CAMeleon: Reconfigurable B(T)CAM in Computational RAM

Zamshed I. Chowdhury, Salonik Resch, Hüsrev Cılasun, Zhengyang Zhao, Masoud Zabihi,
Sachin S. Sapatnekar, Jian-Ping Wang and Ulya R. Karpuzcu
{chowh005,resc0059,cilas001,zhaox526,zabih003,sachin,jpwang,ukarpuzc}@umn.edu
University of Minnesota
Minneapolis, Minnesota, USA

ABSTRACT

Embedded/edge computing comes with a very stringent hardware resource (area) budget and a need for extreme energy efficiency. This motivates repurposing, i.e., reconfiguring hardware resources on demand, where the overhead of reconfiguration itself is subject to the very same tight budgets in area and energy efficiency. Numerous applications running on resource constrained environments such as wearable devices and Internet-of-Things incorporate CAM (Content Addressable Memory) as a key computational building block. In this paper we present CAMeleon – a novel energy-efficient compute substrate which can seamlessly be reconfigured to perform CAM operations in addition to logic and memory functions. CAMeleon has a similar level of latency to conventional CAM designs based on SRAM and emerging memory technologies (such as STT-MTJ, ReRAM and PCM), however, performs CAM operations more energy-efficiently, consumes less area, and can support traditional logic and memory functions beyond CAM operations on demand thanks to its reconfigurability.

CCS CONCEPTS

• **Hardware → Emerging architectures**; **Spintronics and magnetic technologies**; *Memory and dense storage.*

KEYWORDS

CAM, CRAM, emerging, non-volatile, PIM, reconfigurable

ACM Reference Format:
Zamshed I. Chowdhury, Salonik Resch, Hüsrev Cılasun, Zhengyang Zhao, Masoud Zabihi,, Sachin S. Sapatnekar, Jian-Ping Wang and Ulya R. Karpuzcu. 2021. CAMeleon: Reconfigurable B(T)CAM in Computational RAM. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22–25, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3453688.3461507

1 INTRODUCTION

Content addressable memory (CAM) is an extensively used functional building block in many mainstream computing systems. Rather than locating stored data using addresses (as in conventional random access memory, RAM), CAM finds information using the content itself – hence the name. Specifically, upon getting a search request for a data content, CAM performs the search on all memory locations simultaneously, and returns the location (or index) of match, if any.

On a per bit basis, Binary CAM (BCAM) can search for only two states, i.e., 0 and 1, which Ternary CAM (TCAM) expands to include also a third *don't care* state, i.e., *X*. This wildcard X makes searching for a data content, that partially matches with stored data, possible, and therefore, can generate multiple matches for a single content search request. In either case, the parallel search capability enables CAM structures to perform low latency data lookup which is desired in many contexts, including but not limited to network devices [1, 11], neuromorphic associative memory [26], big-data analytics [12, 22], pattern recognition [13, 31], data compression [24], reconfigurable computing [25] and application-specific acceleration [15].

Emergence of edge computing has further increased the range of applications where parallel content based search is critical to overall system performance. Examples include object detection [6], neuromemristive circuits and near-sensor binary deep neural networks for edge computing devices [17, 18]. Besides fast CAM search, operation in resource constrained environments (such as wearable devices and Internet-of-Things, IoT) require very low area and energy consumption. At the same time, constrained hardware resources make reconfigurability an increasingly desired feature in these environments [16], to best match dynamically changing computational demand of the workload, specifically, to deliver the optimal performance without any waste in area and/or energy by repurposing hardware resources on demand. However, reconfiguration itself incurs an overhead which can easily become prohibitive considering the extremely tight budgets in area and energy.

Be it based on traditional CMOS or emerging technologies (STT-MTJ [10], ReRAM [13, 33] or PCM [27]), typical CAM designs suffer from either high area overhead or energy consumption (or both). Moreover, none is practically reconfigurable, hence re-purposing CAM cells on demand during runtime to perform regular memory or even logic operations is out of question. On the other hand, PIM substrates –already by construction– can perform logic and regular memory operations within the same array with minimal reconfiguration overhead. By exploiting array regularity, adding CAM operations on top would be an attractive solution as long as the reconfiguration overhead can be kept at bay[1]. While various recent PIM proposals target edge-computing systems [2, 29], none explores this opportunity.

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8393-6/21/06…$15.00
https://doi.org/10.1145/3453688.3461507

[1]Designs that use CAM to perform very restricted and limited number of logic operations are not considered as PIM enabled CAM architectures in this context.

In this paper, we propose a novel reconfigurable architecture, CAMeleon, targeting edge and embedded environments, which seamlessly adds CAM functionality to nonvolatile PIM. CAMeleon supports both BCAM and TCAM operations utilizing *in-place* logic processing capabilities of PIM. The underlying PIM substrate is the non-volatile (spintronic) Computational RAM (CRAM) [4], which is shown to be a versatile and highly area/energy-efficient platform for resource constrained environments [29], where CAMeleon does not incur any changes to the cell architecture. CAMeleon's energy-efficiency comes from –unlike existing CAM proposals regardless of the underlying memory technology– not requiring specially tuned dedicated sense amplifiers to perform CAM operations. Moreover, CAMeleon can switch between PIM and (B/T)CAM operations with minimal reconfiguration overhead. In a nutshell, the contributions of this paper are as follows:

(1) To the best of our knowledge, CAMeleon is a unique reconfigurable architecture fusing generic BCAM/TCAM functionality with PIM (i.e., conventional memory and logic operations).
(2) We cover the HW/SW co-design in detail, that enables integration of CAM and PIM functionality with trivial reconfiguration overheads.
(3) We show that CAMeleon has comparable search latency to fastest known CAM designs, while providing a higher area/energy-efficiency.

