Boolean Logic Function Synthesis for Generalised Threshold Gate Circuits

Marek A. Bawiec
marek.bawiec@pwr.wroc.pl

Maciej Nikodem
maciej.nikodem@pwr.wroc.pl

The Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology, Poland

ABSTRACT

This paper analyses negative differential resistance (NDR) based logic circuits operating in monostable-bistable transition logic element (MOBILE) regime. We formulate theoretical foundation for formal model of the generalised threshold gate (GTG) and prove that GTG can implement any, n-variable Boolean function. Moreover, we propose the algorithmic approach to GTG structure generation problem.

Categories and Subject Descriptors

B.2 [**Arithmetic And Logic Structures**]: Miscellaneous

General Terms

Algorithms, Design, Theory

Keywords

GTG, Nanoscale Devices, Logic Synthesis, NDR

1. INTRODUCTION

Exploration of negative differential resistance (NDR) and its application to logic circuits focuses on theoretical analyses [2, 3, 4, 8, 9] and physical implementations [5, 6]. Circuits proposed by Avedillo [2, 3] and Berezowski [4] are based on the monostable-bistable transition logic element (MOBILE) concept [1]. Basic MOBILE circuit that operates in such regime, consists of two NDR elements: a load (NDR$_l$) and a driver (NDR$_d$) connected in series (Fig. 1–B). By applying bias voltage V_{bias}, which varies between $0V$ and V_{DD} (Fig. 1–A) the circuit switches from monostable OUT$_m$ to one of the bistable states: OUT$_{b1}$ – low voltage or OUT$_{b2}$ – high voltage. The actual output voltage depends on the peak currents I_{pl} and I_{pd} relation (Fig. 1–C). Since the NDR element with smaller peak current switches to high resistance state when V_{bias} increases thus circuit reaches OUT$_{b1}$ state if $I_{pl} < I_{pd}$ and OUT$_{b2}$ if $I_{pl} > I_{pd}$.

Operation of the circuit composed of NDR elements is controlled by four-phase V_{bias} that changes similarly to the traditional clock signal (Fig. 1–A): I-st phase (V_{bias} increases) – circuit evaluates the output switching from monostable to one of bistable states; II-nd phase (V_{bias} high) – circuit remains in the state selected in phase I, independently of the actual peaks I_{pl} and I_{pd} relation; III-rd phase (V_{bias} falls) – circuit returns to the initial monostable state; IV-th phase (V_{bias} low) – circuit remains in monostable state and awaits for I_{pl} and I_{pd} relation adjustments.

Adjusting the relation between I_{pl} and I_{pd} allows to control the output voltage in the bistable state thus enabling to implement Boolean functions. This can be achieved if NDR$_l$ and/or NDR$_d$ is paralleled with another NDR and a transistor that operates as a switch. This concept was presented by Avedillo et al. [2, 3] and Berezowski [4]. They proposed to build circuits from the serially connected NDR$_l$ and NDR$_d$ paralleled by branches of a single NDR-transistor pair [2] or a single NDR serially connected with serial–parallel (SP) transistor network [3, 4].

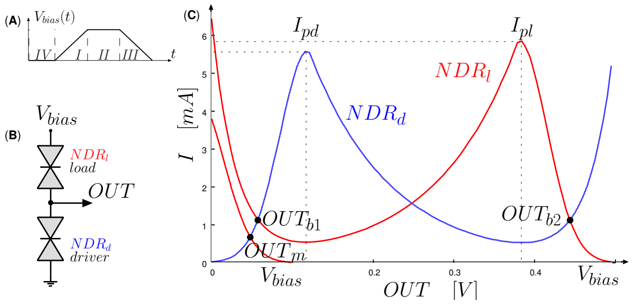

Figure 1: (A)–V_{bias} clocking scheme, (B)–basic MOBILE circuit, (C)–$I(V)$ characteristic.

In this paper we extend the concept presented in [4], propose a modified model of the GTG circuit, prove that GTG enables to implement any n–variable boolean function, and give a GTG synthesis algorithm.

