Synthesis and Optimization of Switching Nanoarrays

Muhammed Ceylan MORGUL and Mustafa ALTUN,

Department of Electronics and Communication Engineering,
Istanbul Technical University, Istanbul, TURKEY
Emails: morgul@itu.edu.tr, altunmus@itu.edu.tr

***Abstract* — In this paper, we study implementation of Boolean functions with crossbar nanoarrays where each crosspoint behaves as a switch. This study has two main parts "formulation" and "optimization". In the first part of formulation, we investigate nanoarray based implementation methodologies in the literature. We classify them as two-terminal or four-terminal switch based. We generalize these methodologies to be applicable for any given Boolean function by offering array size formulations. In the second part of optimization, we focus on four-terminal switch based implementations; we propose a synthesis method to implement Boolean functions with optimal array sizes. Finally, we perform synthesis trials on standard benchmark circuits to evaluate the proposed optimal method in comparison with previous nanoarray based implementation methods. The proposed synthesis method gives by far the smallest array sizes and offers a new design paradigm for nanoarray based computing architectures.**

Keywords—switching nanoarrays; logic synthesis; optimization

I. INTRODUCTION

CMOS transistor dimensions have been shrinking for decades in an almost regular manner. Nowadays this trend has reached a critical point and it is widely accepted that the trend will end in a decade [1]. Even Gordon Moore, who made the most influential prediction in 1965 about CMOS size shrinking (Moore Law), accepted that his prediction will lose it validity in near future [2]. At this point, research is shifting to novel forms of nanotechnologies including molecular-scale self-assembled systems [3-4]. Such technologies have apparent advantages over conventional CMOS technologies, such as high density and easy manufacturability. Unlike conventional CMOS that can be patterned in complex ways with lithography, self-assembled nanoscale systems generally consist of regular structures. Logical functions and memory elements are achieved with arrays of crossbar-type switches. In this study, we target this type of switching arrays where each crosspoint behaves as a switch, either two-terminal or four-terminal. This is illustrated in Figure 1. We implement Boolean functions by considering array sizes. Table 1 compares different implementation methodologies for few XOR functions (Parity functions) regarding the array sizes. The columns "diode based" and "transistor based" represent two-terminal switch based implementation methodologies. These methodologies have been proposed to implement simple logic functions [5-6]. In this study, we generalize them to be applicable for any given Boolean function with offering array size formulations. The last two columns represent four-terminal switch based implementation methodologies that offer favorably better results. The results shown in bold from the last column are taken from our synthesis method proposed in this study that implements Boolean functions with optimal array sizes.

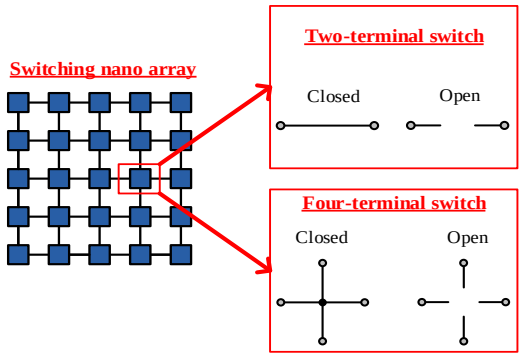
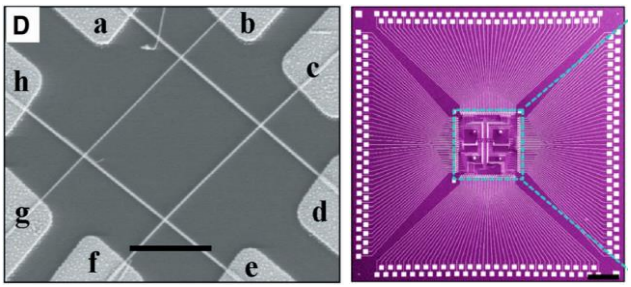


Fig. 1. A switching crossbar nanoarray modeled with two-terminal and four-terminal switches.

TABLE 1
ARRAY SIZES FOR NANOARRAY COMPUTING MODELS; $XOR_2 = X_1 \oplus X_2$, $XOR_3 = X_1 \oplus X_2 \oplus X_3$, AND $XOR_4 = X_1 \oplus X_2 \oplus X_3 \oplus X_4$.