The rest of this paper is organized as follows: Section 2 presents CAM and CRAM basics, Section 3 details HW/SW co-design of CAMeleon, Sections 4 and 5 provide the evaluation, and Section 6 concludes the paper.

2 BASICS

2.1 CAM Architecture

Fig. 1 illustrates basic hardware organization for CAM. CAM table comprises a 2-D array of CAM cells, which store key words (typically 32–128 bits) along each row and which are connected to *match lines*. Each CAM cell typically stores both key and inverted key bits. The query to search, stored in the query register and connected to the CAM table through search lines, connects a bit from each CAM cell to the corresponding match lines. The match line indicates whether the data stored in that row is a match for the query input. All match lines are fed to an encoder that determines the match location, i.e., index. TCAM typically requires one additional memory cell for each CAM cell that stores a wildcard bit, i.e., X (= 0/1). In case of BCAM, one match for each query is expected –unlike TCAM where more than one match is possible and therefore, a priority encoder is employed that outputs the match location with highest priority.

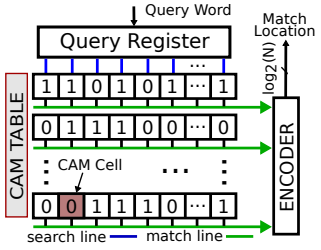

Figure 1: Generic CAM architecture.

2.2 Computational RAM (CRAM) Basics

CRAM cells are organized in a 2-D layout (*tile*) where each cell consists of a non-volatile memory device and an access transistor. Fig. 2(a) illustrates a CRAM cell. CRAM cells in CAMeleon, without loss of generality, use spintronic memory devices, STT-MTJ in particular, which features superior endurance and low energy[2]. Each STT-MTJ contains two adjacent layers of ferromagnets –fixed and free layers, insulated by a thin tunneling layer. The fixed layer has its spin orientation fixed; the free layer, controllable. A current through the MTJ that is greater than a threshold (critical current, I_{crit}) can change spin orientation, where the direction of the current sets the polarity of the orientation. The relative spin orientation of the free layer to the fixed layer results in high (R_{high}) or low (R_{low}) resistance, interpreted as logic 1 or logic 0, respectively. One terminal of each MTJ is connected to a bit select line (BSL) through an access transistor that is controlled by the word line (WL). The other MTJ terminal is tied to a wire, called logic line (LL). Each row of the tile has a separate WL, whereas all cells in each column are connected to a {LL, BSL} pair. Along each column of the tile, there are two groups of BSL, namely Even BSL (EBSL) and Odd BSL (OBSL). When not in logic mode, EBSL and OBSL are virtually indistinguishable in terms of their functionality. All signals are driven by a tile controller.

Memory operations: For read/write operation, WL is set to logic *HIGH* (as shown in red in Fig. 2(a)). A voltage for read, applied between LL and BSL, results in a current through MTJ (which evolves as a function of the MTJ resistance) that is sensed to determine the data content. For writes, the voltage applied between LL and BSL is set to the write voltage which is enough to create a current $> I_{crit}$.

Logic operations: Fig. 2(b) illustrates a logic gate, formed by three CRAM cells along a column (transposed for illustration). The input cells, *Input1* and *Input2*, store the input bits (as encoded by their respective MTJ resistances), and the output cell, *Output*, is preset to logic 0 or 1 depending on the logic operation. All *WL*s to the input and output cells are set to logic *HIGH*. Thereby all columns perform the same logic operation in parallel since all *WL*s run along rows. The inputs are connected to different group of BSLs (either OBSL or EBSL) than the output. The *HIGH* on *WL* connects the cells to the corresponding BSL. In logic mode, LL acts as a connecting wire. Now, a voltage (V_{gate}) is applied between EBSL and OBSL, which generates currents through input MTJs (I_1 and I_2) that sink through the output MTJ. If this combined current (depending on the stored bits, i.e., resistance states of *Input1* and *Input2*) is greater than I_{crit}, then the resistance state of the output MTJ switches, otherwise, the preset resistance state remains – effectively implementing the corresponding truth table. The use of a preset and gate specific V_{gate} during logic operation makes CRAM Boolean complete.

Putting it all together, the equivalent circuit in Fig. 2(c) shows how the gate specific voltage V_{gate}, applied between EBSL and OBSL, conducts a current through input cells (R_1 and R_2). The combined current, $I_{OUT} = I_1 + I_2$, flows through the output cell (R_{OUT}). For simplicity the resistance of access transistors are not shown. The orientation of the free layer in the output cell is switched (in a direction specified by the direction of I_{OUT}) if $I_{OUT} > I_{crit}$, causing the R_{OUT} to change accordingly.

[2]CRAM cells can support various memory devices [5, 28].

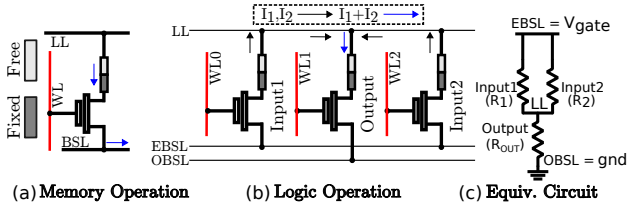

Figure 2: CRAM cell architecture and operation.

