Synthesis for Width Minimization in the Single-Electron Transistor Array

Chian-Wei Liu, Chang-En Chiang, Ching-Yi Huang, Yung-Chih Chen, Chun-Yao Wang, *Member, IEEE*, Suman Datta, *Fellow, IEEE*, and Vijaykrishnan Narayanan, *Fellow, IEEE*

***Abstract*— Power consumption has become one of the primary challenges to meet Moore's law. For reducing power consumption, single-electron transistor (SET) at room temperature has been demonstrated as a promising device for extending Moore's law due to its ultralow power consumption in operation. Previous works have proposed automated mapping approaches for SET arrays that focused on minimizing the number of hexagons in the SET arrays. However, the area of an SET array is the product of the bounded height and the bounded width, and the height usually equals the number of inputs in the Boolean function. Consequently, in this paper, we focus on the width minimization to reduce the overall area in the mapping of the SET arrays. Our approach consists of techniques of product term minimization, branch-then-share (BTS)-aware variable reordering, SET array architecture relaxation, and BTS-aware product term reordering. The experimental results on a set of MCNC and IWLS 2005 benchmarks show that the proposed approach saves 45% of width compared with the work by Chiang *et al.*, which focused on hexagon count minimization, and also saves 13% of width compared with the work by Chen *et al.*, which focused on width minimization.**

***Index Terms*— Area minimization, mapping algorithm, single-electron transistor (SET).**

I. Introduction

REDUCING power consumption has become one of the primary challenges in chip designs to meet Moore's law. To deal with this issue, many ultralow-power devices have been explored. Because the power consumption of single-electron transistors (SETs) [3]–[6], which work with only one or a few electrons during switching operations, is ultralow, SETs are considered a promising candidate that substitutes conventional CMOS devices for future VLSI/system-on-chip designs [7]–[11].

Manuscript received August 4, 2014; revised November 4, 2014; accepted December 3, 2014. Date of publication February 26, 2015; date of current version November 20, 2015. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3 and Grant MOST 103-2221-E-155-069, in part by the National Science Council of Taiwan under Grant 102-2221-E-007-140-MY3, Grant NSC 101-2221-E-155-077, Grant NSC 101-2628-E-007-005, Grant NSC 100-2628-E-007-031-MY3, and Grant NSC 102-2221-E-155-087, and in part by National Tsing Hua University, Hsinchu, Taiwan, under Grant 102N2726E1.

C.-W. Liu, C.-E. Chiang, C.-Y. Huang, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: smilelcw@gmail.com; kian3256@hotmail.com; s986516@m98.nthu.edu.tw; wcyao@cs.nthu.edu.tw).

Y.-C. Chen is with the Department of Computer Science and Engineering, Yuan Ze University, Chungli 32003, Taiwan (e-mail: ycchen.cse@saturn.yzu.edu.tw).

S. Datta is with the Department of Electrical Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: sdatta@engr.psu.edu).

V. Narayanan is with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: vijay@cse.psu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2386331

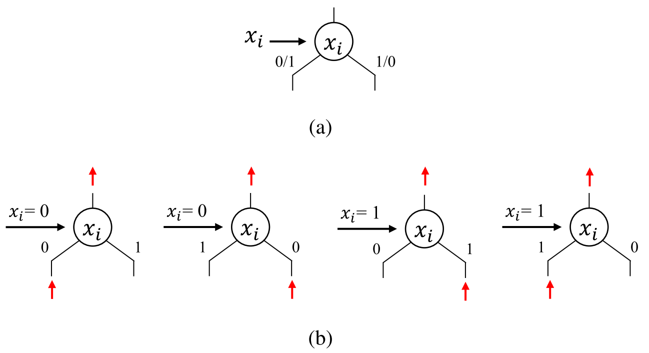


Fig. 1. (a) Node device in the SET array. (b) Behavior of the node device.

The SET devices were studied in many research groups [12]–[17]. A recent effort in [15] experimentally confirmed a reconfigurable single-electron device that can be operated in short, Coulomb blockade, and open-mode operations. This paper also indicates that scaling of the device dimension beyond the proof-of-concept experimental device will enable a room temperature operation. Hence, designs that use these devices become imperative. The SET device exploits a multigate quantum well-field-effect-transistor structure, which uses a pair of split gates with a separation of 80 nm, to achieve the single-electron device behavior at temperature of 4.2 K.

Although the SET is a promising candidate device to substitute the CMOS, it has a poor driving capability and poor threshold control due to the involvement of only one or a few electrons in the switching process. Therefore, the conduction mechanism of the conventional CMOS-based logic is not applicable to SETs. As a result, a binary decision diagram (BDD)-based architecture [18] was proposed as a suitable platform for implementing logic functions using SETs [19]. Therefore, a Boolean circuit can be implemented by mapping its BDD onto a BDD-based SET array that is represented as a hexagonal nanowire network controlled by Schottky wrap gates [3], [20], [21].

In the BDD-based SET array, each node device, as shown in Fig. 1(a), has two receiving edges and one sending edge. It works like a switch that receives the electrons from its

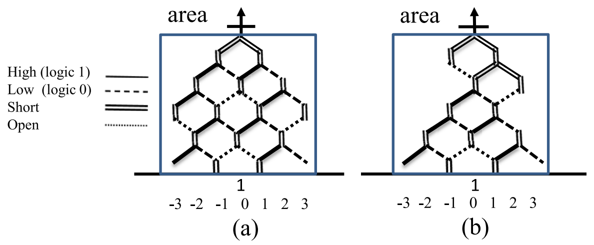


Fig. 2. (a) Original SET array. (b) Area of the SET array is intact even though the number of hexagons is reduced.

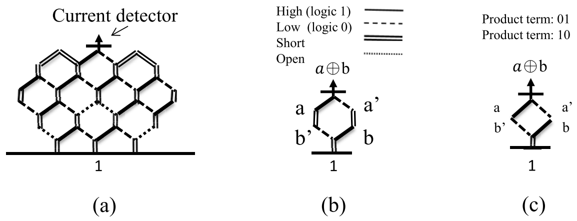


Fig. 3. (a) SET array fabric. (b) Example of a ⊕ b. (c) Simplified diamond-shaped network of a ⊕ b [23].

preceding devices through either its left or its right edge and sends the electrons to its succeeding device depending on the control variable, as shown in Fig. 1(b). The control variable is the input variable of a Boolean circuit. The node device can be realized by controlling nanowires with wrap-gate SET that has two operating modes: 1) active-high (high) and 2) active-low (low). Either the left or the right edge can be high or low. For example, in Fig. 1(b), when the control variable x_i equals 0 and the left (right) edge is low, the node device receives the electrons through the left (right) edge. Similarly, when the control variable x_i equals 1 and the left (right) edge is high, the node device receives the electrons through the left (right) edge. Furthermore, all node devices at the same row in the SET array are controlled by a single-input variable.

The realization of the BDD-based SET architecture in [21] is fixed and not amendable to functional reconfiguration. This structure cannot be changed to implement another different function due to the involvement of an etching process in its realization. Thus, a reconfigurable version of SET that uses wrap gate tunable tunnel barriers was proposed in [22]. This device can be operated in three distinct modes with respect to wrap gate bias-voltages: 1) active; 2) open; and 3) short. Thus, it enables functional reconfiguration and increases the flexibility and reliability of the BDD-based SET arrays [3], [19]–[21].

In parallel with the advances of SET array realization, the automatic synthesis methods were proposed in [23] and [24]. These works sequentially mapped each product term of a Boolean function instead of its whole BDD to solve the mapping problem coming from the nonplannar, i.e., having crossing edges, BDD representation. Furthermore, the proposed mapping approach in [1] focuses on reducing the number of hexagons in the mapped SET arrays using reordering techniques. This paper statically analyzed product terms and variables for extracting features to share nodes and edges in the mapping process. For simplicity, these previous works only allow the same type of configuration, either (high, low) or (low, high), in the whole SET array.

The major purpose of the automatic mapping approach should be to map a circuit within a given SET array area successfully. Although approaches in [1], [23], and [24] minimized the number of hexagons mapped in SET arrays, the reduction of mapped area might not be significant. Minimizing the number of hexagons would not reduce the SET device cost and the SET array area. For instance, Fig. 2(a) and (b) shows an example that different numbers of hexagons result in the same area of SET array. In fact, the area of an SET array is determined by its bounded height and bounded width.[1] Width minimization is more important for an automatic mapping approach to map a circuit into an SET array successfully. As a result, in this paper, we propose an approach to focus on minimizing the width of the SET arrays.

Let us discuss factors influencing the width of mapped SET array. First, fewer product terms usually result in a mapping with a smaller width. Therefore, minimizing the number of product terms is a feasible method. Second, if node or edge sharing among product terms is more, the width could be reduced as well. Finally, if the configuration in each row of SET array is relaxed to different types, more mapping opportunities can be exploited for width minimization. Thus, in this paper, we propose an approach that consists of four techniques: 1) product term minimization; 2) branch-then-share (BTS)-aware variable reordering; 3) SET array architecture relaxation; and 4) BTS-aware product term reordering for minimizing the width of SET arrays.

We conduct the experiments on a set of MCNC and IWLS 2005 benchmarks. The experimental results show that the proposed approach can minimize the width of SET arrays by saving 45% of width, compared with a previous method [1], which focused on the hexagon count minimization. In addition, comparing with a previous work [2], which focused on width minimization, our approach can save 13% of width.

