CCF Transactions on High Performance Computing (2020) 2:272–281
https://doi.org/10.1007/s42514-020-00038-5

REGULAR PAPER

An STT-MRAM based reconfigurable computing-in-memory architecture for general purpose computing

Yu Pan[1] · Xiaotao Jia[1,2] · Zhen Cheng[1,2] · Peng Ouyang[1] · Xueyan Wang[1] · Jianlei Yang[2,3] · Weisheng Zhao[1,2]

Received: 21 February 2020 / Accepted: 27 May 2020 / Published online: 29 July 2020

Abstract
Recently, many researches have proposed computing-in-memory architectures trying to solve von Neumann bottleneck issue. Most of the proposed architectures can only perform some application-specific logic functions. However, the scheme that supports general purpose computing is more meaningful for the complete realization of in-memory computing. A reconfigurable computing-in-memory architecture for general purpose computing based on STT-MRAM (GCIM) is proposed in this paper. The proposed GCIM could significantly reduce the energy consumption of data transformation and effectively process both fix-point calculation and float-point calculation in parallel. In our design, the STT-MRAM array is divided into four subarrays in order to achieve the reconfigurability. With a specified array connector, the four subarrays can work independently at the same time or work together as a whole array. The proposed architecture is evaluated using Cadence Virtuoso. The simulation results show that the proposed architecture consumes less energy when performing fix-point or float-point operations.

Keywords Computing-in-memory · Reconfigurable · STT-MRAM · General purpose computing

1 Introduction

Over the past decades, as the size of data scale growing exponentially with time, the computational demands for data analytics applications are becoming even more forbidding. However, regarding conventional von Neumann architecture, the overhead of the data communication between the processor and the memory units results in huge performance degradation and energy consumption, that called von Neumann bottleneck (Kim et al. 2003; Wulf and McKee 1995). For example, when compares the cost of computation (a double-precision fused multiply add) with communication (a 64-bit read from an off-chip SRAM), the ratio of communication energy to computation energy is 50 X at 40 nm (Keckler et al. 2011). More serious a DRAM access consumes 200 times more energy than a floating-point operation (Kang et al. 2020). Such off-chip accesses become increasingly necessary as data scale grow larger, and even the cleverest latency-hiding techniques cannot conceal their overhead.

In order to overcome this bottleneck, a hopeful approach is to embed processing capability into the memory, termed computing-in-memory (CIM). Several in-memory computing schemes have been proposed by some studies based on SRAM and DRAM (Wang et al. 2019; Biswas and Chandrakasan 2018; Li et al. 2017; Gao et al. 2019). The data communication overhead has been greatly reduced. However, as volatile memory, the static power consumption of SRAM and DRAM has become an important factor affecting its performance. This makes it difficult to meet the ultra-low power consumption demand of in future big-data-based application scenarios.

Recently, a lot of studies have demonstrated that non-volatile memories (NVMs), such as resistive random access memory (RRAM) (Fujiki et al. 2018; Imani et al. 2019; Cheng et al. 2018; Cai et al. 2019; Liu et al.

✉ Xiaotao Jia
jiaxt@buaa.edu.cn

✉ Weisheng Zhao
weisheng.zhao@buaa.edu.cn

1 Beijing Advanced Innovation Certer for Big Data and Brain Computing, School of Microelectronics, Fert Beijing Research Institute, Beihang University, Beijing 100191, China

2 Beihang-Goertek Joint Microelectronics Institute, Qingdao Research Institute, Beihang University, Qingdao 266101, China

3 Beijing Advanced Innovation Certer for Big Data and Brain Computing, School of Computer Science and Engineering, Fert Beijing Institute, Beihang University, Beijing 100191, China

2013), phase change memory (PCM) (Li et al. 2016), and spin transfer torque-based magnetoresistive RAM, (STT-MRAM) (Kang et al. 2015, 2017, 2020; Deng et al. 2013; Jain et al. 2017; Zhang et al. 2019; Chowdhury et al. 2017; Chen and Wang 2009; Zabihi et al. 2018), can provide inherent logical computing capabilities that enable efficient logical computing to be embedded in memory as their resistance-based storage mechanisms. At the same time, these non-volatile memory devices have the advantages of non-volatile, low power consumption and high density. These advantages make it possible for CIM architectures based on non-volatile memory to fundamentally solve the von Neumann bottleneck problem.

Most of the proposed NVM-based CIM architectures can only perform some application-specific logic functions. For example, design in (Deng et al. 2013) can only support Boolean logic operation and simple 1-bit fulladder, some studies take advantage of the crossbar structure of RRAM to achieve matrix vector multiplication efficiently. Research (Fujiki et al. 2018) said they designed a general-purpose processor based on RRAM, but this scheme could only support fixed-point computation and was not applicable to the application of float-point computation. Further, the architectures proposed in those works do not support reconfigurable computing which limits their deployment in many problems. Neural network has more than fixed point operation and logic operation, so a general computing-in-memory architecture is more meaningful.

In this paper a reconfigurable CIM architecture for general purpose computing (GCIM) is proposed based on STT-MRAM. The contributions of the work are summarized as follows:

- An STT-MRAM array based architecture is proposed to support data-parallel in-memory computing. In the proposed architecture, the memory array acts as the role of both data storage and data processor. Thus it could resolve the von Neumann bottleneck issue that significant reduce the energy consumption.
- Simple but efficient peripheral circuits are designed based on memory unit circuits that make our architecture supporting general purpose computing of both fix-point addition/multiplication and float-point addition/multiplication.
- The proposed architecture is reconfigurable. The whole memory array is divided into some subarrays. With a specified array connector, the subarrays can work independently to finish different tasks at the same time or work together to finish one task.
- The GCIM architecture is evaluated by Cadence Virtuoso. The simulation results show that the proposed architecture consumes less energy when performing fix-point or float-point operations.