Logic gates to support CAM operations: The only logic gates required to perform CAM operations are *NOR* and *AND*. The truth table of a 2-input NOR gate is shown in Table 1. The output cell (*Output*) is preset to 0. The NOR gate specific voltage, V_{NOR} is determined such that I_{crit} is exceeded by I_{00} only. Therefore, *Output* switches from 0 to 1 only if both inputs are 0, and retains its preset value for other input combinations. The same biasing conditions implement an *AND* gate with a preset of 1 (and opposite current direction), where *Output* switches from 1 to 0 for all input combinations but 11.

$Input_1$	$Input_2$	$Output$	$I_{OUT} = I_1 + I_2$	
0 (R_{low})	0 (R_{low})	1	I_{00}	$> I_{crit}$
0 (R_{low})	1 (R_{high})	0	I_{01}	$< I_{crit}$
1 (R_{high})	0 (R_{low})	0	$I_{10} = I_{01}$	$< I_{crit}$
1 (R_{high})	1 (R_{high})	0	I_{11}	$< I_{crit}$

Table 1: 2-input NOR truth table (*Output* preset = 0).

3 CAMELEON ARCHITECTURE

3.1 Overview

Fig. 3 shows a generic CAM algorithm. A typical CAM handles equal-length key and query words: *Key* words are stored in unique locations inside CAM structure, and *Query* words (from a query pool, one at a time) are used to search those unique memory locations simultaneously, to find the exact (partial) match(es) between the key words and the query in case of BCAM (TCAM). For every bit of a key word, both that bit and the corresponding inverted bit are stored in CAM to form a *bit-pair*. Query words are written to a structure inside CAM array, i.e., the *query register*, one at a time, before firing search. The process repeats for each query word in the pool.

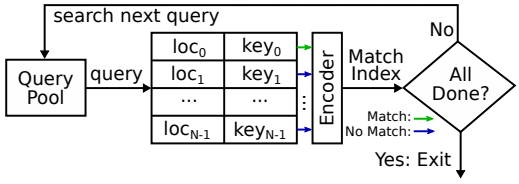

Figure 3: Generic CAM algorithm.

3.2 CAM operations in CRAM

Basics: For each bit in a key word, there is a *bit-pair* stored in the corresponding column of the CRAM tile. Specifically, all bit-pairs of a key word go to the same column. Each bit-pair in a key word thereby forms a CAM cell along the CRAM column; multiple CAM cells residing in a CRAM column. As typically one set of key words is searched for many query words, the initial overhead of writing key words to CRAM columns is amortized over many CAM search operations. The *query register* stores a query word that generates necessary signals to perform CAM operations in CRAM. After search is complete, search outcomes are directly stored in CRAM which are subsequently read out and fed to (priority) encoders. The mechanism for CAM search in CRAM is simple: only if a bit in the query and key words match, a CRAM cell storing logic 0 at that bit position (either key or inverted key bit) is connected to the LL along the column. Therefore, by construction, a subsequent *NOR* of all such cells along the column (which keep the search outcomes for the entire key, query pair) generates a 1 to indicate a *match*, iff all bits match.

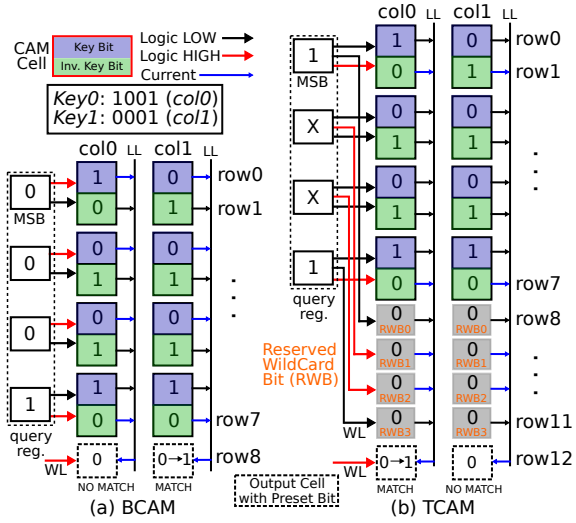

Figure 4: (B/T)CAM search in CAMeleon.

BCAM operation: Fig. 4(a) illustrates the principle of BCAM search in CRAM with an example 9 × 2 tile (where red lines indicate logic HIGH). The two columns (*col0* and *col1*) store *Key0* and *Key1* respectively. The columns are 9-bit long where *row 0 - row 7*, in each column, store the bit-pairs corresponding to 4-bit key words in adjacent CRAM cells. Each CAM cell spans two rows, hence incorporates two CRAM cells: #CAM cells/column = #bits in key or query word. For example, in Fig. 4(a), in each column, the two CRAM cells in *row 0* and *row1* together form a CAM cell. Each bit in the query register selects either of the rows in a CAM cell; if the query bit is 0 (1), it selects the stored (inverted) key bit. The cell in the selected row gets connected to the *LL* in the respective column, to serve as an input to the subsequent *NOR*. In Fig. 4(a), in each column, the *output cell* to *NOR* (in row8), connected to a different BSL group than the rest of the cells in the column, is preset to logic 0. On a per column basis, as each query bit selects one cell to connect to *LL*, applying V_{NOR} across the OBSL and EBSL (i.e., across cells storing key or inverted key bits, and the output cell) effectively creates a 4-input *NOR* gate that switches the output cell (from 0 to 1) iff all 4 input cells are logic 0. This marks a match between the query bits and the key bits stored in the respective column. In Fig. 4(a), the first query (0001) selects in both columns the cells in rows 0, 2, 4 and 7 (indicated in red), which renders all 0 cells in *col1*, hence, a 1 at *NOR* output to indicate a match. This is not the case for the content of the selected cells in *col0*: LL is connected to all logic 0s

but one, which is not enough to generate a current to switch the output cell, hence, no match.