	Two-terminal switch based nanoarray models		**Four-terminal switch** based nanoarray models	
	Diode based [7]	Transistor based [8]	Four-terminal [9]	Four-terminal **(Proposed)**
XOR_2	2×5 array 10 switches	4×4 array 16 switches	2×2 array 4 switches	2×2 array **4** switches
XOR_3	4×7 array 28 switches	6×8 array 48 switches	4×4 array 16 switches	3×3 array **9** switches
XOR_4	8×9 array 72 switches	8×16 array 128 switches	8×8 array 64 switches	3×5 array **15** switches

Although this study is at the technology-independent level, the targeted two-terminal and four-terminal switching arrays have applications in variety of emerging technologies including nanowire crossbar arrays [8-10], magnetic switch-based structures [11], arrays of single -electron transistors [12], and memristive arrays [13]. Furthermore, switching nanoarrays have true potential for commercial fabrication [16]. Figure 2 shows a SEM image of a 2x2 nano-crossbar array made by n-type nanowires and a complete fabricated chip of a nanocomputer.



a) b)

Fig 2. SEM image of a) a 2x2 nano-crossbar array [4] and b) a complete fabricated chip [16].

The paper is organized as follows. In Section II, we investigate nanoarray based implementation methodologies and

propose generalized array size formulations. In Section III, we focus on four-terminal switch based implementation techniques and propose a synthesis method to implement Boolean functions with optimal array sizes. In Section IV, we evaluate our synthesis methods on standard benchmark circuits. In Section V, we discuss the contributions of this study.

A. Definitions

Consider k independent **Boolean variables**, $x_1, x_2, \ldots, x_k$. **Boolean literals** are Boolean variables and their complements, i.e., $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_k, \overline{x_k}$. A **product (P)** is an AND of literals, e.g., P = $x_1\overline{x_2}\,x_3$. A **sum-of-products (SOP)** expression is an OR of products. **An irredundant sum-of-products (ISOP)** expression is an SOP expression with minimum number of products.

f and g are **dual Boolean functions** iff

$f(x_1, x_2, \ldots, x_k) = \bar{g}(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k})$.

Given an expression for a Boolean function in terms of AND, OR, NOT, 0, and 1, its dual can also be obtained by interchanging the AND and OR operations as well as interchanging the constants 0 and 1. For example, if $f(x_1, x_2, x_3) = x_1x_2 + x_1\overline{x_3}$ then $f^D(x_1, x_2, x_3) = (x_1 + x_2)(x_1 + \overline{x_3})$. A trivial example is that for $f = 1$, the dual is $f^D = 0$.

II. Implementation Methodologies and Formulations

We investigate three major implementation methodologies developed for switching nanoarrays. We classify them as two-terminal or four-terminal switch based.

A. Two-terminal switch based methodologies

These methodologies consider each crosspoint of an array as a two-terminal switch that behaves like a diode or a CMOS transistor. This is illustrated in Figure 3. Since diodes and CMOS transistors conduct current through their two terminals that are anode & cathode for diodes and source & drain for CMOS transistors, they are fundamentally two-terminal switches.

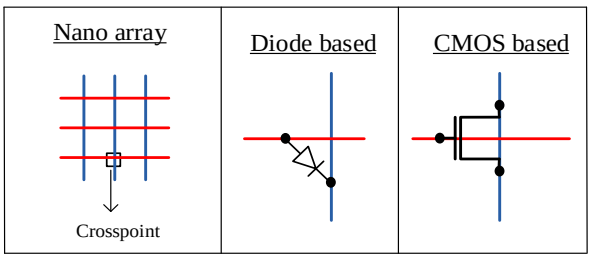
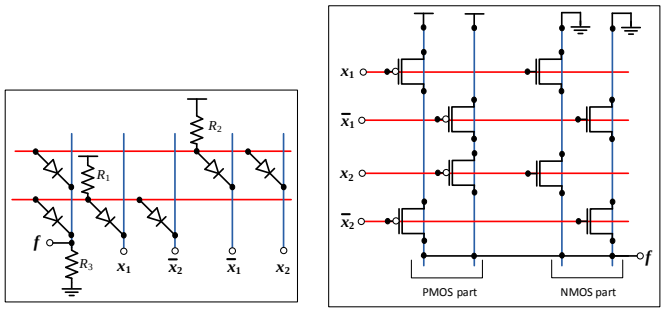


Fig. 3. A switching crossbar nanoarray modeled with diode and CMOS based two-terminal switches.

