Non-Volatile Look-up Table Based FPGA Implementations

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels, Mohammad Alfailakawi*
Laboratory of Computer Engineering, Delft University of Technology, the Netherlands
*Computer Engineering Department, University of Kuwait, Kuwait
Email: {L.Xie,H.A.DuNguyen,M.Taouil,S.Hamdioui,K.L.M.Bertels}@tudelft.nl;alfailakawi.m@ku.edu.kw

Abstract—Many emerging technologies are under investigation to realize alternatives for future scalable electronics. Memristor is one of the most promising candidates due to memristor’s non-volatility, high integration density, near-zero standby power consumption, etc. Memristors have been recently utilized in non-volatile memory, neuromorphic system, resistive computing architecture, and FPGA to name just a few. An FPGA typically consists of configurable logic blocks (CLBs), programmable interconnects, configuration, and block memories. Most of the recent work done was focused on using memristor to build FPGA interconnects and memories. This paper proposes two novel FPGA implementations that utilize memristor-based CLBs and their corresponding automatic design flow. To illustrate the potential of the proposed implementations, they are benchmarked using Toronto 20, and compared with the state-of-the-art in terms of area and delay. The experimental results show that both the area (up to 4.24x) and delay (up to 1.46x) of the novel FPGAs are very promising.

I. INTRODUCTION

As transistors gradually approach their inherent physical device limits, CMOS technology faces major challenges such as increased leakage, saturated performance gain, and reduced reliability [1,2]. To address these challenges, novel technologies, such as carbon nanotube, graphene transistors, and memristors [3]; are proposed as alternatives for future scalable electronics. Memristors are one of the most promising candidates due to their non-volatility, high integration density, and near-zero standby power [3,4]. Memristors-based design have been proposed for non-volatile memory [4], neuromorphic system, resistive computing architecture [5,6], and field programmable gate array (FPGA) [7–9].

Many novel memristor-based FPGAs, called MemFPGA for short, have been reported recently. MemFPGAs typically employ the classical island-style architecture [10] which consists of configurable logic blocks (CLBs), programmable interconnect, and block RAM (BRAM). Each CLB consists of look-up tables (LUTs) and a D flip-flop (DFF). Both CLBs and the programmable interconnect use memories to store configuration information. In MemFPGAs, memristors are utilized in the following fashion:

- As configuration memory for CLBs and interconnects [7]
- Used in the implementation of programmable interconnect [8].
- Implement BRAMs and DFFs [9].

In this work, we propose the use of memristors to improve the implementation of LUTs within FPGA, something that have not been done before. This paper proposes two FPGA implementations using memristor-based LUTs along with an appropriate Electronic Design Automation (EDA) process. The cost of both implementations in terms of area and delay is analyzed.

The rest of this paper is organized as follows. Section II briefly describes memristor-based logic. Section III presents the proposed FPGA implementation followed by an EDA flow in Section IV. Section V evaluates the proposed approaches and the paper is concluded in Section IV.

II. FUNDAMENTALS OF MEMRISTOR LOGIC

This section starts with the characteristics of memristors, followed by an overview of memristor logic, finally presents two memristor logic styles used in this work.

A. Electronic Characteristics of Memristor

Fig. 1 shows I-V characteristics of a typical memristor [4]. The memristor switches from one resistive state to another when voltage across the device is greater than its threshold voltage $V_{th}$. Otherwise, it stays in its current resistive state. The the high-to-low switching is referred to as SET ($R_L$) resistance, while low-to-high switching is referred to as RESET($R_L$).

B. Overview of Memristor Logic

There are four types of memristor logic that have been previously proposed, namely, threshold [11], majority [11], implication [12], and Boolean logic [13,14]. Since LUTs are commonly based on Boolean logic, we will limit our discussion to memristor-based Boolean logic. Memristor-based Boolean logic can be classified into two styles depending on how logic states are represented. One style uses high and low voltages to represent logic 1 and 0 as is referred to as memristor-ratioed logic (MRL) [13]. On the other hand, when
high/low resistance is to represent logic 1 and 0, then it is referred to as Resistive Boolean logic (RBL) [14]. Next, we describe MRL and RBL Boolean logic.

