Power optimization techniques for associative processors

Hasan Erdem Yantir, Ahmed Eltawil, Smail Niar, Fadi Kurdahi

Journal of Systems Architecture 90 (2018) 44–53

Power optimization techniques for associative processors

Hasan Erdem Yantı r [a, *], Ahmed M. Eltawil [a], Smail Niar [b], Fadi J. Kurdahi [a]

[a] Center for Embedded and Cyber-physical Systems, University of California, Irvine, Irvine, CA 92697, USA
[b] Laboratoire d'Automatique, de Mcanique, et d'Informatique Industrielles et Humaines (LAMIH), Polytechnic University Hauts-de-France, Valenciennes, 59313, France

Keywords:

Low-power processor
In-memory computing
Associative processors
SIMD

a b s t r a c t

The toughness and complexity of the computational problems which human beings tackle rise faster than the computational platforms themselves. Moreover, the dark silicon era negatively effects the traditional computational platforms and contributes unfavorably to this gap. These situations require the alternative computing paradigms, ranging from multi-core CPUs to GPUs and even untraditional paradigms such as in-memory computing. Associative processing (AP) is a promising candidate for in-memory computing where the computation is performed on the memory rows without moving the data. Even though APs propose a good solution for the memory bottleneck, their power density poses an issue because of the huge switching activity on the rows happens during the operations. In this study, we seek a low-power AP implementation by proposing architectural and instructional improvements to decrease the switching activity. The simulations on various benchmarks from different domains show that the proposed low-power AP methods provide energy reduction up to 48% with a negligible impact on the area and performance.

1. Introduction

Today's demand for high-speed processing cores with minimal power consumption requires more computational power ready in the integrated circuits. Even though transistor scaling continues and transistor counts in an integrated circuit still grow, the power density expectations specified in Dennard Scaling [1] have been broken down because of the current leakages which in turn violates the power density limits critically. The increasing the frequency method does not perform favorably also due to the intractable consequence of the heat increase. This situation creates a shift towards multi-core processors as an alternative method for the sake of performance increase. This paradigm aims to increase the number of cores in a central processing unit (CPU), thus improving the overall performance as well. However, placing many cores running at high frequencies still impose a threat to overall power consumption. Microprocessor trends show that their power densities get close to that of a nuclear reactor [2]. As an alternative approach, graphical processing units (GPUs) focus on the throughput instead of the latency as in CPUs to tackle the parallel tasks utilizing many cores simultaneously. The main idea of GPUs is increasing the number of cores by making them simpler with limited latency achievements. These simpler cores also require simpler controller units which in turn facilitate placing of much more cores with a similar degree of power density. The GPUs show a successful trend in high-performance computing as seen in the deep learning benchmarks where the performance has the greatest importance [3]. However, GPUs still have drawbacks of memory bottlenecks as like CPUs. GPUs require data transfer minimization as much as possible in order to maximize computation/communication ratio. If the amount of the transferred data is high, GPUs cannot hide the data transfer overhead. On the other hand, today's many benchmarks require large amounts of data to be processed (e.g., machine learning, artificial intelligence, etc.). For this reason, research regarding low-power alternatives for the traditional computing methods is highly attractive.

The current dilemma in the traditional computing methodologies pushes researchers to work on alternative techniques to fulfill the demand of ever growing applications. The main limitations can be summarized as; (1) power density, and (2) memory bottleneck. Theoretically, the most memory efficient paradigm is the in-memory computing which is an unconventional method that aims to process the data inside the memory without moving it. *Associative processor (AP)* is an in-memory computational platform for massively parallel computing [4]. APs combine logic with memory structures and virtually eliminate need for memory load/store operations during computations. This feature largely overcomes the memory bottleneck problem of traditional Von Neumann architectures since there is no inter-dependence between memory and processor. APs can be considered as a kind of Single Instruction Multiple Data (SIMD) processors, however, they have much simpler cores since a memory row behaves as a single processor core despite GPUs [5]. This simplicity in the cores provides much better results in terms of energy when compared with the traditional processors.

DOI : 10.1016/j.sysarc.2018.08.006

APs can be considered as a candidate of next-generation processors on the trend of CPU to GPU where the cores become simpler and the focus shifts towards the throughput. In addition to simpler cores and controller, APs process the data inside the memory without moving it. APs can act as a standalone processor to perform a variety of benchmarks. However, in general purpose computing, their usage as an on-chip accelerator [6–8] is more appropriate since they are a kind of SIMD processor and sometimes such parallelism is not needed during the overall flow of a benchmark.

Even though there are area efficient alternatives of SRAM-based AP architectures such as Resistive Associative Processor (RAP) [8] based on memristor CAM [9] or AC-DIMM based on STT-RAM [10,11], these alternatives are energy inefficient. For instance, single bit write operation on SRAM consumes around less than 1fJ [12], conversely, a memristor write operation consumes 100 times more energy [13] (i.e., single bit write in the RAP requires two memristor writes). The high energy consumption together with a smaller area contributes to the energy density problem that is one of the biggest issues in the dark silicon era. For this reason, in this study, an SRAM-based associative processor (SAP) is referenced as the associative processor (AP).

Even though APs are a good solution for the memory bottleneck, their energy efficiency is limited since they require the high switching activity on memory during the computation. However, there are limited studies on this problem since this approach has been regaining importance recently. As an example, in [14], the energy savings in the APs are obtained by approximate associative computing where some bits in the CAM are either extracted from the computation or the switching range of the memristor is shrunk. Similar to APs, there are other low-power methods applied to other in-memory computing architectures or CAMs used for similar purposes. For example, in [15], an architectural extension to GPUs are proposed to avoid the frequent re-executions by recalling their results directly from a cache-like resistive CAM (RCAM) and memory structure tied to the every FPU in the GPU. The CAM structure facilitates approximate matching, therefore providing energy savings at the expense of an acceptable decrease in quality. In [16], a low power in-memory computing platform using a novel 4-terminal magnetic domain wall motion (4T-DWM) device is proposed for in-memory computing. An approximate processing in-memory architecture (called APIM) is proposed in [17] which uses the analog behavior of memristors in addition and multiplication operations.

In this study, we investigate the low-power associative computing methods to aid in fulfilling the limitations of current computing paradigms. The proposed methods cover AP architectures used as either standalone processors or on-chip accelerators. The contribution of this study to the research community can be summarized as follows:

- The cycle accurate behaviors of the AP operations are analyzed to detect the possible deficiencies which cause surplus energy consumption.
- The matching circuit in the APs are modified to achieve lower switching activity. The proposed circuit prevents the unnecessary compare cycles which are not needed inherently in the process of associative computing.
- The new truth tables for some operations (multiplication and absolute value) which provide more energy savings by exploiting the proposed architecture further are introduced for the first time.

The rest of the paper is organized as follows: In the following section, the background of AP's are presented in detail. Section 3 gives the motivation of the proposed low-power methodologies in Section 4. Experimentation and evaluation results are discussed in Section 5. The final section concludes the work.

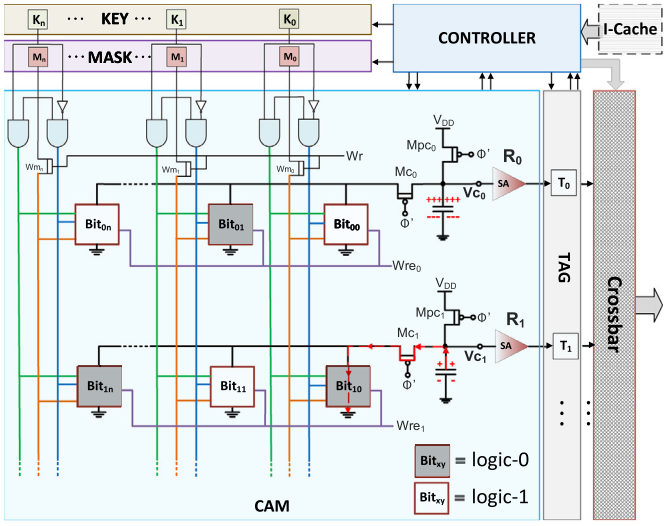


Fig. 1. Architecture of an associative processor.