2 Background

2.1 Magnetic tunnel junction

Figure 1 is a typical magnetic tunnel junction with perpendicular magnetic anisotropy (PMA-MTJ). It is composed of three layers: two ferromagnetic layers (CoFeB), and they are separated by intermediate oxidation layer (MgO). One of the ferromagnetic layers is reference layer whose magnetization direction is fixed. The other ferromagnetic layer is free layer whose magnetization direction could be anti-parallel or parallel with that of reference layer. If the magnetization direction of the free layer is parallel with the reference layer, MTJ presents a low resistivity state (P), and the resistance value is represented by R_P, representing the binary data "0". In contrast, if the magnetization direction of the free layer is anti-parallel with the reference layer, MTJ presents a high impedance state (AP), and the resistance is represented by R_{AP}, representing the binary data "1". The characteristics of the resistance difference of the MTJ in two states can be described by the tunneling magneto-resistance ratio (TMR): $\mathrm{TMR} = (R_{AP} - R_P)/R_P$. MTJ state can be flipped by applying a polarized current injection with spin transfer torque (STT) mechanism. The MTJ state is switched from P state to AP state if the injected current ($I_{P\rightarrow AP}$) flows through the MTJ from free layer to reference layer. On the contrary, the MTJ state is switched from AP state to P state if $I_{AP\rightarrow P}$ is injected. If the current through the MTJ is greater than its critical reverse current I_{C0}, after a period of time, its magnetization direction will change with a probability as Fig. 2 shows (Vincent et al. 2015).

2.2 In-memory computing mechanism based on 2T1MTJ STT-MRAM array

Figure 3a is a 2T1MTJ memory unit consists of two NMOSs and one MTJ (Chowdhury et al. 2017). SL is used in both memory mode and computing mode. MBL and MWL are used in memory mode, CL and CBL are used in computing mode. In memory mode, the red part does not work while

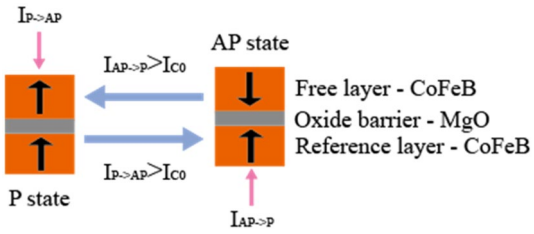


Fig. 1 Perpendicular magnetic anisotropy magnetic tunnel junction (PMA-MTJ)

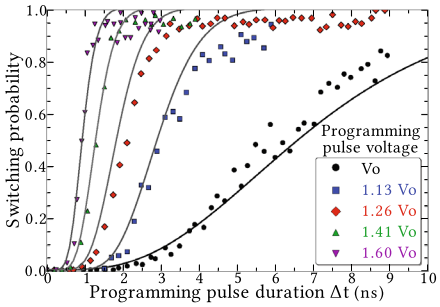


Fig. 2 Experimental measurements of the switching probability for different programming voltages with respect to the duration of the applied programming pulse (Vincent et al. 2015)

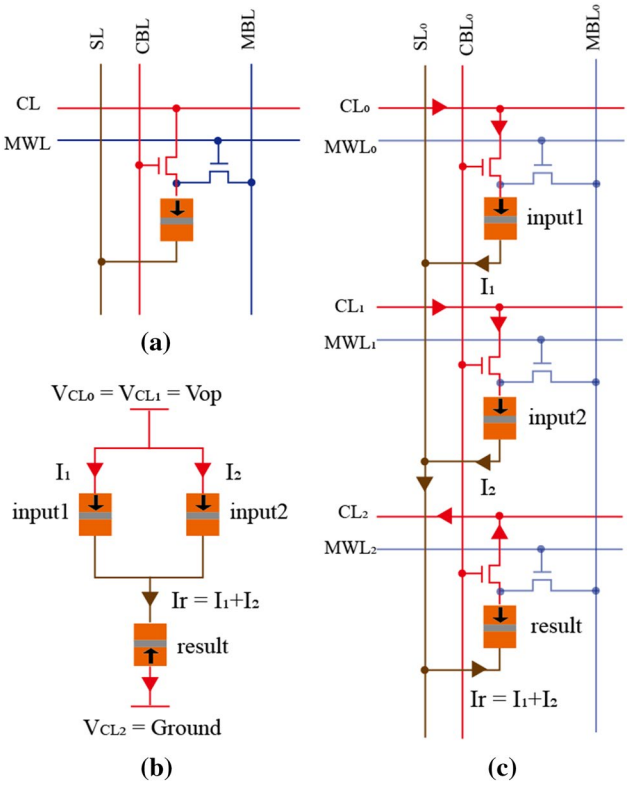


Fig. 3 **a** 2T1MTJ memory unit. The red part works in computing mode while the blue part works in memory mode. The brown line works in both computing and memory mode. **b** Simplified diagram of two-input logic computing. **c** Schematic diagram of two-input logic computing

the blue part dose. According to the MTJ inversion mechanism described in the previous section, the resistance state of MTJ can be changed by adding a suitable bias voltage between SL and MBL, that is, write data "0" or "1". SL and MBL are connected to pre-charged sensing amplifier (PCSA) shown in Fig. 4f (Zhao et al. 2009). The resistance of reference MTJ in PCSA is $Rref = (R_P + R_{AP})/2$. It can sense the resistance state of the MTJ connected to SL and MBL and read out the data from $\boldsymbol{Q_m}$. The inverse of the data can be output from $\overline{\boldsymbol{Q_m}}$ too.

