A more recent work has proposed an FPGA-based implementation of an RTM-based scratchpad memory [13]. To execute an application utilizing the RTM IP, the authors assign a subset of the memory addresses to the IP, and the corresponding memory requests are completed using the *AXI-Lite* protocol. However, as presented in the article, ERMES features a very large resource usage while only being able to emulate a very small RTM.

To address these limitations of available emulation and simulation frameworks, we present the cycle-accurate *NvMISC* framework. Figure 2 shows an overview of the proposed framework. We make the following novel

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) under the X-ReAp project (Project number 380524764).

Y. Zhao, S. Ullah and A. Kumar are with the Chair of Processor Design, CfAED, Technische Universität Dresden, Germany E-mail: yuzh500b@msx.tu-dresden.de, salim.ullah@tu-dresden.de, akash.kumar@tu-dresden.de

S. S. Sahoo is with IMEC, Belgium. E-mail: siva.satyendra.sahoo@imec.be

This article has been accepted for publication in IEEE Embedded Systems Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LES.2023.3299202

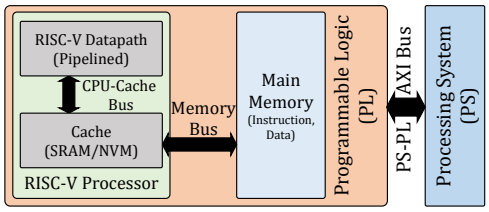


Figure 2: Proposed *NvMISC* framework

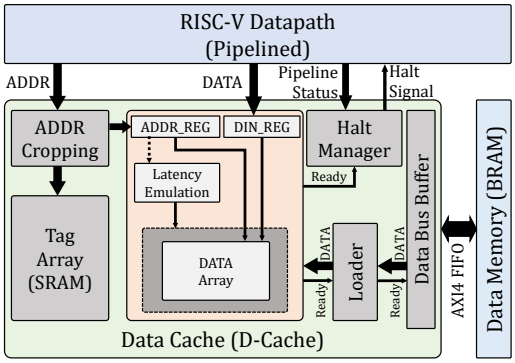


Figure 3: Cache Emulation Architecture

contributions in this work: wide

1) *NvMISC* Framework: We provide an FPGA-based cycle-accurate RISC-V implementation that provides interfaces for emulating NVMs-based L1 data caches (D-Caches).
2) NVMs RTL Models: We provide RTL-based D-Cache models for two of the most widely explored NVM technologies, i.e., RTM and STT-RAM.
3) To help the research community overcome NVMs-related exploratory challenges, the proposed framework is open-source at https://cfaed.tu-dresden.de/pd-downloads.

II. Proposed *NvMISC* Design

A. RISC-V Core Generator

In *NvMISC* framework, RISC-V core generator implements the L1 cache data array based on the designer-provided configurations. For the RTM-based data cache, we have explored the number of shifts per clock cycle (*shiftPerCycle*), the number of access ports (*accessPorts*), and the architectural layout of the track, i.e., *horizontal- or ring-shaped*. For the STT-RAM-based cache, we support configurations for specifying the read and write latencies (*readLatency, writeLatency*). Similarly, for the traditional SRAM-based cache, a designer can define the memory access latency (*accessLatency*).

1) Cache Data Array: Figure 3 presents an overview of the proposed architecture for emulating various technologies-based caches. In this architecture, the address register (ADDR_REG) stores the access address during each cache access. In case of a cache write operation, the corresponding data is stored in the data register (DIN_REG). The various sub-modules in the cache emulation architecture, i.e., *Latency Emulation*, *Cache Loader*, and the *Halt Manger*, are instantiated differently for different architectural configurations. The *Latency Emulation* block is responsible for introducing extra delays in every cache access to implement the designer-provided configurations. For example, for the STT-RAM emulation, a counter value is compared with the *readLatency* (or *writeLatency*) in every cache access to match the provided configurations. The corresponding data array can be accessed (for reading or writing) only after the counter comparison returns true. Similarly, for RTM emulation, the proposed architecture computes the difference between the current and the last access addresses. The address difference value is then utilized to compute the number of required shifts according to the RTM configuration values, i.e., *shiftPerCycle*, *accessPorts*, and the architectural layout of the track. The RTM-based cache model then utilizes a counter to generate a delay according to the computed number of shifts value and thus models the shift latency between two consecutive cache accesses. In our proposed implementation, we assume that each cache line forms a bundle of RTM tracks and will shift together during each access.

