Efficient Implementation of Multiplexer and Priority Multiplexer using 1S1R ReRAM Crossbar Arrays

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

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

***Abstract*—Memristive devices have been shown to have low leakage power, non-volatile storage capability and high storage density. In addition, by using stateful logic approaches, hybrid CMOS nano-crossbar arrays offer functionalities such as arithmetic operations, which make them ideal target for in-memory computing. Multiplexers are useful circuits that are used in a wide variety of applications such as encoding-decoding, signal routing, data communications and data bus control. In this paper, we report the first study on implementation of multiplexers using 1S1R crossbar arrays. An efficient mapping of the multiplexers is presented, with logarithmic delay, in terms of number of control signals— *n*. Physics-based circuit simulations are performed to validate our approach.**

I. Introduction

Redox-based resistive switches (ReRAM) are one of the most promising non-volatile storage technologies [1]. Furthermore, the capability of such devices to perform stateful logic operations [2], [3], [4] enable it as an ideal platform for in-memory computing, in a bid to alter the traditional von Neumann computing model. Multiple arithmetic circuits [5], [6], [7] and programmable processors [8] have been implemented using CMOS-ReRAM hybrid crossbar platforms.

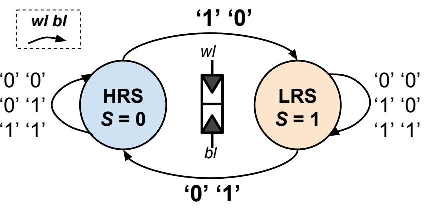
In large passive crossbar arrays low ohmic paths can cause additional currents, so called sneak paths. This sneak paths can be especially for the read-out major problems, since they can add up and change the read-out result from a '0' to a '1'. To prevent these and enable large passive crossbar arrays devices with a higher selectivity, e.g. a complementary resistive switch (1CRS), consisting of two anti-serially connected cells or a selector in series to the ReRAM device, are needed [9], [1].

Efficient multiplexer synthesis is a challenging problem in conventional technologies [10]. Multiplexers are circuits that are used to select one from many data lines, based on the value of the control signals. Multiplexers are used in a wide variety of applications such as signal routing, data communications and data bus control. Priority multiplexers find applications where the selection lines have a defined priority and based on the priority of the selection lines, the data signal is chosen.

This paper presents the efficient implementation of multiplexer and priority multiplexer using 1S1R arrays, with an optimal delay of $O(log_2 n)$ cycles, where n is the number of control signals, for both circuits. We also present the results of circuit-simulation of the proposed schemes by using VerilogA models of the ReRAM devicess and analysis of the circuits in terms of number of devices, number of cycles and energy consumption.

A. Logic Operations of 1S1R devices

The ReRAM device is operated using two input lines – the wordline wl and bitline bl which modifies its internal state S. In Fig. 1, the truth table of a CRS-logic device is shown. The state switches only for two input combinations, which is also clearly observable in the FSM.



wl	*bl*	*NS*
'0'	'0'	*S*
'0'	'1'	0
'1'	'0'	1
'1'	'1'	*S*

Fig. 1: 1S1R device logic operation realizing Majority.

It can be observed that next state NS of the device is nothing else but the function $M_3(S, wl, \overline{bl})$ [8], where M_3 is the 3-input Boolean majority function. Thereby, the internal state S of the device acts as an input when computation of the next state NS. The Boolean majority function along with invertor is a universal logic operator and therefore can be utilised to implement any Boolean function. In this work, the challenge is to efficiently implement the multiplexer functionalities with M_3 operator. A further challenge is to minimize the device count and overhead of crossbar array operations, which directly results into the area complexity.

II. Multiplexer

A multiplexer (MUX) with 2^n input data signals requires n control signal to select the desired input signal. We explain the working principle of a 4-input multiplexer and its implementation using 1S1R arrays, which can be extended to a general case. The schematic of a 4-input multiplexer is shown in Fig. 2 (a) and the corresponding Boolean equation presented in (1).

$$f_4 = \overline{s_0}.\overline{s_1}.a_0 + \overline{s_0}.s_1.a_1 + s_0.\overline{s_1}.a_2 + s_0.s_1.a_3 \qquad (1)$$

, where a_j represent the data signals and s_i are the control signals. Operators . and + represent Boolean AND and OR operations respectively. $\overline{a}$ represents the negated value of Boolean variable a.