Figure 3c is a schematic diagram of two input logic computing. And its simplified figure is shown in Fig. 3b. The MTJ of the two input units *input1* and *input2* are parallel, and then connected in series with the MTJ of the result unit. In computing mode, the blue part in Fig. 3c does not work while the red part dose. CL_0 and CL_1 of the two input cells *input1* and *input2* are given the same high-voltage V_{op} according to the calculation type, and the CL_2 voltage of the result unit is set to *gnd*. After a period of time, the computing result can be written into the result unit (the result unit is initialized to a low-resistance state before computing). Tables 1, 2 and 3 show the truth table and the voltage V_{op} of CL for *nor*, *nand* and *not* operation, respectively. This in-memory computing mechanism writing the result into memory array while computing. Then it can output the result by a regular read operation. It is different from some in-memory computing mechanisms that compute result by means of sense amplifier and output the result at the same time. The in-memory computing method used in this paper has an advantage in the computing of more complex computings that has lots of intermediate results involved in the subsequent computing, such as multi-layer neural network. Because it does not need to write intermediate results into the memory array so that it can directly carry out the subsequent computing.

3 Proposed computing-in-memory architecture

3.1 Architecture overview

Figure 4a is the reconfigurable CIM architecture for general purpose computing based on STT-MRAM (GCIM) proposed in this paper. It consists of 2T1MTJ STT-MRAM array, RD (row decoders), CD (column decoder), BL Dris (BL drivers), MDACs (modified DACs), PCSA (pre-charged sensing amplifier), Mser&Cor (modified shifter and connector), RSIC-V controller, parser and register. GCIM could significantly reduce the energy consumption of data transformation and effectively process both fix-point calculation and float-point calculation in parallel. It can work in both memory mode and computing mode simultaneously or non-simultaneously, which is reconfigurable. The detailed structures that make up GCIM is described in detail below.

STT-MRAM array: The array size can be configured based on the application requirements. The larger the array, the higher the design complexity and the power consumption, while the higher the computing capability. Indeed, the configuration of array size makes a tradeoff between the circuit power and computing capability. In this paper, we

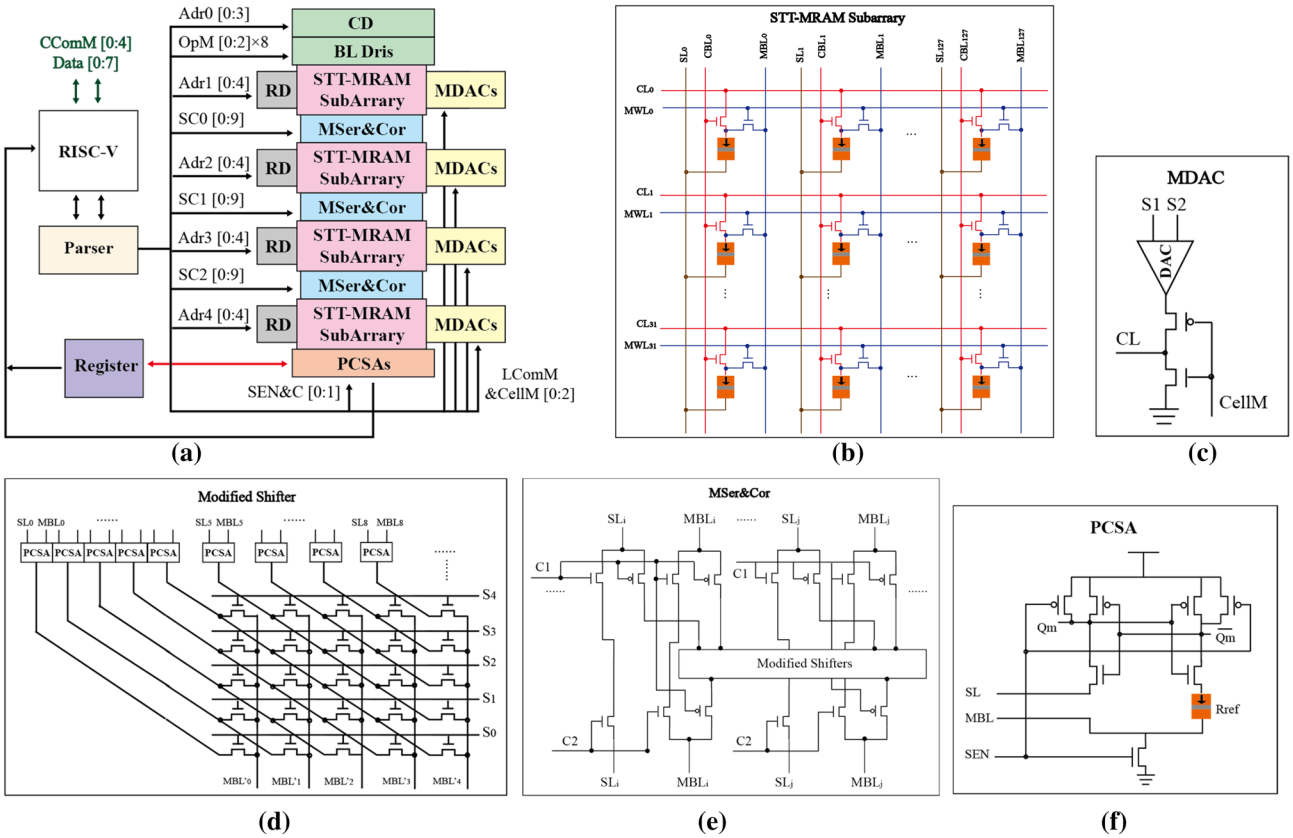


Fig. 4 **a** Proposed computing-in-memory-computing architecture. **b** 2T1MTJ STT-MRAM Subarrary. **c** Modified DAC that can output five voltage levels: *V*nor, *V*nand, *V*not, *V*min and ground. **d** Modified shifter. **e** Connector. **f** Pre-charged sensing amplifier circuit