2) Pipeline Management: We propose a *Halting Manager* to stall the pipeline during cache accesses to handle the variable (additional) delay of NVMs-based models. The *Halting Manager* stalls the *Execution* and *Writeback* stages of the pipeline if a read or write request is issued to the cache and resumes the operation upon receiving a ready signal from the data array. Similarly, it is responsible for stalling the *Execution* and *Writeback* stages of the pipeline during a cache refill process and resumes the operation after an entire cache line has been refilled.

3) Data Bus Buffer: The data bus between the D-Cache and the main memory implements the AXI4 protocol and reads a burst of data from the main memory during a cache miss. Due to the variable and multi-cycle latency of the data array (NVMs), a buffer is used to reconcile the faster data flow incoming from the data bus and the slower operation of the cache *Cache Loader*, described in the next section. The buffer begins storing the content after detecting an incoming burst from the data bus. After asserting that the entire burst of data has been stored in the buffer, the buffer notifies the loader to start the cache line refilling process.

4) Cache Loader: The *Cache Loader* is responsible for loading the values from the data bus buffer into the cache data array during a cache miss. While writing a cache line, the *Cache Loader* ensures to follow the additional write latencies of STT-RAMs and RTMs as defined by the designer-provided configurations. After an entire cache line is refilled, it notifies the *Halting Manager* to resume the pipeline.

B. Peripheral Components

The RISC-V processor's instruction and data buses are routed through an X-BAR and connected to the instruction and data memories. To better emulate the longer latency imposed by main memories (DRAMs), we have included buffers between the memory busses and the X-BAR. The data and instruction memories (DRAM) are also accessible to the

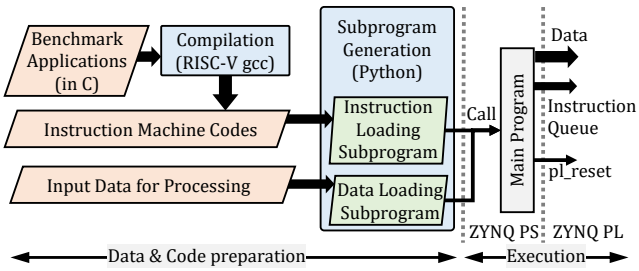


Figure 4: Software setup of *NvMISC*

Table I: Emulation System Specifications

Component	Description
ARM Processor	Arm Cortex-A53@1.5GHz
RISC-V Processor	VexRiscv Softcore
PL frequency	187.5MHz
L1 I-Cache	4kiB, directly-mapped
L1 D-Cache	up to 128kiB, directly-mapped
Instruction Memory	64kiB (BRAMs)
Data Memory	512kiB (BRAMs)

ZYNQ processing system (PS) to allow the control of these memories from the ZYNQ PS.

C. Software Setup

Figure 4 shows the different processes in the software setup of *NvMISC*. Benchmark applications are compiled into RISC-V instructions, with the last instruction writing a debug value to a specific memory location (*DEBUG_LOC*) in DRAM data memory. After loading the benchmark application's binaries and data in the instruction and data memory, the main program, executed on the PS, performs a programmable logic (PL) reset function to allow the RISC-V core to jump to the start of the instruction queue. Along with the reset function, the PS also executes a local timer and continuously polls the *DEBUG_LOC* to identify the completion of the RISC-V execution and compute the total execution time of the benchmark application.

III. Experiments and Results

A. Experiment Setup

We have utilized the VexRISC, an open-source RISC-V core generator written in SpinalHDL [14], for implementing our proposed *NvMISC* framework. For this work, we have used Xilinx Vitis 2022.1 design suite to implement *NvMISC* on an Ultra96v2 single-board computer equipped with the ZYNQ ZU3EGA484 MPSoC. In this implementation, we have used a default clock frequency of 187.5MHz. All benchmark applications are written in C and initially executed using test datasets on a standard x86-based general-purpose computer. These results are used to assess the functional correctness of *NvMISC*. Afterward, the benchmark applications are cross-compiled using the RISC-V GCC compiler, and the RISC-V-based application execution results are obtained using the UART connection to compare with the *golden answers*. Table I and Table II provide the detailed specifications of the emulated system.