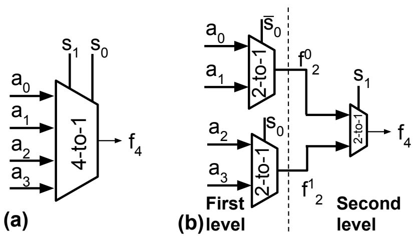


Fig. 2: 4-to-1 MUX (a) Schema (b) Implementation using 2-to-1 MUX

We construct the 4-to-1 MUX by using a hierarchy of 2-to-1 MUX operations, as shown in Fig. 2. In general, an n-to-1 MUX can be constructed using $log_2(n)$ levels of 2-to-1 MUX, which allows the multiplexer to complete operation in $O(log_2(n))$ cycles. A 2-to-1 MUX is represented by the Boolean equation 2, where a_0 and a_1 are the data signals and s is the control signal.

$$f_2 = \overline{s}.a_0 + s.a_1 \tag{2}$$

A. *Multiplexer steps*

Without loss of generality, we explain the proposed schema using 2-bit wide data signals for a 4-to-1 multiplexer, as shown in Figure 4.

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

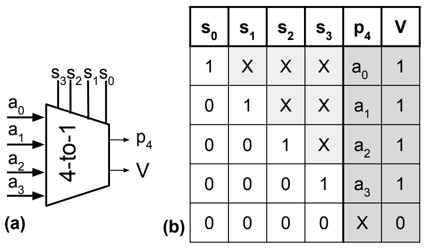
Step 2: The select signal s_0 is loaded into array $A0$ and $A2$, and the inverted signal $\overline{s_0}$ is loaded to array $A1$ and $A3$.

Step 3: The data a_{ij} is ANDed with the contents of device A_{ij} by applying a_{ij} to the wordlines and '1' to the bitlines. The resultant values are shown as v_{ij}, where i is th array number and j is the bit number of the data.

Step 4: The second product term of the 2-to-1 MUX is read out and ORed with the first product term to complete the first level of 2-to-1 MUX operations. The resultant values are shown as f_2^{ij}.

Step 5: For the second level 2-to-1 MUX, the result of the first level is ANDed with the appropriate s_i control signal. The resultant values are shown as w_{ij}.

Step 6: Similar to step 4, second product term of the second level 2-to-1 MUX is read out and ORed with the corresponding first product term. This completes the 4-to-1 MUX operation and the result is stored in the array $A0$.



s_0	s_1	s_2	s_3	p_4	V
1	X	X	X	a_0	1
0	1	X	X	a_1	1
0	0	1	X	a_2	1
0	0	0	1	a_3	1
0	0	0	0	X	0

Fig. 3: Priority multiplexer (a) Schema (b) Truth table

B. *Analysis of the proposed scheme*

For n control signals and 2^n data signals each of width m-bits, the proposed scheme requires 2^n 1S1R arrays, each with m wordlines and 1 bitline.

Sl#		A0		A1		A2		A3
1.	'0'	**X**	'0'	**X**	'0'	**X**	'0'	**X**
	'0'	**X**	'0'	**X**	'0'	**X**	'0'	**X**
		'1'		'1'		'1'		'1'
2.	'1'	0	s_0	0	'1'	0	s_0	0
	'1'	0	s_1	0	'1'	0	s_0	0
		s_0		'0'		s_0		'0'
3.	a_{00}	$\overline{s_0}$	a_{10}	s_0	a_{20}	s_0	a_{30}	$\overline{s_0}$
	a_{01}	$\overline{s_0}$	a_{11}	s_0	a_{21}	s_0	a_{31}	$\overline{s_0}$
		'1'		'1'		'1'		'1'
4.	v_{10}	v_{00}	'1'	v_{10}	v_{30}	v_{20}	0	v_{30}
	v_{11}	v_{01}	'1'	v_{11}	v_{31}	v_{21}	0	v_{31}
		'0'		0		'0'		'1'
5.	'0'	f_2^{00}	0	v_{10}	s_1	f_2^{10}	0	v_{30}
	'0'	f_2^{01}	0	v_{11}	s_1	f_2^{11}	0	v_{31}
		s_1		0		'1'		0
6.	w_{10}	w_{00}	0	v_{10}	0	w_{10}	0	v_{30}
	w_{11}	w_{01}	0	v_{11}	0	w_{11}	0	v_{30}
		'0'		0		'1'		0
		f_4^0		v_{10}		w_{10}		v_{30}
		f_4^1		v_{11}		w_{11}		v_{30}

Fig. 4: Steps of 4-to-1 multiplexer

Two cycles are required for initializing and loading data into the arrays. For completing operation, each 2-to-1 MUX levels requires two cycles— one for performing the AND operations to obtain the product terms and one for performing the OR of the product terms. There are $log_2(n)$ levels of 2-to-1 MUX therefore the schema would require $2 + 2log_2(n)$ cycles.