TCAM operation: As indicated in Fig. 4(b), the same search mechanism for BCAM from Fig. 4(a) applies to TCAM search with two changes: i) a method to search the wildcard bits in query word – stored as a bit-mask in a separate register (i.e., bit-mask register) of equal size to the query register; and ii) additional bits in each column to help in wildcard search – *Reserved Wildcard Bits* (RWB) in *row8-row11*. RWBs are equal to the number of key bits stored in a column (4 in this example). Each RWB corresponds to one unique CAM cell in that column and stores a constant value (logic 0). When a query bit is marked as a wildcard bit (by setting the corresponding bit position in the bit-mask register to logic 1), the corresponding RWB is selected instead of any row of the corresponding CAM cell, effectively bypassing the BCAM match mechanism. When a query bit is not marked as a wildcard bit, on the other hand, the search proceeds exactly in the same way as BCAM search, as depicted in Fig.4(a). In this case, as well, if all cells selected to be connected to a LL are logic 0s, the *output cell* (in *row12*) in the respective column switches – marking a match. In Fig.4(b) this is the case for *col0* for the query (*1XX1*) since all 4 cells in *col0* selected to be connected to LL have logic 0 (in rows 1, 7, 9 and 10), which doesn't apply to *col1*.

Multi-gate logic operations: At the core of the search logic lies the *NOR* operation to generate the match outcome, which has as many inputs as the number of bits in the key (or equivalently query) words. While typical key (query) lengths tend to be fixed, CRAM logic gates cannot support arbitrary number of inputs – hence a limit applies to the #inputs of *NOR* along a column. As a workaround, we chunk query word into groups of bits and perform the search with one chunk at a time, sequentially. Each search in this case involves a n-bit *NOR* which is feasible to implement in CRAM where n is the #bits in the chunk. To generate the final match outcome, we feed the output of each such *NOR* gate to an *AND* gate, on a per column basis, which generates a 1 (to indicate a match) iff all *NOR* outputs are 1.

Row selection logic (RSL): Fig. 5(a) and (b) show the RSL for BCAM and TCAM, respectively (red = logic HIGH), implemented using conventional gates. Recall that each CAM cell spans two rows in a column. Each query bit in BCAM selects either of the rows in the corresponding CAM cell simply using a NOT gate. In TCAM, if a query bit is marked as a wildcard bit, neither of these rows are selected; instead the row with the corresponding *RWB* is selected. As an example, in Fig. 5(b), the first query bit selects *row1* and the last bit in bit-mask register selects the row *RWB3*. The TCAM RSL becomes the equivalent to the BCAM RSL if all bits in the bit-mask register are logic 0. RSL signals drive the rows of all CRAM tiles involved in search (in CAM mode).

During regular (non-CAM) CRAM operations, CRAM tile controllers (responsible for driving rows in each tile) bypass RSL signals for CAM operations– thereby making *all* cells available for CRAM operations.

3.3 Hardware Organization

CAMeleon incorporates a collection of CRAM tiles that store the key words – i.e., a sequence of *bit-pairs* corresponding to CAM

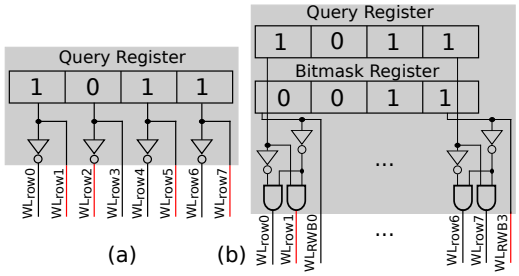

Figure 5: Row selection logic for (a) BCAM and (b) TCAM.

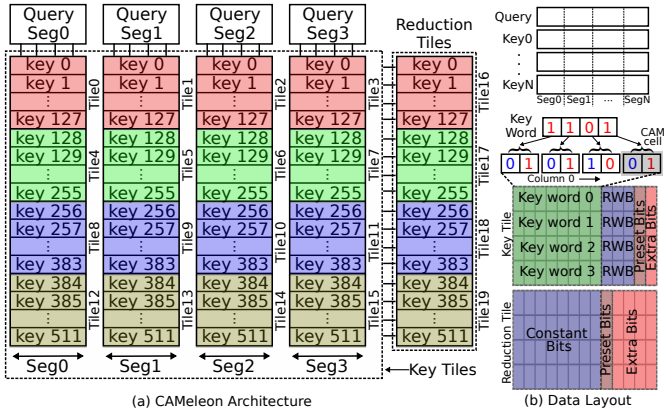

Figure 6: Hardware organization of CAMeleon (transposed to simplify illustration).

cells – along columns. All cells along a column that store the bit-pairs (and that represent inputs to the *NOR* operation to determine match outcome) are connected to same BSL group; cells keeping the *NOR* output, to the opposite. The query word is stored in the query register that is shared among tiles that perform search with that query.