The rest of this paper is organized as follows. Section II reviews the SET device and introduces the background. Section III introduces the proposed approach for minimizing the mapping width of SET arrays. Section IV shows the experimental results. The conclusion is drawn in Section V.

II. Background

This section introduces the SET device and the background of this paper.

A. Reconfigurable BDD-Based SET Array

A reconfigurable BDD-based SET array consists of three layers: 1) Input signal layer; 2) configuration layer; and 3) device layer. The input signal layer involves metal wires to conduct input signals to SET devices. The configuration layer involves metal wires to control the operation modes of SET devices, e.g., high, low, short, and open. The bottom layer is the device layer that is composed of SET device

[1]The height of an SET array is usually the same as the number of inputs in a circuit.

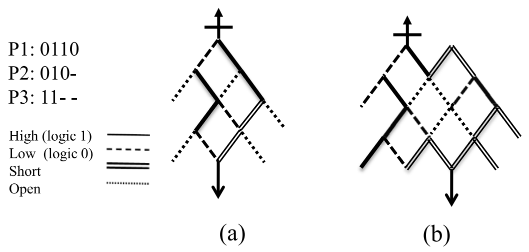


Fig. 4. (a) Constraint-free mapping. (b) Mapping with the fabric constraint.

network [2], [22]. A reconfigurable BDD-based SET array can be represented as a hexagonal network, as shown in Fig. 3(a) [22]. There is a current detector at the top and a current source, represented as 1, at the bottom. When the electrons are transported from the current source to the current detector through a conducting path, which is controlled by the input variables, the current is detected and the output value of the Boolean circuit is 1; otherwise, it is 0. All the sloping edges in the SET array can be configured as high, low, short, or open. A high edge indicates that the corresponding node device operates in active mode controlled by a variable x. Conversely, a low edge is controlled by x', and it is an electrical opposite of a high edge. Furthermore, a short (open) edge is electrically short (open), where the corresponding SET device operates in short (open) mode. For example, Fig. 3(b) shows an implementation of a $\oplus$ b. The current detector detects the current, i.e., the function is evaluated as 1, when either (a = 1, b = 0) (left path) or (a = 0, b = 1) (right path). This behavior exactly represents the function of a $\oplus$ b.

B. Notation

For ease of discussion, a simplified abstract diamond fabric was proposed in [23] and used in [1] and [24]. In the simplified diamond fabric, since all the vertical edges of the hexagons are electrically short, only the sloping edges are preserved. Fig. 3(c) is the corresponding diamond-shaped network of the hexagonal network in Fig. 3(b).

C. Fabric Constraint

Fig. 4(a) shows a configured SET array without imposing any mapping constraint. However, there is a practical mapping constraint called fabric constraint that has to be met. The constraint limits the configurations of an SET array due to the manufacturing limitation [1], [14], [22]–[24]. Hence, our work maps the SET arrays under the fabric constraint as well.

A reconfigurable SET device involves metal wires to configure itself into high, low, short, or open mode. The configuration of metal wires physically dominates the SET array area [2]. In manufacturing, an SET array would have a higher circuit density when the combinations of metal wires in the configuration layer are reduced [22]. To reduce the area, we only configure the combination of the metal wires of an SET device as one of (high, low), (low, high), (short, short), or (open, open), and both the (high, low) and (low, high) configurations are not allowed to appear in the same row simultaneously. This constraint on the configuration combination is called the fabric constraint. We use the notation (high, low) or (low, high) to represent the configuration of each row in an SET array. Fig. 4(b) shows an SET array that meets the fabric constraint. In Fig. 4, P1 $\sim$ P3 are product terms, where "–" means don't care.

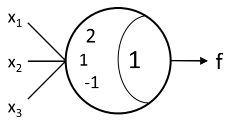


Fig. 5. LTG implementing the function $f = x_1 + x_2 x_3'$.

III. Proposed Approach

In this section, we introduce the proposed approach that consists of four techniques for the width minimization of SET arrays: 1) product term minimization; 2) BTS-aware variable reordering; 3) SET array architecture relaxation; and 4) BTS-aware product term reordering. We will discuss them in Sections III-A–III-D.

A. Product Term Minimization

In general, fewer product terms usually result in a mapping with a smaller width. Therefore, in addition to the BDD-based approach used in [1], [23], and [24], we can use other methods to figure out the product terms of a circuit. Note that these product terms obtained have to be mutually disjoint to each other as obtained from its BDD.[2] This is because only one path can be conducted at a time in an SET array from its physical characteristic [22]. As a result, we compute disjoint product terms from a threshold network representation of a circuit. The threshold network representation of a circuit can be obtained from [25]

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq T + \delta_{\text{on}} \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < T - \delta_{\text{off}}. \end{cases} \quad (1)$$

A threshold network is a network composed of linear threshold gates (LTGs). The parameters of an LTG are weights w_i: $i = 1 \sim n$, which correspond to inputs x_i: $i = 1 \sim n$, and a threshold value T [26]. Parameters δ_{on} and δ_{off} are positive numbers that are used for solving the defect tolerance issue of an LTG implementation. In this paper, we consider a threshold network as a functional representation of a Boolean network and do not consider the implementation issue. Thus, we assume that both parameters δ_{on} and δ_{off} are zero. The output f of an LTG is evaluated by (1) [27], [28].

For example, in Fig. 5, the LTG generates 1 if $2x_1 + x_2 - x_3 \geq 1$, and generates 0 otherwise. Since $\{x_1 = 1\}$ or $\{x_2 = 1, x_3 = 0\}$ can uniquely make the LTG become 1, the Boolean function it represents is $f = x_1 + x_2 x_3'$.

[2]General logic optimization methods, e.g., Espresso, are not applicable to this paper, since the resultant product terms are not disjoint.

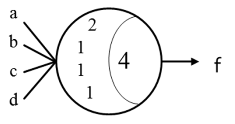


Fig. 6. LTG with a critical input a.

In this paper, we assume the weights and the threshold value of an LTG are integers, and use a weight-threshold vector $\langle w_1, w_2, \ldots, w_n; T\rangle$ to represent an LTG. We also transform the negative weights of an LTG into positive ones by applying a positive–negative weight transformation procedure [26] for ease of analysis in this paper.

1) Background of LTG: We review the background of an LTG [29]—input grouping, critical input, and critical-effect vector—that will be used in our product term computation procedure.

a) Input grouping: Given an LTG, the input grouping is a process that separates the inputs and its corresponding weights into different groups. We know that the relationship between the weighted summation and the threshold value determines the output value of an LTG. If the weight of one input is equal to the threshold value, this input can change the output from 0 to 1 alone. On the other hand, if the weight of an input is smaller than the threshold value, this input needs the weights from the other inputs to change the output from 0 to 1. Thus, the inputs of an LTG can be separated as one or more groups based on this idea.

b) Critical input: Given a single-group LTG, an input x_j with its corresponding weight w_j is critical if and only if the LTG satisfies (2), where n is the number of inputs in this gate

$$\sum_{i=1, i\neq j}^{n} w_i < T. \tag{2}$$

That is, the LTG will become a useless gate[3] when x_j is assumed 0. For example, in Fig. 6, input a is critical because the summation of weights of inputs $b \sim d$ is less than the threshold value.

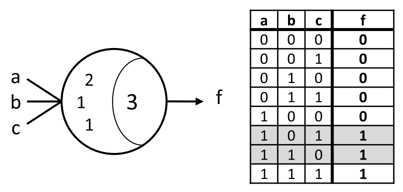
By the definition of the critical input, we know that assigning 0 to a critical input of a single-group LTG can make this gate become a useless LTG. This characteristic of a single-group LTG can be used to compute the product terms of an LTG.

c) Critical-effect vector: Given a single-group LTG, it has a critical-effect if it satisfies (3), where n is the number of inputs in this gate

$$\sum_{i=1}^{n} x_i w_i = T. \tag{3}$$

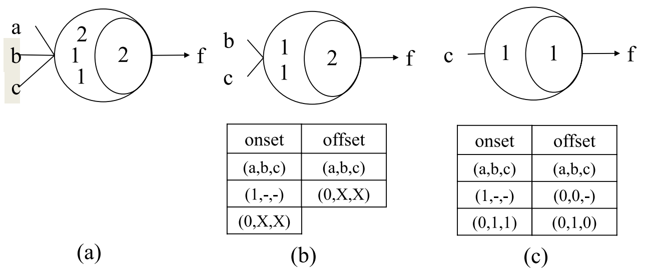
An input assignment that satisfies the requirement of the critical-effect for an LTG is called a critical-effect vector. For example, in Fig. 7, input assignments 101 and 110 are the critical-effect vectors of LTG $\langle 2, 1, 1; 3\rangle$ due to the satisfaction of (3). Changing any input from 1 to 0 in the critical-effect vectors will also change the output from 1 to 0. In addition, (3) is a necessary condition if the given LTG is obtained from an ILP-based synthesis algorithm [25]. That means all the critical-effect vectors of an LTG satisfy (3).

[3]Given a nonempty LTG, it is a useless LTG if and only if it always has the output value of 0.



a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Fig. 7. LTG and its critical-effect vectors [29].