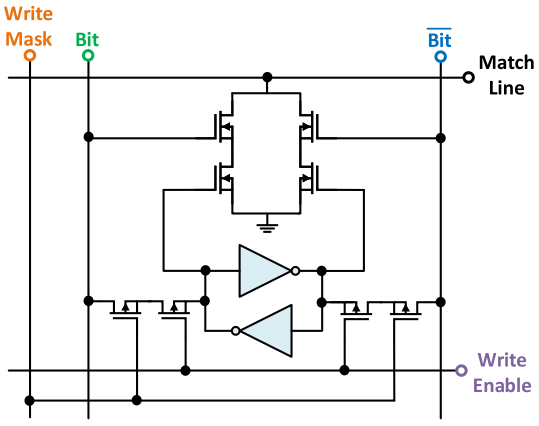


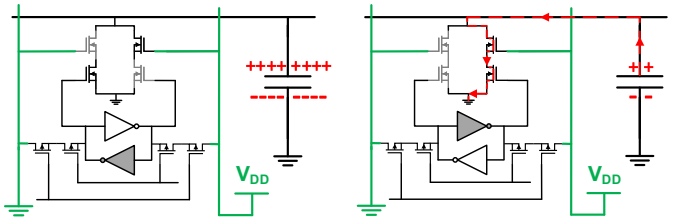
Fig. 2. SRAM-based CAM cell.

2. Background

2.1. Architecture

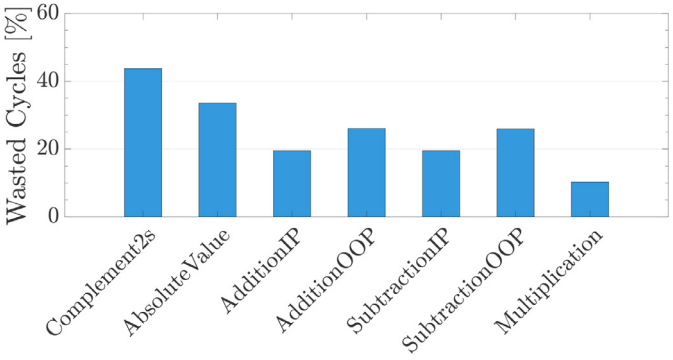
The essential part of an associative processors (AP) is a content addressable memory (CAM) that is capable of identifying the locations of the rows which contain the informations that is searched for inside the CAM (i.e., compare) and the writing the data to the specified columns of the identified locations (i.e., write). Fig. 1 shows the architecture of an AP that consists of a CAM, controller, specific (key, mask, and tag) registers, an instruction cache (I-Cache), and an optional crossbar. The CAM holds the data on which instruction performed. The instructions are stored in the I-Cache in an ordered manner. The controller is responsible for feeding the instructions and decoding them. As a result of decoding, the controller generates the corresponding key and mask values. The *key* register contains the value that is compared against or written. The *mask* register indicates which bit or bits are activated during comparison or write. In addition to them, an optional crossbar (interconnection switch) can be used for data exchange between the rows APs. This interconnection provides data transfer for the applications that require pipelined set of APs (e.g., FFT, filtering, etc.).

In the CAM, each cell stores a single bit either logic-0 or logic-1. Fig. 2 shows an SRAM-based cell used to store one-bit together with an additional circuitry for masked search (bit) and write operations (write mask and enable). For better visualization, the connection points of the cell are shown in the same color with its corresponding wires in Fig. 1.



(a) Match (b) Mismatch

Fig. 3. Typical evaluation phases of a AP cell for the match (a), mismatch (b) states.



Algorithm 1. In-place subtraction (**B = B − A**) in the AP.

In this cell, the one-bit data is stored by a coupled inverter and accessed through the outer circuitry. During the compare, the mask register is initialized to point to the interested columns and the key is set to the value searched inside the CAM. Even though key length equals to the CAM length, the only key values are taken into account where the corresponding mask registers are logic-1. During the compare, a matching circuit attached to each row distinguishes the rows that matched with the combination of given key and mask values from the mismatched ones [8]. Basically, this circuit uses two phases to differentiate the matched and mismatched rows; *pre-charge* and *evaluate*. In the pre-charge phase, the capacitors at each row (row capacitors) of the CAM are pre-charged. During the evaluate phase, a search word is applied to the columns. Only rows carrying matching data will retain charge because the matching transistors on the leakage path of the cell remain switched off. On the other hand, the other rows (i.e., the rows that store a different value than the key) leak their charges since their matching transistors will conduct. Fig. 3 shows these two cases explicitly for the SRAM-based CAM cell. If the stored bit is the reverse of the searched bit, the matching transistors (on the top of the coupled inverter in Fig. 2) forms a short path to the ground and the charge across the capacitor leaks on this path. If they are same, the path becomes closed and only a small amount of charge can leak. The figure indicates the paths of the closed transistors as black and the closed paths as gray. The charges on a row capacitance leak from the mismatched cell, where both series transistor turn on, creating a path to the ground as shown in Fig. 3b. On the other hand, Fig. 3a shows the state of the SAP cell in case of a match. In the figure, the CAM-cell stores a logic-1 value and the "01" pattern is applied to the cell to look for logic-1. Since this a match case, the inverter feeding to the right matching circuit closes the transistor while other one opens. On the other hand, "01" pattern closes the left matching transistor and opens the right one where no path to the ground is available through the matching transistors. The sense amplifier senses the residual charge across the capacitor after the evaluate phase and compares it with a given reference voltage (V_{th}). As a result of comparison, it generates a logical correspondences of match and mismatch cases as logic-1 and logic-0 respectively. For example, if we set the key to 100 and mask to 110, the tag bits of the corresponding rows whose third and second bits are logic-1 and logic-0 respectively becomes logic-1 and the rest is logic-0. In order to write to the specified columns of the matched CAM rows, both write enable (*Wre*) and write mask (*Wm*) inputs of the cell are asserted. Write enable input selects the activated rows which is related with the tag (i.e., the matched rows after a compare operation). On the other hand, write mask activates the columns which is to be written. The mask register is used to control the write mask signal where its value is propagated to write mask port of the cells only during the write phase. Lastly, the value to be written and its reverse are applied to bit and $\overline{\text{bit}}$ columns respectively.

2.2. Operation

The AP performs the operations by consecutive compare and write phases. The values compared or written are referenced from a lookup table (LUT). Each operation has a specific LUT. By utilizing consecutive compare and write cycles with a corresponding LUT, any function that can be done on a sequential processor can be implemented in APs as a parallel operation. As an example, to perform an AND operation on two input columns and then write the result in another column initialized as all 0s, the LUT of AND operation (R = A&B) is applied to the CAM where CAM is searched for "11" in the input columns (A and B) and "1" is written to the result column (R) of the tagged rows. In other words, the truth table of the function is applied to perform this function. The more complicated operations (such as addition, subtraction, multiplication, etc.) require the bigger truth tables and take more than one steps.