The high level architecture of CAMeleon is shown in Fig. 6(a). The first step is to divide the query and key words into smaller equal length segments (as captured by the top portion in Fig. 6(b)). This serves two purposes: i) Storing very long key words (i.e., bit-pairs) in one column of a tile compromises signal integrity across first and last rows, ii) Long query words would require either CRAM logic gates with large #inputs (more prone to process variation of the MTJ, the peripheral circuitry and V_{NOR}); or to perform logic operations in >2 steps – contributing to latency and energy overheads. CAMeleon stores these segments in columns in *key* tiles where CAM search is performed on each of these {Query, Key} segment pairs, in parallel. Since each query bit is matched against the corresponding bit in the key word, such parallel CAM operations are bit-independent across all key tiles.

Fig. 6(a) illustrates an example where 512 key words and a query are divided into 4 segments each. Each key word segment is stored in a separate key tile. For example, key tiles 0-3 store segments 0-3 of key words 0-127. Tiles 0, 4, 8 and 12 store segment 0 of keys 0-511 and share the same RSL signals corresponding to the query segment 0.

Search outcomes from individual {Query, Key} segment pairs, i.e., *partial search outcomes*, are reduced to single match/no match indication by the corresponding reduction tile (Fig. 6(a)). This is

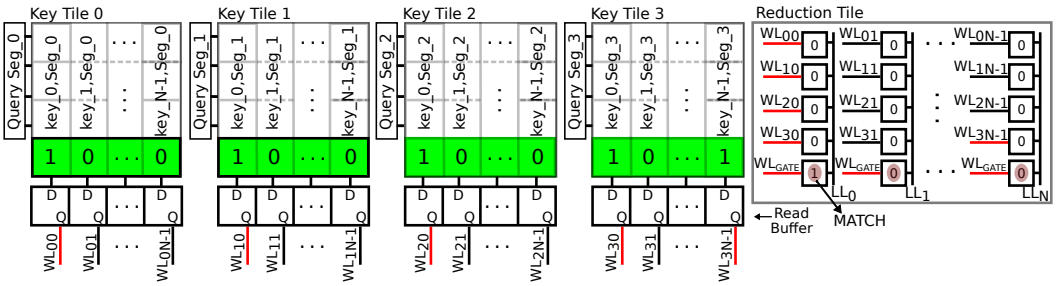

Figure 7: Reduction operation in CAMeleon.

a special purpose CRAM tile where *WL* of individual cells can be independently selected. Similar to the key tiles, *output cells* in the last row of the reduction tile hold the outcome of the reduction operations, i.e., (0)1 for (Mis)match.

Data layout: Fig. 6(b) shows the data layout of key and reduction tiles, with rows and columns transposed to simplify illustration. RWBs always store logic 0 when configured for TCAM. *Preset Bits* in each (key/reduction) tile act as *output cells*. The remaining bits in each column are used as *Extra* bits which do not participate in CAM search, rather provide flexibility in scheduling regular CRAM logic operations. Both *Preset* and *Extra* bits are connected to same BSL group, whereas the rest of the bits in each column are connected to the opposite.

Reduction operation: The reduction operation is illustrated in Fig. 7. Each N-column key tile stores one segment each from N key words and the corresponding segment of the query word, e.g., *Key Tile 0* stores *segment 0* from {key 0, key 1, ... key N-1} and performs search for *segment 0* of the query word. The last row in each key tile (green in figure) holds the *partial search outcome* from (B/T)CAM operation. Each key tile has a *read buffer* (RB), i.e., a D-FF array that stores the *partial search outcome* bits (which uses regular CRAM read). The outputs (Q) from RB connect to individual WLs (i.e., individual cells) along a row in the *reduction tile*. For example, the RB outputs from *Key Tile 0* connect to WL_{00}, WL_{10}, and so on, where WL_{XY} refers to the WL of a cell at *row X* and *column Y* in a reduction tile.

Accordingly, depending on the data stored in RBs (1/0), the individual WLs in the corresponding reduction tile get activated (colored *red*). In each reduction tile, all cells along a column with an active WL get connected to LL, by construction, effectively creating a logic gate configuration in that column. Except the last row, all cells in a reduction tile store a fixed 0 throughout CAM search. The last row in each reduction tile, i.e., *reduction output*, is preset to logic 0 before reduction operation begins. *Reduction output* keeps the result of the *NOR* operation where each partial match is represented by a logic 0 at its input. This *NOR* (along all columns of the reduction tile, simultaneously) outputs a logic 1 only if #cells connected to a LL = #{query,key} segment pairs.

The #rows and #columns in a reduction tile depends on the #segment pairs and #keys stored in each key tile connected to it respectively (= $K \times S$ CRAM cells where K = #key words and S = #segment pairs). The discussion on multi-gate logic operations described earlier applies here as well, depending on S and the maximum #inputs required by CAMeleon logic operations.

When not configured to perform CAM search, all cells in a (key or reduction) tile are available for CRAM logic and memory operations. In order to use *all* cells in the key and reduction tiles in regular CRAM operations, WL signals from corresponding tile controllers are *OR*ed with the signals from RSL and RB, respectively. Fig. 8 illustrates the idea. In case of CAM operations, RSL (RB) signals are used to drive the rows in a key (reduction) tile; otherwise the WL signals generated by the tile controller take precedence.

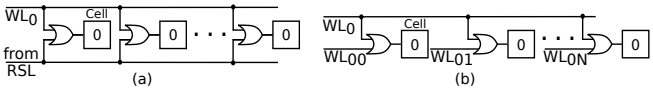

Figure 8: WL usage in (a) key and (b) reduction tiles.

