Fast Comparator Implementation using 1S1R ReRAM Crossbar Arrays

Debjyoti Bhattacharjee[1], Anne Siemon[2], Eike Linn[2], Stephan Menzel[3], and Anupam Chattopadhyay[1]

Email : {*debjyoti001,anupam*}*@ntu.edu.sg,* {*siemon,linn*}*@iwe.rwth-aachen.de, st.menzel@fz-juelich.de*
[1]Nanyang Technological University, Singapore
[2]Institut für Werkstoffe der Elektrotechnik (IWE 2), RWTH Aachen University, Germany
[3]Peter Grünberg Institute (PGI-7), Forschungszentrum Jülich, Germany

***Abstract*—Low leakage power and non-volatile storage capabilities have marked memristive devices as promising technology for future data storage. Furthermore, capability to natively realize universal Boolean operators prompted researchers to exploit it for Programmable Logic-in-Memory (PLiM), deep neural network and neuromorphic computing platforms. Proliferation of such platforms rely on efficient mapping of key computing blocks onto ReRAM crossbar arrays. Comparators are fundamental circuits forming the basis of decision logic in processing units. In this paper, we report the first study on implementation of comparators using 1S1R ReRAM crossbar array. Our major contribution is an efficient mapping of the comparator logic, with logarithmic delay and a linear number of devices in terms of the number of data width— *n*. Simulation studies are performed using 4-bit data signals to validate our approach.**

I. Introduction

One of the most promising technologies for logic and memory applications is redox-based resistive switches (ReRAM) [10]. Large passive crossbar arrays can be realized by means of devices such as a select device in series to a memristive device (1S1R) or a complementary resistive switch (1CRS), consisting of two anti-serially connected cells, that prevent parasitic currents [3]. Multiple feasible logic-in-memory designs have been proposed using ReRAM cells. These propositions fall grossly in the classes of general-purpose Programmable Logic-in-Memory (PLiM) computing platform [9], machine learning applications within the broader perspective of neuromorphic computing domain [2], [5], [7], [8] and finally, a series of dedicated circuit proposals [1], [12]. For either of these propositions, localized decision-making circuits play an important role, which is achieved in traditional CMOS designs using comparator. To the best of our knowledge, there has been no proposition of in-memory comparator circuits for ReRAM crossbar array, which we address in this paper.

An identity or equality comparator, compares the inputs and produces an output HIGH if both of the inputs are equal. Such circuits can be used in electronic locks and security devices, where a binary password consisting of multiple bits as input, has to be compared against a preset password. A magnitude comparator compares n-bit binary strings and outputs HIGH if the magnitude of the first input is greater than or equal to the magnitude of the second input. For example, a magnitude comparator is required for checking if a value is greater than a specified threshold to initiate further action. The schema for identity and magntitude comparator for 4-bit data signals is shown in Fig. 1.

This paper presents an efficient implementation of identity and magnitude comparators, using ReRAM arrays, with an optimal delay of $O(log_2(n))$ cycles using $O(n)$ devices, for both the circuits. We also present the results of circuit simulation of the proposed schemes and analyze the circuits in terms of number of devices and number of cycles.

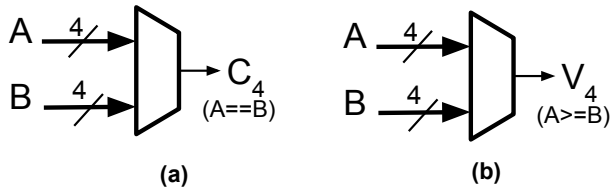

Fig. 1: Schema for (a) Identity comparator (b) Magnitude comparator

II. Modelling and Logic concepts of ReRAM devices

In this work, a Valence Change Mechanism (VCM) ReRAM device and an additional selector are assumed. The ReRAM device used in the performed simulations, is represented by means of VerilogA model. The model proposed in [11] has been used, with the parameter set for median kinetics, where R_0 was changed to 69 kΩ. The selector device was fitted to the presented selector of [6]. The simulations were performed using Cadence® Spectre® .

For logic operations, the device can be interpreted as a finite-state-machine (FSM), as shown in [4]. This FSM presents the transitions and states corresponding to the input signals applied to the two terminals offered by the device, shown in Fig. 2 (a). Here terminal one is connected to the wordline (*wl*) and terminal two is connected to the bitline (*bl*). Only two transitions are feasible one from the 0 state to the 1 state by applying a '1' to the *wl* and '0' at the *bl*, whereas the other transistion starts from the 1 state and ends at the 0 state by applying a '0' at the *wl* and a '1' at the *bl*. All other input combinations won't change the state. The 1 state is represented by a low resistive state (LRS) and the 0 state by a high resistive state (HRS), so that the current level at read-out steps determine the stored values.