Table 1 Truth table of *nor* operation

Input1	*Input2*	***nor (In1, In2)***	$V_{op}=V_{nor}$
0 (P)	0 (P)	1	$V_{nor}/[(R_P//R_P)+R_P] > I_{C0}$
0 (P)	1 (AP)	0	$V_{nor}/[(R_{AP}//R_P)+R_P] < I_{C0}$
1 (AP)	0 (P)	0	
1 (AP)	1 (AP)	0	

$R_A//R_B$ is parallel resistance of R_A and R_B, $R_A//R_B=(R_A{*}R_B)/(R_A+R_B)$

Table 2 Truth table of *nand* operation

Input1	*Input2*	***nand (In1, In2)***	$V_{op}=V_{nand}$
0 (P)	0 (P)	1	$V_{nand}/[(R_{AP}//R_P)+R_P] > I_{C0}$
0 (P)	1 (AP)	1	
1 (AP)	0 (P)	1	
1 (AP)	1 (AP)	0	$V_{nand}/[(R_{AP}//R_{AP})+R_P] < I_{C0}$

$R_A//R_B$ is parallel resistance of R_A and R_B, $R_A//R_B=(R_A{*}R_B)/(R_A+R_B)$

Table 3 Truth table of *not* operation

Input1	***not (In1)***	$V_{op}=V_{not}$
0 (P)	1	$V_{not}/(R_P+R_P) > I_{C0}$
0 (P)	1	
1 (AP)	0	$V_{not}/(R_{AP}+R_P) < I_{C0}$
1 (AP)	0	

chose a 128 × 128 array size to evaluate the architecture. In previous works, the STT-MRAM array is usually treated as a whole unit which makes it able to handle high complex computation but limits its reconfigurability. In our design, it is divided into four 32 × 128 subarrarys as shown in Fig. 4b. Thanks to the division of four subarrays, GCIM can support shift operation between two subarrarys based on the modified shifters. Furthermore, it is beneficial to have stronger parallelism and reconfigurability with the proposed connector as described below.

Modified shifter and connector: As you can see, there is a modified shifter and connector between two adjacent subarrays. The modified shifter consists of a barrel shifter and PCSAs as shown in Fig. 4d and it allows to shift data

between two subarrays. Figure 4e shows the connector that can regulate the connection between two subarrays. They can work in the following three states reconfigurability according to its input data: (1) the modified shifter is connected between the two subarrays to perform shift operations, (2) two adjacent subarrays are connected and work like an array, (3) the two subarrays are independent to each other and work separately.

Row decoder: The output of row decoders is connected to MWLs and its function is the same as the row decoders used in conventional memories. In order to support shift operation and leverage underlying parallelism, each subarray is equipped with a 5–32 row decoder in this design.

Column decoder: In order to support 8-bit computation, the 128 columns of the memory array are divided into 16 groups. Each group consists of eight columns. In our architecture, a 4–16 column decoder is used to enable eight adjacent columns simultaneously to support 8-bit computations.

BL driver: BL drivers provide corresponding voltage for SLs, WBLs and CBLs based on operation mode like computing, reading, writing "0" and writing "1".

Modified DAC: It consists of a 2-bit DAC and two MOSs. The output of modified DACs is connected to CWLs as shown in Fig. 4c. It can provide five distinguished voltage levels for CWLs: *V*nor, *V*nand, *V*not, *V*min and *Gnd*.

PCSA: PCSAs as shown in Fig. 4f are used for read operations as described in the previous section.

RSIC-V controller: This kind of connection described above increases the reconfigurability and parallelism of the architecture. But more powerful controller is needed to support its reconfigurability and parallelism, therefore a RSIC-V is used to control the data flow and control the above devices to work in order under different operation needs.

Parser: The parser parses the commands of RSIC-V and sends them to each device separately.

Register: Registers are used in the judgment steps of complex computations (e.g., floating-point fulladder).

The arrows in Fig. 4a represent the data flow of GCIM. The green arrows are the communication with outside. *CComM [0:4]* (complex computing mode) tells RSIC-V what kind of computing will do next, like floating-point multiplication, fixed-point multiplication, floating-point fulladder, etc. *Data [0:7]* tells RSIC-V the 8-bit data that need to be stored. Black arrows represent the communication between internal devices and RSIC-V. *Adr0 [0:3]* (Address0) is 4-bit-input of column decoder and *Adri* (Addressi), i = 1, 2, 3 or 4, is 5-bit-input of row decoders. *OpM [0:2]* (Operation mode) is 3-bit-singnal to BL driver. *SCi [0:9]* (shifter and connector), i = 1, 2, 3 or 4, is 10-bit input signal to Mser&Cors and the lower 8 bits are inputs for shifter (S1–S8 in Fig. 4d) while the high two bits are inputs for connector (C1 and C2 in Fig. 4e). *LComM&CellM [0:2]* (logic computing mode and cell mode) is 3-bit input to MDAC (2 bits for L1 and L2, and 1 bit for CellM in Fig. 4c). And 1-bit-SEN is for PCSA. Red arrow is the communication between PCSAs and register.