Pipelining: Pipelining CAMeleon in order to reduce search latency is straight-forward, where RBs in key tiles can act as pipeline registers. If the latency of key (reduction) tiles is T_{ktiles} (T_{rtiles}), the total search latency of CAMeleon would be $T_{CAMeleon} = T_{ktiles} + T_{rtiles}$. Both T_{ktiles} and T_{rtiles} strongly depend on the number of logic gate operations (during search) the tiles perform. Typically, key and reduction tiles perform similar number and type of operations, so a balanced pipeline is possible, where $T_{CAMeleon} = \max(T_{ktiles}, T_{rtiles})$ applies. In this case, read buffers store the partial outcomes from key tiles, corresponding to a query, which reduction tiles use to derive the final search outcome (i.e., *reduction output*), while key tiles begin with the search of the next query from the query pool.

4 EVALUATION SETUP

The performance evaluation of CAM search using CAMeleon, in terms of latency/search and energy/search/bit, is performed with an in-house step accurate simulator, where each step is a logic operation, e.g., NOR. The energy and latency of *NOR* and *AND* gates are derived from the electrical equivalent circuits (see Section 2). Table 2 lists all STT-MTJ parameters used in the evaluation [14, 20, 23]. The low power (LP) and high performance (HP) variants of current MTJs – CLP, CHP, CHPA(ggressive) are considered to capture the sensitivity of CAMeleon performance to device technology parameters. Similar variants are considered for future (projected) MTJ (FLP and FHP), as well. All peripheral overheads (including RSL and query register related), at 22nm technology node, are derived from NVSIM [7] and HSPICE simulations, by accounting for parasitic effects such as wire capacitance.

Dataset: As CAM key word dataset, 1024 128-bit words are randomly generated and each is used to generate 1000 128-bit query words with randomly placed (bit position has no impact on performance or correctness of search output) wildcard bits– a large enough dataset for average search performance characterization. The (default) #wildcard bits is assumed to be 50% of the query length

(= 64). 128 key tiles (64 × 64) are required to store the key dataset. Each key (and query) is divided into 8 (= 128/16) segments of 16-bit length. For reduction, 16 (64 × 64) reduction tiles are required (each reduction tile reduces 64 key words). The total memory footprint is ∼ 72 KB (∼ 57% is used during CAM operations) – the entirety of which is available to be used as a regular CRAM array.

Logic gate configuration: The (default) #inputs to CRAM *NOR* gates is 8 (corresponding to a good trade-off between overall performance and reliability of logic operations), and each key tile performs two such logic operations in sequence (since 16 key bits are stored in each column of key tiles) before feeding a 2-input CRAM *AND* gate.

Table 2: Technology Parameters.

Parameter	CLP	CHP	CHPA	FLP	FHP
MTJ Type	Interfacial PMTJ				
MTJ Diameter (nm)	45			10	
TMR (%)	133			500	
RA Product ($\Omega\mu m^2$)	5			1	
I_{crit} (μA)	40	90	180	0.79	10
Switch. Latency (ns)	3	1	0.3	1	0.3
$R_P, R_P, R_{Trans.}$ ($K\Omega$)	3.15, 7.34, 1			12.7, 76.39, 1	

Baselines for comparison: To quantify the performance improvement of CAMeleon, 8 state-of-the-art baseline TCAM designs are selected, with different device technologies, including SRAM, STT-MTJ, ReRAM and PCM [3, 8, 9, 13, 19, 21, 30, 32]. The numbers reported for the baselines and CAMeleon exclude encoder overhead at the output.

5 EVALUATION

5.1 Performance analysis

Fig. 9 provides the energy/search/bit and latency/search characterization (normalized to [3]; the lower the better). Overall, CAMeleon with CLP consumes less energy than STT-MTJ- [21], ReRAM- [13, 19] and PCM-based [9] designs. With future (projected) MTJ-variants (FLP and FHP), the energy consumption reduces even further and CAMeleon outperforms all baselines. The baselines with STT-MTJ, ReRAM and PCM use the memory devices to only store CAM data, unlike CAMeleon which also performs computation (i.e., CAM search) with the memory devices – making CAMeleon more sensitive to device technology parameters.

On the latency front, performance of CAMeleon is dominated by the switching latency of the STT-MTJ devices. Due to longer switching latency of CLP, CAMeleon-CLP suffers from the longest search latency across the board. As MTJ variants (CHP, CHPA, FLP and FHP) exhibit increasingly lower latency, CAMeleon recovers latency significantly, e.g., by a decrease of 8.1× from CLP to CHPA. Although the baselines outperform CAMeleon in terms of search latency, it comes with a significant energy (e.g., [21] consumes 5.5× more energy than CAMeleon-CLP) and area penalty (e.g., [32] uses 5× more transistors/cell than CAMeleon). Table 3 compares all baselines and CAMeleon in terms of area overhead (#Transistors/cell). The SRAM-based baseline [3], while consuming less energy than most baselines, suffers from a high area overhead (16T/cell) – making it difficult to fit in a tight area budget imposed by embedded/edge hardware. CAMeleon, on the other hand, has smaller area footprint than most baselines, except for [9], [19] and [13] which have similar or slightly smaller footprint at the expense of higher energy consumption, e.g., [13] consumes 25.8× more energy than CAMeleon-CLP. Considering the finely tuned dedicated sense amplifiers for CAM search – required by all these baselines (in addition to read sense amplifiers), CAMeleon is even more area efficient.