(b)

onset	offset
(a,b,c)	(a,b,c)
(1,-,-)	(0,X,X)
(0,X,X)	

(c)

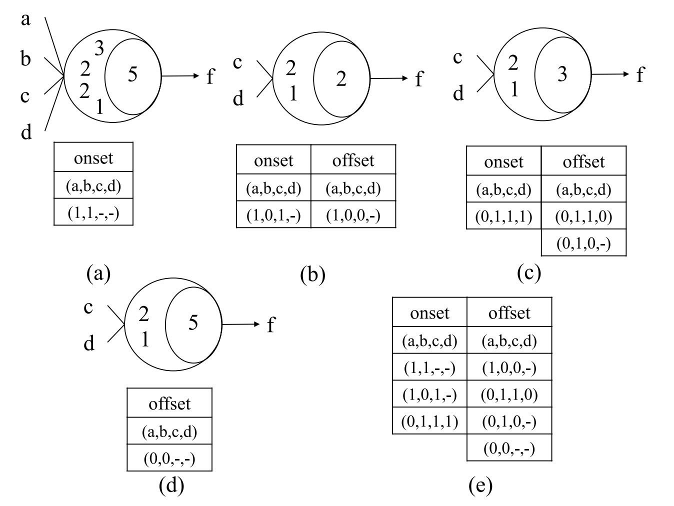
onset	offset
(a,b,c)	(a,b,c)
(1,-,-)	(0,0,-)
(0,1,1)	(0,1,0)

Fig. 8. Example for demonstrating the computation of onset and offset from an LTG having a single-input group. (a) LTG $\langle 2, 1, 1; 2\rangle$. (b) Temporarily modified LTG considering a = 0 and the derived onset and offset. (c) Temporarily modified LTG considering $(a, b) = (0, 1)$ and the final onset and offset of the LTG $\langle 2, 1, 1; 2\rangle$.

2) Onset and Offset Computation for an LTG: Before computing the product terms of a threshold network, we have to compute the onset and offset of each LTG in the threshold network. We first consider the case that an input is a single-input group of an LTG. Since this input can independently change the output from 0 to 1 when it is assigned 1, we can first compute the onset that involves this input of 1. We use an example to demonstrate this procedure. Given an LTG $\langle 2, 1, 1; 2\rangle$, as shown in Fig. 8(a), this LTG has two groups and represents the function $f(a, b, c) = a + bc$. If we assign 1 to the input a, the output becomes 1 under any combinations of b and c. Hence, we can realize $(a, b, c) = (1, -, -)$ is one of the onsets of this LTG, where "$-$" denotes don't care. Then, we consider the situation that the input a is assigned to 0. Since a can be ignored when it is assigned to 0, the LTG can be temporarily modified as $\langle 1, 1; 2\rangle$, as shown in Fig. 8(b) for deriving other onset elements. In Fig. 8(b), we also explicitly express the undetermined onset and offset of $(a, b, c) = (0, X, X)$ in the table, where X denotes unknown. The onset and offset expressions with X have to be refined to the expressions that do not contain X in the succeeding procedure.

Next, we consider the situation that an input is a critical input of an LTG. According to the definition of a critical input, we know that if the critical input is assigned to 0, the output is 0. Thus, one of the offset of this LTG is the one that involves this critical input of 0. For example, in the modified LTG of Fig. 8(b), the input b is a critical input.[4] Hence, we can realize $(b, c) = (0, -)$ is one of the offset of this modified LTG. Combining the input a of 0 with $(b, c) = (0, -)$,

[4]The input c is also a critical input, we can also consider the input c first.



(a)

onset
(a,b,c,d)
(1,1,-,-)

(b)

onset	offset
(a,b,c,d)	(a,b,c,d)
(1,0,1,-)	(1,0,0,-)

(c)

onset	offset
(a,b,c,d)	(a,b,c,d)
(0,1,1,1)	(0,1,1,0)
	(0,1,0,-)

(d)

offset
(a,b,c,d)
(0,0,-,-)

(e)

onset	offset
(a,b,c,d)	(a,b,c,d)
(1,1,-,-)	(1,0,0,-)
(1,0,1,-)	(0,1,1,0)
(0,1,1,1)	(0,1,0,-)
	(0,0,-,-)

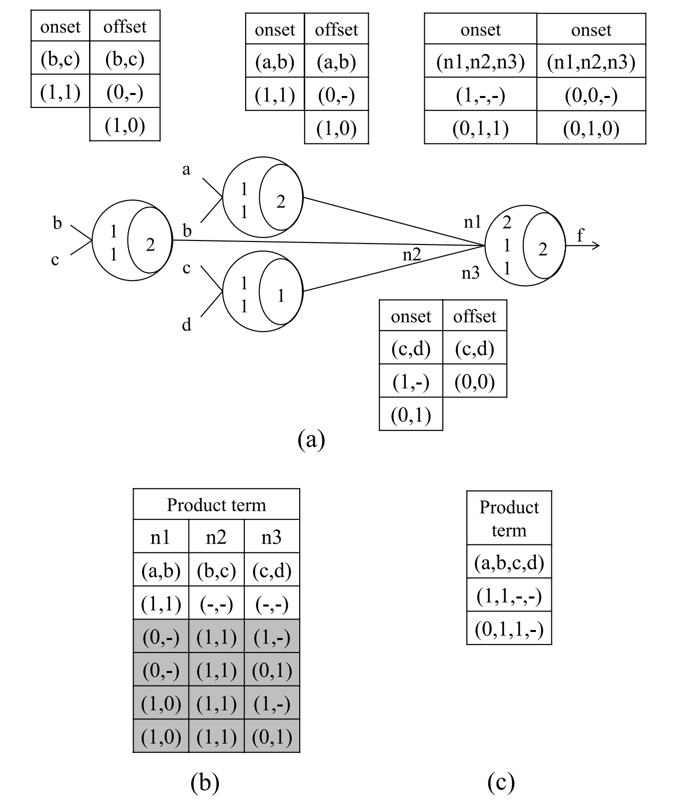
Fig. 9. Example for demonstrating the computation of onset and offset from an LTG having no single-input groups and no critical inputs. (a) LTG $\langle 3, 2, 2, 1; 5\rangle$ and one of its onset. (b) Modified LTG and derived onset and offset considering $(a, b) = (1, 0)$. (c) Modified LTG and derived onset and offset considering $(a, b) = (0, 1)$. (d) Modified LTG and derived onset and offset $(a, b) = (0, 0)$. (e) Final onset and offset.

we obtain $(a, b, c) = (0, 0, -)$, which is one of the offset of this LTG, as shown in the table of Fig. 8(c). Then, we consider the situation that the input b is assigned to 1 [$(a, b) = (0, 1)$]. Note that we also have to modify this LTG where b is assigned to 1. Since the input b has been assigned to 1, we remove the input b from the LTG and decrease the threshold value by the weight of input b from the LTG. Thus, the resultant LTG becomes $\langle 1; 1\rangle$, as shown in Fig. 8(c). For this modified LTG in Fig. 8(c), we can compute its onset and offset as $(c) = (1)$ and $(c) = (0)$, respectively. Combining $(a, b) = (0, 1)$ with c, we obtain another onset $(a, b, c) = (0, 1, 1)$ and another offset $(a, b, c) = (0, 1, 0)$ of this LTG. Since c is the last input of the LTG, the onset and offset of the original LTG $\langle 2, 1, 1; 2\rangle$ have been completely derived, as shown in Fig. 8(c).

In the example of Fig. 8, we use the concepts of single-input group, critical input, and an LTG modification process to derive the onset and offset of an LTG. However, some LTGs might have neither single-input groups nor critical inputs. Thus, for these LTGs, we derive their critical-effect vectors instead. Since the inputs have been sorted by weights, we can easily obtain the critical-effect vectors by solving (3) [30]. By the definition of critical-effect vector, we know that the output of a critical-effect vector is always 1. Hence, the critical-effect vector is definitely the onset of the LTG. For the LTG that has more than one critical-effect vectors, we first derive the critical-effect vector that has the least number of 1. This is because this critical-effect vector can form a bigger product term,[5] such that the total number of product terms can be minimized. We use an example, as shown in Fig. 9, to demonstrate this computation process with the critical-effect vectors.

Given an LTG $\langle 3, 2, 2, 1; 5\rangle$, as shown in Fig. 9(a), this LTG represents the function $f(a, b, c, d) = a(b + c) + bcd$. Since there is no single-input groups and no critical inputs in this LTG, we derive the critical-effect vector having the least number of 1, 1100. From this critical-effect vector, we can compute $(a, b, c, d) = (1, 1, -, -)$ is one of the onset of this LTG, as shown in Fig. 9(a). Then, we consider the remaining combinations $(a, b) = (1, 0), (0, 1)$, or $(0, 0)$, as shown in Fig. 9(b)~(d), respectively. Consider the inputs $(a, b) = (1, 0)$, because the inputs (a, b) are $(\mathbf{1}, \mathbf{0})$, the threshold value of this LTG is modified as $5-3 \times \mathbf{1}-2 \times \mathbf{0} = 2$ and the modified LTG is shown as Fig. 9(b) by removing the inputs a and b. Since this modified LTG has a single-input group, we can easily compute its onset as $(c, d) = (1, -)$ and offset as $(c, d) = (0, -)$ by the method mentioned previously. Combining $(a, b) = (1, 0)$ with $(c, d) = (1, -)$ or $(0, -)$, we obtain one onset of the original LTG $(a, b, c, d) = (1, 0, 1, -)$; one offset $(a, b, c, d) = (1, 0, 0, -)$. Similarly, if the inputs (a, b) is $(0, 1)$, or $(0, 0)$, the LTG is modified as Fig. 9(c) or (d), respectively. In Fig. 9(c), the onset is derived as $(0, 1, 1, 1)$, and the offset is derived as $(0, 1, 1, 0)$ and $(0, 1, 0, -)$. In Fig. 9(d), because this modified LTG is a useless LTG, we can realize $(a, b, c, d) = (0, 0, -, -)$ is one of the offset of LTG. In summary, all the onset and offset of the LTG $\langle 3, 2, 2, 1; 5\rangle$ are derived, as shown in Fig. 9(e).