III. Priority Multiplexer

A priority multiplexer selects from one of the n data signals, based on the n control signals, which have a predefined priority. Basically, the priority multiplexer selects input signal a_k, if control signal s_k is '1' and none of the other control signals with priority more than s_k are '1'. If none of the select signals are '1', then the output is invalid. A 4-bit priority multiplexer is represented by the truth table in Fig. 3 (b) and the following equations.

$$p_4 = s_0.a_0 + \overline{s_0}.s_1.a_1 + \overline{s_0}.\overline{s_1}.s_2.a_2 + \overline{s_0}.\overline{s_1}.\overline{s_2}.s_3.a_3 \tag{3}$$

$$V = s_0 + s_1 + s_2 + s_3 \tag{4}$$

where s_j and a_k represent the control and data signals respectively. Priority of control signal s_j is greater than s_k, if $j < k$. The output signal valid V is '1' when the output is valid, otherwise it is low.

We propose a $O(log_2(n))$ implementation for priority multiplexer with n data signals. To obtain a nearly balanced tree structure for computation, the equation for p_n is recursively decomposed into two parts, such that each part has equal number of minterms. The terms common in each decomposed part are factored out and . these parts are decomposed again. The decomposition is continued till only two minterms remain per part.

The factored terms are of the form $\overline{s_0}.\overline{s_1}...\overline{s_k}$, where k is an even number. This can be re-written as $\overline{s_0 + s_1 + ... + s_k}$ which can be easily computed using a OR-reduction tree implemented using M_3 gates followed by a NOT operation.

We demonstrate the decomposition using 4-to-1 priority multiplexer. In the first stage, the equation 4 is decomposed into two parts which have to be ORed to get result of the overall equation :

1) $s_0.a_0 + \overline{s_0}.s_1.a_1$
2) $\overline{s_0}.\overline{s_1}.s_2.a_2 + \overline{s_0}.\overline{s_1}.\overline{s_2}.s_3.a_3$
   $= \overline{s_0 + s_1}.(s_2.a_2 + \overline{s_2}.s_3.a_3)$

The first decomposed part of the equation has only two terms and hence is not decomposed any further. The second part has common term $\overline{s_0}.\overline{s_1}$, is factored out and is expressed as $\overline{s_0 + s_1}$. The remaining part of the equation has two terms which completes the decomposition.

Recall that each 1S1R device can be interpreted as $M_3(S, wl, \overline{bl})$ gate i.e. a 3-input majority gate with the third input negated and S, wl and bl represent the internal state, the wordline and the bitline of the device. The decomposed equation can be implemented using 1S1R devices as shown in the Fig. 5, where all the nodes at the same level are evaluated in the same clock cycle. The square nodes represent the inputs to the priority multiplexer and the circles represent the M_3 computation nodes. The dark grey trapezoid nodes state the ReRAM array name in which a M_3 node is evaluated. Due to lack of space, we do not show the steps explicitly.

A. Analysis of the proposed scheme

The number of cycles required by the proposed scheme for computation of p_n can be stated using the following recursive formulation. Let $T(n)$ be the number of cycles required for completing operation of a priority multiplexer with n data signals, where $n = 2^k$, $k \geq 1$ and is an integer.

$$T(n) = \begin{cases} T(\frac{n}{2}) + 2 & \text{if } n > 2 \quad (5a) \\ 3 & \text{for } n = 2 \quad (5b) \end{cases}$$

The priority multiplexer equation is decomposed into two parts with half the number of product terms and hence $T(\frac{n}{2})$ cycles would be required to evaluate each part simultaneously, one cycle is required for ANDing the second part of the equation with the factored terms and one cycle is required for ORing the two parts of the equation to compute the overall equation. For evaluating a priority multiplexer with two data signals, 3 cycles are needed, which forms the base case of the recursive formulation. Additionally in the beginning, two cycles are needed for initialization of the devices and for loading initial state of the devices. Hence, the number of cycles for computing p_n of a priority multiplexer with n data signals is $T(n) = 2log_2n + 3$. By using a AND-reduction tree for computation of status signal V and considering 2 cycles needed for initialization and loading of initial states, $log_2n + 2$ cycles would be required. Since computation of p_n and V can be performed in parallel, the priority multiplexer would require $log_2n + 3$ cycles to complete operation.