Fig. 4 shows the step by step execution of in-place subtraction (B = B − A) of two 1×4 4-bit vectors, A and B, i.e. $B[i] \leftarrow B[i] - A[i], i = 0 \ldots 3$. The first row in the figure presents the LUTs for the in-place subtraction where referenced LUT entry for the corresponding column is highlighted as shaded. Other tables show the progress in the CAM contents together with the key/mask registers and tag status where the direction of the flow is indicated by arrows. The LUT for in-place subtraction has four entries in a specific order required to perform the operation correctly and mainly performs the single bit full subtraction. The order which LUT entries is applied on the bits of the vectors is specified in the comment column where NC stands for *no change*. NC entries have no effect on the operation, so not applied to the CAM, so LUT for in-place subtraction consists of 4 entries.

Algorithm 1 shows the controller flow for this addition operation. To perform the operation, *SubtractionIP(8, 7, 4, 3, 0)* instruction is executed where A and B vectors are between the columns of 3–0 and 4–7 in the CAM and Br (borrow) is in the 8th column. Initially, A contains (i.e., bits 3–0) the values of [−3; 7; −2; 1] and B (i.e., bits 7–4) contains the values of [−8; 1; 5; 6] as shown in the upper-left (initial) CAM. The subtraction is done in a bit-serial, world parallel mode, that is, a bit-wise subtraction is performed on each row of Fig. 4. In the figure, each step consists of a compare phase and its following write phase. For each CAM in the figure, the highlighted value from the compare column of the LUT is searched inside the masked columns of the CAM simultaneously and the matched rows are tagged as logic-1. After that, the values specified in the Write columns of the LUT are written to the indicated columns of the matched rows. For example, in the first step, 001 is searched in the Br and the first bits of A and B (column 8, 4, and 0 respectively). There is a match in the first and fourth rows, so the B and Cr values of these rows are changed as logic-1 in the following write phase. Combination of four tables in a row of the figure corresponds to a *LUT pass* where a single-bit subtraction is performed, so the mask locations are shifted left (except Br) at each row. This process is repeated for every bit position and for every LUT entry. Since there are four LUT entries and four bits, the total operation takes 16 steps. Finally, the value stored in B becomes [−5; −6; 7; 5] which is equal to B-A (i.e., [−8–(−3); 1–(7); 5–(−2); 6–(1)]).

As inferred from Algorithm 1, subtracting a vector from another (in-place subtraction) that are m-bit wide takes $10m$ cycles ($4m$ compares

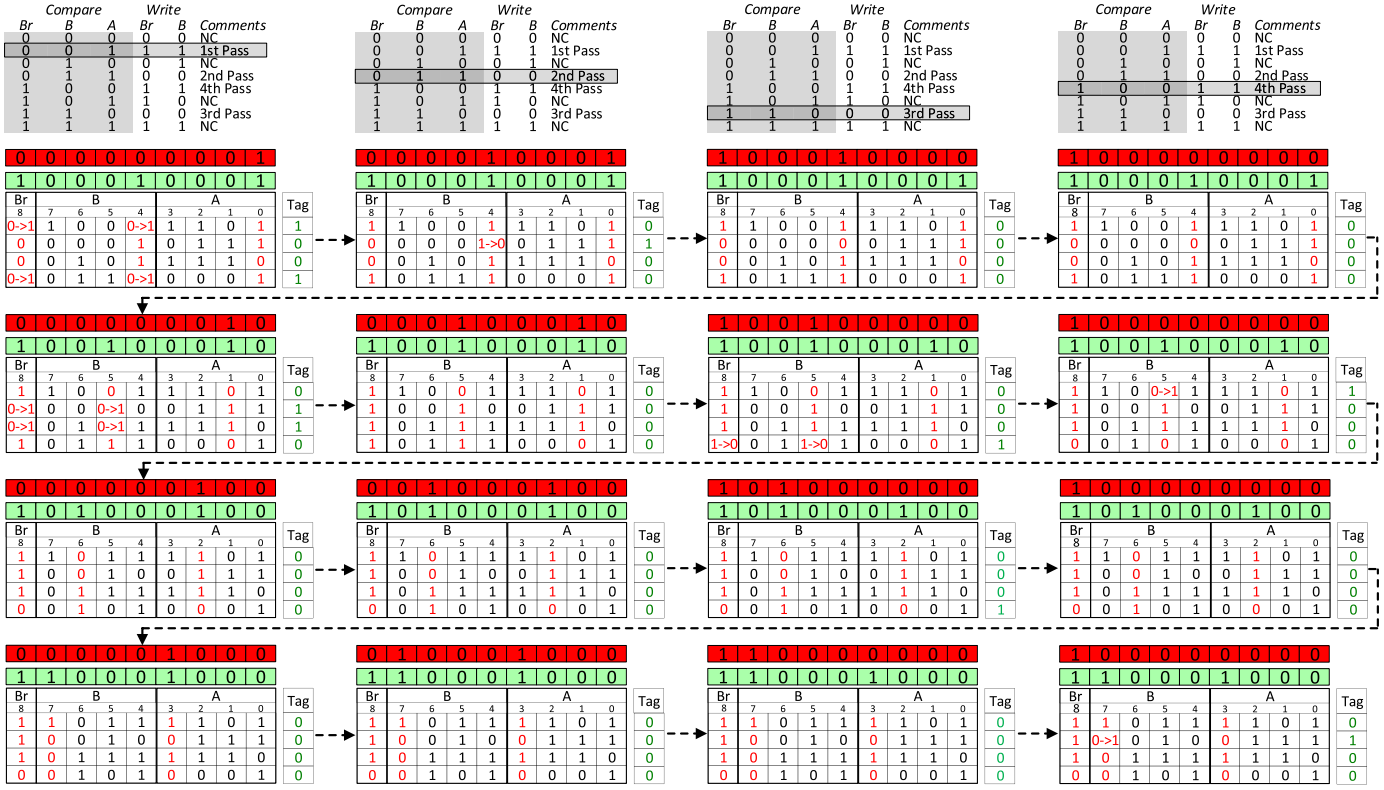


Fig. 4. Vector subtraction operation on 4-bit four number pairs. The sequence of compare and write operations are shown for a complete vector subtraction.

Table 1
Running time and area evaluation of primitive AP operations/instructions .

Function	Runtime	Area/Row	Complexity
NOT	$2m$	$2m$	$\mathcal{O}(m)$
AND	$2m$	$3m$	$\mathcal{O}(m)$
OR	$6m$	$3m$	$\mathcal{O}(m)$
Addition (IP, S/U)	$10m$	$2m+1$	$\mathcal{O}(m)$
Addition (OOP, S/U)	$11m$	$3m+1$	$\mathcal{O}(m)$
Subtraction (IP, S/U)	$10m$	$2m+1$	$\mathcal{O}(m)$
Subtraction (OOP, S/U)	$11m$	$3m+1$	$\mathcal{O}(m)$
2's Complement	$6m$	$2m+1$	$\mathcal{O}(m)$
Absolute Value	$8m$	$2m+1$	$\mathcal{O}(m)$
Multiplication (U)	$10m^2$	$4m$	$\mathcal{O}(m^2)$
MAC (U)	$10m^2+10m$	$4m$	$\mathcal{O}(m^2)$
Multiplication (S)	$10m^2+4m-14$	$8m+4$	$\mathcal{O}(m^2)$

IP:in-place, OOP:out-of-place, S:signed, U:unsigned, m:bitwidth.

and $6m$ writes), independently of the vector size. Table 1 shows the runtime (total number of passes), area usage per CAM row (in terms of bits), and algorithmic complexities of each arithmetic and logical operations in the AP. As stated in the table, the run time of a vectorial operation on m-bit n numbers depends only on the number of bits (m), not on the number of vectors (n). In the AP, all instructions except multiplications have a linear control flow ($\mathcal{O}(m)$) similar to in-place subtraction (see Algorithm 1) where LUT, mask and key locations are replaced with the correspondence of each operation. In multiplication, the control flow includes two nested loops so their run times are quadratic in terms of bit width, $\mathcal{O}(m^2)$. In this case, the AP outperforms the traditional processors in vectorial operations if the vector size (n) is big enough. Moreover, AP has no additional costs for data moving costs (e.g. data access, cache misses) and additional data storage costs which impose additional overhead to single-core and vector processors.