2. MOTIVATION AND PREVIOUS WORK

Avedillo at al. [2] proposed a multi-threshold threshold gate (MTTG) built with NDR devices that have different parameters (i.e. peak currents). Circuit consisting of $2n$ branches, and $2n + 2$ properly adjusted NDR elements is capable of computing weighted sum of n binary inputs and to quantise its result. This allows to implement any 2-variable and small number of $n > 2$-variable boolean functions (e.g. AND, OR). Unfortunately [2] gives no method of circuit synthesis (i.e. selecting parameters of NDR elements) for a given boolean function. Successive paper by Avedillo et al. [3] proposed a modified structure that utilises one or many, serially connected transistors to switch each NDR element. This allows to implement all n–variable boolean functions, however, extends the set of input variables yielding circuit of up to 2^n+1 branches. This is infeasible due to two major concerns: (i) inaccuracies in physical design of NDRs accumulate and may result in erroneous quantisation; (ii) power consumption increases with the number of branches in the circuit.

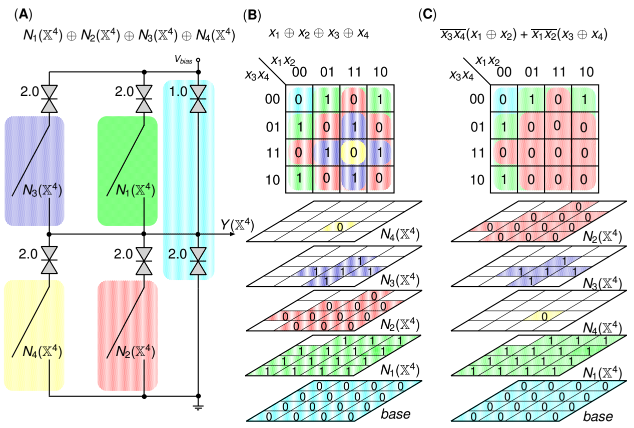

Figure 2: GTG circuit example implementing $Y(\mathbb{X}^4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ (A) circuit structure, (B) correct and (C) incorrect GTG model interpretation.

Berezowski [4] extended approach of a generalised threshold gate (GTG) and proposed to buit circuit out of series of identical NDR elements, serially connected with SP transistor network without a complementary transistor pair. Such an approach has several advantages: (i) reduced number of branches; (ii) no need to implement NDR elements with different parameters; (iii) no need for complementary transistor pair. Berezowski gave an iterative formula that describes GTG behaviour and used it to show that all 4–input boolean functions can be implemented with at most 6 branches. This was done by exhaustive search and neither proof, that $n+2$–branch GTG can implement any n–input boolean function, nor the synthesis method was given.

Recent paper by Pettenghi et al. [7] extends the model [4] by using additional inverters and introducing complementary signals. This results in simplified circuit structure, but introduces additional transistors and requires a complementary transistor pair.

3. NEW MODEL

Without loosing generality we can assume that $I_{pd} > I_{pl}$ and thus $Y(0,\ldots,0) = 0$. If so, then the top branches set '1' to the function output, while the bottom branches reset the output to '0'. Following the [4] function implemented with GTG can be described in a recursive fashion:

$$Y_l(\mathbb{X}^n) = \begin{cases} 0 & l = 0 \\ Y_{l-1}(\mathbb{X}^n) + N_l(\mathbb{X}^n) & l = 2k-1 \\ Y_{l-1}(\mathbb{X}^n)\overline{N_l(\mathbb{X}^n)} & l = 2k \end{cases}. \tag{1}$$

Unfortunately, (1) is not a GTG general description, since the order of $N_i(\mathbb{X}^n)$ functions (N_i for short) influences the resulting function $Y(\mathbb{X}^n)$. Consider a 4-input binary logic function (Fig. 2)

$$Y(\mathbb{X}^4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4, \tag{2}$$

which can be implemented with 6–branch GTG where:

$$\begin{aligned} N_1(\mathbb{X}^4) &= x_1 + x_2 + x_3 + x_4, \\ N_2(\mathbb{X}^4) &= x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4, \\ N_3(\mathbb{X}^4) &= x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4, \\ N_4(\mathbb{X}^4) &= x_1x_2x_3x_4. \end{aligned} \tag{3}$$