In summary, CAMeleon outperforms a wide-range of baselines, in terms of area or energy (or both), while maintaining a comparable search latency. CAMeleon, in BCAM mode, exhibits similar energy (∼ 0.1% less than corresponding TCAM numbers) on average.

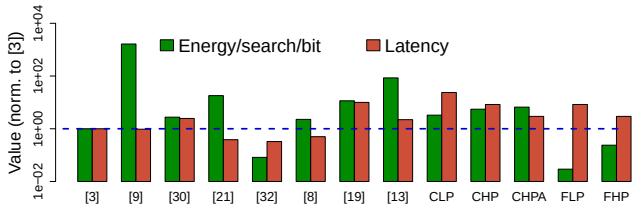

Figure 9: Energy and latency comparison (normalized to [3]).

5.2 Sensitivity analysis

TCAM energy consumption is sensitive to the #wildcard bits in the search query. Fig. 10 captures the relationship between energy/search/bit and % of Wildcard bits in varying lengths of query word (normalized to 25% wildcard share). The energy consumption decreases, although insignificantly (∼ 1%), with increasing #Wildcard bits in query. More wildcard bits tend to yield more matches between {query, key} segment pairs– resulting in lower energy consumption due to *AND* gates with all logic 1 inputs, which doesn't incur switching.

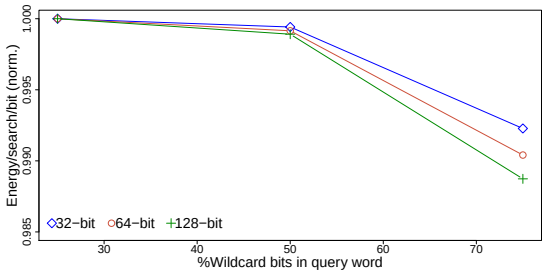

Figure 10: Sensitivity of CAMeleon to #wildcard bits.

Query length: Energy consumption in CAMeleon depends on the query length, as well, although insignificantly. Table 4 lists the energy consumption in CAMeleon when the query length is varied between 32 and 128 bits (normalized to 128-bit). The energy consumption (per search per bit) tends to decrease as query length increases – indicating good scalability. This is because the dominant search energy component (gate energy; > 60% of total) does not scale in proportion to the query length, e.g., gate energy with 64-bit query is ∼ 1.72× of that with 32-bit query – resulting in lower energy/search/bit (vs. 32-bit) for 64-bit query.

Process Variation: The reliability of CAMeleon operations depend on the correct switching events in CRAM logic operations. High #inputs to logic operations could exhibit switching for incorrect input data. To understand the impact of process variation on CAMeleon

Table 3: CAMeleon TCAM cell comparison against baselines.

Parameter	[3]	[9]	[30]	[21]	[32]	[8]	[19]	[13]	CAMeleon
Device	SRAM	PCM	STT-MTJ	STT-MTJ	STT-MTJ	STT-MTJ	ReRAM	ReRAM	STT-MTJ
Tech. node (nm)	40	22	40	40	40	22	14	45	22
Cell	16T	3T-3R	10T-4M	9T-2M	15-4M	6T-2M	3T-1R	2T-2R	3T-3M
Word Length (bits)	144	128	144	144	144	256	128	8	128
Logic-capable	No								Yes

Table 4: Sensitivity to query length.

Query length (#bits)	32	64	128
Energy/search/bit (norm.)	1.29	1.11	1.00

functionality, we considered variation in STT-MTJ and V_{NOR}. For MTJ low resistance (R_P), we assume a σ of 10% with 0.1% variation of TMR (which captures the variability in oxide thickness and surface area), and a 5% standard deviation for V_{NOR}. Our Monte Carlo analysis for an 8-input *NOR* gate, with 10^8 iterations, shows correct switching behavior $\sim$ 100% of the time even under our conservative assumptions. We also introduced incorrect switching behaviour in this 8-bit *NOR* gate to output a logic 1 when 7 (instead of all 8) inputs are logic 0 (with default query and key configurations). Since, in order to get an incorrect match between a {query,key} pair, all corresponding segments have to yield erroneous match through such a (faulty) gate (which is very unlikely), there was no erroneous match in CAM output for the queries.

Gate Width: Higher #inputs results in lower overall CAMeleon latency (more query bits are searched with each logic operation) and lower energy (which decreases quadratically with #inputs), however, with increasing probability of incorrect switching behavior, i.e., error in CAM search output. For example, with 16-input *NOR* gate, the latency and energy consumption of CAMeleon reduce by 1.95× and 3×, respectively, relative to the 8-input gate based design. Such a rich trade-off space is attractive for approximate CAM search, which we leave to future work.

6 CONCLUSION

The constrained execution environment in edge and embedded computing domains, where CAM represents an ubiquitous functional block, requires low-overhead reconfigurability to re-purpose hardware resources, in order to stay within very tight area and energy budgets. In this paper we present CAMeleon, a unique reconfigurable hardware solution which fuses spintronic PIM and (B/T)CAM functionality in a seamless and effective fashion. We show that CAMeleon can outperform a wide-range of CAM baselines, in terms of area or energy consumption (or both), while maintaining comparable search latency – and unlike any of the baselines, while also supporting PIM functionality.

7 ACKNOWLEDGMENTS

This work was supported in part by NSF grant no. SPX-1725420.