3. Motivation

Fig. 6 shows the waveform of one-bit subtraction where the operations correspond to the second row in the 4-bit subtraction example (see Fig. 4). The figure shows the voltage changes across the row capacitors (V_{SARX} where X corresponds to the row number) together with the reference threshold voltage (Vth). In the figure PC, E, WR labels correspond to the pre-charge, evaluate, and write phases respectively. After an evaluate cycle, if the voltage drops below the threshold, the sense amplifier outputs a logic-0 and logic-1 if the voltage is above the threshold.

APs perform the operations as bitwise with the combination of LUT passes on the bits generally from the least-significant bit (LSB) through the most-signification bit (MSB). A single LUT pass means completion of all passes inside a LUT. In the usual process of APs, each LUT pass performs an operation on the single bit of each operand. As an example, the first row in Fig. 4 corresponds to a LUT pass which consists of four compare operations. During each of these comparisons, a LUT entry is searched inside the CAM. During this LUT pass, a single bit subtraction operation is performed on the first bits of A and B. In the first compare operation of this LUT pass (i.e., searching 001 in the locations of Br and the first bits of B and A respectively), the first and fourth rows of the CAM are matched. It means that these rows are participated the subtraction operation during this pass. In the associative processing, it is impossible to get another match within a LUT pass. Therefore, these two rows can be extracted from the computation during the following three compare operations. In other words, these rows have already participated to the subtraction operation during the first compare cycle, so there is no need to query them again during the rest of the computation of subtraction on the first digits. In a similar manner, during the operation on the second bits of B and A (the second row of Fig. 4, the second and third rows are matched during the first compare. In the waveform of this operation (Fig. 6), it is not possible to get another match for these rows between the interval 9 ns and 14 ns (i.e., after the first match). On the other hand, the pre-charge capacitors of these rows

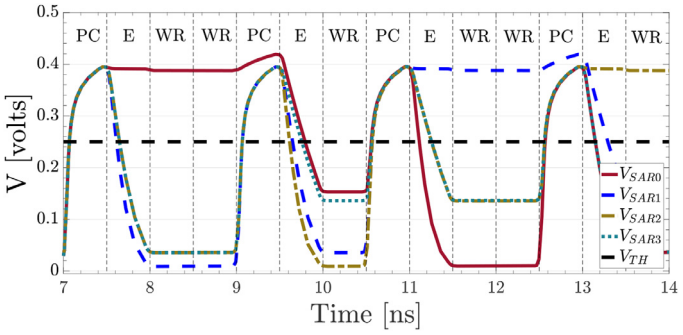


Fig. 5. Unnecessary (wasted) cycle percentages of the fundamental arithmetic operations in the AP.

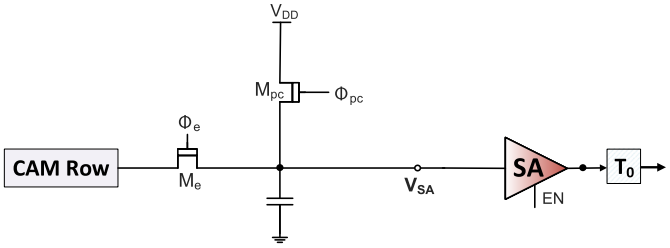


Fig. 6. Waveform of single-bit subtraction which corresponds to the second row of Fig. 4.

are still pre-charged three times during this period even though these cycles are simply wasted and the capacitor is charged and uncharged unnecessarily during these cycles.

If the data is assumed as randomly distributed with equal probability, the Eq. (1) gives the number of wasted cycles. In the equation, m, n_r, n_c, n_{lut} represent the bit width of the operand, number of rows in the CAM, number of compared columns, and number of entries in the LUT of the corresponding operation. If the operation is multiplication (not linear time), the m in the formula must be replaced with m^2. According to the formula, 18.75% of the compare cycles are wasted for the in-place addition operations on m-bit, n numbers.

$$f_w(m, n_r, n_c, n_{lut}) = m \cdot \sum_{i=1}^{n_{lut}} \frac{m}{2^{n_c}} \cdot (n_{lut} - i) \quad (1)$$

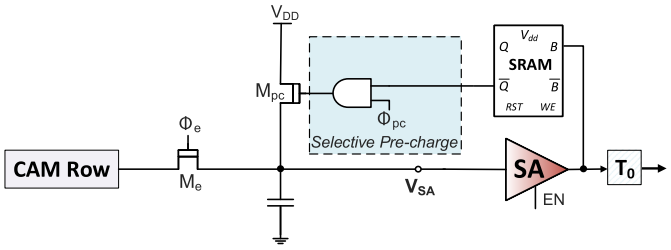
Fig. 5 shows the percentage of the wasted cycles when these operations are performed on the AP simulator detailed in Section 5 on 16-bit 1024K random number pairs. The results indicate that percentage of the wasted cycles are more than theoretical results since LUT tables cover only the part of all cases. According to the figure, the unnecessary cycles range from 11.43% to 77.79% with a mean of 36.45%.

In addition to them, some operations spent unnecessary cycles not within a LUT pass but also within a group of LUT passes (i.e., in a longer period). For example, the absolute value operation in the AP performs unnecessary pre-charge and evaluate cycles even though the input number is positive. Similarly, the multiplication operation in the AP wastes these cycles even if the multiplicand is 0. All these cases in the AP cause the high switching activity on the rows and lead to more energy consumption consequently.

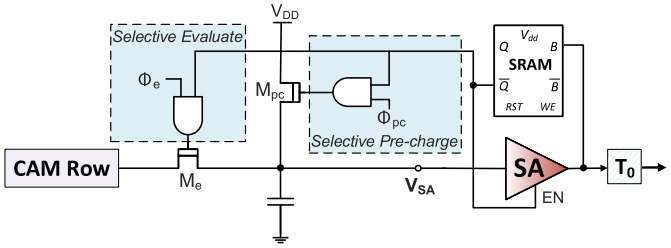
4. Low-power methodologies

4.1. Selective pre-charge

As discussed in Section 3, a considerable portion of the compare cycles are spent unnecessarily. This is because the row capacitors are pre-charged even though it is known as a priori that it is going to discharge definitely during the rest of the LUT pass. If a mechanism prevents these

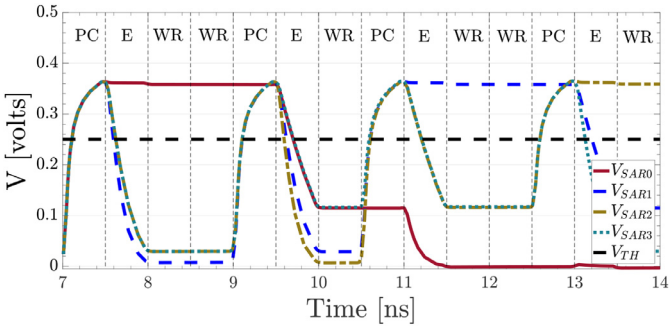


(a) Selective pre-charge mechanism.