It can be easily verified that evaluating (1) for (3) yields (2). However, changing function order by swapping $N_2(\mathbb{X}^4)$ and $N_4(\mathbb{X}^4)$, and evaluating (1) again gives

$$Y_1(\mathbb{X}^4) = \overline{x_3x_4}\,(x_1 \oplus x_2) + \overline{x_1x_2}\,(x_3 \oplus x_4)\,. \tag{4}$$

This suggests that the related circuit implements different function (Fig. 2–C). Moreover, swapping $N_1(\mathbb{X}^n)$ and $N_3(\mathbb{X}^n)$ gives

$$Y_2(\mathbb{X}^4) = x_2\overline{x_3} + \overline{x_1}x_4 + x_3\overline{x_4} + x_1\overline{x_2}. \tag{5}$$

This obviously cannot be true since the order of branches is not important in physical implementation. This property of (1) model limits its practical application and brings in the need for a new model that will be indifferent to $N_i(\mathbb{X}^n)$ functions order. Proper order of $N_i(\mathbb{X}^n)$ functions for the (1) model can be identified from the analysis of the way a MOBILE circuit operates – assuming $Y(0,\ldots,0) = 0$, switching on the upper branch turns the output to '1' while subsequent activation of bottom branch turns it back to '0'. The next upper branch switches it again to '1' and so on. Activating upper and bottom branches in turns means that for every two functions $N_i(\mathbb{X}^n)$, $X_j(\mathbb{X}^n)$ $i < j$ the following equation has to be satisfied

$$N_i(\mathbb{X}^n)N_j(\mathbb{X}^n) = N_j(\mathbb{X}^n) \tag{6}$$

Eq.(6) determines the proper order of $N_i(\mathbb{X}^n)$ functions for the (1) model but also brings in additional properties:

$$\begin{aligned} \overline{N_i}N_j &= \left(\overline{N_i} + \overline{N_j}\right)N_j = \overline{N_iN_j}N_j = \overline{N_j}N_j = 0, \\ N_i + N_j &= N_i + N_iN_j = N_i\left(1 + N_j\right) = N_i, \\ \overline{N_i}\ \overline{N_j} &= \overline{N_i + N_j} = \overline{N_i}, \\ N_i\overline{N_j} &= N_i\overline{N_j} + \overline{N_i}N_j = N_i \oplus N_j. \end{aligned} \tag{7}$$

Let m denote the total number of $N_i(\mathbb{X}^n)$ functions. Since $m = 2k + \delta$ for some k and $\delta = 0$ if m is even or $\delta = 1$ if m is odd, thus the GTG model (1) can be simplified to

$$Y(\mathbb{X}^n) = \bigcup_{i=1}^{\lfloor \frac{m}{2} \rfloor} N_{2i-1}(\mathbb{X}^n)\overline{N_{2i}(\mathbb{X}^n)} + \delta N_m(\mathbb{X}^n). \tag{8}$$

The following theorem states that (8) can be further transformed to negation–free EXOR sum of $N_i(\mathbb{X}^n)$ functions.

Theorem 1. *GTG circuit compound of $m+2$ branches and m unate functions $N_i(\mathbb{X}^n)$ such that $N_i(\mathbb{X}^n)N_j(\mathbb{X}^n) = N_j(\mathbb{X}^n)$ for any $1 \le i < j \le m$, implements boolean function*

$$Y(\mathbb{X}^n) = \bigoplus_{i=1}^{m} N_i(\mathbb{X}^n). \tag{9}$$

Proof. For any $1 \le i < j < k < l \le m$ we have:

$$\begin{aligned} N_i\overline{N_j} &+ N_k\overline{N_l} = \\ &= N_i\overline{N_j}\left(\overline{N_k} + N_l\right) + N_k\overline{N_l}\left(\overline{N_i} + N_j\right) \\ &= N_i\overline{N_j}\,\overline{N_k\overline{N_l}} + N_k\overline{N_l}\,\overline{N_i\overline{N_j}} = N_i\overline{N_j} \oplus N_k\overline{N_l}. \end{aligned} \tag{10}$$

In a similar way it can be shown that:

$$N_i\overline{N_j} + N_k = N_i\overline{N_j} \oplus N_k \tag{11}$$

for any $1 \le i < j < k < l \le m$. Considering (8) it follows that $m+2$–branch GTG circuit implements function $Y(\mathbb{X}^n) = \bigoplus_{i=1}^{m} N_i(\mathbb{X}^n)$. $\square$

The following theorem states that $m = n+2$ branches are enough to implement any n–variable boolean function.