The number of devices $D(n)$ required for computation of p_n with n data signals can be computed as follows:-

$$D(n) = \begin{cases} 2D(\frac{n}{2}) + \frac{n}{4} & \text{if } n > 2 \quad (6a) \\ 2 & \text{for } n = 2 \quad (6b) \end{cases}$$

$D(\frac{n}{2})$ devices are required for computation of the each part of the decomposed equation. $\frac{n}{4}$ devices are required for computation of the OR of the $\frac{n}{2}$ variables in the factored out term. There are $log_2(\frac{n}{4})$ factored out terms with two variables in the recursive formulation, for which an additional device is required for inverting the OR of the variable in the factored out term, which require an additional $log_2(\frac{n}{4})$ devices. The OR of the variables ($\#variables > 2$) in other factored terms can be inverted by reusing the devices, without contributing additional delay. For computation of status signal V using a OR-reduction tree, $\frac{n}{2}$ devices are required. Therefore the total number of devices required is $\frac{n(3+log_2(n))}{4} + log_2(\frac{n}{4}) + \frac{n}{2} = O(nlog_2(n))$.

IV. Simulation

This section presents the results of circuit simulation, using Cadence® Spectre®. To account for the very small filament radius of the median kinetic device in [11], the resistance R_0 is chosen to be 69 kΩ. The voltage is adjusted to 2.4V to enable a clock frequency of 50ns.

In Fig. 6, the simulation results of the multiplexer scheme, for 2-bit input data a_0 = 01, a_1 = 10, a_2 = 00 and a_3 = 11 with control signal s_1s_0 = 01. The orange waveforms depict the voltages applied to the wordline and bitlines of the arrays, while the green waveforms depict the information read out in form of current. The result, available in array $A0$ is read out and is 10 as expected. The simulation results of the priority multiplexer for 2-bit input data signals — a_0 = 01, a_1 = 10, a_2 = 00 and a_3 = 11, with control signals $s_3s_2s_1s_0$ = 0011, is shown in Fig. 7. As signal s_1 has greater priority than s_0, signal 10 is the output signal, available in array $P0$ and since atleast one control signal is 1, the output valid signal V is 1, which is available in array $S0$. Both the schemes have been tested exhaustively for correctness using all possible input combinations. For the specified inputs, the multiplexer and the priority multiplexer schemes consume 73 pJ and 105.24 pJ, respectively.

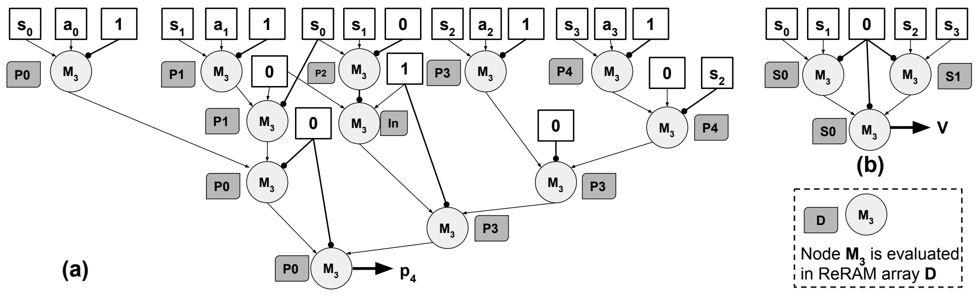
V. Conclusion

In this work, efficient in-memory schemes with $O(log_2n)$ delay for multiplexer and priority multiplexers have been proposed using 1S1R arrays. We benchmarked our proposal by circuit simulations of multiplexer and priority multiplexer with 2-bit data signals and presented estimates of delay, area (in terms of device count) and energy. We believe this work will be useful for design of larger circuits, where multiplexers play an important role such as, encoding-decoding circuits.

References

[1] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, no. 21, pp. 2632–2663, 2009.
[2] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.
[3] 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, p. 305205, 2012.
[4] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 33–36, 2009.

Fig. 5: Priority Multiplexer Computation using M_3 gates (a) Priority multiplexer output p_4 (b) Status V



[5] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A Complementary Resistive Switch-Based Crossbar Array Adder," *IEEE JETCAS*, vol. 5, no. 1, pp. 64–74, 2015.

[6] L. Ni, Y. Wang, H. Yu, W. Yang, C. Weng, and J. Zhao, "An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary RRAM crossbar," in *Proceedings of ASP-DAC*, pp. 280–285, Jan 2016.

[7] D. Bhattacharjee, F. Merchant, and A. Chattopadhyay, "Enabling In-Memory Computation of Binary BLAS using ReRAM Crossbar Arrays," in *24th IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC*, 2016.

[8] P. Gaillardon, L. Amaru, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. D. Micheli, "The programmable logic-in-memory (plim) computer," in *DATE*, pp. 427–432, 2016.

[9] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature materials*, vol. 9, no. 5, pp. 403–406, 2010.