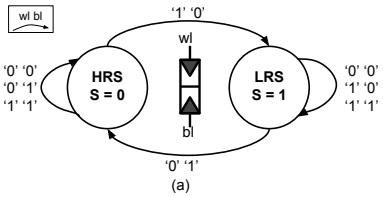

Fig. 2: Logic operation using ReRAM devices

TABLE I: Steps of 4-bit identity comparator

Sl#		X_0		X_1		X_2		X_3		Y
1	'0'	×	'0'	×	'0'	×	'0'	×	'0'	×
		'1'		'1'		'1'		'1'	'0'	×
									'0'	×
									'0'	×
										'1'
2	a_0	0	a_1	0	a_2	0	a_3	0	a_0	0
		'0'		'0'		'0'		'0'	a_1	0
									a_2	0
									a_3	0
										'0'
3	b_0	a_0	b_1	a_1	b_2	a_2	b_3	a_3	b_0	a_0
		'1'		'1'		'1'		'1'	b_1	a_1
									b_2	a_2
									b_3	a_3
										'0'
4	'1'	m_0	'1'	m_1	'1'	m_2	'1'	m_3	0	n_0
		n_0		n_1		n_2		n_3	0	n_1
									0	n_2
									0	n_3
										'1'
5	x_1	x_0	'1'	x_1	x_3	x_2	'1'	x_3		
		'1'		0		'1'		0		
6	x_{23}	x_{01}	0	x_1	'1'	x_{23}	0	x_3		
		'1'		0		0		0		
Res		c_4		x_1		x_{23}		x_3		

III. Identity comparator

An identity comparator compares the value of the inputs and generates a HIGH output only when both the inputs are identical, otherwise the output is LOW. An identity comparator for 4-bit values can be represented by the following equation:

$$c_4 = (a_0 \leftrightarrow b_0).(a_1 \leftrightarrow b_1).(a_2 \leftrightarrow b_2).(a_3 \leftrightarrow b_3) \quad (1)$$

$$a \leftrightarrow b = a.b + \overline{a + b} \quad (2)$$

where a_i and b_i represent the i^{th} bits of input data signals a and b respectively. $\overline{a}$ represents the negated value of Boolean variable a. Operators ., + and $\leftrightarrow$ represent Boolean AND, OR and XNOR operations respectively. The XNOR operation can be expressed in terms of AND and OR operations as shown in (2).

A. Identity comparator steps

Without loss of generality, we demonstrate the implementation of identity comparator using ReRAM arrays, for 4-bit inputs, as show in Table I.

Step 1: All the 1S1R arrays are initialized to a known state 0 by applying '0' to the wordlines and '1' to the bitlines.

Step 2: The data a_i is loaded into each of the array X_i by applying a_i to the wordlines and '0' to the bitlines. Another copy of input A is loaded to the array Y.

Step 3: In this step, $m_i = a_i.b_i$ is computed in the arrays X_i. In the array Y, $n_i = a_i + b_i$ is computed.

Step 4: The XNOR(x_i) for each bit position is computed in this step by ORing m_i and $\overline{n_i}$ in array X_i.

Step 5: The results pairs from the previous step x_0 , x_1 are ANDed and x_2 , x_3 are ANDed in this step.

Step 6: In the final step, the final output is computed by ANDing x_{01} and x_{23}.

In general, the identity comparator compares the individual bits of the inputs using XNOR operations to determine if the corresponding bits are identical. Thereafter, a balanced AND-tree is used to determine if the two input data signals are equal. For the 4-bit identity comparator, steps 5-6 performs the AND-tree computation.

B. Analysis of the proposed scheme

For designing n-bit identity comparator, $2n$ devices are required - n arrays of size 1×1 device and one array with n-wordlines and 1 bitline i.e. with n devices. Two cycles are required for initializing and loading data into the arrays. Two cycles are needed to compute the XNOR of individual bits of the input data signals. Thereafter, $log_2(n)$ cycles are needed to compute the AND of the XNOR results of the individual bit positions, since we use a balanced AND tree for computation. Thus, the scheme requires $4 + log_2(n)$ cycles for computing the required function.