Theorem 2. *GTG circuit consisting of $n+2$ branches can implement any n–variable boolean function.*

Proof. According to Reed-Muller canonical expression any n-variable boolean function can be represented as:

$$\begin{aligned} Y(\mathbb{X}^n) &= a_0 \oplus \ (a_1x_1 \oplus a_2x_2 \oplus \cdots \oplus a_nx_n) \\ &\quad \oplus(a_{12}x_1x_2 \oplus a_{13}x_1x_3 \oplus \cdots \oplus a_{n-1n}x_{n-1}x_n) \\ &\quad \oplus \ \cdots \oplus (a_{12\cdots n}x_1x_2\cdots x_n), \end{aligned} \tag{12}$$

for $a_i \in \{0,1\}$. Reed-Muller formula consist of at most 2^n minterms that are products of at most n variables with no complementary signals. To simplify the notation we denote products from (12) as $I_i(\mathbb{X})$ (I_i for short) and change the indices so they will now correspond to the consecutive numbers rather than variables:

$$\begin{aligned} Y(\mathbb{X}^n) &= a_0 \oplus (a_1I_1 \oplus a_2I_2 \oplus \cdots \oplus a_nI_n) \\ &\quad \oplus (a_{n+1}I_{n+1} \oplus a_{n+2}I_{n+2} \oplus \cdots \oplus a_{\frac{(n-1)n}{2}}I_{\frac{(n-1)n}{2}}) \\ &\quad \oplus \ \cdots \oplus (a_{2^n-1}I_{2^n-1}). \end{aligned} \tag{13}$$

In the above equation I_i denotes product of some variables such that for any $i \ne j$, $I_i \ne I_j$. Using this notation any n-variable Boolean function can be represented as EXOR sum of some $I_i(\mathbb{X})$ functions:

$$Y(\mathbb{X}^n) = a_0 \oplus \bigoplus_{i\in\Omega} I_i(\mathbb{X}), \tag{14}$$

where $\Omega = \{j : a_j = 1\}$. Moreover, for any two terms $I_i(\mathbb{X})$ and $I_j(\mathbb{X})$ from (14), one of two cases occur:

1. $I_iI_j = I_i$ or $I_iI_j = I_j$,
2. $I_iI_j \ne I_i$ and $I_iI_j \ne I_j$.

If 1 holds for every $i \ne j$ then (14) represents the formal model of the GTG and (12) can be directly implemented.

In the other case, i.e. when 2 holds, then:

$$I_i \oplus I_j = (I_i + I_j) \oplus I_iI_j, \tag{15}$$

$$(I_i + I_j)\, I_iI_j = I_iI_j. \tag{16}$$

Eq (16) is analogous to (6) which means that by proper grouping of terms I_i and I_j we can transform any n–variable Reed-Muller canonical expression, to the GTG model form (9). This proves that any n–variable boolean function can be implemented with GTG circuit.

To show that at most $n+2$ branches are required it is enough to realize that there is at most n functions I_i, I_j, being negation free sum-of-products, such that either $I_iI_j = I_i$ or $I_iI_j = I_j$ holds. Additional two branches are required to set the output for all-zero input. $\square$

4. SYNTHESIS

Our circuit synthesis algorithm utilizes two auxiliary functions: Sort and Count. Sort$(Y(\mathbb{X}^n))$ is a function that given an EXOR sum of $N_i(\mathbb{X}^n)$ terms, outputs $Y(\mathbb{X}^n)$ with EXOR terms ordered according to the smallest number of variables in products and the biggest number of terms in a sum.

Example 1: Given $Y(\mathbb{X}^n) = x_1 \oplus (x_1 + x_3x_4) \oplus x_1x_3x_4 \oplus (x_1x_2 + x_1x_3 + x_2x_3x_4)$ as an input, Sort(Y) outputs: $(x_1 + x_3x_4) \oplus x_1 \oplus (x_1x_2 + x_1x_3 + x_2x_3x_4) \oplus x_1x_3x_4$.