Boolean functions are implemented by using conventional techniques from diode-resistor logic and CMOS logic with an important constraint regarding nanoarray structures. Boolean functions should be implemented in their sum-of-products (SOP) forms; other forms such as factored or BDD can not be used since these forms require manipulation/wiring of switches that is not applicable for self-assembled nanoarrays. Figure 4 shows implementation of a Boolean function XOR_2 with diode and CMOS based nanoarrays.



a) b)

Fig. 4. a) Diode and b) CMOS based nanoarrays implementing $XOR_2 = x_1 \oplus x_2$ with 2×5 and 4×4 arrays, respectively

Array size formulations: Given a target Boolean function $\boldsymbol{f}$, we derive formulas of the array sizes required to implement $\boldsymbol{f}$. This is shown in Table 2. For diode based implementations, each product of $\boldsymbol{f}$ requires a row (horizontal line), and each literal of $\boldsymbol{f}$ requires a column (vertical line) in an array. Additionally, one extra column is needed to obtain the output. For CMOS based implementations, each product of $\boldsymbol{f}$ and $\boldsymbol{f^D}$ requires a column, and each literal of $\boldsymbol{f}$ requires a row in an array. As an example shown in Figure 4, $\boldsymbol{f}$ = XOR_2 = $x_1\overline{x_2} + \overline{x_1}x_2$ has 4 literals and 2 products; $\boldsymbol{f^D} = x_1x_2 + \overline{x_1}\,\overline{x_2}$ has 2 products. This results in array sizes of 2×5 and 4×4 for diode and CMOS based implementations, respectively. Note that both formulas, for diode and CMOS, always result in optimal array sizes; no further reduction is possible.

TABLE 2
Array Size Formulas for Diode and CMOS Based Implementations

Type	Array Size Formulas
Diode	(number of products in $\boldsymbol{f}$) x ("number of literals in $\boldsymbol{f}$"+ 1)
CMOS	(number of literals in $\boldsymbol{f}$) x ("number of products in $\boldsymbol{f}$" + "number of products in $\boldsymbol{f^D}$")

B. Four-terminal switch based methodology

This methodology considers each crosspoint of an array as a four-terminal switch. This is illustrated in Figure 5. Boolean functions are implemented with top-to-bottom paths in an array by taking the sum (OR) of the product (AND) of literals along each path. This makes Boolean functions implemented in their sum-of-products (SOP) forms. Figure 6-a) and Figure 6-b) show the implementations of a Boolean function XOR_2 in an array and lattice representations, respectively. Figure 6-c) shows a lattice of four-terminal switches implementing a Boolean function $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$. The function is computed by taking the sum of the products of the literals along each path. These products are $x_1x_2x_3$, $x_1x_2x_5x_6$, $x_2x_3x_4x_5$, and $x_4x_5x_6$.

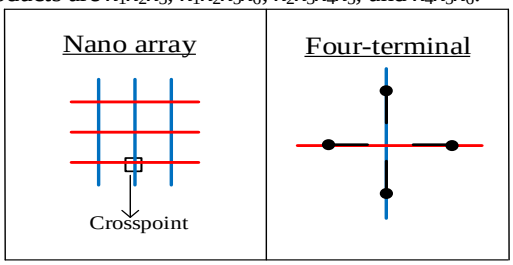
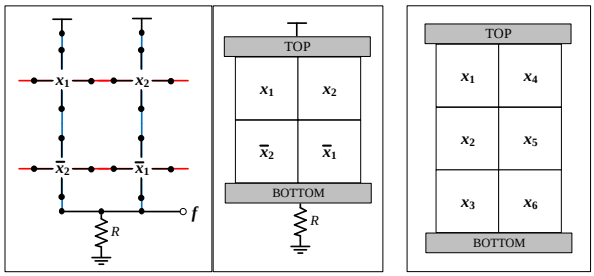


Fig. 5. A switching crossbar nanoarray modeled with four-terminal switches.



a) b) c)

Fig. 6. a) Four-terminal switch based nanoarray and b) its lattice representation implementing $XOR_2 = x_1 \oplus x_2$ with a size of 2×2 c) Four-terminal switch based lattice implementing $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.