IV. Magnitude comparator

A n-bit magnitude comparator compares two n bit signals A and B and gives an output HIGH if $A \geq B$, otherwise LOW i.e. the value represented by the bit string A is greater than or equal to the value represented by the bit string B. For simplicity, we assume that $n = 2^k$, $k \in \mathcal{N}$. The magnitude comparison can be performed by using a recursive formulation. To do so, we define intermediate variables — g_i, s_i and e_i for comparison of single bit i and g_{ij}, s_{ij} and e_{ij} for comparing bits i to j, of the inputs. The terms g_{ii}, s_{ii} and e_{ii} are identical to g_i, s_i and e_i, respectively.

$$g_i = a_i > b_i = a_i.\overline{b_i} \quad (3)$$

$$g_{ij} = a_{ij} > b_{ij} \quad (4)$$

$$s_i = a_i \leq b_i = \overline{a_i} + b_i = \overline{g_i} \quad (5)$$

$$g_{ij} = a_{ij} \leq b_{ij} = \overline{g_{ij}} \quad (6)$$

$$e_i = a_i \geq b_i = a_i + \overline{b_i} \quad (7)$$

$$e_{ij} = a_{ij} \geq b_{ij} \quad (8)$$

Let us partition the 2-bit input signal strings A and B into two equal substrings. Say, $A = [a_1|a_0]$ and $B = [b_1|b_0]$. We can say that $A \geq B$ if $a_1 > b_1$ or if $a_1 == b_1$ and $a_0 >= b_0$. Formally, the comparator function V_2 can be therefore defined using the variables g and e, as shown in (9).

$$V_2 = g_1 + e_1.e_0 \quad (9)$$

This recursive formulation can be generalized for comparing inputs with larger number of bits, as presented in

(10) and (11). The comparator function V_n is identical to the value of $e_{n-1,0}$.

$$g_{ij} = a_{ij} > b_{ij} = g_{(i,\lceil \frac{i+j}{2} \rceil)} + e_{(i,\lceil \frac{i+j}{2} \rceil)}.g_{(\lfloor \frac{i+j}{2} \rfloor, j)} \quad (10)$$

$$e_{ij} = a_{ij} \geq b_{ij} = g_{(i,\lceil \frac{i+j}{2} \rceil)} + e_{(i,\lceil \frac{i+j}{2} \rceil)}.e_{(\lfloor \frac{i+j}{2} \rfloor, j)} \quad (11)$$

For computing V_2 and other intermediate variables for two bits (g_{32}, e_{32}, etc) on 1S1R arrays, we propose an optimization to compute V_2 in a single cycle. Given $g_i = 1$, it is implies that $e_i = 1$, thereby g_i, e_i and V_2 can be re-written as

$$\begin{aligned} g_i &= g_i.e_i && (12)\\ e_i &= g_i + e_i && (13)\\ V_2 &= g_1.e_1 + (g_1 + e_1).e_0 \\ &= g_1.e_1 + g_1.e_1 + g_1.e_0 \\ &= M_3(g_1, e_1, e_0) && (14) \end{aligned}$$

Since 1S1R arrays are capable of computing $M_3(S, wl, \overline{bl})$ natively and $g_1 = \overline{s_1}$, we can express V_2 as shown in (15), which allows V_2 to be computed in a single clock cycle.

$$V_2 = M_3(e_1, e_0, \overline{s_1}) \quad (15)$$

Without loss of generality, we demonstrate the implementation of comparator function using this formulation for 4-bit numbers using ReRAM arrays. We can compute the V_4, i.e. g_{30} by using the recursive formulation, (10) and (11), and the corresponding computation tree expressed in terms of M_3 is shown in Fig. 3. The steps involved in computation of 4-bit

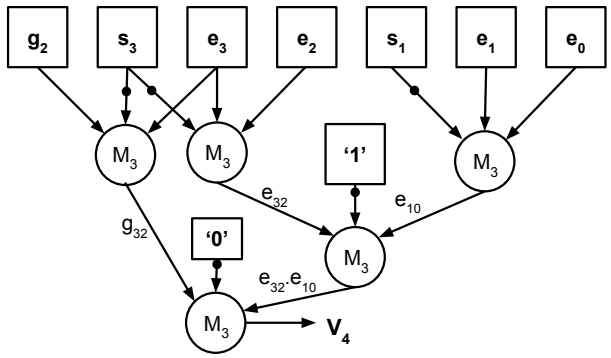

Fig. 3: 4-bit magnitude comparator computation tree

magnitude comparator using 1S1R arrays is stated below.

Step 1. In this step, the array G_2 is intialized to 0 and the rest of the arrays are intialized to 1.

Step 2. The variables s_3, e_3, g_2, e_2, s_1, e_1 and e_0 are computed in the arrays S_3, E_3, G_2, E_2, S_1, E_1 and E_0 respectively.

