Optimized Implementation of Memristor-Based Full Adder by Material Implication Logic

Mehri Teimoory*, Amirali Amirsoleimani†‡, Jafar Shamshi**, Arash Ahmadi†,
Shahpour Alirezaee‡, Majid Ahmadi†

Email: {mehri_teimoory}@yahoo.com {amirisol,alirezee,ahmadi}@uwindsor.ca
{jafarshamsi}@elec.iust.ac.ir {ahmadi}@razi.ac.ir

*Department of Electrical Engineering, Islamic Azad University, Science and Research Branch, Kermanshah, Iran
†Department of Electrical and Computer Engineering, University of Windsor, Windsor, Ontario, Canada
‡Department of Electrical Engineering, Razi University, Kermanshah, Iran
**Department of Electrical Engineering, Iran University of Science and Technology, Tehran, Iran
††Department of Electrical Engineering, Islamic Azad University, Science and Research Branch, Kermanshah, Iran

Abstract— Recently memristor-based applications and circuits are receiving an increased attention. Furthermore, memristors are also applied in logic circuit design. Material implication logic is one of the main areas with memristors. In this paper an optimized memristor-based full adder design by material implication logic is presented. This design needs 27 memristors and less area in comparison with typical CMOS-based 8-bit full adders. Also the presented full adder needs only 184 computational steps which enhance former full adder design speed by 20 percent.

Index Terms—Memristor, Material Implication, Full Adder, CMOS, Artificial Intelligence.

I. INTRODUCTION

Logic computing has recently drawn wide attention due to emergence of memory-resistor or memristor devices [1]. Memristor’s non-volatility good scalability, low power consumption and compatibility with CMOS structures makes them an ideal device for various applications.

Logic operations with memristors are promising for nano computing circuits. Several approaches are presented for implementing logic circuits with memristors. Hybrid CMOS-memristor based logic and stand-alone memristor based logic are two main logic family [1]. Hybrid CMOS-Memristor based logic is based on integrating CMOS with memristors only for computational purpose. Memristor Ratioed Logic (MRL) is presented in [2]. In this family, logical states represented in voltage logic gates based on MRL are not capable to store their last output value. The other drawback in MRL design is its need to the conventional CMOS inverter gate for building the NAND and NOR gates. The stand-alone memristor based logics are more promising than hybrid peer. Crossbar architecture is also one of the family logic with memristor presented in [3], where the computation takes place in crossbar structure of memristors. Since in this family of logic the fabrication is simple and its high density make it to be a good candidate for future computing architectures. Other type of innovative memristor-based logic gates are developed and presented in [4] and [5], where the resistance represents logical state unlike MRL. Therefore they can store output of the gates as memристance of the output memristor. Both of these families can perform functions inside a crossbar array but they need a sequencer to bias memristors. Material implication which is known IMPLY is presented in [4][5].

This logic family executes basic Boolean operations by applying memristor based IMPLY logic gate.

In this paper a serial eight-bit memristor based full adder is implemented based on material implication (IMPLY) logic. This full adder is designed with 27 memristors and it needs 184 computational steps for completing its operation which is faster than previous designs [6-7].

This paper is organized as follows. Section II introduces IMPLY logic gate and its utilization in building different logic gates. Design and serial implementation of one-bit memristor based full adder as well as the proposed 8-bit adder are presented in section III. Consequently the section IV includes the conclusion of the paper.

II. MATERIAL IMPLICATION (IMPLY) LOGIC

[p] IMPLY [q] is a logic function that is called material implication. This logic statement means “p implies q” or “if p then q”. In Table 1 a truth table of material implication logic is presented. This logic is similar to answering a question. p is considered as a question and q as its answer. When p is wrong any answer results in a true output (logic 1) whether it is wrong or correct one. In other cases when p is true the answer truthfulness would determine output. Only case three of IMPLY logic truth table as it is shown in Table 1 has FALSE (logic 0) output. In this case a correct question (p = 1) has wrong answer (q = 0). Based on the output values for each cases of IMPLY logic this function is the equivalent to,

\[ p \text{ IMPLY } q = p \rightarrow q = (\neg p) \lor q \]

TABLE. 1. TRUTH TABLE OF IMPLY FUNCTION.

| Case | p | q | p IMPLY q |
|------|---|---|-------|
| 1    | 0 | 0 |   1   |
| 2    | 0 | 1 |   1   |
| 3    | 1 | 0 |   0   |
| 4    | 1 | 1 |   1   |