[10] S. Mitra, L. J. Avra, and E. J. McCluskey, "Efficient multiplexer synthesis techniques," *IEEE Des. Test*, vol. 17, pp. 90–97, Oct. 2000.

[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 *ISCAS*, pp. 1420–1423, 2014.

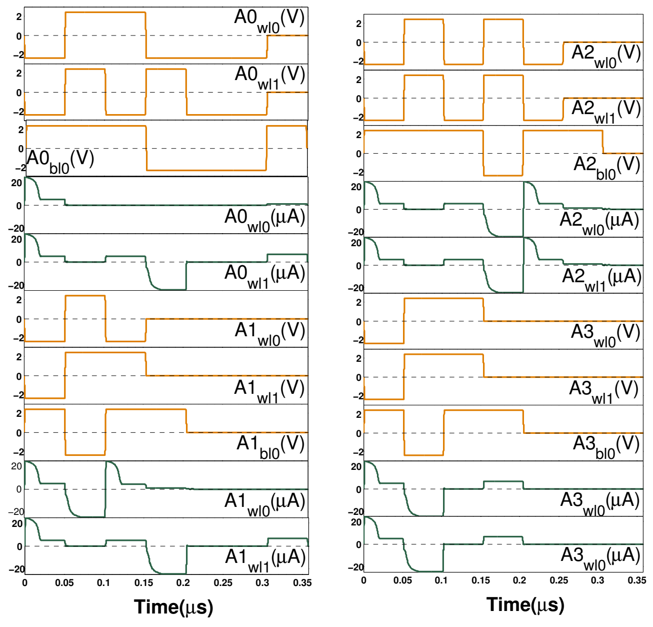


Fig. 6: Multiplexer Simulation Waveforms

[The orange waveforms represent the voltage applied to input lines (wordlines/bitlines). The green waveform represent the information stored in the device, read out in form of current.]

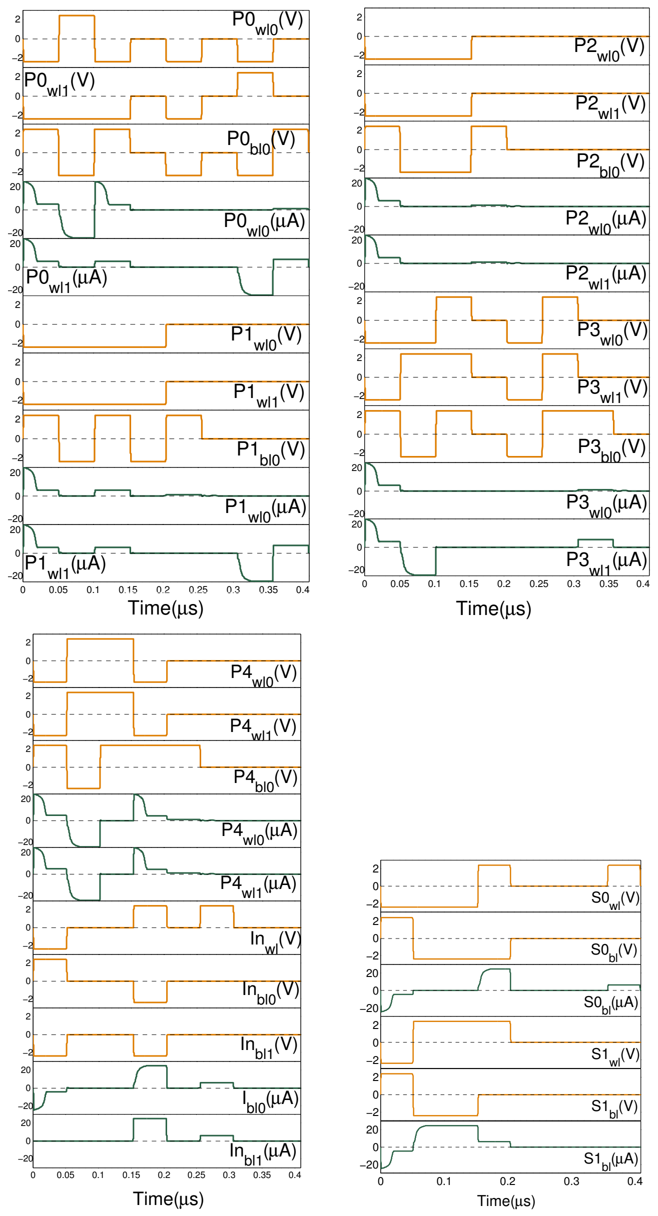


Fig. 7: Priority Multiplexer Simulation Waveforms