REFERENCES

[1] Gordon Brebner et al. 2014. High-speed packet processing using reconfigurable computing. *IEEE Micro* 34, 1 (2014).
[2] Wei-Hao Chen et al. 2019. CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nat. Electron.* 2, 9 (2019).
[3] Woong Choi et al. 2018. Low cost ternary content addressable memory using adaptive matchline discharging scheme. In *ISCAS*. IEEE.
[4] Zamshed Chowdhury et al. 2017. Efficient in-memory processing using spintronics. *IEEE CAL* 17, 1 (2017).
[5] Zamshed I Chowdhury et al. 2019. Spintronic in-memory pattern matching. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 5, 2 (2019).
[6] Nazgul Dastanova et al. 2018. Bit-plane extracted moving-object detection using memristive crossbar-cam arrays for edge computing image devices. *IEEE Access* 6 (2018).
[7] Xiangyu Dong et al. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE TCAD* 31, 7 (2012).
[8] Rekha Govindaraj et al. 2017. Design and analysis of STTRAM-based ternary content addressable memory cell. *ACM JETC* 13, 4 (2017).
[9] Qing Guo et al. 2011. A resistive TCAM accelerator for data-intensive computing. In *MICRO*. IEEE.
[10] Qing Guo et al. 2013. Ac-dimm: associative computing with stt-mram. In *ISCA*.
[11] Jhih-Yu Huang et al. 2018. TCAM-based IP address lookup using longest suffix split. *IEEE/ACM Trans. Netw.* 26, 2 (2018).
[12] Li-Yue Huang et al. 2014. ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-word length-capacity for normally-off instant-on filter-based search engines used in big-data processing. In *Symposium on VLSI Circuits Digest of Technical Papers*. IEEE.
[13] Mohsen Imani et al. 2016. Resistive configurable associative memory for approximate computing. In *DATE*. IEEE.
[14] Guenole Jan et al. 2014. Demonstration of fully functional 8Mb perpendicular STT-MRAM chips with sub-5ns writing for non-volatile embedded memories. In *VLSI-Technology*. IEEE.
[15] Roman Kaplan et al. 2017. A resistive cam processing-in-storage architecture for dna sequence alignment. *IEEE Micro* 37, 4 (2017).
[16] Wei-Pau Kiat et al. 2020. An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications. *Microprocessors and Microsystems* 73 (2020).
[17] Olga Krestinskaya et al. 2018. Binary weighted memristive analog deep neural network for near-sensor edge processing. In *NANO*. IEEE.
[18] Olga Krestinskaya et al. 2019. Neuromemristive circuits for edge computing: A review. *IEEE Trans. Neural Netw. Learn. Syst* 31, 1 (2019).
[19] Shuangchen Li et al. 2016. Nvsim-cam: a circuit-level simulator for emerging nonvolatile memory based content-addressable memory. In *ICCAD*. IEEE.
[20] Hiroki Maehara et al. 2011. Tunnel magnetoresistance above 170% and resistance–area product of 1 Ω (μm) 2 attained by in situ annealing of ultra-thin MgO tunnel barrier. *Appl. Phys. Express* 4, 3 (2011).
[21] Shoun Matsunaga et al. 2012. A 3.14 um 2 4T-2MTJ-cell fully parallel TCAM based on nonvolatile logic-in-memory architecture. In *VLSIC*. IEEE.
[22] Xuan-Thuan Nguyen et al. 2018. An FPGA-based hardware accelerator for energy-efficient bitmap index creation. *IEEE Access* 6 (2018).
[23] Hiroki Noguchi et al. 2015. 7.5 A 3.3 ns-access-time 71.2 μW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture. In *ISSCC*. IEEE.
[24] Naoya Onizawa et al. 2012. High-throughput low-energy content-addressable memory based on self-timed overlapped search mechanism. In *ASYNC*. IEEE.
[25] Somnath Paul et al. 2008. Reconfigurable computing using content addressable memory for improved performance and resource usage. In *DAC*.
[26] Yuriy V Pershin et al. 2011. Neuromorphic, digital, and quantum computation with memory circuit elements. *Proc. IEEE* 100, 6 (2011).
[27] Bipin Rajendran et al. 2011. Demonstration of CAM and TCAM using phase change devices. In *IMW*. IEEE.
[28] Salonik Resch et al. 2019. PIMBALL: Binary Neural Networks in Spintronic Memory. 16, 4, Article 41 (Oct. 2019). https://doi.org/10.1145/3357250
[29] Salonik Resch et al. 2020. MOUSE: Inference In Non-volatile Memory for Energy Harvesting Applications. In *MICRO*. IEEE.
[30] Byungkyu Song et al. 2016. A 10T-4MTJ nonvolatile ternary CAM cell for reliable search operation and a compact area. *IEEE Trans. Circuits Syst. II Express Briefs* 64, 6 (2016).
[31] Hsiang-Jen Tsai et al. 2017. Energy-efficient TCAM search engine design using priority-decision in memory technology. *IEEE TVLSI* 25, 3 (2017).
[32] Chengzhi Wang et al. 2018. A novel MTJ-based non-volatile ternary content-addressable memory for high-speed, low-power, and high-reliable search operation. *IEEE Trans Circuits Syst I Regul Pap* 66, 4 (2018).
[33] Hasan Erdem Yantır et al. 2018. A Hybrid Approximate Computing Approach for Associative In-Memory Processors. *IEEE Trans. Emerg. Sel. Topics Circuits Syst.* 8, 4 (2018), 758–769. https://doi.org/10.1109/JETCAS.2018.2852701