A. IMPLY logic gate

The memristor-based IMPLY logic gate constructed by two memristor (P and Q) and one resistor (R_G). The polarity of the
memristors is depicted in Fig. 1. Also the $R_G$ connects the memristors into the ground. Two voltage biases for IMPLY logic gate, $V_{\text{cond}}$ and $V_{\text{set}}$ are imposed into $P$ and $Q$ memristors doped side respectively for $P$ IMP $Q$ operation. The $V_{\text{cond}}$ voltage magnitude is lower than $V_{\text{set}}$ ($|V_{\text{cond}}| < |V_{\text{set}}|$). In this design the proposed IMPLY gate inputs are stored as resistance of the $P$ and $Q$ memristors. Interestingly, output of this gate is the final changed memristance value of $Q$ memristor. For setting input of IMPLY logic gate initial memristances of this memristors should be set based on the considered input values of each case showed in Table 1. For this purpose two bias sources are required to set initial memristance of memristors to $R_{\text{ON}}$ and $R_{\text{OFF}}$. For implementing logic zero memristors memristance should be in its highest value $R_{\text{OFF}}$. The memristor should be inversely biased. Therefore $V_{\text{clear}}$ voltage source is applied to set memristors initial memristances to OFF state. On the other hand, for setting logic one with memristor, its memristance should be changed to its lowest value $R_{\text{ON}}$. $V_{\text{set}}$ voltage source can be used for forward bias of the memristor. It should be mentioned both of these bias voltages should be higher than threshold voltage of the memristors. Also pulse width of the proposed voltage should be long enough for reaching the memristance of the memristor to the considered values ($R_{\text{ON}}$ or $R_{\text{OFF}}$). For different input combinations stated in Table 1 IMPLY logic gate produces different outputs. In case 1, the memristances of $P$ and $Q$ should be both set to $R_{\text{OFF}}$ initially. $R_G$, $R_{\text{ON}}$ and $R_{\text{OFF}}$ are assumed 10 kΩ, 1 kΩ and 100 kΩ respectively. For this both memristors are biased with $V_{\text{clear}}$, which is considered -1 V. Then $V_{\text{cond}}$ and $V_{\text{set}}$ biases are connected to $P$ and $Q$ memristors respectively. For simulation $V_{\text{cond}}$ and $V_{\text{set}}$ are 0.5 V and 1 V respectively. By imposing these voltages based on voltage division between $R_G$ ($R_G = 10$ kΩ) and resistance of memristor $Q$ ($R_{\text{OFF}} = 100$ kΩ), voltage of the node between two memristor and $R_G$ becomes negligible. The voltage across memristor $Q$ is,

$$V_Q = \frac{R_G}{R_{\text{ON}} + 2R_G} (V_{\text{set}} + V_{\text{cond}})$$  \hspace{1cm} (2)

This makes current flows from $V_{\text{set}}$ to ground. As the charge passes through memristor $Q$ its memristance is decreasing until it reaches to its minimum value $R_{\text{ON}}$. Therefore the final memristance ($R_{\text{ON}}$) of $Q$ is the desired output (logic 1) of the IMPLY gate for case 1 of the truth table. The case 1 is simulated for IMPLY gate with mentioned characteristic in Fig. 2. This case is determined the write time of the circuit.

In case 2, the memristances of $P$ and $Q$ should be set to $R_{\text{OFF}}$ and $R_{\text{ON}}$ respectively. This would be done by biasing $P$ and $Q$ memristors with $V_{\text{clear}}$ and $V_{\text{set}}$ respectively to allocate desired input to memristors. Then $V_{\text{cond}}$ and $V_{\text{set}}$ voltage bias sources should connect to $P$ and $Q$ to start IMPLY logic operation. Since memristance of memristor $Q$ ($R_{\text{ON}} = 1$ kΩ) is negligible in comparison with $R_G$ resistance ($R_G = 10$ kΩ), in this case voltage of the common node of memristors and resistor would be about 1 V. The voltage across memristor $Q$ in this case is determined through,

$$V_Q = V_{\text{set}} - \frac{R_G}{R_{\text{OFF}} + 2R_G} V_{\text{cond}} \approx 0$$  \hspace{1cm} (3)