[5]A bigger product term means that it contains more minterms.



(a)

onset	offset
(b,c)	(b,c)
(1,1)	(0,-)
	(1,0)

onset	offset
(a,b)	(a,b)
(1,1)	(0,-)
	(1,0)

onset	onset
(n1,n2,n3)	(n1,n2,n3)
(1,-,-)	(0,0,-)
(0,1,1)	(0,1,0)

onset	offset
(c,d)	(c,d)
(1,-)	(0,0)
(0,1)	

(b)

Product term		
n1	n2	n3
(a,b)	(b,c)	(c,d)
(1,1)	(-,-)	(-,-)
(0,-)	(1,1)	(1,-)
(0,-)	(1,1)	(0,1)
(1,0)	(1,1)	(1,-)
(1,0)	(1,1)	(0,1)

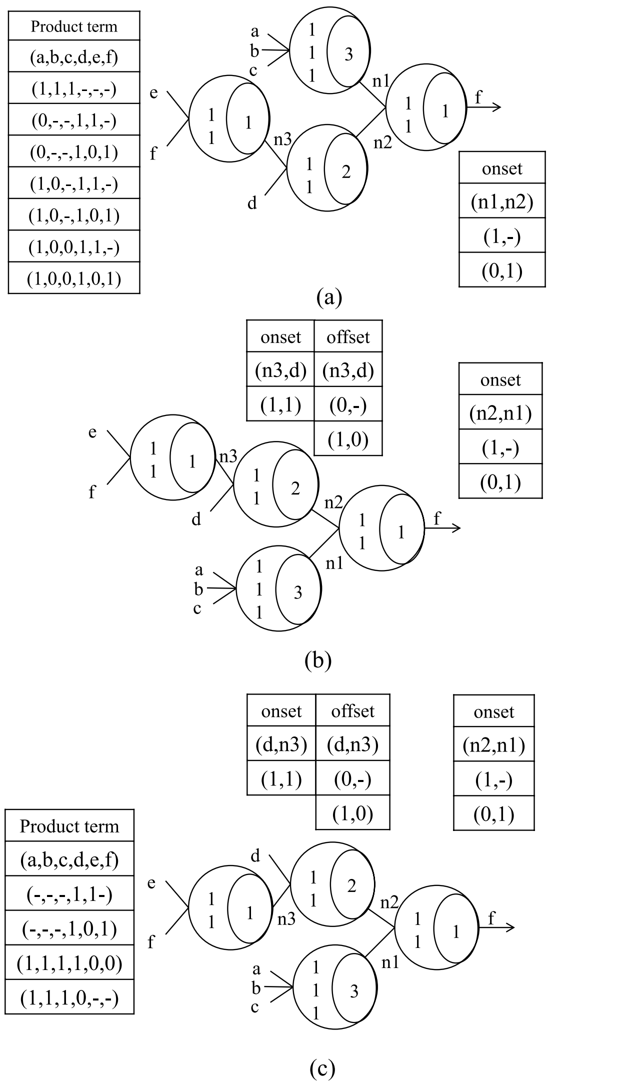
(c)

Product term
(a,b,c,d)
(1,1,-,-)
(0,1,1,-)

Fig. 10. Example for demonstrating the computation of product terms from a threshold network. (a) Threshold network and the precomputed onset and offset of each LTG. (b) Derived input assignments from (a). (c) Resultant product terms.

In this computation process, we can ensure that the product terms in the onset and offset are disjoint. This is because we disjointly decompose the input space.

3) Product Term Computation From a Threshold Network: After computing the onset and offset of each LTG, we then compute the product terms of a threshold network. Given a single-output threshold network, we compute its disjoint product terms from the primary output (PO) toward the primary



Product term
(a,b,c,d,e,f)
(1,1,1,-,-,-)
(0,-,-,1,1,-)
(0,-,-,1,0,1)
(1,0,-,1,1,-)
(1,0,-,1,0,1)
(1,0,0,1,1,-)
(1,0,0,1,0,1)

onset
(n1,n2)
(1,-)
(0,1)

onset	offset
(n3,d)	(n3,d)
(1,1)	(0,-)
	(1,0)

onset
(n2,n1)
(1,-)
(0,1)

onset	offset
(d,n3)	(d,n3)
(1,1)	(0,-)
	(1,0)

onset
(n2,n1)
(1,-)
(0,1)

Product term
(a,b,c,d,e,f)
(-,-,-,1,1-)
(-,-,-,1,0,1)
(1,1,1,1,0,0)
(1,1,1,0,-,-)

Fig. 11. Example of product term minimization via swapping the symmetric inputs. (a) Original threshold network and the computed product terms. (b) Threshold network after swapping n_1 with n_2. (c) Resultant threshold network after swapping n_3 with d from (b) and the computed product terms.

inputs (PIs) in a breadth-first search manner. The product terms (onset) of the whole threshold network then can be derived level by level. This is because for two connected LTGs, $G_i \rightarrow G_{i+1}$, the input space of G_{i+1} represents the function of G_i. In particular, if one input of G_{i+1} is assigned to 1 (0), it represents the onset (offset) of G_i. Thus, each input of an LTG can be replaced by the onset (offset) of its fanin gate when the input assignment is 1 (0).

We use an example, as shown in Fig. 10, to demonstrate the product term computation from a threshold network. Fig. 10(a) shows a given threshold network and the precomputed onset and offset of each LTG.

The elements in the onset of the LTG f $\langle 2, 1, 1; 2\rangle$ are $(n_1, n_2, n_3) = (1, -, -)$ and $(0, 1, 1)$, as shown in Fig. 10(a). For the first element $(n_1, n_2, n_3) = (1, -, -)$, we assign 1 to n_1. Thus, we use the onset of n_1 to derive the input assignments, $(a, b) = (1, 1)$. Then, we also assign $-$ to n_2 and n_3, and the corresponding input assignments are $(b, c) = (-, -)$ and $(c, d) = (-, -)$, as shown in the fourth row of Fig. 10(b). Similarly, the second element $(n_1, n_2, n_3) = (0, 1, 1)$ in the

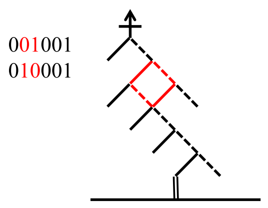


Fig. 12. Example of 2-bit BTS.

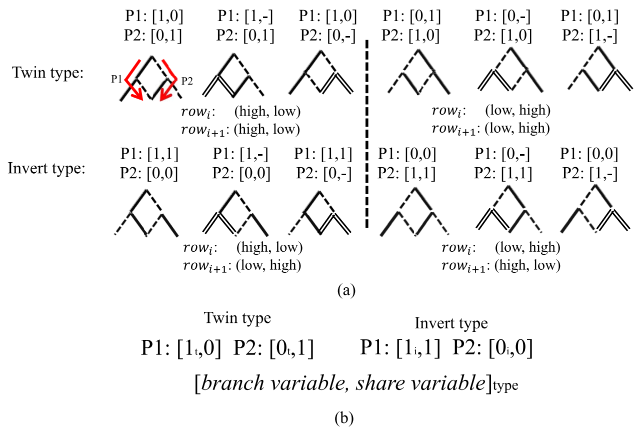


Fig. 13. (a) Types of BTS. (b) Notations of BTS.

onset is also used to compute the product terms of LTGs, as shown in the last four ($2 \times 1 \times 2$) rows of Fig. 10(b).

After having the input assignments in the product term table of Fig. 10(b), we need to further refine these input assignments such that no conflict exists for a single input. For example, in the fourth row of Fig. 10(b), the input assignments of $(a, \mathbf{b})$, $(\mathbf{b}, c)$ are $(1, \mathbf{1})$, $(-, -)$, respectively. Because "$-$" represents either 0 or 1, we can assign b as 1 without causing any conflicts. Thus, the input assignment of the fourth row can be refined as $(a, b, c, d) = (1, 1, -, -)$, as shown in the third row of Fig. 10(c).

For the product term that has conflict values, 0 and 1, in one variable, we discard this product term. For example, in the last three rows of Fig. 10(b), there exists a value conflict to either variable b or c. Therefore, we discard all of them. As a result, the resultant product terms of this threshold network are obtained and summarized in Fig. 10(c).

Next, we propose a technique that swaps the symmetric inputs in an LTG to obtain a smaller set of product terms. Since the don't care bit "$-$" covers 1 and 0, and having more "$-$" in a product term generally causes the product term set of a function smaller, we prefer to create more don't care bits in one product term during the computation process. Based on this preference, we propose a method that changes the locations of symmetric inputs in an LTG as follows, so that more "$-$" appear in one product term.