Count$(Y(\mathbb{X}^n))$ is a function that outputs the number of EXOR terms in $Y(\mathbb{X}^n)$ expression, e.g. Count$(Y(\mathbb{X}^n))$ outputs 4 when given $Y(\mathbb{X}^n)$ from the last example.

Algorithm 1 Reed-Muller based GTG circuit synthesis

```
Require: $n$–variable Boolean function $Y(\mathbb{X}^n)$
Ensure: NDR$_l$ vs. NDR$_d$ relation, and $N_i(\mathbb{X}^n)$ functions
 1: Transform $Y(\mathbb{X}^n)$ to Reed-Muller canonical form, i.e.
        $Y(\mathbb{X}^n) = a_0 \oplus \bigoplus_i N_i(\mathbb{X}^n)$,
 2: if $Y(0^n) = 0$ then NDR$_l >$ NDR$_d$
 3: else NDR$_l <$ NDR$_d$,
 4: $Y(\mathbb{X}^n) = $ Sort$(Y(\mathbb{X}^n))$,
 5: set $i = 1$, $j = 2$,
 6: if $N_i(\mathbb{X}^n)N_j(\mathbb{X}^n) \ne N_k(\mathbb{X}^n)$ for $k = i, j$ then
 7:     set $N_i(\mathbb{X}^n) \leftarrow N_i(\mathbb{X}^n) + N_j(\mathbb{X}^n)$,
 8:     set $N_j(\mathbb{X}^n) \leftarrow N_i(\mathbb{X}^n)N_j(\mathbb{X}^n)$,
 9:     $Y(\mathbb{X}^n) = $ Sort$(Y(\mathbb{X}^n))$,
10: else set $j = j + 1$,
11: if $j >$ Count$(Y(\mathbb{X}^n))$ then $i = i + 1$, $j = i + 1$,
12: if $i <$ Count$(Y(\mathbb{X}^n))$ then goto 6-th step,
```

Example 2: Synthesis of a 3-variable Boolean function
$Y(x_1, x_2, x_3) = x_1x_2 + x_2\bar{x_3} + \bar{x_1}\bar{x_2}x_3$.

1. Reed-Muller expression for a given function equals
$Y_{RM}(x_1, x_2, x_3) = x_2 \oplus x_3 \oplus x_1x_3$.
2. since $Y(0^3) = 0$ thus NDR$_l >$ NDR$_d$, set $i = 1$, $j = 2$,
3. since $N_1N_2 \ne N_k$ for $k = 1, 2$ thus
   - $N_1 \leftarrow N_1 + N_2 = x_2 + x_3$ and $N_2 \leftarrow N_1N_2 = x_2x_3$,
   - $Y = (x_2 + x_3) \oplus x_2x_3 \oplus x_1x_3$,
4. since $i, j \le$ Count(Y) thus keep i, j and go to step 6,
5. since $N_1(\mathbb{X}^3)N_2(\mathbb{X}^3) = N_2(\mathbb{X}^3)$ thus $j = j + 1 = 3$,
6. since $i, j \le$ Count(Y) thus keep i, j and go to step 6,
7. since $N_1(\mathbb{X}^3)N_3(\mathbb{X}^3) = N_3(\mathbb{X}^3)$ thus $j = j + 1 = 4$,
8. since $j >$ Count(Y) thus $i = i + 1 = 2$, $j = i + 1 = 3$ and go to step 6,
9. since $N_2(\mathbb{X}^3)I_3(\mathbb{X}^3) \ne N_k(\mathbb{X}^3)$ for $k = 2, 3$ thus
   - $N_2 \leftarrow x_2x_3 + x_1x_3$, $N_3 \leftarrow x_1x_2x_3$,
   - $Y = (x_2 + x_3) \oplus (x_1x_3 + x_2x_3) \oplus x_1x_2x_3$,
10. since $i, j \le$ Count(Y) thus go to step 6,
11. since $N_2(\mathbb{X}^3)N_3(\mathbb{X}^3) = N_3(\mathbb{X}^3)$ thus $j = j + 1 = 4$,
12. since $j >$ Count(Y) thus $i = i + 1 = 3$, $j = i + 1 = 4$,
13. since $i =$ Count(Y) thus finish and output:

$$\begin{aligned} &\text{NDR}_l > \text{NDR}_d, && N_2(\mathbb{X}^3) = x_1x_3 + x_2x_3, \\ &N_1(\mathbb{X}^3) = x_2 + x_3, && N_3(\mathbb{X}^3) = x_1x_2x_3. \end{aligned}$$