Therefore the voltage across memristor $Q$ would be zero. As a result, the memristor $Q$ state is remained unchanged as its previous state (logic 1). Case 4 is similar to this case. Its only difference with case 2 is setting procedure for inputs of memristors. The voltage across memristor $Q$ is determined by,

$$V_Q = \frac{R_G}{R_{\text{ON}} + 2R_G} (V_{\text{set}} + V_{\text{cond}})$$  \hspace{1cm} (4)

In case 3, the memristance of $Q$ is set to $R_{\text{OFF}}$. After applying $V_{\text{cond}}$ and $V_{\text{set}}$ voltage bias sources the logic state remain unchanged. The voltage across the memristor $Q$ is,

$$V_Q = \frac{R_G}{R_{\text{ON}} + R_G} V_{\text{cond}} \equiv V_{\text{cond}}$$  \hspace{1cm} (5)

This voltage is not enough for changing the state of memristor. Although this voltage is forced the internal state of $Q$ to change to $R_{\text{ON}}$ (logic 1) it cannot be changed to its minimum value. This is called state drift. In this state final memristance of $Q$ becomes lower than initial input memristance ($R_{\text{OFF}}$).

B. Implementing logic functions by IMPLY logic

By combining IMPLY logic gate and FALSE gate any logic structure can be produced [4-5]. FALSE gate is a logic gate which yields zero as its output for any input. This circuit is consisting of three memristors for implementing NAND logic based on IMPLY and FALSE functions. $P$ and $Q$ are inputs and $S$ is output. The NAND function for $P$ and $Q$ is written by IMPLY and FALSE logic through,

$$S = Q \text{ IMP}(P \text{ IMP } S) = \sim Q \lor ((\sim P) \lor S)$$  \hspace{1cm} (6)

By allocating zero to $S$ then
\[ S = \neg Q \lor ((\neg P) \lor 0) = (\neg Q) \lor (\neg P) = \neg (P \land Q) \] (7)

This shows for the first step FALSE operation should be applied to memristor \( S \), which means \( V_{\text{clear}} \) should be applied to memristor \( S (\neg \text{IMP} \ 0) \). For implementing \( P \text{ IMP} \ S \), voltage pulses of \( V_{\text{cond}} \) and \( V_{\text{set}} \) are applied to memristors \( P \) and \( S \) respectively. Final step is producing output \( S = (Q \text{ IMP} S) \). For this step \( V_{\text{cond}} \) and \( V_{\text{set}} \) are applied to memristors \( Q \) and \( S \) respectively. The memristance of \( Q \) is considered as output of this logic. The truth table for each step is shown in Table 2 for NAND logic gate. Similarly other logic gates can be implemented based on this procedure. Different logic operations made through IMPLY logic and FALSE function is introduced in Table 3.

III. 8-BIT MEMRISTOR BASED FULL ADDER BASED ON MATERIAL IMPLICATION (IMPLY) LOGIC

In CMOS based 8-bit full adder two eight-bit inputs are entered and one eight-bit output number is extracted. Single CMOS 8-bit full adder comprises of 400 CMOS transistors. Several implementations were applied for designing 8-bit full adder based on material implication logic. The proposed method in [6] has used 712 computational steps and it also needs 29 memristors for implementing an 8-bit full adder. In [7] 27 memristors are used for implementing an 8-bit full adder with 232 computational steps. The proposed method is based on architecture which is depicted in Fig. 3, where the output of the proposed full adder is determined by,

\[ S = A \oplus B \oplus C_{\text{in}} \] (8)
\[ C_{\text{out}} = (A \cdot B) + (C_{\text{in}} \cdot (A \oplus B)) \] (9)

where \( A, B \) are inputs and \( C_{\text{in}} \) is carry in for one bit full adder. Also \( C_{\text{out}} \) and \( S \) are carry out and output of the full adder. In this design XOR, AND and OR logics should be implemented by material implication logic. For implementing XOR function unlike typical implementation of XOR logic, as mentioned in Table 3, two optimized equivalent IMPLY based XOR designs are proposed. These implementations are developed based on design requirements of the proposed serial full adders. These implementations have decreased computational steps of XOR logic. The proposed IMPLY based XOR implementations are,

\[ A \oplus B = (A \text{ IMP} B) \text{ IMP} \left( A' \text{ IMP} B' \right) \] (10)