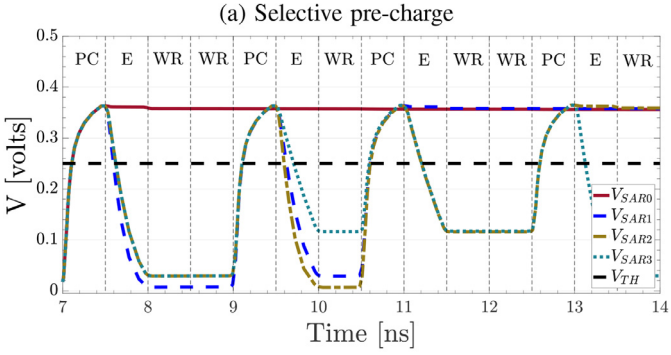


(b) Selective evaluate mechanism.

Fig. 7. Traditional AP row without any mechanism.



(a) Selective pre-charge



(b) Selective pre-charge & evaluate

Fig. 8. Selective pre-charge (a) and evaluate (b) mechanisms for low-power AP.

unnecessary cycles from occurring, the wasted cycles shown in Fig. 5 can be eliminated for energy saving. In order to accomplish this mechanism, these rows must be differentiated from the others by tagging them so that in the next cycle the pre-charge operation can be done selectively only on the untagged rows. These tagging operations can be done by using a single-bit memory cell (SRAM) placed in each row.

Fig. 7 shows the traditional row in an AP where the CAM row is directly connected to the pre-charge capacitor and the sense amplifier (SA). Fig. 8a shows a single row of an AP modified to facilitate the selective pre-charge mechanism (SPC). For tagging purpose, a single SRAM is added to the row. Initially, each SRAM stores the logic-0 value which means that these rows are the candidate for a possible match. Within a LUT pass, if a row is matched, the value of SRAM is changed as logic-1.

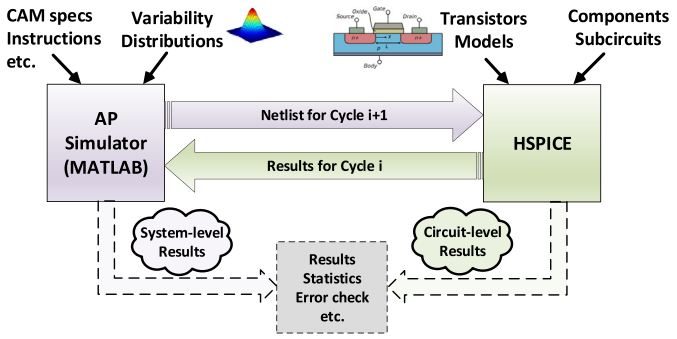


Fig. 9. Waveform of single-bit addition which corresponds to the second row of Fig. 4 when selective pre-charge (a) and evaluate (b) mechanisms enabled.

A logic-1 value stored in the SRAM indicates that this row is matched in any of the previous cycles within a LUT pass and it is not possible that it will match again within the current LUT pass. The value of the SRAM is kept until the end of the LUT pass to exclude the corresponding row from the pre-charge operations. At the end of each LUT pass, the SRAMs in the rows are reset to convert the all values to logic-0. In the figure, it is shown that the circuit also has modifications in the input of the pre-charge switch transistor (M_{pc}). Instead of directly controlling the switch by Q_{pc} signal coming from the controller, the output of the SRAM is connected to an AND gate together with the Q_{pc} signal so that this transistor is controlled by both signals. Even though controller pulls the Q_{pc} signal up to logic-1 in the beginning of a compare cycle, the capacitor cannot be charged if SRAM holds a logic-1. In this way, wasted pre-charge cycles are avoided.

Fig. 9a shows the waveform of the operation previously shown in Fig. 6 when selective pre-charge mechanism is enabled. In the figure, the voltage across the row capacitance of the row 0 (V_{SAR0}) is matched in the first cycle and its SRAM register is set as logic-1. Since this register disables the pre-charging in the following cycles within a LUT pass, this capacitor cannot be charged in the following three pre-charge phases. In this way, the upcoming tree pre-charge cycles are avoided and the energy is saved. At the end of the LUT pass, this SRAM register is reset after the forth compare (13 ns). The same situation applies to the second row (row 1) after the third pre-charge.

4.2. Selective evaluate

Selective pre-charge is an effective method in avoiding unnecessary pre-charge operations. On the other hand, the charge across the capacitor leaks in the following evaluate cycles unnecessarily without preceding pre-charges. One improvement in addition to selective pre-charge can be *selective evaluate (SE)* where evaluate cycles can be performed selectively as well so that the charge across the capacitor cannot be lost and is locked in the capacitor for the next LUT pass. For the architecture to support this mechanism, the same SRAM from selective pre-charge mechanism can be used for this purpose, however, the circuit needs another AND gate at the input of the evaluate transistor (M_e). This AND gate prevents the unnecessary evaluate operations in the row which are already matched in one of the previous cycle. Fig. 8b shows the modified architecture to enable selective evaluate together with selective pre-charge. The combination of these two methods is called as *selective compare (SC)* since a compare operation consists of pre-charge and evaluate phases. In the overall, the selective compare method requires an additional two AND gates and single SRAM cell.

Fig. 9b shows the waveform of the same operation in Fig. 6 where both selective pre-charge and evaluate mechanism enabled. The figure demonstrates that after a match, the following pre-charge cycles are disabled as well as the charge does not leak during the following evaluate cycles. On the other hand, in the upcoming evaluate cycles, the preserved charge across the capacitor is interpreted as logic-1 in the sense amplifier. In order to avoid this situation, the sense amplifier is also disabled if SRAM stores a logic-1 by connecting its output to the sense amplifier enable input (EN) (see Fig. 8b). In this way, the total energy consumption of the sense amplifiers can be decreased as well since unnecessary sensing operations are avoided like pre-charge and evaluate phases.

Table 2
Modified LUT for the multiplication .

Cr	*R*	*B*	*A*	*Cr*	*R*	*Comment*
X	X	X	0	–	–	Once
0	0	1	1	0	1	2nd Pass
0	1	1	1	1	0	1st Pass
1	0	0	1	0	1	3rd Pass
1	1	0	1	1	0	4th Pass

4.3. Modified LUTs

The methodology of selective compare requires an SRAM cell that is able to keep the one-moment history of the previous status. Instead of exploiting this opportunity in a LUT pass (i.e., short-term), this history can be used for a longer period for some AP operations. In this section, the same SRAM cell is used to lower the energy consumption further in the multiplication and absolute value operations by modifying their LUTs. This method is called as modified LUTs (ML).

4.3.1. Multiplication

For the unsigned multiplication in the AP ($R = A \times B$), the LUT is applied to all bits of B for each bit of A. Indeed, this table performs the addition operation between B and R if the A's bit is logic-1. On the other hand, if A's bit is logic-0, the partial addition operation is still done even though it has no effect on the results ($R = R + 0$) and all the cycles are wasted. In here instead of enabling selective compare mechanism in fine-grain, it can be employed in the coarse grain to get more advantage from it. In other words, instead of keeping the history during a single LUT pass (i.e., addition of single bit), the history can be kept linger during the multiplication between the single bit of A and the B (i.e., many LUT passes).

Table 2 shows the modified LUT for the low-power multiplication. The table adds an additional LUT entry which looks for a logic-0 in the column of A. This compare operation is performed once at the beginning of each partial addition (i.e., at the beginning of the inner loop). To illustrate, it is executed m times for the multiplication of two m-bit numbers where the total number of compare cycles is $4m^2$. If the A's bit is logic-0, this row is simply excluded from the following partial addition. However, during this operation selective compare cannot be performed as well since there is a single SRAM and it keeps the history whether A is logic-1 or 0 (not the history through the LUT pass). Since more rows are excluded by just checking single bit (average of 50% of the rows), it is expected to get more energy savings when compared to selective compare.