3.2 General computing methods based on the proposed architecture

3.2.1 Full adder and ripple-carry adder

GCIM can work as a full adder (FA) based on Eqs. (1) and (2). It figures out carry *Cout* in two steps (*MIN* operation and *not* operation) and needs two *xnor* operations to get sum *S*. Table 4 shows the process of computing R = *xnor* (A, B) which needs three steps. Memory unit U_A and U_B store the input operands A and B. Memory unit U_{R0} and U_R store intermediate results and final results respectively, and they are initialized to 0 before computing. Step 1 writes the result R0, of *nand* (A, B) to U_{R0}. A *not* (R0) operation makes memory unit U_R get the result of *and* (A, B). Finally, the *xnor* (A, B) result can be written to memory unit U_R after step3. Based on the simulation in Sect. 4, it needs only 13.328 pJ to calculate 8-bit full add in parallel:

$$C_{out} = (MIN(A + B + C_{in}))', \tag{1}$$

$$S = A\ xnor\ B\ xnor\ C_{in}. \tag{2}$$

The proposed architecture can also work as a ripple-carry adder (RCA). It can be used to complete the unsigned fixed-point addition based on Eqs. (3) and (4) (in which, "i" represents the i-th element of the operand), which is similar

Table 4 The process of computing *xnor* (A, B)

Memory unit	U_A	U_B	Initial		Step1 *nand* (A, B)	Step2 *not* (*R0*)	Step3 *nor* (A, B)
			U_{R0}	U_R	U_{R0}	U_R	U_R
Data stored in the memory unit	0	0	0	0	1	0	1
	0	1	0	0	1	0	0
	1	0	0	0	1	0	0
	1	1	0	0	0	1	1

to Eqs. (1) and (2). Table 5 shows the process of computing 8-bit unsigned fixed-point addition A + B. In the table, Row0 and Row1 store the 8-bit-operands A and B, and Row2 is used to store carrays. The *MIN* operation results of data in Row0, Row1 and Row2 are stored in Row3. In step1, it needs to finish *MIN* operation to get $\boldsymbol{C}_{i+1}{}'$ and output $\boldsymbol{C}_{i+1}$ from $\overline{\boldsymbol{Q}_{\boldsymbol{m}}}$ of PCSAs, then write $\boldsymbol{C}_{i+1}$ to corresponding unit to next column, which is a serial procedure. Next, 8-bit Sum S can be figured out after two 8-bit *xnor* operations, which is 8-bit-parallel procedure that needs 6 steps. Step 2 and 5 are *nand* operations, step 3 and 6 are *not* operations, and step 4 and 7 are *nor* operations, operands of all these operations are A and B, and operation results are stored in corresponding rows as shown in the table. As reported in Sect. 4, it needs only 21.037 pJ to calculate 8-bit unsigned fixed-point addition:

$$C_{i+1} = (MIN(A_i + B_i + C_i))', \tag{3}$$

$$S_i = A_i \, xnor \, B_i \, xnor \, C_i. \tag{4}$$

3.2.2 Signed fixed-point addition and subtraction

In this design, signed fixed-point addition and subtraction are computed based on Eqs. (5) and (6):

$$[D + E]_{complement} = [D]_{complement} + [E]_{complement}, \tag{5}$$

$$[D - E]_{complement} = [D]_{complement} + [-E]_{complement}. \tag{6}$$

It needs the following steps to achieve 8-bit-[D + E] based on the proposed architecture.

Step 1: Computing complements of D and E, respectively. Since the rules for the complement of positive and negative numbers are different, the sign bits need to be read out and stored in the register, and RISC-V determine if "Invert" and "+ 1" for the operands based on it. It executes "+ 1" by the following rules: Reading data from low bit to high bit, when the first "0" presents, write "1" in its position as well as write "0" in its lower position and other bits remain the themselves.

Step 2: Computing unsigned fixed-point addition $[D]_{complement} + [E]_{complement}$ as described in Sect. 3.2.1.

Step 3: Getting the result of D + E by computing the complement of the result in Step2. The computing of 8-bit signed fixed-point subtraction D − E needs to invert the sign bit of E before Step1, and other steps is the same as 8-bit-D + E. As reported in Sect. 4, it need only 26.757 pJ to achieve 8-bit signed fixed-point addition.

3.2.3 Floated-point addition

A floated-point data N can be presented as $N = 2^{N_E} \times N_M$, N_E and N_M are exponent and mantissa of N. GCIM stores binary N_E and N_M as data N. There are four steps to achieve a floated-point addition R = X + Y ($R = 2^{R_E} \times R_M$, $X = 2^{X_E} \times X_M$, $Y = 2^{Y_E} \times Y_M$) as described below. As reported in Sect. 4, it need only 85.803 pJ to achieve 8-bit floated-point addition.

Step 1: The exponent bits of two operands X_E *and* Y_E are subtracted ($X_E - Y_E$) based on the methods described in Sect. 3.2.2.

Step 2: Read out the result of Step1 and store it in register to prepare for the judgment in the next step.

Step 3: According to the sign bit of $X_E - Y_E$ in the register, RSIC-V decides the exponential bit of the result R_E and which operand's mantissa bits should be shifted as well as the shift length. For example, if $X_E > Y_E$, then GCIM will copy X_E as R_E and shift Y_M right by $|X_E - Y_E|$ bits under the control of RSIC-V. For ease of description, we use M_M to represent the mantissa of Y after shift. Thanks for the subarray design of the architecture, it can achieve shift operation directly within the array.

Step 4: Add the shifted mantissa of operand Y and mantissa of operand X ($M_M + X_M$) as the mantissa of its result R_M. According to the ripple-carry adder described in Sect. 3.2.1, R_M can be calculated easily. Furthermore, if the

Table 5 Computing process of 8-bit unsigned fixed-point addition A + B

		Col0	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	*op*
	Row0	A0	A1	A2	A3	A4	A5	A6	A7		
	Row1	B0	B1	B2	B3	B4	B5	B6	B7		
Step1	Row2	C0	C1	C2	C3	C4	C5	C6	C7		*write*
	Row3	C1′	C2′	C3′	C4′	C5′	C6′	C7′	CO′		*min*
Step2	Row4	(AiBi)′									*nand*
Step3	Row5	AiBi									*not*
Step4	Row5	Ai *nxor* Bi									*nor*
Step5	Row6	((Ai *nxor* Bi)Ci)′									*nand*
Step6	Row7	(Ai *nxor* Bi)Ci									*not*
Step7	Row7	Si = Ai *xnor* Bi *xnor* Ci								CO	*nor*

operands are signed data, it takes the sign bit of X and Y as the sign bit of M_M and X_M respectively to execute signed fixed-point addition according to Sect. 3.2.2.