Array size formulation: Given a target Boolean function $\boldsymbol{f}$, the array size formula was proposed by Altun and Riedel [9] that is shown in Table 2. In their implementation, each product of $\boldsymbol{f}$ and $\boldsymbol{f^D}$ require a column and a row, respectively, in an array. As an example shown in Figure 6-a), $\boldsymbol{f}$ = XOR_2 = $x_1\overline{x_2} + \overline{x_1}x_2$ and $\boldsymbol{f^D} = x_1x_2 + \overline{x_1}\,\overline{x_2}$ have both 2 products. This results in an array size of 2×2.

TABLE 3
ARRAY SIZE FORMULA FOR FOUR-TERMINAL SWITCH BASED IMPLEMENTATION

Type	Array Size Formula
Four-terminal	(number of products in $\boldsymbol{f}$) x (number of products in $\boldsymbol{f^D}$)

Examining the array size formulas in Table 2 and Table 3, we see that while the formulas in Table 2 always result in optimal sizes, but the sizes derived from the formula in Table 3 that is for four-terminal switch based arrays, are not necessarily optimal. In the following section we propose an algorithm that finds an optimal size implementation of any given target Boolean function.

III. OPTIMIZATION

Finding whether a certain array with assigned literals to its switches implements a target function is the main problem in finding optimal sizes. This problem requires to check if each assignment of 0's and 1's to the switches, corresponding to a row of the target function's truth table, results in logic 1 (a top-to-bottom path of 1's exists). To check this we have to enumerate all top-to-bottom paths that is exponentially growing with the array size. Therefore any algorithm that finds optimal sizes should have exponential time complexity with respect to the array size so is our algorithm.

Our algorithm finds optimal array sizes to implement given target Boolean functions with arrays of four-terminal switches in four steps:

1) Obtain irredundant sum-of-products (ISOP) expressions of a given-target function $\boldsymbol{f_T}$ and its dual $\boldsymbol{f_T^D}$. Determine the upper bound on the array size using the formula in Table 3:
Upper Bound (UB): (number of products in $\boldsymbol{f_T}$) x (number of products in $\boldsymbol{f_T^D}$).
The implementable **lower bound (LB)** values are taken from the lower bound table proposed in [9].

2) List the array shapes (RxC) (which are in between **LB** and **UB**) into the 'List of Implementable Nanoarray Shapes' and sort them regarding of array sizes, in ascending order. While ordering, first take the array shape which has lower number of rows (e.g. if the k^{th} shape is "3x4", then the $(k+1)^{th}$ shape can be "4x3".). Suppose that there are total of **N** different shapes in the list. For step-3, start with n=1 ($1 \le n \le N$).

3) Compute the value of the following statement for the n^{th} shape.
The Statement: An array which has the shape in the n^{th} line of the list is implementable for $\boldsymbol{f_T}$.
If the statement is **TRUE**
Change **UB** to the **RxC (**save the design);
Go to the step-4;
If the statement is **FALSE**
Increase the number "n" by 1 (n=n+1);
Repeat step-3

4) Declare that **UB** is **optimal size** for given-target function $\boldsymbol{f_T}$ can be realized in.

