Many modern applications require frequent data movement between the CPU and the main memory, which consumes a significant amount of energy. To mitigate the cost of data movement, recent works propose a new computing paradigm called processing-using-memory (PUM), which leverages electrical interactions between memory cells to realize useful computation within the memory arrays, without the need for additional CMOS logic. Compared to adding discrete logic near memory arrays, PUM eliminates significantly more data movement and can enable thousands to millions of ways of data-level parallelism [3, 10]. A PUM-capable memory cell can be used as either (1) a multi-bit device capable of analog operations such as multiplication (e.g., [1,2]), or (2) a single-bit device capable of digital operations such as Boolean algebra (e.g., [7, 9, 13]).

While PUM can be implemented using both conventional (e.g., DRAM, SRAM) and emerging (e.g., MRAM, PCM, ReRAM) memories, prior works build PUM architectures around a specific memory technology. Unfortunately, this creates a significant barrier to the development of PUM-based systems. Due to DRAM scaling issues [8, 11], manufacturers are hesitant to alter the design of DRAM arrays in fear of hurting yield, and while SRAM-based PUM avoids such concerns, it significantly compromises storage density compared to other memory technologies. Emerging memory technologies offer the promise of high-density PUM, but issues remain with the at-scale production of each of them. Given the uncertainty of which technology will emerge as the dominant replacement for DRAM, it becomes unclear which of the previously-proposed PUM architectures will be viable, which in turn prevents systems developers from solving software-level challenges holding back the adoption of PUM [3].

In our JETCAS 2022 paper, we aim to decouple PUM architectures as much as possible from (1) a specific memory technology, as well as (2) a specific logic family (e.g., the MAGIC logic family [6] provides driving voltages and circuit-level support for either NOR or NAND, while the FELIX logic family [4] provides this for NOR, NAND, and NOT in the same array). For this work, we build upon our prior work on RACER [13], a highly-scalable PUM architecture that efficiently enables MAGIC-NOR-based processing inside small memory arrays. We start by developing interface circuits that allow the RACER architecture to be compatible with any PUM logic family. These circuits ensure that most of RACER’s circuitry can stay the same as we adapt the architecture to new logic families and to other resistive memory technologies (including both 1S1R-based crossbars and 1T1R-based arrays).

Building upon our technology-agnostic version of RACER, we seek to design a new logic family that is practical for the types of memory devices that we can manufacture today. Unfortunately, the MAGIC and FELIX proposals require constraints for the device switching thresholds that are difficult to achieve with today’s resistive memory technology prototypes (as was also recognized by prior work [5,14]). We propose a new logic family, OSCAR, that enables NOR and OR using resistive memories with significantly-relaxed constraints, which are widely compatible with existing device technologies.

**Background: RACER Architecture.** RACER [13] uses bit pipelining to achieve high performance with small $n \times n$ ReRAM tiles (e.g., $n = 64$). A $w$-bit word is stored across $w$ tiles as shown in Figure 1. RACER’s bit-serial operations iteratively apply Boolean logic operations one tile at a time. To pass intermediate values (e.g., carry-out bits) between tiles, RACER uses a resistive-memory-based buffer that connects to a tile using programmable pass gates. This allows tiles to act as per-bit pipeline stages, while buffers act as pipeline registers between stages, enabling the concurrent computation of $w \times n$ bit-serial operations. Using MAGIC-based NOR operations in ReRAM arrays, RACER achieves a $107 \times$ speedup and $189 \times$ energy savings compared to a modern 16-core Xeon CPU for microkernels from a range of important domains (image processing, linear algebra, signal processing, neural-network-based classification, string matching).

**Technology Interface Circuits.** Figure 2a illustrates the necessary circuitry to enable bit-pipelining. The pipeline controller consists of one micro-op queue per tile. Each queue holds a sequence of micro-ops, which are commands that tell RACER which Boolean primitive to apply to a set of columns in the tile. RACER originally directly connected the pipeline controller to the tiles, and translated each micro-op to predetermined voltage signals according to the ReRAM-based MAGIC [6] logic family. Thus, RACER would need a significant redesign to integrate with a different logic family/technology. We address this inflexibility by decoupling the control circuitry from the tiles using modified decode & drive units. These units act as interface circuits that allow RACER to work with other logic families, by isolating the technology-agnostic aspects of the controller away from the technology-specific switching voltages.

We design the decode & drive units by observing that for all existing resistive PUM logic families (e.g., [4, 6, 12]), one of three voltages is asserted to each column of a tile depending on its "role": (1) one or two columns that serve as inputs of the Boolean operation are asserted with $V_{\text{in}}$; (2) one column that serves as the output is asserted with $V_{\text{out}}$; and (3) all other (i.e., idle) columns are asserted with $V_{\text{float}}$. The values of these assertion voltages depend on the specific logic family and on the Boolean operation (e.g., NOR, NAND). Once a decode & drive unit (Figure 2b) receives a micro-op (1) in the

---


*We use the term resistive memory to refer broadly to resistance-based non-volatile memories (e.g., PCM, MRAM, ReRAM), while we use ReRAM to refer specifically to oxide-based switches (often referred to as memristors).
existing logic families, OSCAR NOR does not require any constraint between \( V_{set} \) and \( V_{reset} \), while OSCAR OR requires \( V_{set} < 2V_{reset} \). As shown in Figure 3a, OSCAR’s constraints are compatible with ratios achievable by typical devices.

With OSCAR, we improve upon RACER’s performance and energy savings by 30% and 37%, respectively (Figure 4), compared to MAGIC NOR, while enabling broad compatibility with current devices. OSCAR-based RACER achieves a 142× speedup and 233× energy savings over a 16-core Xeon CPU. Notably, RACER now outperforms CASCADE [2], a state-of-the-art analog PUM dot product accelerator, for several matrix-based microkernels by an average of 3.16×. We conclude that by adapting RACER to other logic families, we can significantly improve the efficiency of Boolean PUM.

**Significance.** While we focus on improving the compatibility of RACER across a wide range of logic families and memory technologies, both our decode & drive circuit design and OSCAR can be adapted to other digital PUM architectures. This can allow others to design architectures that abstract away technology-specific details from the architecture and from software developers. We believe that this is a crucial step to enabling the development of the currently-missing software stack for non-dot-product PUM, which today is a significant barrier to widespread commercialization. Without this abstraction, there is little incentive to develop a toolchain for an architecture that may lose its relevance in a few years as device technologies and commercial processes evolve.

**References**