3.2.4 Fixed-point and float-point multiplication

Figure 5 shows a dot notation representation for a 4×4 wallace tree multiplier. GCIM can achieve fixed-point multiplication based on the rules shown in Fig. 5. Firstly, GCIM worked as fulladder to figure out S and Cout of first three partial products as shown in orange part. Next, GCIM also worked as fulladder to get S and Cout of first S and Cout as well as the fourth partial product as shown in purple part. Finally, GCIM worked as RCA to compute the final result. Thanks to our modified shifter and connector, which moves data to align the corresponding bits so that it makes sure the results of fulladder and RCA are correct. Making use of the computing described above, GCIM can figure out the result of float-point multiplication easily. Firstly, the symbol bit of result is calculated by an *xnor* operation on the symbol bit of two operands. Secondly, the exponential bits of two operands are calculated by signed addition to get the exponential bits of the result. Finally, mantissa bits of the result are figured out by a fixed-point multiplication.

4 Experiments and results

The MTJ/CMOS devices hybrid GCIM architecture is evaluated by Cadence Virtuoso with 45 nm CMOS and 40 nm MTJ technologies. We used the advanced MTJ to evaluate the proposed architecture and the MTJ parameters are shown in Table 6. We set the TMR to 500 present based on the discussion in (Zabihi et al. 2018). And (Hirohata et al. 2015) predicted that a TMR of 1000% at room temperature will be attainable by 2024. Benefit from the high TMR of advanced MTJ, the error rate is greatly reduced. We evaluate the runtime and energy required when using GCIM to support general purpose computings, such as Boolean logic computings, fixed-point and float-point addition and multiplication, etc.

We firstly evaluate the performance of the peripheral circuits using Verilog language and Cadence Virtuoso and the detailed results are shown in Table 7. Compared to regular 2T1MTJ memory, the design needs extra DAC and BLdriver to support computing, needs Mser&Cor to explore its reconfigurability and parallelism, and needs RSIC-V to provide a powerful controller. Both GCIM and regular 2T1MTJ memory need row decoder and column decoder to provide addresses. Regular memory needs sensing circuits to read

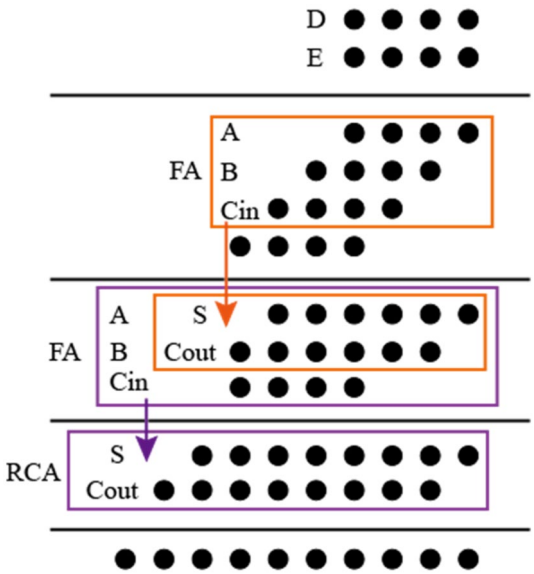


Fig. 5 Dot notation representation for a 4×4 wallace tree multiplier

Table 6 Main parameters of MTJ

MTJ type	PMA-MTJ
MTJ area	10 nm×10 nm
TMR	500%
RA product	1 Ωμm^2 (Maehara et al. 2011)
t_{ox}	0.8 nm

Table 7 The power consumption of peripheral circuits

	Spec	Power (mW)	Area (μm^2)
DAC (Fujiki et al. 2018)	2-bit DAC + 2MOSs	0.000051	0.000006
Row decoder	5–32	0.0721	123
Column decoder	4–16	0.0431	67
BLdriver	–	0.0074	24
Mser&Cor	8 level	0.61	100
RSIC-V	–	0.1	11,601
PCSA (Zhao et al. 2009)	–	0.077	4

Table 8 The evaluation results of 8-bit basic Boolean logic computings used in this design

	not	*nand*	*nor*	*MIN*
V (mV)	220	90	48	60
Time (ns)	3	5	20	6
Array energy (fJ)	3.27	1.835	2.14	1.351
Total energy (pJ)	0.63451	1.0292	4.0685	1.2293
Power (mW)	0.21129	0.2056	0.20336	0.20457
Error rate	0	0	0.65%	1.95%

Table 9 Evaluations of some common general purpose computings

8-bit com	Time (ns)	Energy (e − 12 J)	Power (mW)
not	3	0.63451	0.2113
and	8	1.6638	0.2080
or	23	4.7031	0.2045
xor	31	6.3599	0.2052
Logical shift left	4	3.2639	0.8154
Logical shift right	4	3.2639	0.8154
Arithmetic shift left	4	3.2639	0.8154
Arithmetic shift right	7	3.9721	0.5672
Select	18	3.7165	0.2064
Unsigned fixed-point addition	136	21.037	0.1547
Unsigned fixed-point subtraction	160	26.757	0.1673
Signed fixed-point addition	160	26.757	0.1673
Signed fixed-point subtraction	164	27.714	0.1733
Absolute value subtraction	167	28.422	0.1733
Unsigned fixed-point multiplication	662	177.16	0.2675
Signed fixed-point multiplication	680	173.41	0.2549
Float-point addition	426	85.803	0.2014

data while GCIM needs sensing circuit (PCSA) to read data or sensing computing results. It could be found from Table 7 that the power of modified shifter and connector is very high. It is because the modified shifter consists of conventional barrel shifter and PCSAs. The 8-level conventional barrel shifter themselves consume lots of power, but it can shift 8 bits data simultaneously and can shift data up to 8 units distance. The good news is that computings that need shift data is only multiplication and the multiplication frequency is not very high. So, it only has slight influence on the performance of the proposed architecture. As for the area breakdown, the RSIC-V occupies the largest area, followed by Mser&Cor and decoders.