Step 3. g_{32} and e_{32} is computed in array G_2 and E_2 by reading out arrays S_3 and E_3 to apply appropriate input at the wordlines and bitlines. Similarly, array S_1 and E_1 are read out for computation of e_{10} in array E_0.

Step 4. In this step, $e_{32}.e_{10}$ is computed in array E_2 by reading out the required value e_{10} from array E_0 and applying it to the wordline and '1' to the bitline.

Step 5. Finally $V_4 = g_{32} + e_{32}.e_{10}$ is computed in array G_2 by reading out the required value $e_{32}.e_{10}$ from array E_2 and applying it to the wordline and '0' to the bitline.

TABLE II: Steps of 4-bit magnitude comparator ($A \geq B$)

Sl#	S_3	E_3	G_2	E_2	S_1	E_1	E_0
1.	'1' × '0'	'1' × '0'	'0' × '1'	'1' × '0'	'1' × '0'	'1' × '0'	'1' × '0'
2.	b_3 1 a_3	a_3 1 b_3	a_2 0 b_2	a_2 1 b_2	b_1 1 a_1	a_1 1 b_1	a_0 1 b_0
3.	'1' s_3 0	'1' e_3 0	e_3 g_2 s_3	e_3 e_2 s_3	'1' s_1 0	'1' e_1 0	e_1 e_0 s_1
4.	0 s_3 0	0 e_3 0	0 g_{32} 0	e_{10} e_{32} '1'	0 s_1 0	0 e_1 0	'1' e_0 0
5.	0 s_3 0	0 e_3 0	$e_{32}.e_{10}$ g_{32} '0'	'1' $e_{32}.e_{10}$ 0	0 s_1 0	0 e_1 0	0 e_0 0
Res	s_3	e_3	V_4	e_{30}	s_1	e_1	e_0

A. Analysis of the proposed scheme

The proposed scheme solves the problem in $log_2(n)$ levels, where each level requires 2 cycle (one cycle for computing $e_i.e_j$ and one step for ORing g_i with the term $e_i.e_j$), except the lowest level with intermediate variables for 2 bits, which can be computed in one cycle by using the proposed optimization. One cycle is required for initializing the arrays and one more cycle for computing intermediate variables for one bit. Thus, the proposed scheme requires $2log_2(n) + 1$ cycles for computation of n-bit magnitude comparator.

The number of devices required is $2n - 1$, since for each bit position i, $0 \leq i < n$, variable g_i or s_i and e_i has to be computed and stored in step 2 of computation, except variable g_0 which the scheme does not require.

V. Simulation

In this section, the results of circuit simulation of the algorithms, are presented. The pulse width is chosen to be 50 ns and the voltage amplitude is 2.4 V. Since parallel reads of multiple devices are assumed, additional wordline amplifiers have been used to support the algorithm. In Fig. 4, the simulation waveforms of the identity comparator is depicted, with input variables a = $(0101)_2$ and b = $(0101)_2$. The final result is read out in the last step of array X_0 and corresponds to 1 state. This means that the input variables are identical, which is the expected result. In Fig. 5, the simulation of the magnitude comparator is presented, for the input variables a = $(1000)_2$ and b = $(0011)_2$. The result is read out in the last step from Array G_2 and corresponds to the 1 state, which implies that $a \geq b$, showing the correctness of the scheme for this input data. For the specified input signals, the power consumption of the identity and magnitude comparator is 97.2 pJ and 74.2 pJ respectively. Both schemes were checked for correctness, by exhaustively testing with all possible input combinations of 4-bit data.

VI. Conclusion

In this work, we have proposed in-memory schemes with $O(log_2(n))$ delay for identity comparator and magnitude comparator using 1S1R arrays. Our proposal has been verified with circuit simulation of the identity and magnitude comparator for 4-bit data signals and presented estimates of delay and area, in terms of number of devices.