A XOR B: FALSE(\( M_0 \)), \( A \text{ IMP} M_{18} \), FALSE(\( M_1 \)), \( B \text{ IMP} M_{1} \), \( A \text{ IMP} B \), \( M_0 \text{ IMP} M_{18} \), FALSE(\( M_0 \)), \( M_1 \text{ IMP} M_{18} \), \( B \text{ IMP} M_0 \)

It takes 9 computational steps.

\[ A \oplus B = \left( A' \text{ IMP} B \right) \text{ IMP} \left( A \text{ IMP} B' \right) \] (11)

A XOR B: FALSE(\( M_0 \)), \( A \text{ IMP} M_{18} \), FALSE(\( M_1 \)), \( B \text{ IMP} M_{1} \), \( M_0 \text{ IMP} B \), \( A \text{ IMP} M_{18} \), FALSE(\( M_0 \)), \( M_1 \text{ IMP} M_{18} \), \( B \text{ IMP} M_0 \)

It takes 11 computational steps. As it can be seen Eqn. 10 and

| Table 2. NAND gate Implementation Truth Table Based IMPLY Function. |
|---------------------------------------------------------------|
| Step 1: FALSE(\( S \)) | Step 2: IMP(\( S \)) | Step 3: IMP(\( S \)) |
|------------------------|------------------|------------------|
| \( S \) | \( P \) | \( S' \) | \( Q \) | \( S'' \) |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |

| Table 3. Logic Gates Implementation Based on IMPLY Function. |
|---------------------------------------------------------------|
| Logic Gate | Implementation based on IMPLY |
|------------------------|------------------|
| NOT \( P \) | \( P \text{ IMP} 0 \) |
| \( P \text{ NAND} Q \) | \( P \text{ IMP} (Q \text{ IMP} 0) \) |
| \( P \text{ AND} Q \) | \( (P \text{ IMP} (Q \text{ IMP} 0)) \text{ IMP} 0 \) |
| \( P \text{ OR} Q \) | \( (P \text{ IMP} 0) \text{ IMP} Q \) |
| \( P \text{ XOR} Q \) | \( (P \text{ IMP} Q) \text{ IMP} ((Q \text{ IMP} P) \text{ IMP} 0) \) |

Eqn. 11 need nine and eleven computational steps. In typical implementation in the first steps input memristors \( (A \text{ and } B) \) are used as inputs of IMPLY logic. This will change their memristances and logic values. Therefore these values should be copied in other memristors for using in subsequent steps. This procedure will result in additional computational steps. In Eqn. 10 and Eqn. 11 by changing IMPLY based XOR logic design there is no need for keeping input values since \( A \text{ and } B \) are required in subsequent steps. Therefore computational steps are reduced. Using this implementation one bit full adder is designed.

Figure 3: Logic schematic of the proposed design for eight-bit full adder.

Figure 4: Proposed memristor-based eight-bit full adder.
This full adder needs only 23 computational steps by applying 6 memristors. A and B are used as input memristors. M₁ and M₂ are applied as functional devices. Also C is used as both input and output. Carry in (Cᵢᵣₐ₃) value is stored in memristor C and its last value becomes carry out (Cᵢ₄ᵣₐ₅). S is the last value stored in Mₛ. For designing N-bit full adder with the proposed design, input memristors of previous determined bits are used as functional memristors for computing purpose of the next bits. This will decrease number of memristors dramatically. Also carry in of each bit is the last value of memristor C. Therefore for Cᵢᵢᵣ₃ and Cᵢ₄ᵣ₅ only one memristor is used for all bits of N-bit full adder. By this technique except first bit which needs six memristors other bits only required three memristors. For implementing 8-bit full adder 27 memristors are used in a row as it depicted in fig. 4. This full adder completes its task in 184 computational steps. It lowers 48 steps in comparison with 8-bit full adder designed in [7]. In Table 4 and Table 5 logics and input and output values are defined in each step.

IV. CONCLUSION

Material implication logic is a significant issue in digital system design. In this paper, an optimized serial implementation technique for 8-bit memristor based full adder is presented. This technique takes 23 computational steps for one-bit full adder. By using 27 memristors an 8-bit full adder is designed, which requires 184 computational steps to complete its task. This shows about 20 percent faster performance in comparison with the techniques presented in [6] and [7].