We have verified our synthesis method for different functions with different number of variables. Verification was based on selecting random functions, describing with Reed-Muller canonical expression and transforming into the GTG expression. Later on we simulate the circuit using PSpice software. The next example presents a synthesis of 5-variable boolean function, and its simulation results. In order to make the synthesis compact, we simplify the notation and write i instead of x_i, and ij instead of x_ix_j. Moreover, we underline

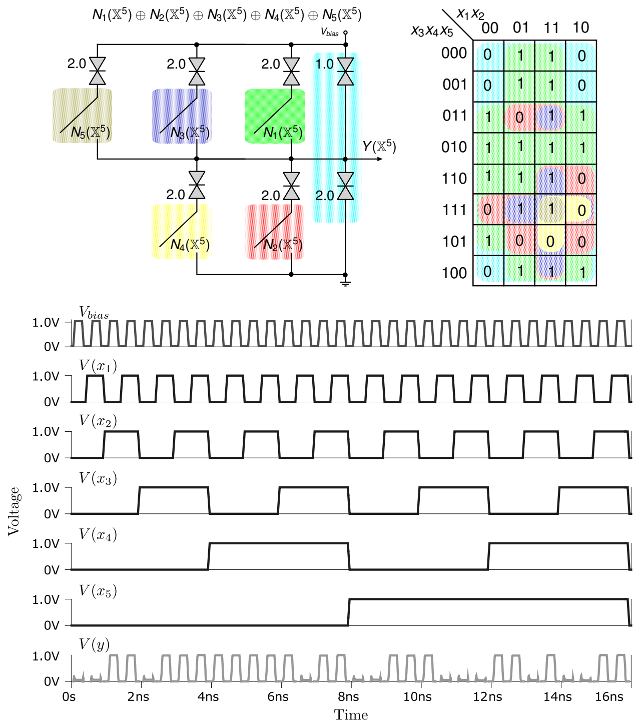

Figure 3: Carnough map, circuit structure and simulation results for 5-variables example.

terms of the EXOR sum that are transformed in each iteration. Figure 3 presents a circuit structure, its GTG model as well as simulation results.

Example 3: Synthesis of 5 variable Boolean function given in Reed-Muller form:

$$Y_{RM} = 2 \oplus 4 \oplus 13 \oplus 35 \oplus 24 \oplus 123 \oplus 245 \oplus 1245 \oplus 1345.$$

1. since $Y(0^5) = 0$ thus $\mathrm{NDR}_l > \mathrm{NDR}_d$, and:

$$\begin{aligned}
Y &= \underline{2 \oplus 4} \oplus 13 \oplus 35 \oplus 24 \oplus 123 \oplus 245 \oplus 1245 \oplus 1345\\
&= \underline{(2+4)} \oplus 13 \oplus 35 \oplus 123 \oplus 245 \oplus 1245 \oplus 1345\\
&= \underline{(2+4+13)} \oplus 35 \oplus (123+134) \oplus 123\\
&\quad \oplus 245 \oplus 1245 \oplus 1345\\
&= N_1 \oplus \underline{(235+345+135)} \oplus \underline{(123+134)}\\
&\quad \oplus 123 \oplus 245 \oplus 1245 \oplus 1345\\
&= N_1 \oplus \underline{(235+345+135+123+134)} \oplus 123\\
&\quad \oplus \underline{245} \oplus (1235+1345) \oplus 1245 \oplus 1345\\
&= N_1 \oplus N_2 \oplus \underline{123} \oplus (1235+1345) \oplus \underline{2345} \oplus 1245 \oplus 1345\\
&= N_1 \oplus N_2 \oplus \underline{(123+2345)} \oplus \underline{(1235+1345)}\\
&\quad \oplus 1245 \oplus 1345 \oplus 12345\\
&= N_1 \oplus N_2 \oplus \underline{(123+2345+1345)} \oplus 1235\\
&\quad \oplus \underline{1245} \oplus 1345 \oplus 12345\\
&= N_1 \oplus N_2 \oplus N_3 \oplus \underline{1235 \oplus 1345}\\
&= N_1 \oplus N_2 \oplus N_3 \oplus (1235+1345) \oplus 12345.
\end{aligned}$$