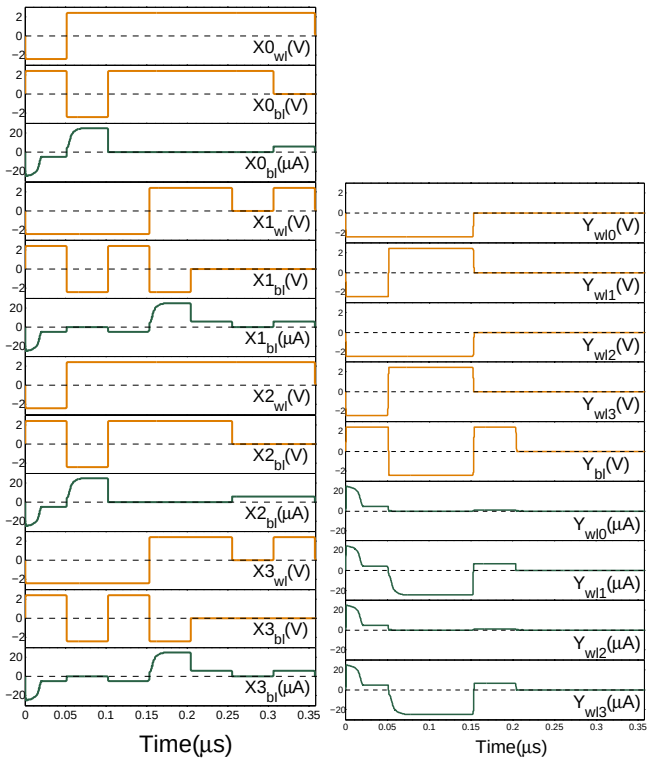

Fig. 4: Simulation waveform for Identity Comparator
[The orange color waveforms represent the voltage applied to input lines. The green color waveforms are the information read out from the arrays]

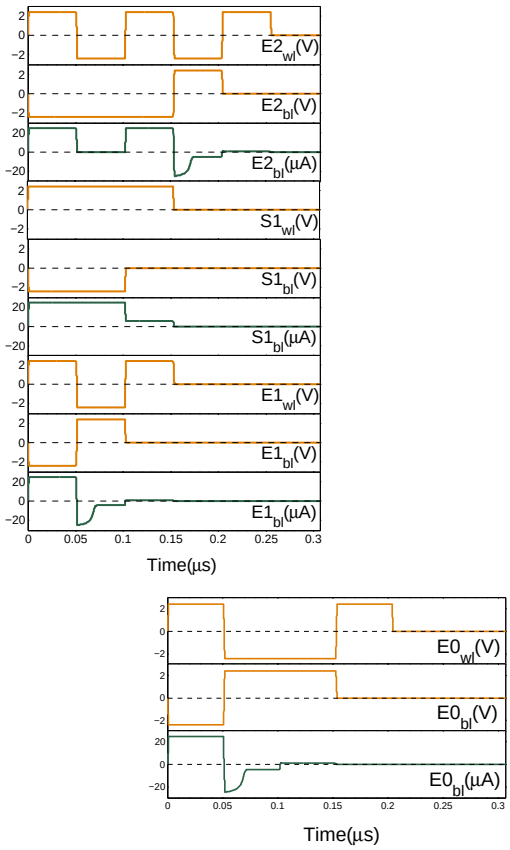

Fig. 5: Simulation waveform for Value Comparator

REFERENCES

[1] A. Siemon, S. Menzel, R. Waser and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 1, pp. 64–74, March 2015.

[2] D. B. Strukov, D.R. Stewart, J. Borghetti, X. Li, M. Pickett, G. M. Ribeiro, W. Robinett, G. Snider, J. P. Strachan, W. Wu, Q. Xia, J. J. Yang and R. S. Williams, "Hybrid cmos/memristor circuits," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 1967–1970.

[3] E. Linn, R. Rosezin, C. Kügeler and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Letters*, vol. 9, pp. 403–406, 2010.

[4] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger and R. Waser, "Beyond von neumann-logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, 2012.

[5] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa and W. Lu, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano Letters*, vol. 12, no. 1, pp. 389–395, 2011.

[6] S. Kim, W. Lee, and H. Hwang, "Selector devices for cross-point reram," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012.

[7] M. P. Sah, H. Kim and L. O. Chua, "Brains are made of memristors," *IEEE Circuits and Systems Magazine*, vol. 14, no. 1, 2014.

[8] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, pp. 61–64, 2015.

[9] P.-E. Gaillardon, L. Amarù, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay and G. De Micheli, "The programmable logic-in-memory (plim) computer," in *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, 2016.

[10] R. Waser, R. Dittmann, G. Staikov and K. Szot, "Redox-based resistive switching memories - nanoionic mechanisms, prospects, and challenges," *Adv. Mater.*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[11] A. Siemon, S. Menzel, A. Marchewka, Y. Nishi, R. Waser, and E. Linn, "Simulation of TaO_x-based complementary resistive switches by a physics-based memristive model," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1420–1423.

[12] T. Breuer, A. Siemon, E. Linn, S. Menzel, R. Wasser and V. Rana, "A HfO_2-Based Complementary Switching Crossbar Adder," *Advanced Electronic Materials*, vol. 1, no. 10, 2015.