Our algorithm is mainly based on finding a design in a certain sized array such that the design implements $\boldsymbol{f_T}$. Our algorithm does not check every possible design. If it did then the algorithm would be intractable even for small sized arrays. For example, if a target function $\boldsymbol{f_T}$ having 6 variables, 8 literals, is tested on a 3x4 array then there are 12^{10} possible designs and 2^6 truth table rows. Note that for each of the 12 switches in the array there are 10 different options; it might be one of the 8 literals, 0, or 1. In this scenario, the algorithm would have to check $12^{10}x2^6$ truth table rows. To overcome this problem, we discard a significant portion of designs to be checked. For this purpose, we offer 3 major improvements:

I) We create a library of reduced number of Rx2 sized sub-designs. We use them to achieve RxC sized designs. While creating sub-designs we exploit the following simple lemmas. First lemma allows us to discard designs implementing a product (s) that does not imply $\boldsymbol{f_T}$. The second lemma allows us to discard designs with "0" assignments to the switches if $\boldsymbol{f_T}$ has a product having a single literal.

Lemma 1: If a design has a path realizing a product $\boldsymbol{p}$ for which $\boldsymbol{f_T \neq f_T + p}$, then the design can not implement $\boldsymbol{f_T}$.
Proof: Since $\boldsymbol{p}$ is not an implicant of $\boldsymbol{f_T}$, then a design including p implements a different function.
Lemma 2: If a function $\boldsymbol{f_T}$ has a single variable product term $\boldsymbol{p=x}$ then the algorithm does not need to assign "0" to the switches.
Proof: All the "0" assignments can be replaced with $\boldsymbol{x}$'s without a loss of generality.

II) If there is a product of $\boldsymbol{f_T}$ such that the number of literals of the product equals to the number of switches in the longest top-to-bottom path in the array, then we settle that particular product onto that particular path.

III) We discard designs having fewer number of total literals than the total number literals of $\boldsymbol{f_T}$.

These improvements make our algorithm much faster. As an example, suppose that XOR_3 is given as a target function for which the improved algorithm runs roughly 400 times faster. For 3x2 sized sub-designs, there are 8^6=262,144 designs. With applying the proposed improvements, this number is reduced to 12,114, roughly 20 times smaller than the unimproved one. Since we use two sub-arrays for XOR_3, for the optimal array size of 3x4, the improved algorithm works 400 times faster.

IV. Experimental Results

TABLE 4
EXPERIMENTAL RESULTS FOR STANDARD BENCHMARK CIRCUITS

Benchmark	CMOS	Diode	4-Terminal	Optimal 4-Terminal
Alu 0	30	18	6	**6**
Alu 1	30	18	6	**6**
Alu 2	30	18	6	**6**
Alu 3	30	18	6	**6**
B12 0	80	32	24	**12**
B12 1	120	70	35	**16**
B12 3	30	20	8	**8**
B12 4	42	28	8	**8**
B12 6	132	77	35	**18**
B12 7	110	66	24	**18**
B12 8	90	70	14	**14**
C17 0	36	18	9	**6**
C17 1	30	20	8	**8**
Clpl 0	64	32	16	**12**
Clpl 1	36	18	9	**9**
Clpl 2	16	8	4	**4**
Clpl 3	144	72	36	**18**
Clpl 4	100	50	25	**15**
Dc1 1	25	10	6	**6**
Dc1 2	72	36	16	**12**
Dc1 5	35	15	12	**6**
Dc1 6	36	18	9	**6**
Ex5 31	156	104	32	**24**
Ex5 33	110	77	21	**21**
Ex5 46	81	54	18	**18**
Ex5 49	72	54	12	**12**
Ex5 50	81	63	14	**14**
Ex5 61	64	48	12	**12**
Ex5 62	49	35	10	**10**
Misex1 1	48	16	8	**8**
Misex1 2	132	55	35	**15**
Misex1 3	156	60	40	**24**
Misex1 4	121	44	28	**16**
Misex1 5	90	45	25	**15**
Misex1 6	143	66	42	**18**
Misex1 7	81	36	20	**15**
Mp2d 4	345	75	90	**24**
Newtag	108	72	32	**18**

In Table 4 we report synthesis results for standard benchmark circuits [14]. We treat each output of a benchmark circuit as a separate target function. The number of products for each target function f_T and its dual f_T^D are obtained through sum-of-products minimization using the program Espresso [15]. The array size values for "**Diode**", "**CMOS**", and "**4-terminal**" are calculated by using the formulas in Table 2 and Table 3. The array size values for "**Optimal 4-terminal**" are obtained using the proposed optimization algorithm in Section III: Optimization.