REFERENCES

[1] Massimiliano Di Ventra and Yuriy V. Pershin. Memcomputing: a computing paradigm to store and process information on the same physical platform. CoRR, abs/1211.4487, 2012.
[2] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Rationale Logic," Proceedings of the International Cellular Nanoscale Networks and their Applications, pp. 1-6, August 2012.
[3] I. Vourkas and G.C. Sirakoulis, “A novel design and modeling paradigm for memristor-based crossbar circuits.” Nanotechnology, IEEE Transactions on, 11(6):1151–1159, 2012.
[4] E. Lehtonen, J. Poikonen, and M. Laiho. Implication logic synthesis methods for memristors. In Circuits and Systems (ISCAS), 2012 IEEE International Symposium on, pages 2441–2444, May.
[5] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, Joshua J. Yang, Duncan R. Stewart, and R. Stanley Williams. ‘Memristive’ switches enable ‘stateful’ logic operations via material implication. Nature,464(7290):873–876, April 2010.
[6] E. Lehtonen and M. Laiho, “Stateful Implication Logic with Memristors,” Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures, pp. 33-36, July 2009.
[7] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, “Memristor-based material implication (imply) logic: Design principles and methodologies,” IEEE Transactions on Very Large Scale Integration (VLSI), vol. PP, pp. 1–13, 2013.

TABLE 4. VALUES AND LOGICS IN EACH STEP FOR PROPOSED DESIGN ONE-BIT FULLADDER.

| Step | Goal | FALSE and IMPLY Logic in each step | Parameters Value and Equivalent Logic | Output |
|------|------|-----------------------------------|--------------------------------------|--------|
| 0    | Setting Initial Values | -                                 | A and B and C is set to initial values |        |
| 1    | FALSE (M₁) | M₁ = 0                             |                                      |        |
| 2    | FALSE (M₂) | M₂ = 0                             |                                      |        |
| 3    | FALSE (M₃) | M₃ = 0                             |                                      |        |
| 4    | Copy A′ to M₂ | A → M₂ | A′ → M₂ |                                      |        |
| 5    | Copy B′ to M₃ | B → M₃ | B′ → M₃ |                                      |        |
| 6    | M₁ → B | A IMP B |                                      |        |
| 7    | A → M₁ | A IMP B′ |                                      |        |
| 8    | (A XOR B) | B → M₃ | M₃ → (A XOR B)′ |                                      |        |
| 9    | FALSE (A) | A → 0 |                                      |        |
| 10   | M₃ → A | A' → (A XOR B) |                                      |        |
| 11   | FALSE (B) | B → 0 |                                      |        |
| 12   | FALSE (M₅) | M₅ = 0 |                                      |        |
| 13   | C → M₆ | C |                                      |        |
| 14   | M₆ → A | C' IMP (A XOR B) |                                      |        |
| 15   | M₆ → A | C' IMP (A XOR B)' |                                      |        |
| 16   | M₆ → B | C IMP (A XOR B)′ |                                      |        |
| 17   | M₆ → B | C IMP (A XOR B)′ |                                      |        |
| 18   | A → B | S |                                      |        |
| 19   | FALSE (M₇) | M₇ = 0 |                                      |        |
| 20   | Finish S execution | B → M₁ | M₁ → (A XOR B XOR C) | S |        |
| 21   | FALSE (C) | C → 0 |                                      |        |
| 22   | M₇ → C | C |                                      |        |
| 23   | Finish Cᵢᵣ₆ execution | M₇ → C | C' → Cᵢᵣ₆ | Cᵢᵣ₆ |        |

TABLE 5. TRUTH TABLE OF MEMRISTOR-BASED 1-BIT FULL ADDER BASED ON IMPLY FUNCTION.

| MEMRISTORS | A | B | Cᵢᵣ₆ | M₁ | M₂ | M₃ | S |
|------------|---|---|------|----|----|----|---|
| INPUTS / OUTPUTS | INPUT 1 | INPUT 2 | Cᵢᵣ₆ | M₁ | M₂ | M₃ | S |
| STEPS | 0 | 10 | 15 | 6 | 12 | 17 | 18 | 0 | 21 | 22 | 23 | 1 | 8 | 9 | 16 | 2 | 5 | 7 | 3 | 4 | 13 | 14 | 19 | | 20 |
| CASE 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | |
| CASE 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | |
| CASE 3 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| CASE 4 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| CASE 5 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | |
| CASE 6 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | | |
| CASE 7 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| CASE 8 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |