Reconfigurable RTD-based Circuit Elements of Complete Logic Functionality

Yexin Zheng and Chao Huang
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061
{yexin, chaoh}@vt.edu

Abstract—**Resonant tunneling diodes (RTDs) have demonstrated promising circuit characteristics of high speed switching property and versatile functionality with negative differential resistance (NDR). In this paper, we propose novel programmable logic elements (PLEs) that can be configured to realize all three- or four-input logic functions. These simple RTD-based circuit elements are implemented with threshold gates (TGs) and multi-threshold threshold gates (MTTGs) by employing programmable monostable-bistable logic element (MOBILE) principles. We also developed a dynamically reconfigurable scheme based on our PLE structures which facilitate nanopipelining without incurring delay overheads.**

I. INTRODUCTION

Although traditional silicon electronics will continue the dominance for the next 10-15 years [1], innovative nanoscale device research advances have visualized great opportunities to surpass the physical barriers of current CMOS technology and continue the projection by Moore's Law [2], [3]. Resonant tunneling diode (RTD) devices have shown promising circuit characteristics in improving both analog and digital circuits. Various RTD models have been proposed [4], [5], [6] and RTD devices and circuits have been reported working at a frequency of several GHz [7], [8], [9]. RTD-CMOS hybrid circuit prototype and integration process [10], [11] were developed to yield higher speed and lower power design and fabrication over pure CMOS circuits.

To harness RTD's ultrafast switching speed, compact and high speed logic circuits are designed by using RTDs in conjunction with heterostructure field effect transistors (HFETs). In [12], [13], a RTD/HFET threshold logic circuit, called MOnostable-BIstable transition Logic Element (MOBILE), achieved more complex functionality with smaller area and lower power consumption. Both synthesis and automatic test pattern generation methodologies have been established targeting on MOBILE-based threshold logic networks [14], [15]. Moreover, the intrinsic latching property of MOBILE devices enables the implementation of nanopipeline architectures [16], [17].

In this paper, we propose the design of our novel programmable logic element (PLE) structures implemented with programmable MOBILE threshold gates (TGs) and multi-threshold threshold gates (MTTGs). The proposed PLEs are proved of complete logic functionality and an efficient configuration bits generation algorithm for PLE structures is also constructed. By adapting a nanopipeling scheme, PLE structures can support dynamic reconfigurability without incurring delay overheads. The contributions of this work are highlighted as follows:

- The simple and novel three- and four-input PLE structures are developed, which consist of three novel programmable gates and two primitive functional gates based on MOBILE TGs and MTTGs. The design simulation testifies functional correctness.
- Compared with [13], our circuit configuration is proved to be able to realize all the logic functions through properly setting the control bits which can be obtained by an effective encoding scheme. Furthermore, only five control bits need to be configured to realize a three-input logic function. This is more compact and efficient than a general look-up table (LUT) solution which requires eight configuration bits.
- By adapting a nanopipelining scheme, the PLE structure is designed to support dynamical reconfiguration without delay overheads. Comparisons between three- and four-input PLE implementations provide an insightful view of design tradeoff.

The rest of this paper is organized as follows. Section II introduces the preliminary concepts and background materials. section III presents the novel PLE topologies. Section IV proves the logic completeness of the PLE's functionality. Section V introduces the algorithm to generate configuration bits. Section VI discusses the dynamic reconfigurability of PLE structures. Section VII demonstrates our experimental results. Finally Section VIII concludes.

II. BACKGROUND

In this section, we introduce some preliminary concepts, specifically, the MOBILE circuit, threshold function, and clocking scheme for nanopipelining.

A. Monostable-bistable transition logic element

The basic MOBILE circuit exploits the negative differential resistance (NDR), an important feature of the RTD's I-V characteristics (see Fig. 1(a)). It consists of two RTD devices (load and driver RTD) connected in series as shown in Fig. 1(d). Driven by a bias voltage V_{CLK} which oscillates between 0V and V_{DD}, the MOBILE circuit switches between a monostable state (S_0 in Fig. 1(b)) and a bistable state (S_1 or S_2 in Fig. 1(c)). The resulting state of a bistable MOBILE circuit depends on the RTD's peak current I_p: the RTD with a smaller peak current will switch to a high resistance state when V_{CLK} increases. For example, if the driver RTD has a lower peak current, the MOBILE becomes stable at state S_2 after the transition and generates a logic high output $V_{out} = 1$. Combined with RTD/HFET branches which are used to modulate the peak currents, MOBILE circuit can implement TGs and more complex MTTGs.

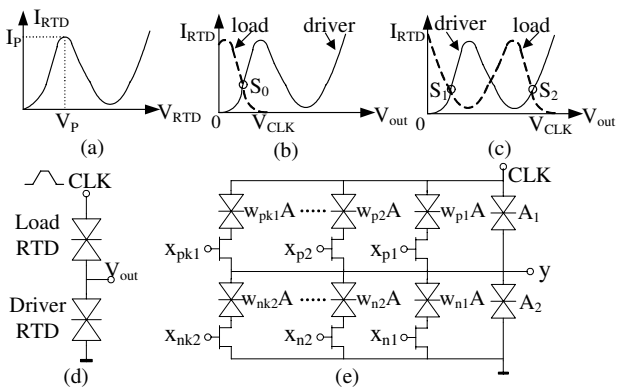

Fig. 1. (a) RTD I-V characteristics, (b) MOBILE operating principle in monostable, (c) MOBILE operating principle in bistable, (d) basic MOBILE circuit, and (e) a generic MOBILE TG

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

B. Threshold and multi-threshold threshold gates

A TG [18] is defined as a logic gate with n binary input variables $\{x_i\}$ (i=1,2,. . .,n), a set of n positive or negative weights $\{w_i\}$ (i=1,2,. . .,n), and a numerical threshold T such that the binary output is 1 when $\sum_{i=1}^{n} w_i x_i \geq T$ and 0 otherwise. This threshold gate can also be denoted by a weight-threshold vector $[w_1, w_2, \ldots, w_n; T]$.

Fig. 1(e) illustrates a generic TG topology based on a RTD/HFET implementation [19], in which current controlling branches are connected in parallel with the MOBILE RTDs (load area A_1 and driver area A_2). The current controlling branches consist of a series combination of an RTD and HFET, where A is the unit RTD area and the weights w_{pi} (i=1,2,. . .,$k1$) and w_{nj} (j=1,2,. . .,$k2$) are determined by the RTD areas. The HFETs behave like switches with x_{pi} (i=1,2,. . .,$k1$) and x_{nj} (j=1,2,. . .,$k2$) as the positive and negative binary inputs, respectively. The MOBILE TG can be simplified to the basic model (Fig. 1(d)) with an equivalent load and driver RTD whose corresponding peak current can be computed as $(\sum_{i=1}^{k1} x_{pi} w_{pi} A + A_1) I_{pd}$ and $(\sum_{j=1}^{k2} x_{nj} w_{nj} A + A_2) I_{pd}$, respectively. Assuming that the peak current density I_{pd} is identical for both load and driver RTDs, the RTD's peak current is proportional to its area. Therefore, the functionality of the MOBILE circuit can be simply determined by the equivalent RTD sizes. The generic MOBILE TG shown in Fig. 1(e) implements the threshold function $[w_{p1}, w_{p2}, \ldots, w_{pk1}, -w_{n1}, -w_{n2}, \ldots, -w_{nk2}; T]$, where the threshold $T = (A_2 - A_1)/A$.

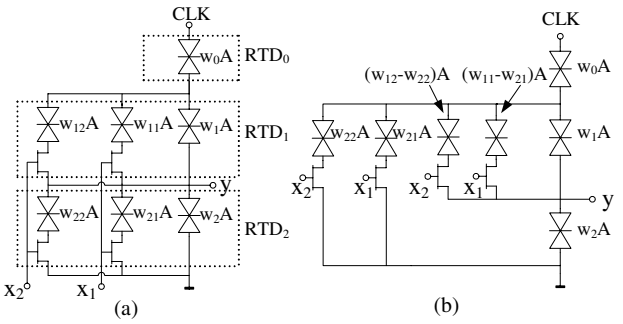

Fig. 2. MOBILE MTTG: (a) basic topology and (b) improved topology

The concept of RTD/HFET TG design can be further extended to implement MTTGs by connecting three or more RTDs in series [20], as shown in Fig. 2(a). Because of the same circuit operating principles as TGs, different MTTG functions can be designed by adjusting the RTD areas to obtain the required current relationship among different equivalent RTDs. A programmable MTTG gate can be achieved by using some of the inputs as control bits [13] to realize different logic functions. Fig. 2(b) demonstrates an alternative circuit topology of the same logic function as the circuit in Fig. 2(a). This alternative implementation can achieve a smaller circuit area and consume less power [21]. The RTD-based circuits proposed in this paper use this improved circuit topology.

C. MOBILE clocking scheme

MOBILE circuits are inherently self-latching as they can preserve the output values when bias voltage V_{CLK} is set to high. Therefore, this property together with a proper clocking scheme can enable nanopipelining operations [17]. A four-phase clocking scheme was introduced in [19] to operate cascaded MOBILE circuit stages (see Fig. 3). In this scheme, each clock period T is divided to four phases with an equal time interval of $T/4$. Phase A is the evaluation phase during which the gate switches from monostable to bistable and evaluates the output. In phase B, the gate holds the result. In the reset phase C, the load capacitor is discharged and the gate returns to its initial monostable state. The

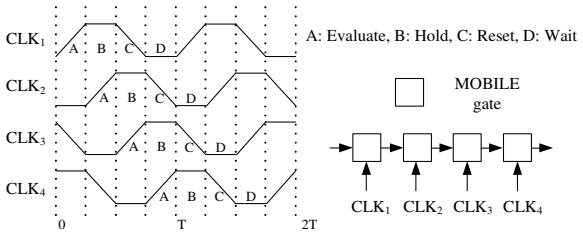

Fig. 3. Cascaded MOBILE circuits and four-phase clocking scheme

gate is inactive in the wait phase D. So in the clocking scheme illustrated in Fig. 3, each clock is delayed by $T/4$ from the previous one to safeguard that the evaluation of a gate only starts after the output of its previous gate becomes valid.

III. PROGRAMMABLE LOGIC ELEMENT

Based on the multi-threshold function implemented by RTD/HFET MTTG topology, we present our three- and four-input PLE structures in this section.

A. Three-input PLE

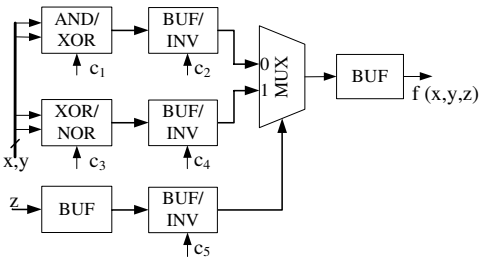

Fig. 4. Three-input PLE

The three-input PLE implementation is shown in Fig. 4, which can realize all 256 logic functions by setting the control bits $\{c_1, c_2, \ldots, c_5\}$. The logic completeness and configuration details will be addressed in Sections IV and V, respectively.

The three-input PLE is composed of three programmable gates, AND/XOR, XOR/NOR, and BUF/INV, and two primitive functional gates, MUX (multiplexer) and BUF (buffer). The programmable gates and MUX that we have designed use the improved MTTG topology introduced in Section II-B, and BUF is implemented by a basic RTD/HFET TG [21]. The gate designs are illustrated in Fig. 5(a)-(e). The area of each RTD is given in the figures and A denotes the unit RTD area. Each of these three programmable gates can realize two different boolean functions depending on the value of the control bit. For example, gate AND/XOR shown in Fig. 5(a) has x_1 and x_2 as inputs, c as control bit, and y as output. When $c = 0$, gate AND/XOR acts as a logic *AND*, otherwise an *XOR*. The selection of the logic functions of the programmable gates by the control bit is presented in Table I.