Table 8 shows the voltage, time, energy, power and error rate of GCIM to perform the basic Boolean logic operations that used in this design. The operands are 8-bit data in this evaluation. One 8-bit *not* operation needs ~ 0.633e − 12 J. That is average a 1-bit *not* operation needs only ~ 0.079e − 12 J energy and the power is ~ 0.027 mW. Monte Carlo method is used to evaluate the error rate and we took 3000 experiments for one operation. Because the error rates of *nand*, *nor* and *not* operations are lower than *and* and *not* operations (Zabihi et al. 2018). We use two basic operations (*nand* plus *not,* or *nor* plus *not*) to achieve *and* and *or* operations.

Table 9 shows some common general purpose computings. Operands of the computings in the table are 8-bit data. GCIM can not only support fix-point computing, but also support float-point computing. The power of most computings are between 0.15 and 0.25 mW. The power of shift computing is a little high which is about 0.8 mW. That is because the high power of the Modified shifter circuit.

5 Conclusion

In this paper, a reconfigurable computing-in-memory architecture for general purpose computing based on STT-MRAM is proposed. It can work as both memory and In-memory computing processor. General purpose computing, such as fixed-point and float-point addition, can be supported by GCIM. And the subarrays can work independently to finish different tasks at the same time or work together to finish one task, which improves the parallelism and reconfigurability of the architecture greatly. The MTJ/CMOS devices hybrid GCIM architecture is evaluated by Cadence Virtuoso. It needs only 85.803 pJ to achieve 8-bit floated-point addition.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China under Grant 61701013, in part by State Key Laboratory of Software Development Environment under Grant SKLSDE-2018ZX-07, in part by National Key Technology Program of China under Grant 2017ZX01032101, in part by State Key Laboratory of Computer Architecture under Grant CARCH201917 and in part by the 111 Talent Program under Grant B16001.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

Biswas, A., Chandrakasan, A.P.: Conv-RAM: an energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications. In: IEEE international solid-state circuits conference-(ISSCC). IEEE, pp. 488–490 (2018)

Cai, F., Correll, J.M., Lee, S.H., et al.: A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. Nat. Electron. **2**(7), 290–299 (2019)

Chen, Y., Wang, X.: Compact modeling and corner analysis of spintronic memristor. In: IEEE/ACM international symposium on nanoscale architectures, pp. 7–12 (2009)

Cheng, M., Xia, L., Zhu, Z., et al.: Time: A training-in-memory architecture for rram-based deep neural networks. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **38**(5), 834–847 (2018)

Chowdhury, Z., Harms, J.D., Khatamifard, S.K., et al.: Efficient in-memory processing using spintronics. IEEE Comput. Archit. Lett. **17**(1), 42–46 (2017)

Deng, E., Zhang, Y., Klein, J.O., et al.: Low power magnetic full-adder based on spin transfer torque MRAM. IEEE Trans. Magn. **49**(9), 4982–4987 (2013)

Fujiki, D., Mahlke, S., Das, R.: In-memory data parallel processor. ACM SIGPLAN Not. **53**(2), 1–14 (2018)

Gao, F., Tziantzioulis, G., Wentzlaff, D.: ComputeDRAM: in-memory compute using off-the-shelf DRAMs. In: Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture, pp. 100–113 (2019)

Hirohata, A., Sukegawa, H., Yanagihara, H., et al.: Roadmap for emerging materials for spintronic device applications. IEEE Trans. Magn. **51**(10), 1–11 (2015)

Imani, M., Gupta, S., Kim, Y., et al.: Floatpim: in-memory acceleration of deep neural network training with high precision. In: Proceedings of the 46th international symposium on computer architecture. ACM, pp. 802–815 (2019)

Jain, S., Ranjan, A., Roy, K., et al.: Computing in memory with spin-transfer torque magnetic ram. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **26**(3), 470–483 (2017)

Kang, W., Zhang, Y., Wang, Z., et al.: Spintronics: emerging ultra-low-power circuits and systems beyond MOS technology. ACM J. Emerg. Technol. Comput. Syst. (JETC) **12**(2), 16 (2015)

Kang, W., Wang, H., Wang, Z., et al.: In-memory processing paradigm for bitwise logic operations in STT–MRAM. IEEE Trans. Magn. **53**(11), 1–4 (2017)

Kang, W., Deng, E., Wang, Z., et al.: Spintronic logic-in-memory paradigms and implementations, pp. 215–229. Springer, Singapore (2020)

Keckler, S.W., Dally, W.J., Khailany, B., et al.: GPUs and the future of parallel computing. IEEE Micro **31**(5), 7–17 (2011)

Kim, A., Austin, T., Baauw, D., et al.: Leakage current: Moore's law meets static power. Computer **36**(12), 68–75 (2003)

Li, S., Xu, C., Zou, Q., et al.: Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In: Proceedings of the 53rd annual design automation conference. ACM, p. 173 (2016)