For an LTG with symmetric-inputs, we first check the locations of "$-$" in its onset and offset. Next, we check the fanin gate sizes[6] of these symmetric inputs, and swap these symmetric inputs so that "$-$" corresponds to a larger fanin gate.

For example, in Fig. 11(a), the LTG f $\langle 1, 1; 1\rangle$ has two symmetric inputs. In its onset, since the location of "$-$" is

[6]The fanin gate size is defined as the number of inputs in the fanin gate.

(a)

	a	b	c	d	e	f	g	h	i
P1	0	-	0	1	1	0	1	-	0
P2	0	-	1	1	1	-	0	-	1
P3	0	-	1	1	0	0	0	1	0
P4	0	-	1	1	0	0	1	-	0
P5	0	-	1	1	1	0	0	-	0
P6	1	-	0	1	-	-	1	1	1
P7	1	-	1	1	1	-	1	1	1

(b)

	d	b	a	c	e	f	g	h	i
P1	1	-	0	0	1	0	1	-	0
P2	1	-	0	1	1	-	0	-	1
P3	1	-	0	1	0	0	0	1	0
P4	1	-	0	1	0	0	1	-	0
P5	1	-	0	1	1	0	0	-	0
P6	1	-	1	0	-	-	1	1	1
P7	1	-	1	1	1	-	1	1	1
MAX:			5	5	4	4	4	4	4

(c)

	d	b	a	c	e	f	g	h	i
P1	1	-	0	0	1	0	1	-	0
P2	1	-	0	1	1	-	0	-	1
P3	1	-	0	1	0	0	0	1	0
P4	1	-	0	1	0	0	1	-	0
P5	1	-	0	1	1	0	0	-	0
MAX:				4	3	4	3	4	4

(d)

	d	b	a	c	e	f	g	h	i
P2	1	-	0	1	1	-	0	-	1
P3	1	-	0	1	0	0	0	1	0
P4	1	-	0	1	0	0	1	-	0
P5	1	-	0	1	1	0	0	-	0
MAX:					2	3	3	3	3

(e)

	d	b	a	c	f	e	g	h	i
P3	1	-	0	1	0	0	0	1	0
P4	1	-	0	1	0	0	1	-	0
P5	1	-	0	1	0	1	0	-	0
MAX:						2	2	2	3

(f)

	d	b	a	c	f	i	e	g	h
P3	1	-	0	1	0	0	0	0	1
P4	1	-	0	1	0	0	0	1	-
P5	1	-	0	1	0	0	1	0	-
MAX:							2	2	2

Fig. 14. (a) Original variable order. (b) Move all-share variables to the left. (c)–(f) Reorder variables according to the quantity of bit values.

the second input and n_1 is a larger fanin gate, we swap n_1 with n_2. The new threshold network is shown in Fig. 11(b). Next, we consider the LTG $n2$ in Fig. 11(b), which also has two symmetric inputs. The location of "—" is in the second input of the offset, and n_3 is a larger fanin gate than d. Thus, we swap n_3 with d accordingly. The resultant threshold network is shown in Fig. 11(c) and the number of computed disjoint product terms is reduced from 7, as shown in Fig. 11(a), to 4.

In this paper, for each output function, we compare the number of product terms computed from the LTG-based method and the BDD-based method, and choose the smaller set of product term for SET array mapping.

B. BTS-Aware Variable Reordering

1) BTS Basis: The concept that two-product terms share some path segment in an SET array was proposed in [1]. These product terms are named BTS product terms, as they branch in one row and merge in the succeeding row such that the remaining path segments are shared with edges. If some product terms are BTS and mapped adjacently, the mapped width can be reduced. As a result, the more BTS are identified in the mapping process, the smaller width results in. For example, in Fig. 12, the product terms 0**01**001 and 0**10**001 are BTS, where only two variables are configured as different types of edges in the second and third rows. These product terms branch at the second row and merge at the third row such that the remaining path segments are shared. Therefore, the width of the resultant array is minimized.

In this paper, we exploit different types of configurations among the rows to create more opportunities for BTS. The configuration for two consecutive rows is classified into two share types: 1) twin type and 2) invert type. Hence, we enumerate bit value combinations for two consecutive rows, as shown in Fig. 13(a), where twin type (invert type) represents that the BTS occurs at two consecutive rows that have the same (opposite) configurations. We use subscripts t (for twin), and i (for invert) to denote the BTS types, as shown in Fig. 13(b). Note that we only mark BTS with t and i at the branch row for simplification. For those variables in the BTS, we call them branch variable and share variable, as shown in Fig. 13(b).

Since the BTS structure is beneficial to width reduction, we would like to identify as many BTS as possible using a two-phase variable reordering method as introduced in the following sections.

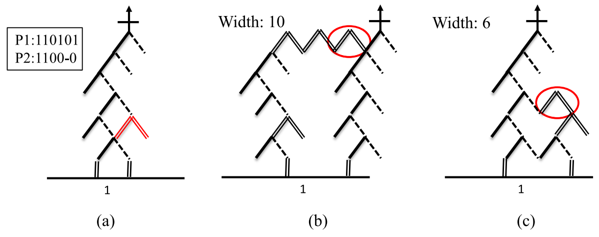


Fig. 15. (a) Invalid path creation for mapping P2. (b) Expansion at the first row for mapping P2. (c) Expansion at the third row for mapping P2.

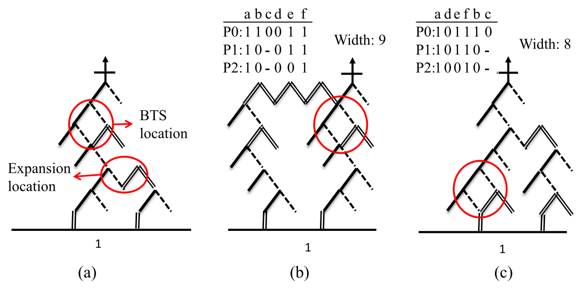
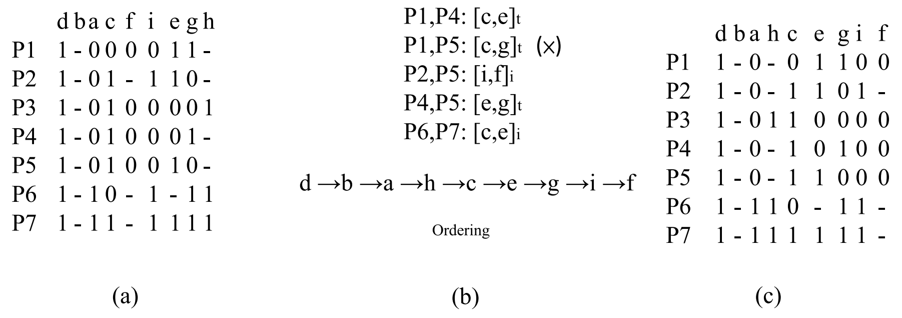


Fig. 16. (a) Invalid path occurs when a product term expands at a location lower than BTS location. (b) Correct mapping when expansion is at the first row. (c) Correct mapping when variables are reordered.

2) Variable Reordering: To create more share edges among product terms, we first move the variables that have the same value among all the product terms, called all-share variables, to the front-end of the variables. Since don't care "—" provides more flexibility to the succeeding mapping process, the all-share variables with don't care are arranged in the back-end of all-share variables.

For example, given seven product terms, as shown in Fig. 14(a), we move all-share variables b and d to the front-end of all variables, as shown in Fig. 14(b). Moreover, since variable b is "—" and d is 1, d is followed by b. Next, the order of the remaining variables is based on their quantities of values, 0, 1, and "—" among all the product terms. Since variables a and c have a maximal number of certain value, one of them is arranged next. For the situation that two variables have the same maximal value, their order is set as the original order. Therefore, we put a next to b. Next, since the determined variable order is used for all the product terms, we determine the order based on the product terms that have more same value. Therefore, we ignore P6 and P7, and choose the variable c as the next variable, as shown in Fig. 14(c).



(a)

	d	b	a	c	f	i	e	g	h
P1	1	-	0	0	0	0	1	1	-
P2	1	-	0	1	-	1	1	0	-
P3	1	-	0	1	0	0	0	0	1
P4	1	-	0	1	0	0	0	1	-
P5	1	-	0	1	0	0	1	0	-
P6	1	-	1	0	-	1	-	1	1
P7	1	-	1	1	-	1	1	1	1

(b)

P1,P4: $[c,e]_t$
P1,P5: $[c,g]_t$ (×)
P2,P5: $[i,f]_i$
P4,P5: $[e,g]_t$
P6,P7: $[c,e]_i$

$d \rightarrow b \rightarrow a \rightarrow h \rightarrow c \rightarrow e \rightarrow g \rightarrow i \rightarrow f$

Ordering

(c)

	d	b	a	h	c	e	g	i	f
P1	1	-	0	-	0	1	1	0	0
P2	1	-	0	-	1	1	0	1	-
P3	1	-	0	1	1	0	0	0	0
P4	1	-	0	-	1	0	1	0	0
P5	1	-	0	-	1	1	0	0	0
P6	1	-	1	1	0	-	1	1	-
P7	1	-	1	1	1	1	1	1	-

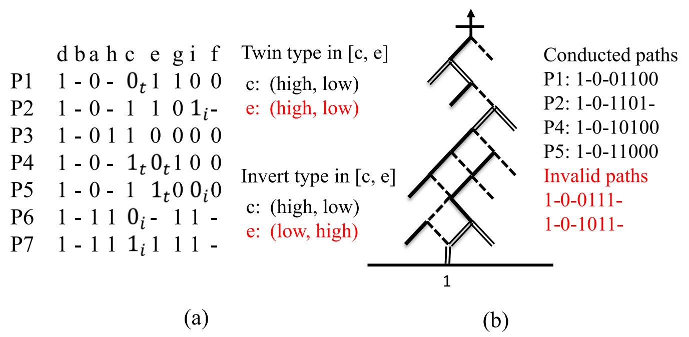
Fig. 17. (a) New product terms after the first phase of variable reordering. (b) BTS collection and variable ordering. (c) New product terms after the second phase of variable reordering.