Table II: Cache Test Configurations for Emulation

D-Cache Type	Description
SRAM	Dual-cycle latency SRAM
STT-RAM	Mid-retention STT-RAM [15]
RTM_1	16-bit long RTM, single shift per cycle
RTM_2	16-bit long ring shaped RTM, single shift per cycle with 4 access ports
RTM_3	16-bit long RTM, 4 shifts per cycle

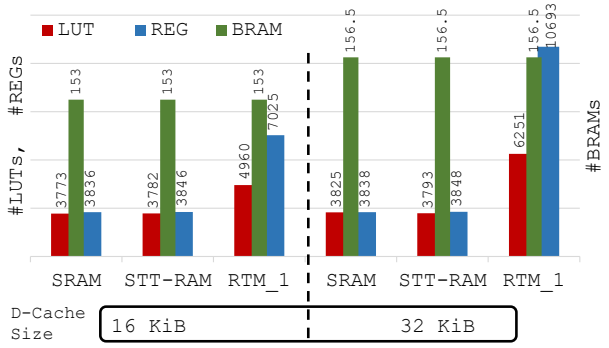


Figure 5: Comparison of Resource Utilization

B. Resource Utilization

Figure 5 compares the resource utilization of SRAM-, STT-RAM-, and RTM-based D-Cache models for two different cache sizes. Compared to the SRAM-based model, the NVMs-based models utilize more Look-up Tables (LUTs) and Registers. For the STT-RAM-based implementation, more logic is needed to compare the counter value against the configured reading and writing latencies. The extra logic results in an increase of up to 0.2% in the total utilized LUTs and registers. Similarly, for RTM-based cache implementations, additional registers are required to hold the last access address information, and extra logic is instantiated to compute the required access delay. Compared to the SRAM-based implementation, the RTM-based model results in an increase of up to 83% and 31% in utilized registers and LUTs, respectively.

The maximum possible delay (number of shifts) plays a more significant role in determining the resource utilization of an RTM configuration. For example, Equation 1 and Equation 2 show the estimation of the maximum number of shifts (L_{max}) and the bit width (N_{width}) of the register to hold the address difference in RTM emulation. In Equation 1, N_{length} and N_{ports} identify the length of the track and the number of access ports per track, respectively. The k_{ring} is the ring coefficient, and for a ring-shaped track, its value is 2, and for horizontal tracks, the value of k_{ring} is one. In general, the reduced number of shifts results in a notable reduction in register and LUT usage. Table III compares the resource utilization of *NvMISC* with ERMES [13], which is an emulator for exploring RTMs only. Compared to ERMES, *NvMISC* features a much simpler design with larger memory capacity and significantly reduced resource usage. The results in Table III show that *NvMISC* provides better host platform adaptability and improved performance.

Table III: Comparing *NvMISC* with State-of-the-art Emulator

Platform	ERMES [13]		*NvMISC*	
Resource	LUT	REG	LUT	REG
32kiB	216k	261k	6.3k	10.7k
64kiB	X		8.9k	18.5k
128kiB			14.5k	35.1k

Table IV: Benchmark Applications

Application	Description
Matmul	Matrix multiplication between a 170×166 array of integers and a 3×3 kernel
Bubblesort	Bubble sorting algorithm on an array of 28K integers
Bitcount	Count number of bits in an array of 28K integers using the Brian Kernighan algorithm

Table V: Comparing Xsim (behavioral) and *NvMISC* timings

Execution Type	Simulation(Xsim)		*NvMISC*	
D-Cache Type	Matmul	Bitcount	Matmul	Bitcount
SRAM	2460s	880s	11.80s	9.10s
STT-RAM	3030s	893s	13.07s	8.10s
RTM	2305s	840s	9.47s	10.89s

$$L_{max} = \frac{N_{length}}{N_{ports} k_{ring}} \quad (1)$$

$$N_{width} = \log_2(L_{max}) \quad (2)$$

C. Benchmark Evaluation

Table IV summarizes the details of the evaluated benchmark applications. Figure 6 shows the execution time results of these applications for various cache configurations. It can be observed that caches emulating STT-RAM cells featuring medium data retention perform notably worse with up to 95% increased processing time than other caches. The increase in the overall execution time of STT-RAM-based cache is due to their long writing latency. Our results also show that by allowing more access ports and implementing ring-shaped RTM tracks, the performance of very large RTM-based caches can be remarkably increased. Furthermore, in such a configuration, the performance of RTM-based cache can reach a point where it noticeably exceeds the performance provided by dual cycle latency SRAMs with an up to 26% reduced execution time. Here, we also demonstrated the ability to emulate and benchmark RTMs operating at higher frequencies, i.e., multiple shifts per cycle RTMs. It can be observed that RTMs having such an architecture offers reduced shifting latency resulting in an up to 16% reduced execution time.