4.3.2. Absolute value

In the APs, the absolute value operation is simply performed by performing 2's complement on negative numbers and copying positive numbers directly. However, while performing 2's complement, the compare cycles spent on the rows having positive numbers are unnecessarily wasted. For this reason, the positive and negative numbers can be discriminated and the separate LUTs can be defined for each of them. In this way, while performing the operation on positive numbers (i.e., copy operation), the negative numbers are excluded and on negative numbers (i.e., the 2's complement operation), the positive numbers are excluded. Table 3 shows the modified LUT table. During the operation, first a logic-0 is searched on the sign bit of the numbers to exclude the

Table 3
LUT for absolute value .

A	*R*	*Comment*	*Flag*	*A*	*Flag*	*R*	*Comment*
1	1	–	0	1	1	1	3rd Pass
			1	0	1	1	1st Pass
			1	1	1	0	2nd Pass

Table 4
Percentage of covered compare cycles in SC and ML .

	Selective compare	Modified LUTs
Absolute value	33.59%	46.86%
Multiplication	10.26%	48.45%

Table 5
Average energy and power results of the SAP .

	Energy	Time	Power
Compare (per row)	5.425 fJ	1ns	5.425μW
Write (per cell)	0.242 fJ	0.5ns	0.484μW
Static energy (per cell)	0.002 fJ	0.5ns	0.005μW

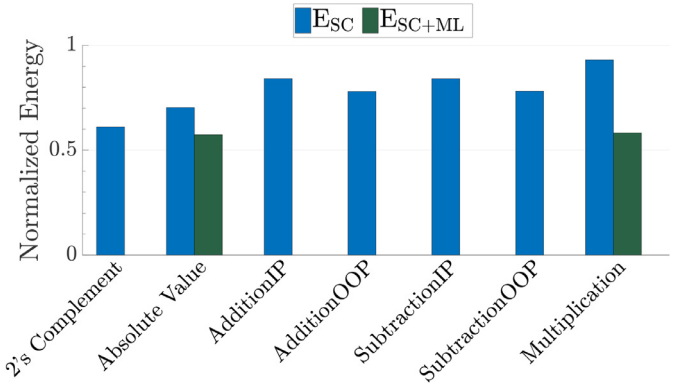


Fig. 10. The simulation framework.

negative numbers and then, copy operation (Table 3a) is performed on the positive numbers. After that, a logic-1 is searched on the sign bit to exclude the positive numbers and 2's complement operation is performed on the negative numbers.

Since the architecture supports only single-bit history, either SC or ML methods can be exploited in these operations, but not both. In order to decide on the better method, the percentages of the covered cycles by each method are presented in Table 4. The table shows that the ML method can cover more compare cycles that the selective compare method. On the other hand, the modified LUT is adding a small number of extra compare cycles while excluding many rows from the computation. The detailed analysis of both methods is revealed in Section 5.

5. Experimentation

5.1. Simulation framework

Fig. 10 shows the simulator framework for the AP architectures. The simulator can perform both system-level and circuit level simulations at the same time in Matlab and HSPICE respectively. As parameters, the simulator accepts circuit models (i.e., transistor), sub-circuits (e.g., sense amplifiers), CAM features (width and height), sweep parameters, initial data, and the instructions. Then MATLAB part of the simulator creates netlists that iteratively drive the HSPICE simulator. For the transistor model, the Predictive Technology Models (PTM) [18] is preferred to simulate high-density memories with 16nm feature sizes [19]. Performance metrics and statistics are obtained by cross-checking the output of both Matlab and HSpice simulations. For the sense amplifier, a low-power, sub-ns amplifier design in [20] is employed in the circuit.

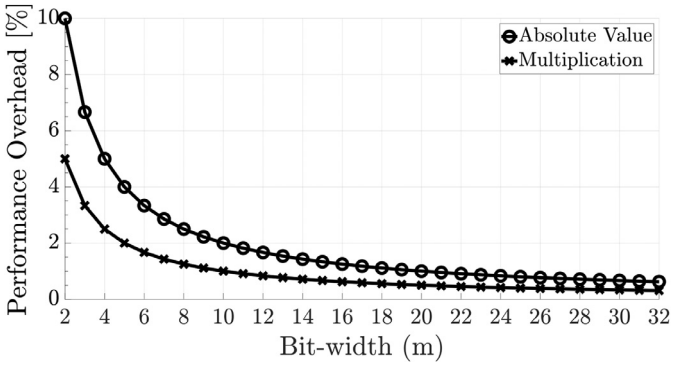


Fig. 11. Energy reduction in arithmetic operations when selective compare and modified LUTs are enabled.

5.2. Arithmetic operations

Fig. 11 shows the normalized energy consumption of the fundamental arithmetic operations on the low-power SAP. The results are obtained by performing the corresponding operations on 16-bit 1 M (2^{20}) numbers (in 2's complement and absolute value) or number pairs (in others). The figure shows energy results of the operations where SC and ML methods are applied. The reported results are normalized with respect to the energy consumption of each operation without any low-power methodology to fit them into the common scale. For absolute value and multiplication, the normalized energy consumptions of ML method are shown as well. All low-power SAP results include the energy overhead of the additional components such as AND gates, SRAMs, extra cycles due to modified LUTs, etc.

According to the figure, the energy reductions obtained from the selective compare methods ranging from 6.95% in multiplication to 38.92% in 2's complement with a mean of 21.58%. On the other hand, the modified LUTs method provides a reduction of 42.59% and 41.74% in absolute value and multiplication even though selective compare results in 29.67% and 6.95% savings respectively. It is proven that exploiting the single-bit history in a longer-term provides more energy saving in these operations. On the other hand, this method affects the performance since it inserts additional compare cycles during the execution. Fig. 12 shows this overhead percentage when the bit-width of the operands ranges from 2 to 32. The performance overhead decreases as the bit-width of the operands increases. For a traditional bit-width of 16, the performance overheads are less than 2%. When compared with energy reduction, this performance overhead is negligible and can be ignored for the sake of energy savings.

5.3. Benchmarks

For the evaluation of proposed low-power AP for the real cases, seven benchmarks from different domains are implemented on the AP. Table 6 shows these benchmarks, their inputs, and parameters during the evaluation. Both selective compare and modified LUTs methods are evaluated on these benchmarks. During the runs, AP is configured according to the corresponding benchmark, so in some cases, more than one AP is pipelined to each other. For example, a 1K FFT requires 10 AP stages. Since the circuit simulation time takes days for the successive iterations of the computationally intensive benchmarks (e.g., 10s thousand cycles for FFT), first, the accurate execution profile (dynamic & static energy, time, match & mismatch statistics, etc.) are obtained from the simulator (Fig. 10), and later these numbers are used in the Matlab simulator to get the precise results.