We keep selecting the variable with a maximal value until all the variables are reordered. Fig. 14(d)–(f) shows the remaining steps, and the order in Fig. 14(f) is the result of first phase.

When space for mapping a product term is not enough, we have to apply expansion operations during the mapping. Without expansion, invalid paths might occur. For example, configuring P2 after P1, as shown in Fig. 15(a), will create invalid paths that represent product terms 1100-1 and 110100. Fig. 15(b) and (c) shows the correct mapping results of Fig. 15(a) with expansion at different rows. From this example, we observed that expanding at a lower row could save more width in an SET array. However, an invalid path might be created when the expansion level is lower than the BTS location. Fig. 16(a) shows such an example that creates an invalid path when expansion occurs at a level lower than BTS location. Fig. 16(b) shows a correct mapping that expands at a higher row. Fig. 16(c) shows a mapping with a smaller width after moving BTS to a lower row using another variable order. Thus, we further reorder the variables such that the BTS location is moved as lower as possible for width minimization.

For identifying BTS, we scan the product terms pairwisely according to the BTS types in Fig. 13. For the last example with an initial variable order in Fig. 17(a), P1 and P4 are BTS with branch variable c and share variable e. After recording all the BTS, we calculate the number of BTS each variable involved. We then choose the branch variable that is involved in most BTS and put its share variable next to it. If this share variable is also a branch variable of another BTS, put its share variable next to it as well. Otherwise, choose a variable from the remaining variables that is involved in most BTS. Finally, we insert the variables that are not branch variables in the front-end of the ordered BTS variables to lower the BTS location.

For example, in Fig. 17(b), c is a branch variable involved in most BTS. Therefore, we choose c and put its share variable e next to it. Since e also is a branch variable, e is followed by its share variable g. Next, we select i from the remaining variables and i is followed by its share variable f. The partial ordering of these variables is $c \rightarrow e \rightarrow g \rightarrow i \rightarrow f$. Finally, we insert h in the front-end of the ordered BTS variables to lower the BTS location. The order becomes $d \rightarrow b \rightarrow a \rightarrow h \rightarrow c \rightarrow e \rightarrow g \rightarrow i \rightarrow f$, as shown in Fig. 17(b). Note that the P1 and P5 are not a BTS anymore since the variables c and g are not adjacent under this variable order.



	d	b	a	h	c	e	g	i	f
P1	1	-	0	-	0_t	1	1	0	0
P2	1	-	0	-	1	1	0	1_i	-
P3	1	-	0	1	1	0	0	0	0
P4	1	-	0	-	1_t	0_t	1	0	0
P5	1	-	0	-	1	1_t	0	0_i	0
P6	1	-	1	1	0_i	-	1	1	-
P7	1	-	1	1	1_i	1	1	1	-

Fig. 18. (a) Type conflict. (b) Group conflict.

Fig. 17(c) shows the new product terms after the second phase of variable reordering. Note that not all BTS structures can be implemented simultaneously under one variable ordering, we will discuss this in Section III-B3.

3) BTS Identification: First, we introduce the share group that we will use in the BTS identification. In each product term, one BTS or consecutive BTS, either twin type or invert type, form a share group. For example, in Fig. 18(a), P4 has two-consecutive BTS forming a share group, P3 has no share group, P5 has two share groups, and the others have one share group.

Then, we define two kinds of conflicts between BTS. The first one is named type conflict. Due to the fabric constraint, one variable can have only one configuration. Hence, the invert type and twin type are not allowed to occur in the same variable. Once the BTS requires both types to appear in the same row, it is called type conflict. For example, in Fig. 18(a), the configuration of variable e is required to be both (high, low) and (low, high) due to the BTS requirement. However, it violates the fabric constraint.

The other conflict is group conflict. If more than one share group are kept in a product term, it is called group conflict. This is because more than one share group in a product term might create invalid paths. For example, in Fig. 18(b), invalid paths 1-0-0111- and 1-0-1011- occur due to two-share groups in a product term. As a result, we choose one share group in P5 to avoid this group conflict.

Fig. 19 shows the process of BTS identification on the same example. Fig. 19(a) shows all the BTS with subscripts t and i where the same BTS type uses the same subscript. We first

	d	b	a	h	c	e	g	i	f
P1	1	-	0	-	0_t	1	1	0	0
P2	1	-	0	-	1	1	0	1_i	-
P3	1	-	0	1	1	0	0	0	0
P4	1	-	0	-	1_t	0_t	1	0	0
P5	1	-	0	-	1	1_t	0	0_i	0
P6	1	-	1	1	0_i	-	1	1	-
P7	1	-	1	1	1_i	1	1	1	-

(a)

	d	b	a	h	c	e	g	i	f
P1	1	-	0	-	0_t	1	1	0	0
P2	1	-	0	-	1	1	0	1_i	-
P3	1	-	0	1	1	0	0	0	0
P4	1	-	0	-	1_t	0_t	1	0	0
P5	1	-	0	-	1	1_t	0	0_i	0
P6	1	-	1	1	0	-	1	1	-
P7	1	-	1	1	1	1	1	1	-

(b)

	d	b	a	h	c	e	g	i	f
P1	1	-	0	-	0_t	1	1	0	0
P2	1	-	0	-	1	1	0	0	-
P3	1	-	0	1	1	0	0	0	0
P4	1	-	0	-	1_t	0_t	1	0	0
P5	1	-	0	-	1	1_t	0	0	0
P6	1	-	1	1	0	-	1	1	-
P7	1	-	1	1	1	1	1	1	-

(c)

Fig. 19. Process of BTS identification. (a) Original BTS types. (b) Updated BTS types after discarding the invert type in variable c. (c) Final result of the BTS identification.

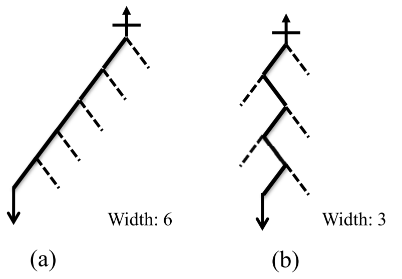


Fig. 20. Mapping result when (a) only one configuration is allowed in an SET array. (b) Hybrid configuration is allowed in an SET array.

eliminate the type conflict. We check each variable to see whether it has both twin and invert types, and choose the type that can support more BTS. In Fig. 19(a), there is a type conflict in variable c. Since the twin type can support three product terms (P1, P4, and P5), which are more than that the invert type can support, we choose the twin type and discard the invert type. After checking all the variables, we check group conflict for each product term. In Fig. 19(b), there is a group conflict in P5. We calculate the number of BTS each share group can support, and choose the first share group in P5. This is because it can support three BTS (P1, P4, and P5), which are more than that the second-share group supports. Fig. 19(c) shows the final result after the BTS identification.

C. SET Array Architecture Relaxation

The previous work [1] constrained the rows of an SET array to be either all (high, low) or all (low, high) for simplifying the mapping algorithm. This restriction, however, may lose the opportunities for achieving a smaller width in SET arrays due to inflexibility. Thus, in this paper, we relax the restriction such that either (high, low) or (low, high) is allowed to configure each row of SET arrays. We call this relaxed architecture hybrid architecture. Note that this architecture relaxation does not violate the mentioned fabric constraint. With the hybrid architecture, more BTS structures are possibly created, and a product term can be mapped with a smaller width, as shown in the example of Fig. 20.

In this stage, to determine the configuration of each row in the hybrid architecture, we check the values in variables among all the product terms under the precomputed variable order. Without loss of generality, we configure the first row as (high, low). If there exists a BTS between two variables, we determine the configuration according to its BTS type. That is, as mentioned in Fig. 13(a), the twin type BTS requires the configurations of two adjacent rows to be identical, and the invert type BTS requires the configurations of two adjacent rows to be different. For the other variables, we set a row's configuration as the same as its previous row if more than half of the variable's bit values are changed among all the product terms. Otherwise, we set the row's configuration as the opposite one to its previous row.

For example, in Fig. 21(a), the configuration of the first row is set as (high, low) by default. For the second row (variable b), since more than half of the product terms change the bit values against the first row, we set the second row as same as (high, low). Then, we determine the following three rows' configurations as (high, low) since they all change more than half of bit values among the product terms. For the sixth row (variable e), we determine it as (high, low) since it is a twin type BTS with the previous row. The seventh row is configured as (high, low) due to the same reason. For the eighth row, we determine it as (low, high) since less than half of the bit values are changed. Similarly, the last row is configured as (high, low). Fig. 21(a) lists the final configurations of the SET array under the hybrid architecture.