TABLE I
LOGIC FUNCTION SELECTION OF PRIMITIVE PROGRAMMABLE GATES

Control bit	Programmable gates			
	AND/XOR	XOR/NOR	BUF/INV	MUX
$c = 0$	$y = x_1 x_2$	$y = x_1 \oplus x_2$	$y = x_1$	$y = x_1$
$c = 1$	$y = x_1 \oplus x_2$	$y = \overline{x_1 + x_2}$	$y = \overline{x_1}$	$y = x_2$

In the PLE shown in Fig. 4, variable z selects an x-y function branch based on its positive or negative phase through a MUX. A BUF is used for z to patch up the signal path to two stages to synchronize data arrivals at the evaluation phase of the MUX. Another buffer is inserted at the output of the MUX in order to complete a whole four-phase clock cycle. Fig. 5(f) presents the HSPICE simulation results of all the MOBILE gates that we have designed.

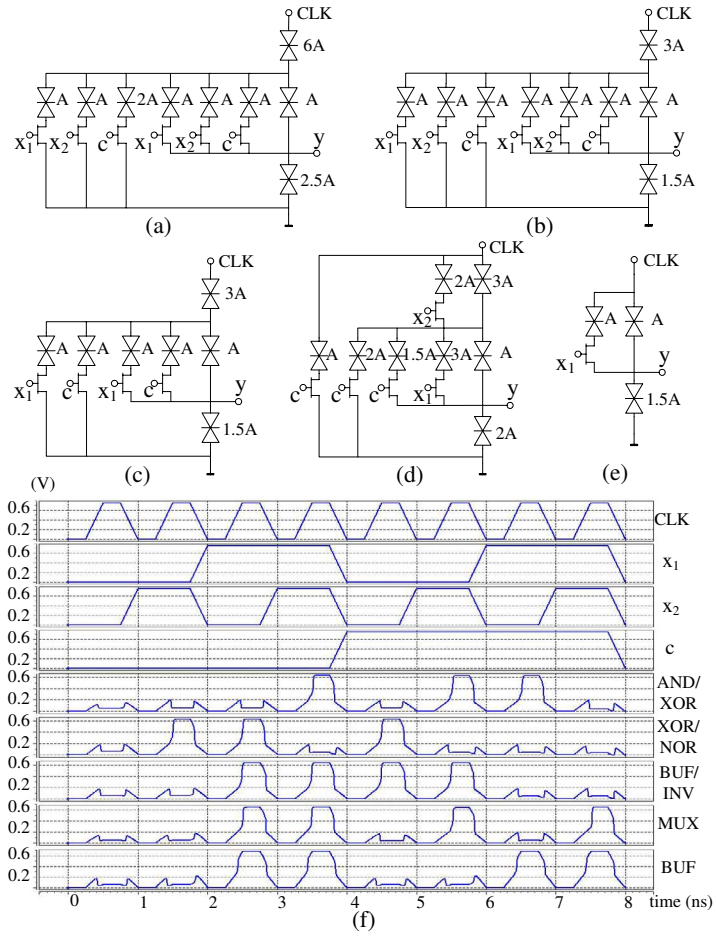

Fig. 5. PLE MOBILE gate designs and HSPICE simulation: (a) AND/XOR, (b) XOR/NOR, (c) BUF/INV, (d) MUX, (e) BUF, and (f) HSPICE simulation

B. Four-input PLE

In our three-input PLE design, a BUF is added at the MUX output to serve as the fourth nanopipelining stage to adapt the four-phase clocking scheme. We also designed a four-input PLE as an alternative approach to fit in the four clock phases, as shown in Fig. 6, by duplicating two structures of the first three stages of the three-input PLE and connecting them to a MUX which serves as a fourth stage. In this manner, a fourth input w is added (as the final MUX selection) to implement all four-input logic functions. The comparisons of three- and four-input PLE implementations are discussed in Section VII.

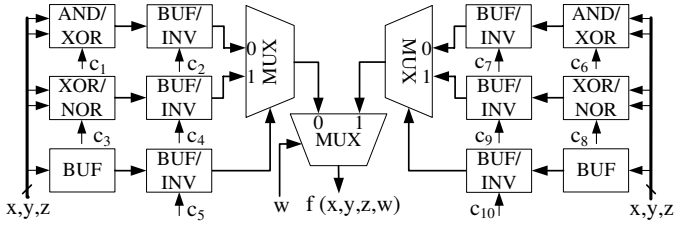

Fig. 6. Four-input PLE

IV. LOGIC COMPLETENESS

The ability of realizing all 256 three-variable logic functions $f(x, y, z)$ by using our three-input PLE is not as obvious as a three-input LUT. In order to prove the logic completeness, we first introduce Shannon Expansion.

Shannon Expansion: given an n-variable boolean function $f(x_1, x_2, \ldots, x_n)$, we have

$$f(x_1, x_2, \ldots, x_n) = \bar{x}_i \cdot f_{\bar{x}_i} + x_i \cdot f_{x_i}$$

$$f_{\bar{x}_i} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

$$f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

$$\forall x_i, \quad i = 1, 2, \ldots, n$$

where $f_{\bar{x}_i}$ and f_{x_i} are, respectively, the negative and positive Shannon cofactors of $f(x_1, x_2, \ldots, x_n)$ with respect to variable x_i. We define the literal set $\mathbf{L}$ as

$$\mathbf{L} = \{l_i\} = \{\bar{x}_1, x_1, \bar{x}_2, x_2, \ldots, \bar{x}_n, x_n, \}, \quad i = 1, 2, \ldots, 2n$$

Accordingly, the cofactor set $\mathbf{F}$ is defined as

$$\mathbf{F} = \{f_i\} = \{f_{\bar{x}_1}, f_{x_1}, f_{\bar{x}_2}, f_{x_2}, \ldots, f_{\bar{x}_n}, f_{x_n}, \}, i = 1, 2, \ldots, 2n$$

Shannon Expansion can be implemented by simply using a MUX with x_i as the select bit and the positive and negative cofactors connected to the positive and negative MUX inputs, respectively. Therefore, the three-input PLE shown in Fig. 4 is a three-variable function by Shannon Expansion on variable z. The positive or negative cofactor is one of the total 16 x-y functions. Unfortunately, the combination of an AND/XOR and BUF/INV gate (the negative branch) as well as the combination of an XOR/NOR and BUF/INV gate (the positive branch) cannot implement all the 16 x-y functions. We denote the function set of $\{x\bar{y}, \bar{x}y, x+\bar{y}, \bar{x}+y\}$ that cannot be implemented as unavailable set S_u and the set of the rest 12 functions as available set S_a.

Theorem: given a three-variable logic function $f(x, y, z)$, there exists at least one variable, with respect to which Shannon Expansion can avoid the cofactors that belong to the unavailable set S_u.

In order to prove it, let us begin with some preliminary concepts. A Boolean function can be canonically expressed as the sum of minterms. For a three-variable function $f(x, y, z)$, it has eight possible minterms: $\{m_1, m_2, \ldots, m_8\}$ representing $\{\bar{x}\bar{y}\bar{z}, \bar{x}\bar{y}z, \ldots, xyz\}$, respectively. Hence, we use a 8×1 matrix $\mathbf{X}$ to canonically represent f.

$$\mathbf{X} = (\ x_1 \quad x_2 \quad \ldots \quad x_8\ )^T$$

$$x_i = \begin{cases} 1 & \text{if } m_i \in f \\ 0 & \text{if } m_i \notin f \end{cases}$$

The expansion cofactor matrix $\mathbf{W}$ is constructed to represent the cofactors of Shannon Expansions with respect to all the variables, which can guide us to quickly determine whether an expansion yields S_u cofactors or not. The general $\mathbf{W}$ matrix for n variables is defined as

$$w_{ij} = \begin{cases} 0 & \text{if minterm } m_j\text{'s cofactor } f_i \text{ is 0} \\ 1 & \text{if minterm } m_j\text{'s cofactor } f_i \in S_a \\ 3 & \text{if minterm } m_j\text{'s cofactor } f_i \in S_u \end{cases}$$

$$\forall i = 1, 2, \ldots, 2n \text{ and } j = 1, 2, \ldots, 2^n$$

Here the weights $\{1, 3\}$ are chosen to distinguish the S_u functions from S_a. As for three input variables, $\mathbf{W}$ is an 6×8 matrix given as follows.

$$\mathbf{W}_{6\times 8} = \begin{pmatrix} 1 & 3 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 3 & 1 \\ 1 & 3 & 0 & 0 & 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 & 3 & 1 \\ 1 & 0 & 3 & 0 & 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 & 0 & 3 & 0 & 1 \end{pmatrix} \begin{matrix} f_{\bar{x}} \\ f_x \\ f_{\bar{y}} \\ f_y \\ f_{\bar{z}} \\ f_z \end{matrix}$$
$$m_1\ m_2\ m_3\ m_4\ m_5\ m_6\ m_7\ m_8$$

where row i represents cofactor f_i in the cofactor set $\mathbf{F} = \{f_i\} = \{f_{\bar{x}}, f_x, f_{\bar{y}}, f_y, f_{\bar{z}}, f_z\}$, and column j represents minterm m_j of $\{m_1, m_2, \ldots, m_8\}$.

We then define our cofactor encoding matrix $\mathbf{F}^*$ as

$$\mathbf{F}^* = (\ f_x^* \quad f_{\bar{x}}^* \quad f_y^* \quad f_{\bar{y}}^* \quad f_z^* \quad f_{\bar{z}}^*\ )^{\mathbf{T}} = \mathbf{W} \cdot \mathbf{X}$$

so that f_i^* is equal to 3 or 5 if cofactor f_i belongs to S_u. In other words, if function $f(x, y, z)$ cannot be implemented on the

three-input PLE by using input z as the MUX select bit as shown in Fig. 4, at least one of the two encoded cofactors $f^*_{\bar{z}}$ and f^*_z is equal to either 3 or 5.

We now prove the theorem by contradiction.

Proof: If the theorem is not true, for all the three encoded cofactor pairs (negative and positive), $\{f^*_{\bar{x}}, f^*_x\}$, $\{f^*_{\bar{y}}, f^*_y\}$, and $\{f^*_{\bar{z}}, f^*_z\}$, at least one encoded cofactor of each pair is equal to 3 or 5. In other words, the theorem is true if we can prove that no matrix $\mathbf{X}$ can yield a 3, 5, or both for all the three encoded cofactor pairs at the same time.

- *Case 1:* Every encoded cofactor pair has a 3. An encoded cofactor of value 3 relates to a violating Shannon cofactor that is a $(x\bar{y})$-style function. Consider matrix $\mathbf{W}_{6\times 8}$: each column (one minterm) only covers two 3s (two corresponding cofactors). Function $f(x, y, z)$ should contain at least two minterms to result in three 3s. Without losing generality, suppose f has minterm m_2 which results in $f^*_{\bar{x}} = 3$ ($f_{\bar{x}} \in S_u$) and $f^*_{\bar{y}} = 3$ ($f_{\bar{y}} \in S_u$). To satisfy the case that every encoded cofactor pair has a 3, at least one of the encoded cofactors of $f^*_{\bar{z}}$ and f^*_z equals 3. Therefore, at least one of the four minterms $\{m_3, m_4, m_5, m_6\}$ is contained in function f. However, no matter which minterm belongs to f, the value of $f^*_{\bar{x}}$ or $f^*_{\bar{y}}$ no longer stays equal to 3, which contradicts the assumption we have. Hence, case 1 is impossible.
- *Case 2:* At least one of the three encoded cofactor pairs has a 5, and the other two pairs have either a 3 or 5. An encoded cofactor of value 5 relates to a violating Shannon cofactor that is a $(x + \bar{y})$-style function. Also without lost generality, suppose minterms m_1, m_2, and $m_4 \in f$ that makes $f_{\bar{x}} \in S_u$ and the cofactor encoding matrix $\mathbf{F}^* = (5\ 0\ 4\ 3\ 1\ 4)$. Since $f^*_y = 3$, only the expansion on variable z is now feasible. To meet the assumption that all the three pairs have either a 3 or 5, f must contain other minterms. In other words, either $f^*_{\bar{z}}$ or f^*_z should equal 5. If $f_{\bar{z}}{}^* = 5$, m_3 and m_7 are contained in f, which will result in $f^*_{\bar{y}} = 4$ and $f^*_y = 7$. Or m_5 and m_7 are contained in f, which will result in $f^*_{\bar{y}} = 7$ and $f^*_y = 6$. Under these two scenarios, Shannon Expansion on y flips from infeasible to feasible. If $f^*_z = 5$, m_8 must belong to f, which results in $f^*_{\bar{y}} = 4$ and $f^*_y = 4$. Now expansion on y becomes feasible. If either m_6 or m_3 is in f, correspondingly, $f^*_{\bar{y}}$ or f^*_y will be 5. However, this will change the value of $f^*_{\bar{x}}$ or f^*_z and contradict with the assumption $f_{\bar{x}} = 5$. Hence, case 2 is impossible.

Combining case 1 and case 2, we conclude that it is impossible that all the three encoded cofactor pairs have a 3, 5, or both at the same time. In other words, no such a function $f(x, y, z)$ whose expansion on every variable can yield a S_u cofactor.

Next, we will use an example to demonstrate how to use encoded cofactor matrix $\mathbf{F}^*$ to choose an expansion variable.

Example 1: Consider a logic function $f(x, y, z) = xy + y\bar{z} + x\bar{z}$. It can be expressed as the sum of minterms:

$$f(x, y, z) = \bar{x}y\bar{z} + x\bar{y}\bar{z} + xy\bar{z} + xyz$$

The corresponding minterm representation for f is

$$\mathbf{X} = (0\,0\,1\,0\,1\,0\,1\,1)^T$$

Therefore,

$$\mathbf{F}^* = \begin{pmatrix} 1 & 3 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 3 & 1 \\ 1 & 3 & 0 & 0 & 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 & 3 & 1 \\ 1 & 0 & 3 & 0 & 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 & 0 & 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 3 \\ 5 \\ 7 \\ 1 \end{pmatrix}$$

Since $f^*_{\bar{x}} = f^*_{\bar{y}} = 3$ and $f^*_x = f^*_y = 5$, Shannon expansions on variable x and y cannot be implemented on our three-input PLE. Actually Shannon Expansion on variable x generates

$$f = \bar{x}(y\bar{z}) + x(y + \bar{z})$$

The cofactors $f_{\bar{x}} = y\bar{z}$ and $f_x = y + \bar{z}$ both belong to S_u. A similar result can be derived by expansion on variable y. Since neither $f^*_{\bar{z}}$ nor f^*_z equals 3 or 5, we choose to expand on variable z, which yields

$$f = \bar{z}(x + y) + z(xy)$$

Fig. 7 shows the PLE implementation for this example. The control bits generating algorithm is presented in Section V. ∎

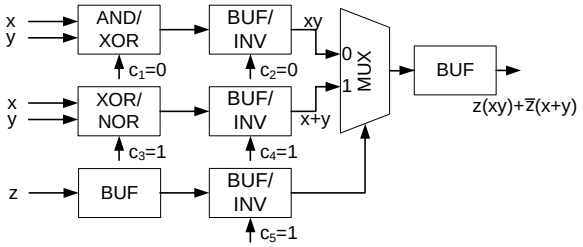

Fig. 7. A configuration example

After picking up a feasible expansion variable, both the negative and positive cofactors of function f belong to the available set S_a and can be respectively mapped to a pair of available boolean functions {AND, NAND, OR, NOR, XOR, XNOR}. As both of the expansion variable and its complement can be fed as the select bit of the MUX, the order of the boolean function pairs can be exchanged. However, since each input branch of the MUX can only implement four of the six boolean functions ({AND, NAND, XOR, XNOR} for negative branch and {OR, NOR, XOR, XNOR} for positive branch), there are still six unordered function pairs that cannot be mapped onto a PLE. They are (AND, AND), (NAND, NAND), (OR, OR), (NOR, NOR), (AND, NAND), and (OR, NOR). Fortunately, the mapping between a cofactor and its boolean function implementation is a many-to-many mapping. Alternative function pairs always exist to result in a feasible mapping.

Based on the previous discussion, we see that the PLE structure is a simple yet powerful logic element. It can realize all 256 three-input functions with a proper configuration of the inputs and control bits. Compared with current FPGA SRAM-based LUTs, our PLE requires only five bits to configure any three-input function rather than eight bits for a 3-LUT.

V. CONTROL BITS GENERATION

In Section IV, an expansion cofactor matrix $\mathbf{W}$ is introduced to quickly determine the feasibility of Shannon Expansions. We similarly construct a cofactor mapping matrix $\mathbf{W}'$ to derive the control bits for three-input PLE implementations. We use a weighted binary encoding scheme. For simplicity, consider the case of expansion on variable z. Since the resulting cofactors have four possible terms $\{\bar{x}\bar{y}, \bar{x}y, x\bar{y}$ and $xy\}$, we assign four different binary weights $\{1, 2, 4, 8\} = \{2^0, 2^1, 2^2, 2^3\}$ to these four possible cofactors. Therefore, the cofactor mapping matrix can be expressed as:

$$\mathbf{W}' = \begin{pmatrix} 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 \\ 1 & 2 & 0 & 0 & 4 & 8 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 4 & 8 \\ 1 & 0 & 2 & 0 & 4 & 0 & 8 & 0 \\ 0 & 1 & 0 & 2 & 0 & 4 & 0 & 8 \end{pmatrix}$$

Since the mapping cofactors derived through $\mathbf{F}' = \mathbf{W}' \cdot \mathbf{X}$ are integers $\in [0, 15]$, a one-to-one mapping to the 16 two-variable functions, we can easily determine the boolean functions by checking their binary encoded values.

Fig. 8 describes the pseudo code of the control bits generating algorithm. It derives the input connection and control bits configuration $C(f)$ for a given function f. First, the cofactor mapping matrix W' and minterm representation X of function f

Input: f
Output: $C(f)$
1: generate matrix X for function f
2: $\mathbf{F'} = \mathbf{W'} \cdot \mathbf{X}$
3: **for** every feasible expansion variable v
4: $B_{\bar{v}} \leftarrow f'_{\bar{v}}$
5: $B_v \leftarrow f'_v$
6: **if** $map(B_{\bar{v}}, B_v)$=positive **then**
7: $c_5 = 0$, get c_1, c_2, c_3 and c_4
8: **return** $C(f)$
9: **else if** $map(B_{\bar{v}}, B_v)$=negative **then**
10: $c_5 = 1$, get c_1, c_2, c_3 and c_4
11: **return** $C(f)$

Fig. 8. Control bits generating algorithm

are multiplied to generate the mapping cofactors $\mathbf{F'}$ (Line **1-2**). As we discussed in Section IV that not all the Shannon Expansions can be implemented on PLEs, we need to perform a feasibility check on the resulting mapping cofactors and choose a feasible one. Because the four cofactor functions in S_u are encoded as $\{2, 4, 11, 13\}$ under this binary encoding scheme, we just search in the variable order and pick up the first expansion whose both encoded positive and negative cofactors do not belong to $S'_u = \{2, 4, 11, 13\}$.

Then we map the encoded cofactors f'_v and $f'_{\bar{v}}$ to the available boolean set **B**=$\{AND, NAND, OR, NOR, XOR, XNOR\}$. If boolean functions $B_{\bar{v}}$ and B_v that realize the negative and positive cofactors can be implemented as the negative and positive PLE branches respectively ($map(B_{\bar{v}}, B_v)$=positive), variable v is connected to the select bit of the MUX with $c_5 = 0$. The corresponding control bits that configure the programmable gates are generated (Line **6-8**). However, due to the asymmetry of the positive and negative PLE branches, it may happen that the positive and negative cofactors can only be mapped to the negative and positive branches, respectively ($map(B_{\bar{v}}, B_v)$=negative). Under such a circumstance, the control bit c_5 is set to 1 which feeds v's complement to the MUX's select bit (Line **9-11**).

Example 2: Consider function $f(x, y, z) = xy + y\bar{z} + x\bar{z}$ in Example 1 again. The minterm matrix representation of f is **X**=$(0\ 0\ 1\ 0\ 1\ 0\ 1\ 1)^T$. Then we calculate the mapping cofactors $\mathbf{F'}$ using the cofactor mapping matrix:

$$\mathbf{F'} = \begin{pmatrix} 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 \\ 1 & 2 & 0 & 0 & 4 & 8 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 4 & 8 \\ 1 & 0 & 2 & 0 & 4 & 0 & 8 & 0 \\ 0 & 1 & 0 & 2 & 0 & 4 & 0 & 8 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 13 \\ 4 \\ 13 \\ 14 \\ 8 \end{pmatrix}$$

Checking the encoded cofactors, we find out that $f'_{\bar{x}} = f'_{\bar{y}} = 4 \in S'_u$ and $f'_x = f'_y = 13 \in S'_u$. This implies that only z can be used as the expansion variable. The encoded cofactors with respect to variable z are mapped to boolean functions: $f'_{\bar{z}} = 14 = (1110)_2 \Rightarrow f_{\bar{z}} = xy + x\bar{y} + \bar{x}y = x + y$ (OR gate) and $f'_z = 8 = (1000)_2 \Rightarrow f_z = xy$ (AND gate). Because $B_{\bar{v}} = OR$ and $B_v = AND$ can only be implemented on the positive and negative PLE branches respectively ($map(B_{\bar{v}}, B_v)$=negative), the control bit c_5 is set to 1 to invert z. The control bits of the negative branch $c_1 = 0$ and $c_2 = 0$ are required to configure an AND gate for $f_z = xy$, while the control bits of the positive branch $c_3 = 1$ and $c_4 = 1$ are required to configure an OR gate for $f_{\bar{z}} = x + y$. The final configuration to realize function $f(x, y, z) = z \cdot (xy) + \bar{z} \cdot (x + y)$ is shown in Fig. 7. ∎

Because of the fact that the four-input PLE is composed of two three-input PLEs, the control bits of a four-input PLE can be derived by a proper modification of the aforementioned algorithm targeting three-input PLEs. For a four-input function $f(x, y, z, w)$, at first an input variable is selected randomly, for example w. Thus the function f can be expressed as Shannon Expansion on variable w: $f = w \cdot f_w + \bar{w} \cdot f_{\bar{w}}$. Then f_w and $f_{\bar{w}}$ are two three-input functions that can be implemented by the two three-input PLE branches of the four-input PLE structure (see Fig. 6). The control bits generating algorithm for three-input PLE is executed twice to obtain the control bits for both function f_w and $f_{\bar{w}}$. Altogether ten control bits are generated corresponding to $\{c_1, c_2, \ldots, c_{10}\}$ of the four-input PLE shown in Fig. 6.