C. Memristor Ratioed Logic (MRL)

The basic gates in MRL are AND, OR, NAND and NOR [13]. Fig. 2 shows an example of a 2-input NAND gate consisting of two memristors and a CMOS inverter (an n-input NAND gate requires n memristors). When only one of the inputs is 1 (e.g., A=1, B=0, see Fig. 2(a)), a current flows through memristor \( M_A \) and \( M_B \) as indicated by the dash-lined resulting in \( \text{RESET} \)ting \( M_A \) and \( \text{SET} \)ting \( M_B \). The voltage of the floating nanowire \( V_x \approx \frac{V_{dd} V_{dd}}{V_{dd} + V_{dd}} \) is \( R_H \geq R_L \) [4], hence the output voltage \( V_o \) is \( V_{dd} \). Cases when both inputs are 1 or 0 can be analyzed similarly and are shown in Fig. 2(b)-(c).

D. Resistive Boolean Logic (RBL)

In RBL, the basic logic gates are NAND, copy, invert (INV), and AND and are shown in Fig. 3 [14]. To illustrate working principle of RBL, a two-input NAND gate is used as an example. A 2-input NAND gate consists of two input memristors (\( M_A \) and \( M_B \)), an output memristor (\( M_o \)), and a resistor \( R_s \) (\( R_L \ll R_s \ll R_H \)). The output memristor must be \( \text{RESET} \) to \( R_H \) before each operation and input ones must be pre-programmed before execution (for brevity, this initialization is not shown). To perform an NAND operation, control voltages \( V_{dd} \) and \( V_{dd} \) \((\frac{V_{dd}}{2} < V_{dd})\) are applied to input and output memristors, respectively. In the case when \( A=1 \) and \( B=0 \), \( M_A = R_H \), and \( M_B = R_L \) (see Fig. 3(a)), the voltage on floating nanowire \( V_x \approx \frac{V_{dd}}{2} \) is \( R_L \ll R_s \ll R_H \) resulting in the voltage across \( M_o \) to be \( V_{mo} \approx \frac{V_{dd}}{2} \ll V_{dd} \), rendering \( M_o \) to stay in \( R_H \) state. Case when input \( A=B=1 \) can be analyzed similarly and is shown at the bottom of Fig. 3(a). Fig. 3(b) shows other gates which work in the similar way as the 2-input NAND gate.

III. TWO FPGA USING MEMRISTOR LOGIC

This section first briefly describes island-style FPGA architecture then presents MRL and RBL based FPGA architectures.

A. Island-Style FPGA Architecture

Fig. 4(a) shows the island-style FPGA architecture [10] which consists of CLBs, connection boxes (CBs), switching boxes (SBs) and BRAMs. Each CLB is composed of a switch matrix and \( N \) basic logic elements (BLEs) as shown in Fig. 4(b). The switch matrix contains multiple MUXes configured by SRAM bits to route \( I \) shared inputs and feedback \( N \) outputs among BLEs. A single \( K \)-input BLE contains a LUT, DFF, MUX and configuration memories (i.e., SRAM) as shown in Fig. 4(c).

\[ f = c_1 x_1 x_2 + c_2 x_1 x_2 + c_3 x_1 x_2 + c_4 x_1 x_2 \]
\[ \text{where } x_i (i=1,2) \text{ are the inputs and } c_i (1 \leq i \leq 4) \text{ are configuration bits.} \]

The LUT can implement a \( K \)-input Boolean function as determined by the configuration memories. The LUT can switch between sequential and combinational mode using the DFF and MUX. CBs and SBs constitute the programmable interconnect to route the signals among CLBs.

The reminder of this section presents two novel FPGAs. The MRL based implementation will be referred to as MFGPA whereas RBL based one will be labeled as RFGPA.

B. MFGPA

MFGPA uses MRL to implement the LUT and switch matrix (SM) of the CLBs while still using CMOS to implement the DFF and 2:1 MUX of BLEs. The output \( f \) of a 2-input LUT can be expressed by Eq.1:

\[ f = c_1 x_1 x_2 + c_2 x_1 x_2 + c_3 x_1 x_2 + c_4 x_1 x_2 \]
and \( c_1 \) with the first row while keeping other memristors on the same row disabled (by not electroforming them [4]). Note that the disabled junction is permanently in a high resistance \( R_D \gg R_H \) [14]. Configuration bit \( c_1 \) is stored in a 3-Transistor-2-Memristor (3T2R) cell as shown in Fig. 5(b) [7]. Two memristors \( M_1 \) and \( M_2 \) form a voltage divider and inverter output \( V_i \) is used as value for \( c_1 \). To configure \( c_1 \) to 0, \( M_1 \) and \( M_2 \) should be programmed to \( R_T \) and \( R_H \) using \( V_{dd} \). During execution stage, \( V_i \) is applied resulting in \( V_{ci} = 0 \) outputted. The case of \( c_1 = 1 \) works in a similar fashion.

The output \( f \) of an N:1 MUX (e.g., \( N = 2 \)) used in SM can be expressed by Eq.2 where \( x_i \) (1 \( \leq \) \( i \) \( \leq \) 2) presents the inputs and \( c_1 \) configuration bit. Fig. 5(c) shows an MRL-based 2:1 MUX as an example. It works similarly as MRL LUT.

\[
f = x_1 c_1 + x_2 c_1 = x_1 c_1 \cdot x_2 c_1 \tag{2}
\]

C. RFPGA

In RFPGA, RBL is used to implement BLEs while it uses the same switch matrix used in MFPGA. The output \( f \) of a \( K \)-input LUT can be expressed by Eq.1; e.g., \( K = 2 \).

\[
f = c_1 x_1 x_2 + c_2 x_1 x_2 + c_3 x_1 x_2 + c_4 x_1 x_2 \tag{3}
\]

Fig. 6(a) shows an RBL implementation of 2-input LUT. The first two rows are used to invert \( x_i \) to \( \bar{x}_i \) (\( i = 1,2 \)) whereas the following four rows implement the four terms in Eq.3 by mapping four NAND gates of Fig. 3(a) on the crossbar. For instance, the expression \( c_1 x_1 x_2 \) is implemented by row 3 where three memristors are placed at columns \( \bar{x}_1 \), \( x_2 \) and \( c_1 \) junctions representing inputs while a memristor is placed on column \( f \) junction signifying the output. The remaining junctions in row 3 are disabled. To calculate output \( f \), an AND gate is mapped on column \( f \) where it reuses the output of the four NAND gates as input hence storing value at the memristor at the junction of row and column \( f \).

To control the crossbar of the BLE, multiple voltage drivers and a CMOS controller are employed. A voltage driver is attached to each nanowire (triangles in Fig. 6(a)) and is used to control the voltage on the nanowires. The controller has two modes of operation, configuration (CFG) and execution (EXE) as described by the state machine shown in Fig. 6(b). The CFG mode consists of two states:

1) RSC: Activate all configuration bits by RESET all memristors to \( R_H \).
2) WR0: Deactivate configuration bits that do not contribute to function implemented (set memristors to \( R_L \)).

The EXE mode consists of seven states:

1) RSL: RESET all memristors to \( R_H \) except output \( f \) and configuration bits (i.e., \( c_i \), 1 \( \leq \) \( i \) \( \leq \) 4).
2) TRO: Transfer output \( f \) to the next BLEs, while receive all inputs of the LUT.
3) RSO: RESET the memristor that stores output \( f \).
4) INV: Invert inputs \( x_i \) to \( \bar{x}_i \) (\( i = 1, 2 \)).
5) CPY: Copy inputs to all NAND gates.
6) NAND: Execute all NAND gates to calculate the items.
7) AND: Execute an AND gate to calculate output \( f \).

To perform the operation of each state in the state machine, control voltages as indicated in Table I needs to be applied to the various nanowires. For instance, during CPY state, all inputs (\( x_i \) and \( \bar{x}_i \)) are copied to all NAND gates by applying \( V_{dd} \) to row INV (row 1–2) and \( V_{dd} \) to NAND (row 3–6) while simultaneously column IN (\( x_i \)) and INN \( \bar{x}_i \) are floating. In order to reduce the impact of sneak path currents on the BLE’s robustness, \( \frac{V_{dd}}{2} \) is applied to rows and columns that are not involved in the operations [4,14].

IV. AN EDA FLOW FOR PROPOSED FPGAS

This section presents a modified EDA flow for the proposed FPGA architectures and evaluate them in term of area and delay.
A. Modified EDA Flow

To automatically implement circuits using the proposed FP-GAs, we modify the standard EDA flow for CMOS FPGA [10] as shown in Fig. 7 where shaded blocks identify modification. Circuit design is first optimized at logic level and then mapped to LUTs using ABC [15]. Thereafter, LUTs are packed into CLBs using T-VPACK [16] where each CLB consists of one or more LUTs depending on configuration setting. Next, CLB netlist is placed and routed using VPR [16]. Finally, the entire FPGA consisting of CLBs and the routing architecture (including CBs and SBs) are estimated in terms of area and delay. To estimate the performance of the proposed architectures, we modify the performance estimation block by adding area and delay models for memristor-based CLBs. As the FPGA architecture is regular, FPGA performance estimation block only need to sum up the area and delay of CLBs and routing architecture [10]. The rest of this section presents area and delay model for the proposed CLBs.

B. Area Model of Memristor CLBs

CLBs of MFPGA and RFPGA can be implemented by a memristor crossbar stacked on top of a CMOS circuit as shown in Fig. 8 [3,4]. Hence, the area of a single CLB (A_{clb}) is estimated by the maximum area of the memristor crossbar (A_{xbar}) and CMOS circuit (A_{cmos}) as given by Eq.4. Crossbar area is estimated as the product of number of junctions (N_{junction}) by the area of a single junction (A_{junction}).

\[
\begin{align*}
A_{clb} &= \max\{A_{xbar}, A_{cmos}\} \\
A_{xbar} &= N_{junction} \cdot A_{junction}
\end{align*}
\]  

(4)

The CMOS part of MFPGA CLB contains inverters, DFFs, and 3T2R memory cells, hence can be represented as the summation of each component’s area as expressed in Eq.5.

\[
A_{cmos,MFPGA} = \sum A_{inv} + \sum A_{memory} + \sum A_{diff}
\]  

(5)

where A_{inv}, A_{memory}, and A_{diff} are the areas of a CMOS inverter, 3T2R memory cell, a DFF, respectively.

The CMOS part of RFPGA CLB contains voltage drivers and controller and thus its area can be expressed in Eq.6.

\[
A_{cmos,RFPGA} = A_{ctrl} + \sum A_{vd}
\]  

(6)

where A_{ctrl} presents the area of the controller and A_{vd} is that of a single voltage driver. The area of the controller can be estimated by a synthesis tool (e.g., Cadence RTL compiler).

Fig. 9 shows a possible implementation of a voltage driver consisting of three pass transistors [17] controlled by three-bit signals C[0:2]. To drive a nanowire with multiple memristors connected as shown on the right side of Fig. 9, the transistors should provide enough current to drive such wires. Therefore, transistors width-to-length ratio W/L should be carefully determined. To program a single memristor, the transistor must supply a current greater than I_a = \frac{V_{dd}}{R_a} [9]. The area of a transistor is typically A_{tr} = 6F^2 [7] where F is the feature size of CMOS technology. To drive n_a active memristors in parallel, W/L should be increased n_a times in order to provide the required current I_w. As a result, A_{tr} of the transistor increases n_a times as given in Eq. 7.

\[
I_w = n_a \frac{V_{dd}}{R_L} \quad \text{and} \quad A_{tr} = 6n_aF^2
\]  

(7)

Further, assume that we have another n_d disabled junctions with each consuming current equal to I_D = \frac{V_{dd}}{R_D}, then the transistor should also compensate for the current through n_d disabled memristors as described in Eq.8.

\[
\begin{align*}
I_w &= n_a \frac{V_{dd}}{R_L} + n_d \frac{V_{dd}}{R_D} = (n_a + R_L R_D n_d) \frac{V_{dd}}{R_L} \\
A_{tr} &= 6(n_a + \frac{R_L}{R_D} n_d)F^2
\end{align*}
\]  

(8)

Finally, the total area A_{vd} of a single voltage driver is as given in Eq.9. Typically, \frac{R_L}{R_D} > 5 \times 10^4 [18] and hence the number of memristors n_a dominates the area of the driver.

\[
A_{vd} = 3A_{tr} = 18(n_a + \frac{R_L}{R_D} n_d)F^2 \approx 18n_aF^2
\]  

(9)
C. Delay Model of Memristor CLBs

The delay of MFPGA CLB is determined by the critical path from inputs to outputs similar to CMOS circuit. The critical path contains an SM MUX, an LUT, a DFF, and a 2:1 MUX. The DFF and 2:1 MUX are implemented using CMOS and the remaining components are implemented using MRL. The delay of SM is modelled by Eq.10.

$$D_{mrl} = 2D_{nw} + 2T_{sw} + D_{nw,row} + D_{nw,col}$$  \hspace{1cm} \text{(10)}$$

where $D_{inv}$ represents the delay of a CMOS inverter, $T_{sw}$ is the switching time of a memristor device, and $D_{nw,row}$ ($D_{nw,col}$) is the delay of a row (column) nanowire. Note that memristors driven by $V_{DD}$ first switch $R_1$ and then memristors driven by $GND$ switch to $R_H$ or vice versa (see Fig. 2(a)) [13], therefore, memristor devices need $2T_{sw}$ to switch. A row or column nanowire is modelled as a transmission line as shown in Fig. 10 and its delay $D_{nw}$ is formulated by Elmore model [19] as given in Eq.11.

$$D_{nw} = \sum_{i=1}^{n} \left[ C_i \left( \sum_{j=1}^{i-1} R_j \right) \right]$$

$$= (n^2 + 4n - 21/8) R_{nw} \cdot C_{nw} \cdot F^2$$

$$n : \text{number of junctions in the nanowire}$$

$$R_i = \frac{2F \cdot R_{nw}}{1 < i \leq n}$$

$$C_i = \frac{3F \cdot C_{nw}}{2}$$

$$C_{nw} = \frac{3F \cdot C_{nw} + 3F \cdot C_{nw}}{2}$$

R FPGA CLB contains several BLEs based on RBL and an SM based on MRL. The delay of SM is modelled by Eq.10. The delay $D_{ble}$ of the BLE of Fig. 6 is modelled by Eq.12.

$$D_{ble} = N_{step} \cdot D_{step} = 7 \cdot D_{step}$$

$$D_{step} = D_{xbar} + D_{ctrl}$$

$$D_{xbar} = T_{sw} + D_{nw}$$

where $D_{ble}$ is the product of execution step number $N_{step}$ and the delay of a single step $D_{step}$. $D_{step}$ is the sum of crossbar delay $D_{xbar}$ and that of the CMOS controller $D_{ctrl}$. The nanowire delay $D_{nw}$ is modelled by Eq.10 and $D_{ctrl}$ is provided by the synthesis tool (e.g., Cadence RTL Compiler).

V. EVALUATION

To evaluate the performance of proposed architectures, a benchmark suite was synthesized and the area and delay characteristics were compared with state-of-the-art FPGA. First, will present experimentation setup, followed by results and discussion, and finally limitations of this work.

A. Experiment Setting Up

Table II summarizes the characteristics of all FPGAs used in this sections. FPGA that employs SRAM as configuration memories is used as baseline implementation. In addition to traditional SRAM based FPGAs, an MemFPGA that replaces SRAM of CLB with 1T1R as described in [20] was implemented and compared to the proposed architectures. All FPGAs in Table II can use routing architecture based on either CMOS [10] or RRAM [7,8]. To highlight the improvement of CLBs, all FPGAs use the same CMOS programmable interconnect.

This experiment uses classical island-style FPGA architecture [10] and Toronto 20 benchmark package [21] which consists of 20 benchmark circuits frequently used in different domains. Different numbers of LUT inputs ($K=3$–$8,10,12$) and numbers of LUTs within a CLB ($N=1,8$) are evaluated using area and delay model described in Section IV. All area and delay various reported are the average for all benchmark circuits synthesized. Since circuit ‘tseng’ of Toronto 20 cannot be synthesized correctly by ABC, it was not included in this experiment. Table III summarizes parameters of FPGA architecture and technology used in the experiments.

B. Results

Area Fig. 11 shows the area required for all four FPGAs. MFPGA and RFPGA typically need smaller area to implement CLBs (see Fig. 11(a)) whereas all need almost similar area to implement their routing architectures. Overall, MFPGA and
RFPGA outperform the baseline and MemFPGA as logic area dominates the total area. Therefore, MFPGA and RFPGA provide a great potential to improve logic integration density. **Delay** Fig. 12 shows the delay of all four FPGAs. The delay of the CLBs within MFPGA is similar to baseline and MemFPGA in case $K \geq 6$. On the other hand, the delay of CLBs within RFPGA are longer than others as each CLB needs several steps to complete its function. The delay of routing architecture in MFPGA and RFPGA are less than the other two FPGAs. Overall, MFPGA performs better than others in case $K \geq 6$ in all three clustering configurations. It is worth noting that each CLB of RFPGA can store data at run time, and hence its LUTs can be pipelined to improve its throughput.

In addition, MFPGA and RFPGA can be further improved if they incorporated with memristor-based routing architectures (e.g., [8]).

C. Limitations

This paper did not estimate power consumption of the proposed FPGAs as power modelling of memristor logics are still not mature [4]. Nevertheless, the proposed FPGAs may consume less power as memristors are non-volatile and hence they may consume less leakage power [3,4]. In addition, as this paper mainly illustrates the potential of FPGAs using memristor logic, technology challenges such as limited endurance, process variations [4] are out of the scope of this paper. These limitations will be studied in our future work.

VI. CONCLUSION

This paper proposed two novel FPGA implementations based on memristor logics. Their performances are intensively evaluated. Compared to the state-of-the-art, the proposed FPGAs provide a potential to improve the logic integration density, and possibly to reduce the delay. Hence, they are promising candidates for the future FPGA design and applications.