D. BTS-Aware Product Term Reordering

In this section, we then introduce the product term reordering algorithm. Since our approach sequentially maps the product terms to form the SET array, the ordering of product terms affects the mapping width. The proposed product term reordering method consists of two steps and they are: 1) grouping tree construction and 2) product term order determination.

1) Grouping Tree Construction: By building a grouping tree for all the product terms, we can realize which product terms have better share relationships with others. With this information, the mapping order can be determined. In the beginning, all the product terms are in one group. We scan all the product terms from the first variable until one product term has different bit values with the other product terms in the same variable. The product terms with the same bit value in the variable are grouped into one group. We keep grouping the product terms until each group contains only one or two product terms, which is called a leaf group. This is because such grouping facilitates the order determination among the product terms. For example, in Fig. 21(b), since the bit values in the variables d and b are the same for all the product terms, and the third variable of P6 and P7 is 1, P6 and P7 are grouped as a leaf group. Next, the similar procedure is applied to the other group of P1 $\sim$ P5. The final grouping tree is shown in Fig. 21(b).

2) Product Term Order Determination: After constructing the grouping tree, we determine the product term order. First, we choose a BTS product term having the lowest branch variable location to map. Then, we choose the product term that belongs to the same BTS. Next, we choose the product term that is within the same leaf group. If there are more than one BTS product term having the same lowest branch variable location, we determine their orders by the levels in the grouping tree in the descending order. If there are no BTS

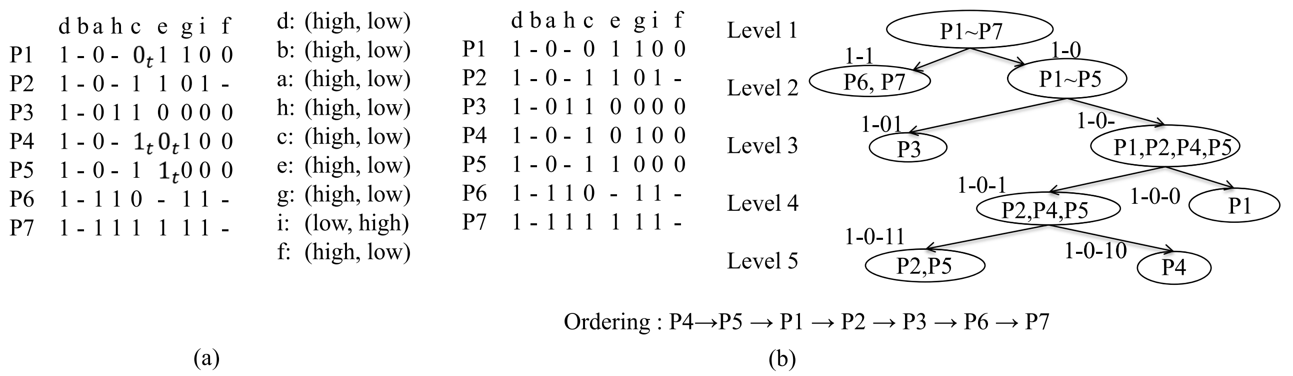


Fig. 21. (a) Determined configuration in hybrid architecture. (b) Demonstrate of product term reordering.

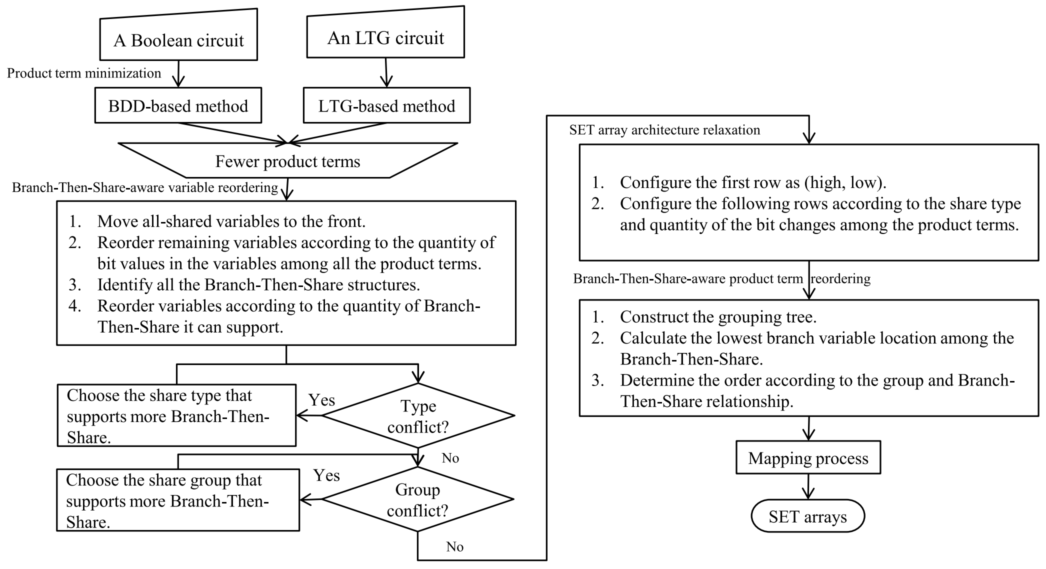

A Boolean circuit
An LTG circuit
Product term minimization
BDD-based method
LTG-based method
Fewer product terms
Branch-Then-Share-aware variable reordering
1. Move all-shared variables to the front.
2. Reorder remaining variables according to the quantity of bit values in the variables among all the product terms.
3. Identify all the Branch-Then-Share structures.
4. Reorder variables according to the quantity of Branch-Then-Share it can support.
Choose the share type that supports more Branch-Then-Share.
Yes
Type conflict?
No
Choose the share group that supports more Branch-Then-Share.
Yes
Group conflict?
No
SET array architecture relaxation
1. Configure the first row as (high, low).
2. Configure the following rows according to the share type and quantity of the bit changes among the product terms.
Branch-Then-Share-aware product term reordering
1. Construct the grouping tree.
2. Calculate the lowest branch variable location among the Branch-Then-Share.
3. Determine the order according to the group and Branch-Then-Share relationship.
Mapping process
SET arrays

Fig. 22. Proposed algorithm.

product terms, we determine the order from the bottom to the top and from the smaller leaf groups to larger ones in the grouping tree. If there are no rules to distinguish two product terms, we follow their original orders.

For example, in Fig. 21(b), we first choose P4, then P5 to map since P4 is a BTS with the lowest branch variable location and P5 is its corresponding BTS product term. Then, we map P1 since it is the corresponding BTS product term of P5. Next, we map P2 since P2 and P5 are in the same leaf group. Next, P3 is mapped according to the level of the group tree. Finally, P6 and P7 are mapped according to the original order. The final order of these product terms is shown in Fig. 21(b).

E. Proposed Algorithm

In this section, we summarize the proposed algorithm, which is shown in Fig. 22. Given the Boolean netlist and the LTG network of a benchmark, we first derive the product terms from its BDD and LTG network, and choose the set having fewer product terms to map. Then, we use a two-phase variable reordering technique to determine a variable order used in the mapping process. In this stage, BTS product terms are analyzed and collected first, and the useful BTS are identified by eliminating conflict cases. Next, we determine the configuration of each row of SET array based on the BTS types and the quantity of bit value changes among all the product terms. Finally, we reorder the product terms based on the group relationship in the grouping tree and BTS relationship. After having the set of reordered product terms, we sequentially map it into an SET array with a minimal width.

IV. Experimental Results

The proposed algorithm was implemented in C language. The experiments were conducted on a set of MCNC [31] and IWLS 2005 [32] benchmarks in a 3.0 GHz Linux platform (CentOS 4.6), as used in [1] and [23]. The format of the benchmarks is *.pla*. For each benchmark, we calculated the number of product terms, hexagons, and BTS, and the width of the mapped SET arrays. We also measured the CPU time for the product term computation (T_{pt}) and the overall algorithm (T_{tal}). Note that the SET array is mapped from each PO of benchmarks. That means the result of each benchmark is the summation of that in each PO.

Table I summarizes the experimental results of the method in [1] and our approach. Columns 1–3 show the benchmark information. To compare with [1], we used the same benchmarks in [1]. Columns 4–9 show the experimental results