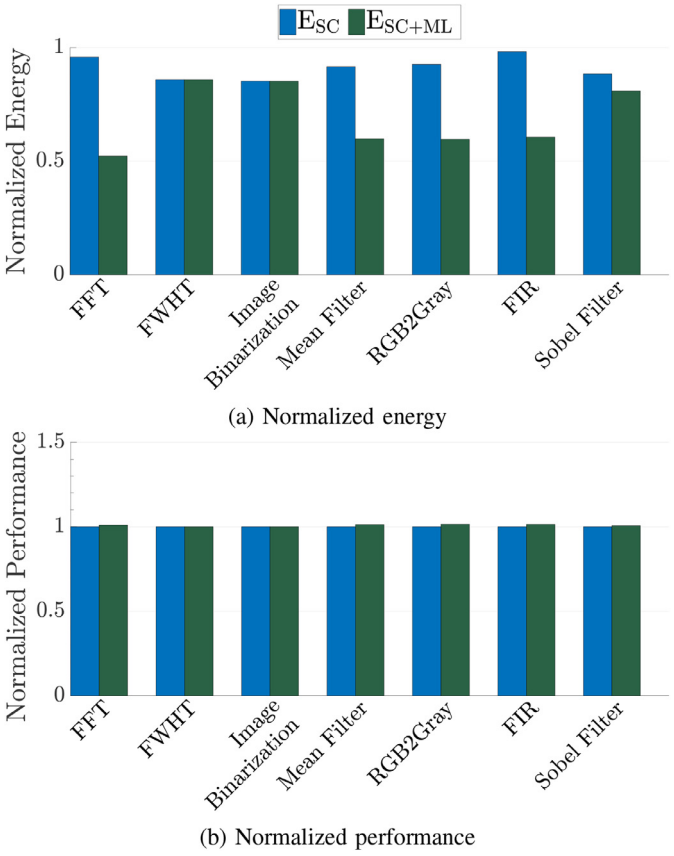


Fig. 12. The performance overhead in 2's complement and multiplication due to the modified LUTs.

Table 6
The evaluated benchmarks and their input sizes .

Benchmark	Domain	Parameters & Input
Sobel Filter	Image processing	512x512 gray image
FIR	Signal processing	8-tap, 512 x 8-bit integers
FFT	Signal processing	1K 16-bit complex numbers
Image Binarization	Image processing	512x512 gray image
RGB2Gray	Image processing	384x512 color image
FastWalsh	Signal processing	256x256 gray image
Mean Filter	Machine vision	512x512 gray image

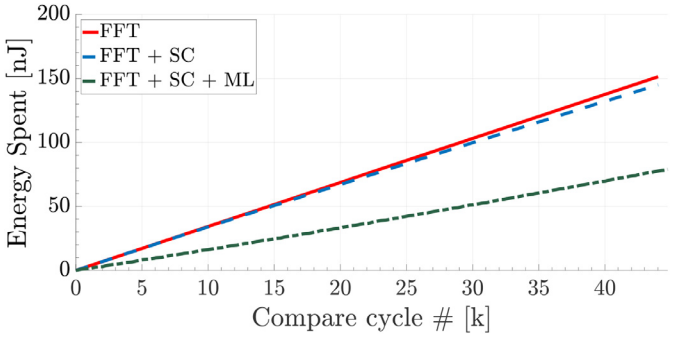


Fig. 13. Normalized energy consumption (a) and performance (b) results of the benchmarks when selective compare and modified LUTs are enabled.

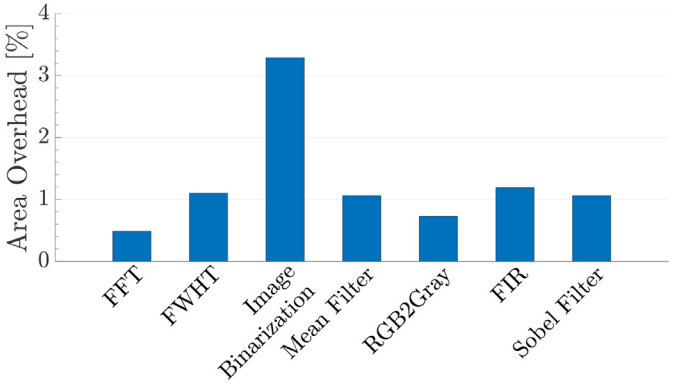


Fig. 14. Energy consumption during the FFT benchmark runs of all three cases.

Algorithm 1: In-place subtraction ($\mathbf{B} = \mathbf{B} - \mathbf{A}$) in the AP

```
1  function SubtractionIP(Br, B.MSB, B.LSB, A.MSB, A.LSB)
2    for i = A.LSB to A.MSB
3      for pass = 1 to 4
4        SetMask(Br, B.i, A.i);
5        SetKey(LUT_SubIP(pass), Br, B.i, A.i);
6        Compare(Mask, Key);
7        Write(LUT_SubIP(pass), Br, B(i));
8      end
9    end
10 end
```

Fig. 15. Area overhead of the benchmarks when selective compare and modified LUTs are enabled.

Fig. 13 shows the normalized energy consumption and performance results of each benchmark for both methods. If the benchmark includes either multiplication or absolute value, it becomes possible to reduce the energy consumption further by modified LUTs. For example, even though the normalized energy of the FFT is reduced by 4.16% when selective compare method is used, a 47.77% improvement is possible when modified LUT is enabled for absolute value and multiplication. The figure shows that exploiting modified LUTs can provide up to 45.51% more savings on top of selective compare. The one reason for this huge saving is that at the first stages of the FFT, the twiddle factors includes lots of zeros so the modified LUT provides an excessive saving on these multiplications. For instance, all twiddle factors are $1+0i$ during the first butterfly stage of an FFT operation. When the data is represented in 16 bits, this corresponds to a single logic-1 in the first bit and all logic-0s in the rest 15 bits for the real part and all logic-0s for the imaginary part. The modified LUT for multiplication in Table 2 handles this case very efficiently where the rows with logic-0 are excluded from the multiplication which in turn provides a considerable energy consumption. The similar benefit is obtained from the absolute value operation as well since it is also used for signed multiplication. In other benchmarks, the energy savings range from 14% to 40%. According to the Fig. 13b, the selective compare methodology has no effect on the performance since the method uses the same LUTs together with the selective compare circuit. On the other hand, the modified LUTs has a minimal effect on the performance between 0.6% (in Sobel Filter) to 1.5% (in RGB2Gray), however it provides 19.1% and 40.4% energy reduction for these benchmarks respectively. Fig. 14 shows the energy trace comparison between the normal AP and the low-power alternatives. The figure shows the trend of the FFT benchmark since it includes nearly all instructions in Table 1. The reported energy is sensed at every compare cycle. The figure shows that the ML follows a much lower energy consumption trend.

As stated in Section 4, the proposed low-power methodologies cause area overhead in the circuit by inserting AND gates and SRAM cells. The Fig. 15 shows these overheads for each benchmark when the AP architecture is configured specifically for these benchmarks. According to the results, the area overhead is between 0.49% in FFT and 3.29% in image binarization with an average of 1.27%. The FFT processor requires a 121×512 cell array together with additional matching circuits in each row, so adding two AND gates and a single SRAM cell does not cause

Table 7
Comparison with other sub-65nm ASIC implementations of FFT .

	Technology	Size (*points*)	Word **width** (*bits*)	Area (mm^2)	Effective **throughput** (MS/s)	Normalized **area efficiency** ($GS/s/mm^2$)	Normalized **power efficiency** ($GS/s/W$)	Normalized FoM ($GS/s/W/mm^2$)
AP	16 nm	2048	16	0.096	268	3.74	7.84	81.35
AP+SC + ML	16 nm	2048	16	0.097	268	3.68	15.66	161.58
AP+SC + ML (VoS)	16 nm	2048	16	0.097	268	3.68	25.08	258.83
[21]	65 nm	1024	16	8.29	240	0.16	129.14	116.86
[22]	45 nm	2048	32	0.97	0.22	0.002	0.25	3.99
[23]	65 nm	2048	12	1.38	20	0.06	8.97	80.79

a significant area overhead. The overhead of the image binarization is the most because it requires relatively smaller CAM (i.e., 17 cells/row). On the other hand, it is explicit that the energy reduction is much more than all these numbers.