D. Platform Performance

To evaluate the performance of our proposed *NvMISC* framework, we compare the execution time (wall clock time) of executing the Matmul and Bitcount benchmarks on the FPGA-based *NvMISC* and Xilinx XSim, an HDL-based cycle-level simulator on an x86 server with 1TB RAM and 6 AMD EPYC 7513 processor enabled. For our current experiment, we have considered dual cycle SRAMs, STT-RAMs, and RTMs featuring a single shift per

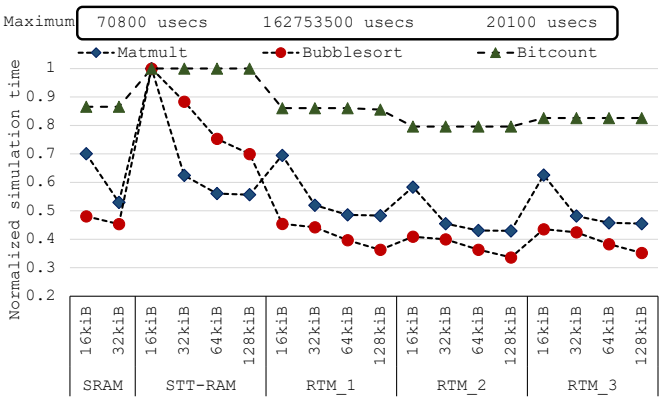


Figure 6: Benchmark Evaluation

cycle in 16kiB D-Cache on Xilinx XSim. The comparison results are shown in Table V. Our proposed emulation platform provides an up to 243× speedup thanks to its purposefully built FPGA setup. Furthermore, the emulation time also includes the time consumed by other processes, such as loading the bitstream and configuring the FPGA. The programmability and fast execution provided by *NvMISC* can be utilized to run more complex benchmark applications for exploring NVMs-based caches.

IV. Conclusion

This work provides a cycle-accurate framework for evaluating the impact of NVMs-based caches on the overall performance of a system. In the future, we aim to extend the framework with hybrid technology-based multi-level caches and various cache placement policies such as set associativity. We also plan to compare the performance results of our framework with other simulation frameworks, such as the RTSim [8], for various benchmark applications.

References

[1] W. A. Wulf et al. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 1995.
[2] T. M. O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX security*, 2007.
[3] C. Escuín Blasco, et al. Stt-ram memory hierarchy designs aimed to performance, reliability and energy consumption. In *ACACES*, 2019.
[4] B. C. Lee, et al. Architecting phase change memory as a scalable dram alternative. In *ISCA*, 2009.
[5] S. Parkin et al. Memory on the racetrack. *Nature nanotechnology*, 2015.
[6] R. Bläsing, et al. Magnetic racetrack memory: From physics to the cusp of applications within a decade. *Proceedings of the IEEE*, 2020.
[7] M. Poremba, et al. Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems. *IEEE CAL*, 2015.
[8] A. A. Khan, et al. Rtsim: A cycle-accurate simulator for racetrack memories. *IEEE CAL*, 2019.
[9] N. Binkert, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 2011.
[10] T. E. Carlson, et al. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC*, 2011.
[11] F. Wen, et al. An FPGA-based Hybrid Memory Emulation System. In *IEEE FPL*, 2021.
[12] T. Lee, et al. FPGA-based prototyping systems for emerging memory technologies. In *IEEE ISRSP*, 2014.
[13] F. Spagnolo, et al. Ermes: Efficient racetrack memory emulation system based on fpga. In *FPL*, Sep 2022.
[14] SpinlHDL. VexRISC-V. https://github.com/SpinalHDL/VexRiscv, 2020.
[15] Z. Sun, et al. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *MICRO*, 2011.