VI. DYNAMIC RECONFIGURABILITY

Generally speaking, one of the performance challenges of dynamic reconfiguration is the relatively long reconfiguration time caused by the requirement of loading a large amount of configuration data through limited internal bandwidth. Thanks for the inherent self-latching property of MOBILE devices, the PLE structure can easily relieve this design bottleneck without introducing any overhead.

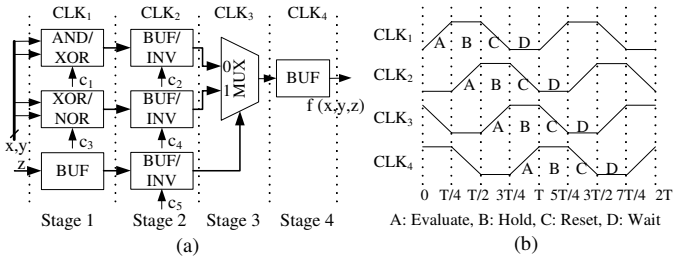

Fig. 9. Nanopipelining: (a) pipeline stages and (b) clocking scheme

Fig. 9(a) shows the separation of the PLE's four nanopipelining stages. As described in Section II-C, four overlapping four-phase clocks (CLK_1, CLK_2, CLK_3, and CLK_4 illustrated in Fig. 9(b)) are supplied to the corresponding stages to facilitate nanopipelining operations. Under this clocking scheme, the MOBILE-based circuits of each stage require the output values of their previous stage to be valid only at the evaluation phase (phase A). Even the inputs change after the evaluation phase, the self-latching property of MOBILE circuits keeps the output values stable during the hold phase (phase B). Therefore, the hold, reset, and wait phases can be used to reconfigure the input connections and control bits.

Suppose the PLE functionality is dynamically reconfigured every clock cycle T (reconfiguration cycle). In the PLE pipeline, the inputs to the stage-1 gates are required to be valid only during the clock phase $[0, T/4], [T, 5T/4], \ldots, [nT, nT + T/4]$ (evaluation phase). Thus the time interval from the end of an evaluation phase to the beginning of next valid evaluation phase, can be used to reconfigure the PLE stage-1 gates including input connection and control bits c_1 and c_3. An example of such a clock phase for stage-1 is $[T/4, T]$ with a reconfiguration slack of $3T/4$. The stage-2 gates, similarly, can take advantage of the hold, reset, and wait phases of CLK_2 (*e.g.*, $[T/2, 5T/4]$) to reconfigure the control bits c_2, c_4, and c_5. The overlapping reconfiguration slacks of different configuration objects form the pipelining reconfiguration scheme. The advantage of this pipelining reconfiguration scheme is that the inactive clock phases of the MOBILE circuits are fully utilized to avoid performance degrading.

VII. EXPERIMENTAL RESULTS

We evaluated our three- and four-input PLE structures in terms of area and performance. MCNC benchmarks were implemented on an array of PLEs. Berkeley's synthesis and verification software ABC [22] was used to extract three- and four-variable logic functions from the benchmark applications.

The area and performance comparisons of the three- and four-input PLE implementations are summarized in Table II. The level

TABLE II
AREA AND PERFORMANCE COMPARISONS OF THREE- AND FOUR-INPUT PLE IMPLEMENTATIONS

Circuit	three-input PLE				four-input PLE				Comparisons			
	Level	PLEs (w/o)	PLEs (1DR)	PLEs (Red.%)	Level	PLEs (w/o)	PLEs (1DR)	PLEs (Red.%)	Latency (w/o)	Area (w/o)	L×A (w/o)	L×A (1DR)
9symml	7	111	43	61	6	78	31	60	1.17	0.71	0.83	0.81
alu4	16	381	61	84	12	287	70	76	1.33	0.66	0.88	0.58
apex6	8	369	102	72	6	238	95	60	1.33	0.78	1.04	0.71
apex7	8	107	27	75	5	82	32	61	1.60	0.65	1.04	0.68
cc	3	39	18	54	2	31	21	32	1.50	0.63	0.95	0.64
count	10	56	19	66	6	37	7	81	1.67	0.76	1.27	2.27
dalu	16	585	114	81	11	398	79	80	1.45	0.73	1.06	1.04
des	9	1925	522	73	6	1534	556	64	1.50	0.63	0.95	0.70
rot	12	300	97	68	8	240	91	62	1.50	0.63	0.95	0.80
z4ml	3	16	8	50	3	11	6	45	1.00	0.73	0.73	0.67

of the circuit, total number of PLEs without dynamic reconfiguration (*w/o*), total number of PLEs with dynamic reconfiguration cycle of $1T$(*1DR*), and reduction of PLE numbers by using dynamic reconfiguration (*Red.%*) are presented for both implementations in major columns *three-input PLE* and *four-input PLE*. The comparisons between two implementations are computed in ratios of three- to four-input on four metrics: *Latency*, *Area*, $L\times A$ *(w/o)*, and $L\times A$ *(1DR)*. Since the latency is proportional to the circuit level, the ratio of latency is the ratio of circuit level under the assumption that the implementations are working at the same clock frequency. The area is proportional to the total number of PLE employed and PLE area.

Since the four-input PLE structure is larger in terms of granularity, it requires less number of total PLEs to implement the function thus reducing the circuit level and overall latency. However this more powerful PLE structure takes approximately twice of area compared to a three-input PLE. Although the total number of the PLEs reduced, the total area required is still larger than the implementation based on three-input PLEs. If we consider the latency-area product without reconfiguration, the three-input PLE-based implementations are slightly better and the selection between these two structures is a tradeoff between performance and area cost. When reconfiguration is implemented, the three-input PLE solutions are more favorable especially from the area cost perspective.

The comparison of PLE numbers required for implementation with and without dynamic reconfiguration demonstrates an average total area reduction of 65% if reconfiguration is applied. Combined with the discussion in Section VI, the reconfiguration process enables area reduction without incurring performance overheads by utilizing the inactive clock phase of MOBILE circuit.