5.4. Figure of merit

The comparison of AP-based implementation of FFT processors with the traditional sub-65 nm approaches is shown in Table 7. The table includes three versions of AP implementation of 2K 16-bit FFT where the first one corresponds to the original implementation, the second one exploits the SC and ML together. The third one corresponds to the case where voltage over scaling is applied to the CAM cells (i.e., core voltage is decreased 10%) and noise margin of the compare operations is narrowed as a result. The table shows the normalized area, power efficiency numbers and a figure of merit in terms of throughput over power density. For a fair comparison, the respective numbers are normalized according to Eqs. (2)–(4) where N corresponds to the FFT size. As shown in the table, AP provides tremendous gain in terms of area efficiency. The gains are shown in both the absolute area numbers as well as the normalized area efficiency as measured in $GS/s/mm^2$. In terms of normalized power efficiency measured in $GS/s/W$, the low-power AP architecture increases the efficiency by 2x making it the second best where it had been third. Finally, in terms of the Figure of Merit (FoM) chosen as the normalized power efficiency per unit area $GS/s/W/mm^2$ where the low-power AP outperforms all other architectures whereas the normal one is not.

$$Normalized\ Area = \frac{Area}{\left(\frac{Tech}{16\ nm}\right)^2 \cdot \left(\frac{Wordlength}{16}\right) \cdot \left(\frac{N \cdot log_2 N}{11 \times 2^{11}}\right)} \quad (2)$$

$$Normalized\ Power = \frac{Power}{\left(\frac{Tech}{16\ nm}\right) \cdot \left(\frac{Wordlength}{16}\right) \cdot \left(\frac{V_{DD}}{0.7\ V}\right)^2 \cdot \left(\frac{N \cdot log_2 N}{11 \times 2^{11}}\right)} \quad (3)$$

$$Normalized\ Throughput = \frac{Throughput}{\left(\frac{16\ nm}{Tech}\right) \cdot \left(\frac{16}{Wordlength}\right) \cdot \left(\frac{N}{2^{11}}\right)} \quad (4)$$

6. Conclusion

In this study, the energy consumption of the associative processors is decreased by power optimization techniques. The proposed techniques decreases the energy consumption considerably by preventing the unnecessary compare cycles which have no effect on the correctness of the operations. The techniques include both architectural (selective compare) and instructional (modification in look-up tables) changes in the architecture with a negligible area and performance degradation. The evaluation section shows that both methods provide a considerable energy savings. For the future work, the other low-power techniques for the general CAM architectural techniques can be applied to AP context and a detailed comparison between the in-memory and traditional processors can be presented.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.sysarc.2018.08.006 .

References

[1] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, A.R. LeBlanc, Design of ion-implanted mosfet's with very small physical dimensions, IEEE J. Solid-State Circ. 9 (5) (1974) 256–268, doi:10.1109/JSSC.1974.1050511.

[2] M. Pant, Microprocessor Power Impacts, 2010.

[3] D.C. Cireşan, U. Meier, J. Masci, L.M. Gambardella, J. Schmidhuber, Flexible, high performance convolutional neural networks for image classification, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two, in: IJCAI'11, AAAI Press, 2011, pp. 1237–1242, doi:10.5591/978-1-57735-516-8/IJCAI11-210.

[4] C.C. Foster, Content Addressable Parallel Processors, John Wiley & Sons, Inc., New York, NY, USA, 1976.

[5] J.L. Potter, Associative Computing: A Programming Paradigm for Massively Parallel Computers, Perseus Publishing, 1991.

[6] L. Yavits, A. Morad, R. Ginosar, Computer architecture with associative processor replacing last-level cache and SIMD accelerator, IEEE Trans. Comput. 64 (2) (2015) 368–381, doi:10.1109/TC.2013.220.

[7] H.E. Yantir, A.M. Eltawil, F.J. Kurdahi, A two-dimensional associative processor, IEEE Trans. Very Large Scale Integr. VLSI Syst. (2018) 1–12, doi:10.1109/TVLSI.2018.2827262.

[8] L. Yavits, S. Kvatinsky, A. Morad, R. Ginosar, Resistive associative processor, IEEE Comput. Archit. Lett. 14 (2) (2015) 148–151, doi:10.1109/LCA.2014.2374597.

[9] J. Li, R.K. Montoye, M. Ishii, L. Chang, 1 mb 0.41 mm^2; 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing, IEEE J Solid-State Circ. 49 (4) (2014) 896–907, doi:10.1109/JSSC.2013.2292055.

[10] Q. Guo, X. Guo, R. Patel, E. Ipek, E.G. Friedman, Ac-dimm: associative computing with stt-mram, in: Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA '13, ACM, New York, NY, USA, 2013, pp. 189–200, doi:10.1145/2485922.2485939.

[11] E. Ipek, Q. Guo, X. Guo, Y. Bai, Resistive Memories in Associative Computing, Springer New York, New York, NY, pp. 201–229. 10.1007/978-1-4419-9551-3_8.

[12] S. Khasanvis, M. Rahman, C.A. Moritz, Heterogeneous graphenecmos ternary content addressable memory, J. Parallel Distrib. Comput. 74 (6) (2014) 2497–2503, doi:10.1016/j.jpdc.2013.08.002. Computing with Future Nanotechnology

[13] Y. Halawani, B. Mohammad, D. Homouz, M. Al-Qutayri, H. Saleh, Modeling and optimization of memristor and stt-ram-based memory for low-power applications, IEEE Trans. Very Large Scale Integr. VLSI Syst. 24 (3) (2016) 1003–1014, doi:10.1109/TVLSI.2015.2440392.

[14] H.E. Yantir, A.M. Eltawil, F.J. Kurdahi, Approximate memristive in-memory computing, ACM Trans. Embed. Comput. Syst. 16 (5s) (2017) 129:1–129:18, doi:10.1145/3126526.

[15] M. Imani, D. Peroni, T. Rosing, Nvalt: non-volatile approximate lookup table for GPU acceleration, IEEE Embed. Syst. Lett. PP (99) (2017) 1, doi:10.1109/LES.2017.2746742.

[16] F. Parveen, S. Angizi, Z. He, D. Fan, Low power in-memory computing based on dual-mode sot-mram, in: 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2017, pp. 1–6, doi:10.1109/ISLPED.2017.8009200.

[17] M. Imani, S. Gupta, T. Rosing, Ultra-efficient processing in-memory for data intensive applications, in: Proceedings of the 54th Annual Design Automation Conference 2017, in: DAC '17, ACM, New York, NY, USA, 2017, pp. 6:1–6:6, doi:10.1145/3061639.3062337.

[18] A.S. University, Predictive Technology Model (ptm), 2012.

[19] S. Sinha, G. Yeric, V. Chandra, B. Cline, Y. Cao, Exploring sub-20nm finfet design with predictive technology models, in: DAC Design Automation Conference 2012, 2012, pp. 283–288, doi:10.1145/2228360.2228414.

[20] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, B. Nauta, A double-tail latch-type voltage sense amplifier with 18ps setup + hold time, in: 2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, 2007, pp. 314–605, doi:10.1109/ISSCC.2007.373420.

[21] M. Seok, D. Jeon, C. Chakrabarti, D. Blaauw, D. Sylvester, A 0.27v 30mhz 17.7nj/transform 1024-pt complex fft core with super-pipelining, in: 2011

IEEE International Solid-State Circuits Conference, 2011, pp. 342–344, doi:10.1109/ISSCC.2011.5746346.
[22] A.S. Beulet Paul, S. Raju, R. Janakiraman, Low power reconfigurable fp-fft core with an array of folded da butterflies, EURASIP J. Adv. Signal Process. 2014 (1) (2014) 144, doi:10.1186/1687-6180-2014-144.
[23] C.H. Yang, T.H. Yu, D. Markovic, Power and area minimization of reconfigurable fft processors: a 3gpp-lte example, IEEE J. Solid-State Circ. 47 (3) (2012) 757–768, doi:10.1109/JSSC.2011.2176163.