2. for given function synthesis algorithm outputs

$$\begin{aligned}
\mathrm{NDR}_l &> \mathrm{NDR}_d,\\
N_1(\mathbb{X}^5) &= x_2 + x_4 + x_1x_3 + x_3x_5,\\
N_2(\mathbb{X}^5) &= x_2x_3x_5 + x_3x_4x_5 + x_1x_3x_5 + x_1x_2x_3\\
&\quad + x_1x_3x_4 + x_2x_4x_5,\\
N_3(\mathbb{X}^5) &= x_1x_2x_3 + x_2x_3x_4x_5 + x_1x_3x_4x_5 + x_1x_2x_4x_5,\\
N_4(\mathbb{X}^5) &= x_1x_2x_3x_5 + x_1x_3x_4x_5,\\
N_5(\mathbb{X}^5) &= x_1x_2x_3x_4x_5.
\end{aligned}$$

5. CONCLUSIONS

While it is possible to use technology-independent synthesis with AND, OR, NOT gates and perform a technology mapping to NDR devices, the challenge in GTG design is to generate individual gate topologies. As shown in the paper, GTGs of fan-in n can implement all Boolean functions of n variables. Even for a very limited fan-in circuits (like typical library cell) only on-fly synthesis is viable approach because the size of full library of 2^{2^n} cells would be way beyond the excessive. Moreover, some Boolean functions can be implemented in many alternative variants that feature different speed, power and area trade-offs.

Synthesizing GTG circuit for a given Boolean function and exploring different viable alternatives considering the underlying combinatorial structure of the circuit topology as well as timing, power, and area constrains, is not a straightforward problem that can be solved manually. The algorithm proposed in this paper, that generates a GTG circuit consisting of at most $n + 2$ branches, given an n variable Reed-Muller canonical expression is the first synthesis method designed for NDR-based circuits.

6. REFERENCES

[1] T. Akeyoshi, K. Maezawa, and T. Mizutani. Weighted sum threshold logic operation of mobile using resonant-tunneling transistors. *Electron Device Letters, IEEE*, vol.14(10), pp.475–477, October 1993.

[2] M. Avedillo, J. Quintana, H. Pettenghi, P. Kelly, and C. Thompson. Multi-threshold threshold logic circuit design using resonant tunnelling devices. *Electronics Letters*, vol.39(21), pp.1502–1504, October 2003.

[3] M. Avedillo, J. Quintana, and H. Pettenghi. Logic models supporting the design of mobile-based rtd circuits. pp.254–259, July 2005.

[4] K. Berezowski. Compact binary logic circuits design using negative differential resistance devices. *Electronics Letters*, vol.42(16), pp.902–903, 2006.

[5] T. Kim, Y. Jeong, and K. Yang. Low-power high-speed performance of current-mode logic d flip-flop topology using negative-differential-resistance devices. *Circuits, Devices & Systems, IET*, vol.2(2), pp.281–287, April 2008.

[6] H. Kim and K. Seo. Noninverted/inverted monostabel-to-bistable transition logic element circuits using three resonant tunneling diodes and their application to a static binary frequency divider. *The Japan Society of Applied Physics*, vol.47, pp.2854–2857, 2008.

[7] H. Pettenghi, M. Avedillo, and J. Quintana. A novel contribution to the rtd-based threshold logic family. *IEEE International Symposium on Circuits and Systems, 2008*, pp.2350–2353, May 2008.

[8] H. Pettenghi, M. J. Avedillo, and J. M. Quintana. Using multi-threshold threshold gates in rtd-based logic design: A case study. *Microelectronics Journal*, vol.39(2), pp.241 – 247, 2008.

[9] Y. Zheng and C. Huang. Reconfigurable rtd-based circuit elements of complete logic functionality. *Asia and South Pacific Design Automation Conference, 2008*, pp.71–76, March 2008.