VIII. CONCLUSION

In this paper, we proposed our novel three- and four-input circuit elements, based on MOBILE TG and MTTG implementations. The functional correctness of the circuit is verified by HSPICE simulation. An efficient control bit generating algorithm is developed to configure the structures to realize all three- and four-variable logic functions. Due to the self-latching property of MOBILE circuits, the reconfigurability achieves an average 65% area reduction without delay overheads.

References

[1] *International Technology Roadmap for Semiconductors (ITRS)*, http://www.itrs.net.

[2] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler, "Molecular electronics: From devices and interconnect to circuits and architecture," *Proc. IEEE*, vol. 91, no. 11, pp. 1940–1957, Nov. 2003.

[3] A. DeHon and K. K. Likharev, "Hybrid CMOS/nanoelectronic digital circuits: Devices, architectures, and design automation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2005, pp. 375–382.

[4] J. N. Schulman, H. J. De Los Santos, and D. H. Chow, "Physics-based RTD current-voltage equation," *IEEE Electron Device Letters*, vol. 17, no. 5, pp. 220–222, May 1996.

[5] J. Sun, G. I. Haddad, P. Mazumder, and J. N. Schulman, "Resonant tunneling diodes: Models and properties," *Proc. IEEE*, vol. 86, no. 4, pp. 641–660, Apr. 1998.

[6] M. Bhattacharya and P. Mazumder, "Augmentation of SPICE for simulation of circuits containing resonant," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 1, pp. 39–50, Jan. 2001.

[7] W. Williamson, S. B. Enquist, D. H. Chow, H. L. Dunlap, S. Subramaniam, P. Lei, G. H. Bernstein, and B. K. Gilbert, "12 GHz clocked operation of ultralow power interband resonant tunneling diode pipelined logic gates," *IEEE J. Solid-State Circuits*, vol. 32, no. 2, pp. 222–231, Feb. 1997.

[8] H. Matsuzaki, T. Itoh, and M. Yamamoto, "A novel high-speed flip-flop circuit using RTDs and HEMTs," in *Proc. Great Lake Symp. VLSI*, Mar. 1999, pp. 154–157.

[9] T. P. E. Broekaert, B. Brar, J. P. A. van der Wagt, A. C. Seabaugh, F. J. Morris, T. S. Moise, E. A. Beam, and G. A. Frazier, "A monolithic 4-bit 2-Gsps resonant tunneling analog-to-digital converter," *IEEE J. Solid-State Circuits*, vol. 33, no. 9, pp. 1342–1349, Sept. 1998.

[10] M. Bhattacharya, S. Kulkarni, A. Gonzalez, and P. Mazumder, "A prototyping technique for large-scale RTD-CMOS circuits," in *Proc. Int. Symp. Circuits & Systems*, May 2000, pp. 635–638.

[11] J. I. Bergman, J. Chang, Y. Joo, B. Matinpour, J. Laskar, N. M. Jokerst, M. A. Brooke, B. Brar, and E. Beam, "RTD/CMOS nanoelectronic circuits: Thin-film InP-based resonant tunneling diodes integrated with CMOS circuits," *IEEE Electron Device Letters*, vol. 20, no. 3, pp. 119–122, Mar. 1999.

[12] K. Maezawa, T. Akeyoshi, and T. Mizutani, "Functions and applications of monostable-bistable transition logic elements (MOBILEs) having multiple-input terminals," *IEEE Trans. Electron Devices*, vol. 41, no. 2, pp. 148–154, Feb. 1994.

[13] M. J. Avedillo, J. M. Quintana, and H. P. Roldan, "Increased logic functionality of clocked series-connected RTDs," *IEEE Trans. Nanotechnology*, vol. 5, no. 5, pp. 606–611, Sept. 2006.

[14] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Synthesis and optimization of threshold logic networks with application to nanotechnologies," in *Proc. Design Automation & Test Europe Conf.*, Feb. 2004, pp. 904–909.

[15] P. Gupta, R. Zhang, and N. K. Jha, "An automatic test pattern generation framework for combinational threshold logic networks," in *Proc. Int. Conf. Computer Design*, Oct. 2004, pp. 540–543.

[16] P. Mazumder, S. Kulkarni, M. Bhattacharya, J. P. Sun, and G. I. Haddad, "Digital circuit applications of resonant tunneling devices," *Proc. IEEE*, vol. 86, no. 4, pp. 664–686, Apr. 1998.

[17] P. Gupta and N. K. Jha, "An algorithm for nano-pipelining of circuits and architectures for a nanotechnology," in *Proc. Design Automation & Test Europe Conf.*, Feb. 2004, pp. 974–979.

[18] Q. Sheng, *Threshold logic*, Toronto, Ryerson Press, 1969.

[19] C. Pacha, U. Auer, C. Burwick, P. Glosekotter, A. Brennemann, W. Prost, F. J. Tegude, and K. F. Goser, "Threshold logic circuit design of parallel adders using resonant tunneling deviecs," *IEEE Trans. VLSI Systems*, vol. 8, no. 5, pp. 558–572, Oct. 2000.

[20] M. J. Avedillo, J. M. Quintana, H. Pettenghi, P. M. Kelly, and C. J. Thompson, "Multi-threshold threshold logic circuit design using resonant tunneling devices," *Electron Letters*, vol. 39, no. 21, pp. 1502–1504, Oct. 2003.

[21] H. Pettenghi, M. J. Avedillo, and J. M. Quintana, "Using multi-threshold threshold gates in RTD-based logic design: A case study," in *Proc. Euro. Nano. System Conf.*, Dec. 2005, pp. 14–16.

[22] Berkeley Logic Synthesis and Verification Group, *ABC: A system for sequential synthesis and verification*, http://www.eecs.berkeley.edu/∼alanmi/abc/.