Li, S., Niu, D., Malladi, K. T., et al.: Drisa: a dram-based reconfigurable in-situ accelerator. In: 2017 50th annual IEEE/ACM international symposium on microarchitecture (MICRO). IEEE, pp. 288–301 (2017)

Liu, B., Hu, M., Li, H., et al.: Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine. In: Proceedings of the 53rd annual design automation conference, pp. 1–6 (2013)

Maehara, H., Nishimura, K., Nagamine, Y., et al.: 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), 033002 (2011)

Vincent, A.F., Locatelli, N., Klein, J.-O., Zhao, W.S., Galdin-Retailleau, S., Querlioz, D.: Analytical macrospin modeling of the stochastic switching time of spin-transfer torque devices. IEEE Trans. Electron Devices **62**(1), 164–170 (2015)

Wang, J., Wang, X., Eckert, C., et al.: A compute SRAM with bit-serial integer/floating-point operations for programmable in-memory vector acceleration. In: 2019 IEEE international solid-state circuits conference-(ISSCC). IEEE, pp. 224–226 (2019)

Wulf, W.A., McKee, S.A.: Hitting the memory wall: implications of the obvious. ACM SIGARCH Comput. Archit. News **23**(1), 20–24 (1995)

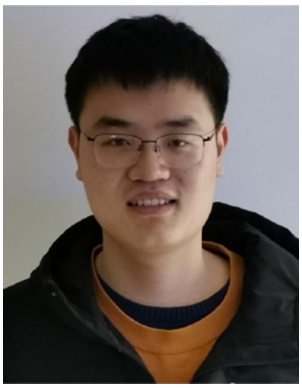
Zabihi, M., Chowdhury, Z., Zhao, Z., et al.: In-memory processing on the spintronic CRAM: from hardware design to application mapping. IEEE Trans. Comput. **68**(8), 1159–1173 (2018)

Zhang, H., Kang, W., Cao, K., et al.: spintronic processing unit in spin transfer torque magnetic random access memor. IEEE Trans. Electron. Devices **66**(4), 2017–2022 (2019)

Zhao, W., Chappert, C., Javerliac, V., et al.: High speed, high stability and low power sensing amplifier for MTJ/CMOS hybrid logic circuits. IEEE Trans. Magn. **45**(10), 3784–3787 (2009)

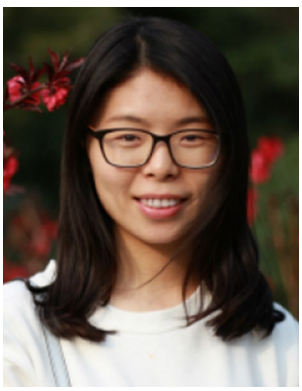
Yu Pan received the B.S. degree in physics from Shandong University, Jinan, Shandong, China, in 2018. She is currently a master student with the School of Microelectronics in Beihang University, Beijing, China. Her research interests include computing-in-memory architecture and circuit design.

Xiaotao Jia received the B.S. degree in mathematics from Beijing Jiao Tong University, Beijing, China, in 2011, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2016. He is currently an Assistant Professor with the School of Microelectronics in Beihang University, Beijing, China. From 2016 to 2019, he was a post-doctor researcher with the Microelectronics in Beihang University, Beijing, China. His current research interests include spintronic circuits, stochastic computing and Bayesian deep learning.



Zhen Cheng received the B.S. degree in communication engineering from Taiyuan University of Technology, Shanxi, China, in 2019. He is currently a master student with the School of Microelectronics in Beihang University, Beijing, China. His research interests include circuit design and EDA.

Peng Ouyang received the B.S. degree in electronic and information technology from Central South University, Changsha, Hunan, China, in 2008, and the Ph.D. degree in electronic science and technology from Tsinghua University, Beijing, China, in 2014. He holds a postdoctoral position with the School of Information, Tsinghua University. His research interests include the embedded deep learning, neuron computing, and reconfigurable computing.



Xueyan Wang received the B.S. degree in computer science from Shandong University, Jinan, China, in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018. From 2015 to 2016, she was a visiting student in University of Maryland, College Park, MD, USA. She is currently a postdoctoral researcher with the School of Microelectronics in Beihang University, Beijing, China. Her current research interests include hardware security and processing-in-memory architectures.

Jianlei Yang received the B.S. degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in computer science and technology with Tsinghua University, Beijing, China, in 2014. He joined Beihang University, Beijing, China, in 2016, where he is currently an Associate Professor with the School of Computer Science and Engineering. From 2014 to 2016, he was a postdoctoral researcher with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania, United States. From 2013 to 2014, he was a research intern at Intel Labs China, Intel Corporation. His current research interests include spintronics and neuromorphic computing systems. Dr. Yang was the recipient of the first place on TAU Power Grid Simulation Contest in 2011, and the second place on TAU Power Grid Transient Simulation Contest in 2012. He was a recipient of IEEE ICCD Best Paper Award in 2013, IEEE ICESS Best Paper Award in 2017, and ACM GLSVLSI Best Paper Nomination in 2015.

Weisheng Zhao received the Ph.D. degree in physics from University of Paris Sud, Paris, France, in 2007. He is currently the Professor with the School of Microelectronics in Beihang University, Beijing, China. In 2009, he joined the French National Research Center (CNRS), as a Tenured Research Scientist. Since 2014, he has been a Distinguished Professor with Beihang University, Beijing, China. He has published more than 200 scientific articles in leading journals and conferences, such as Nature Electronics, Nature Communications, Advanced Materials, IEEE Transactions, ISCA and DAC. His current research interests include the hybrid integration of nano-devices with CMOS circuit and new nonvolatile memory (40-nm technology node and below) like MRAM circuit and architecture design. He is currently the Editor-In-Chief for the IEEE Transactions on Circuits and Systems I: Regular Paper. He is an IEEE Fellow.