Examining the numbers in Table 4, we always see the same sequence from the worst to the best result as "**CMOS**", "**Diode**", "**4-terminal**", and "**Optimal 4-terminal**". This demonstrates that nanoarray models based on four-terminal switches overwhelm those based on two-terminal switches regarding the array size. Further, the numbers obtained by our optimal synthesis method compares very favorably to the numbers obtained by previous methods.

V. Conclusion

In this paper, we extensively investigate computing models developed for switching nanoarrays. We classify them as two-terminal or four-terminal switch based. We derive array size formulations in terms of the properties of given Boolean functions. We synthesize arrays of four-terminal switches to implement Boolean functions with optimal array sizes. We perform synthesis trials on standard benchmark circuits to evaluate the proposed optimal method in comparison with previous methods by using their derived formulas. The proposed synthesis method gives by far the smallest array sizes and offers a new design paradigm for nanoarray based computing architectures. With this promising motivation, we seek to develop our algorithm to make it useful for complex benchmark functions.

Acknowledgment

This work is supported by TUBITAK (The Scientific and Technological Council of Turkey) Career Program #113E760.

References

[1] "Overall Technology Roadmap Characteristics". International Technology Roadmap for Semiconductors. 2010. Retrieved 2013.
[2] Dubash, M. Moore's Law is dead, says Gordon Moore. Techworld. com, 13 (2005).
[3] Ariga, Katsuhiko, et al. "Two-dimensional nanoarchitectonics based on self-assembly." Adv. in colloid & interface science 154 (2010)
[4] Whitesides G. M. and Grzybowski B.. Self-assembly at all scales. Science, 295(5564):2418-2421, (2002).
[5] Chen, Zhihong, et al. "An integrated logic circuit assembled on a single carbon nanotube." Science 311.5768 (2006): 1735-1735.
[6] Yan, Hao, et al. "Programmable nanowire circuits for nanoprocessors." Nature 470.7333 (2011): 240-244.
[7] Huang, Yu, et al. "Logic gates and computation from assembled nanowire building blocks." Science 294.5545 (2001): 1313-1317.
[8] Snider, Greg. "Molecular-junction-nanowire-crossbar-based inverter, latch, and flip-flop circuits, and more complex circuits composed, in part, from molecular-junction-nanowire-crossbar-based inverter, latch, and flip-flop circuits." U.S. Patent No. 6,919,740. 19 Jul. 2005.
[9] Altun, Mustafa, and Marc D. Riedel. "Logic synthesis for switching lattices." Computers, IEEE Transactions on 61.11 (2012): 1588-1600.
[10] Dehon, André. "Nanowire-based programmable architectures." ACM J. on Emerging Tech. in Computing Sys. (JETC) 1.2 (2005): 109-162.
[11] Khitun, Alexander, Mingqiang Bao, and Kang L. Wang. "Spin wave magnetic nanofabric: A new approach to spin-based logic circuitry." Magnetics, IEEE Transactions on 44.9 (2008): 2141-2152.
[12] Chen, Yung-Chih, et al. "Automated mapping for reconfigurable single-electron transistor arrays." Proceedings of the 48th Design Automation Conference. ACM, 2011.
[13] Levy, Yifat, et al. "Logic operations in memory using a memristive Akers array." Microelectronics Journal (2014).
[14] K. McElvain, "IWLS93 benchmark set: Version 4.0, distributed as part of the IWLS93 benchmark distribution, http://www.cbl.ncsu.edu:16080/benchmarks/lgsynth93/," 1993.
[15] R. K. Brayton, C. McMullen, G. D. Hachtel, and A. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, 1984.
[16] Yao. J.; Yan, H.; Das, S.; Klemic, J. F.; Ellenbogen, J. C.; Lieber, C. M. Nanowire nanocomputer as a finite-state machine. Proc. Natl. Acad. Sci. U.S.A. (2014), 111, 2431– 2435.