TABLE I
COMPARISON OF EXPERIMENTAL RESULTS BETWEEN [1] AND OURS

bench.	$\|PI\|$	$\|PO\|$	[1]						ours						$ratio\ of\ W(\%)$
			$\|PT\|$	$\|HEX\|$	W	$\|BTS\|$	T_{pt}(s)	T_{tal}(s)	$\|PT\|$	$\|HEX\|$	W	$\|BTS\|$	T_{pt}(s)	T_{tal}(s)	
C17	5	2	8	54	31	1	<0.01	0.09	8	50	26	3	<0.01	0.08	83.87
cm138a	6	8	48	460	199	8	<0.01	0.10	48	88	64	40	<0.01	0.07	32.16
x2	10	7	33	397	134	4	<0.01	0.10	33	377	103	13	<0.01	0.11	76.87
cm151a	12	2	25	521	109	0	<0.01	0.09	25	560	108	2	<0.01	0.09	99.08
cm162a	14	5	37	578	156	0	<0.01	0.09	37	668	120	13	<0.01	0.10	76.92
cm163a	16	5	27	391	113	4	<0.01	0.09	27	473	77	10	<0.01	0.09	68.14
cu	14	11	24	415	103	0	<0.01	0.09	22	382	78	4	<0.01	0.12	75.73
cmb	16	4	26	376	122	6	<0.01	0.09	26	88	35	22	<0.01	0.10	28.69
pm1	16	13	41	586	156	14	<0.01	0.10	37	356	81	20	<0.01	0.10	51.92
pcle	19	9	45	751	183	8	<0.01	0.10	45	501	91	28	<0.01	0.10	49.73
cc	21	20	57	1040	191	2	<0.01	0.09	54	873	129	24	<0.01	0.10	67.54
c8	28	18	94	2026	427	6	<0.01	0.11	88	2324	231	38	<0.01	0.24	54.10
count	35	16	184	4590	755	0	<0.01	0.15	184	4080	403	121	<0.01	0.13	53.38
unreg	36	16	64	1515	257	0	<0.01	0.11	49	2879	209	1	<0.01	0.09	81.32
cht	47	36	92	3556	349	7	<0.01	0.13	91	5288	336	17	<0.01	0.14	96.28
cm85a	11	3	49	686	174	0	<0.01	0.09	49	809	168	22	<0.01	0.09	96.55
sct	19	15	142	3168	620	11	<0.01	0.11	134	3591	448	77	<0.01	0.12	72.26
i1	25	16	38	1190	159	9	<0.01	0.09	31	607	76	11	<0.01	0.10	47.80
lal	26	19	160	3312	613	10	<0.01	0.11	160	4183	407	92	<0.01	0.14	66.39
pcler8	27	17	68	1920	315	21	<0.01	0.10	68	2056	208	23	<0.01	0.11	66.03
b9	41	21	352	9112	1520	34	<0.01	0.24	200	10343	649	97	<0.01	0.21	42.70
apex7	**49**	**37**	**1440**	**49004**	**5859**	**31**	**<0.01**	**1.36**	**1440**	**103003**	**4576**	**730**	**<0.01**	**22.40**	**78.10**
example2	85	66	430	14402	1517	93	<0.01	0.50	403	43579	1393	150	<0.01	0.76	91.83
steppermotordrive	29	29	795	22994	3250	20	<0.01	0.35	667	26949	2106	327	<0.01	0.54	64.80
usb_phy	113	116	401	28960	1527	57	<0.01	0.64	389	44930	1161	143	<0.01	0.42	76.03
sasc	133	129	1407	54987	5883	47	<0.01	4.01	794	10706	2284	351	<0.01	0.88	38.82
i2c	**147**	**142**	**3187**	**115944**	**9756**	**220**	**<0.01**	**12.10**	**1858**	**243730**	**4610**	**1145**	**4.35**	**17.90**	**47.25**
simple_spi	**148**	**144**	**3065**	**129039**	**12483**	**263**	**<0.01**	**13.05**	**1959**	**324618**	**5872**	**971**	**<0.01**	**11.77**	**47.04**
total			12339	451974	46961	876	0.79	34.28	8926	838091	26049	4495	5.26	57.1	–
ratio			1	1	1	1	1	1	0.72	1.85	0.55	5.13	6.66	1.67	–
ave			–	–	–	–	–	–	–	–	–	–	–	–	65.40

TABLE II
COMPARISON OF EXPERIMENTAL RESULTS BETWEEN [2] AND OURS

bench.	$\|PI\|$	$\|PO\|$	[2]				ours				$ratio\ of\ W(\%)$
			$\|PT\|$	$\|HEX\|$	W	T_{tal}(s)	$\|PT\|$	$\|HEX\|$	W	T_{tal}(s)	
C17	5	2	8	45	22	<0.01	8	50	26	0.08	118.18
cm138a	6	8	48	332	129	0.01	48	88	64	0.07	49.61
x2	10	7	40	498	122	0.02	33	377	103	0.11	84.43
cm151a	12	2	17	372	75	<0.01	25	560	108	0.09	144.00
cm162a	14	5	41	849	152	0.01	37	668	120	0.10	78.95
cm163a	16	5	31	647	114	0.02	27	473	77	0.09	67.54
cu	14	11	23	285	95	0.01	22	382	78	0.12	82.11
cmb	16	4	26	80	55	<0.01	26	88	35	0.10	63.64
pm1	16	13	49	591	136	0.02	37	356	81	0.10	59.56
pcle	19	9	45	581	116	0.02	45	501	91	0.10	78.45
cc	21	20	53	1180	193	0.01	54	873	129	0.10	66.84
c8	28	18	85	2208	249	0.05	88	2324	231	0.24	92.77
count	35	16	200	3844	444	0.07	184	4080	403	0.13	90.77
unreg	36	16	48	2424	194	0.03	49	2879	209	0.09	107.73
cht	47	36	81	3528	340	0.06	91	5288	336	0.14	98.24
sct	19	15	153	3521	439	0.05	134	3591	448	0.12	102.05
i1	25	16	38	322	97	0.01	31	607	76	0.10	78.35
lal	26	19	171	5455	543	0.06	160	4183	407	0.14	74.95
pcler8	27	17	67	1756	234	0.04	68	2056	208	0.11	88.89
b9	41	21	376	20674	1190	0.26	200	10343	649	0.21	54.54
example2	85	66	447	23964	1144	0.49	403	43579	1393	0.76	121.77
total			2047	73156	6083	1.24	1770	83346	5272	3.1	–
ratio			1	1	1	1	0.86	1.14	0.87	2.5	–
ave			–	–	–	–	–	–	–	–	64.43

reported in [1]. Columns 10–15 show the corresponding results of our approach. Column 16 shows the ratio of reduction on the width of each benchmark. For example, a large benchmark, simple_spi, has 148 PIs and 144 POs. The previous work cost 13.05 s to map 3065 product terms into an SET array with 129 039 hexagons and 12 483 widths while our approach reduced the number of product terms to 1959 and mapped an SET array with 324 618 hexagons and 5872 widths in 11.77 s. The ratio of reduction on the width is 47.04%.

According to Table I, we find that our width minimization approach reduces 28% of product terms and 45% of the width compared with the approach in [1] on average even though the number of hexagons increases. The increase of hexagon count does not always widen the SET arrays and the hexagon count is not our minimization objective. If we consider the ratio of width in each benchmark, the width ratio between our approach and [1] can be down to 28.69%, and the average width ratio is 65.40%. The reasons for the width-saving are that our approach creates fewer product terms, more BTS (more than five times on average), and more flexibility in the SET architecture.

As compared with the results in [1], the CPU time overhead for the benchmark i2c comes from the product term computation. This is because we use two methods to extract product terms and the LTG-based product term extraction causes much time. In addition, the CPU time overhead for the benchmark apex7 comes from the sophisticated mapping algorithm with SET architecture relaxation, while the number of product terms is not reduced. However, we observed that if our approach can obtain much fewer product terms for a benchmark, we could reduce the overall CPU time, e.g., the benchmark simple_spi is the case. This CPU time saving comes from the fact that the number of product terms to be mapped in the mapping procedure is a dominating factor in the overall synthesis process.

We also compare our result with [2], which focused on the width minimization of SET arrays as well. Table II shows the comparison between [2] and our approach.

In Table II, Columns 1–3 show the benchmark information, only the benchmarks reported in [2] are shown here. Columns 4–7 show the experimental results reported in [2]. Columns 8–11 show the corresponding results of our approach. Column 12 shows the ratio of width reduction of each benchmark. According to Table II, our approach reduces 14% of product term number and 13% of width compared with [2], with 1.9 s CPU-time overhead.

In [2], the dynamic product term reordering might obtain better results than ours in some circuits due to large path sharing, which is similar to our BTS method. However, we used both product term minimization and BTS-aware variable reordering to minimize the width. Moreover, we also relax the SET architecture to make the mapping more flexible and create more BTS. As a result, on average, we can reduce 14% of the number of product terms and 13% of width compared with [2].

V. Conclusion

Synthesis algorithms for SET arrays have been rapidly developed in recent years due to a promising achievement in SET technology. In this paper, we propose a mapping approach for reducing the width of SET arrays. Our approach analyzes the major factors that influence the width of SET arrays: 1) the number of product terms; 2) the number of BTS product terms; and 3) the hybrid architecture. The corresponding variable reordering and product term reordering techniques are also proposed. The experimental results show that our approach is effective for minimizing the width of SET arrays.

Our future work is to consider the issue of reconfigurability for defective devices existing in the SET arrays during mapping.

References

[1] C.-E. Chiang *et al.*, “On reconfigurable single-electron transistor arrays synthesis using reordering techniques,” in *Proc. Design, Autom. Test Eur.*, Mar. 2013, pp. 1807–1812.

[2] Y.-H. Chen, J.-Y. Chen, and J.-D. Huang, “Area minimization synthesis for reconfigurable single-electron transistor arrays with fabrication constraints,” in *Proc. Design, Autom. Test Eur.*, Mar. 2014, pp. 1–4.

[3] H. Hasegawa and S. Kasai, “Hexagonal binary decision diagram quantum logic circuits using Schottky in-plane and wrap gate control of GaAs and InGaAs nanowires,” *Phys. E, Low-Dimensional Syst. Nanostruct.*, vol. 11, nos. 2–3, pp. 149–154, Oct. 2001.

[4] S. Kasai and H. Hasegawa, “GaAs and InGaAs single electron hexagonal nanowire circuits based on binary decision diagram logic architecture,” *Phys. E, Low-Dimensional Syst. Nanostruct.*, vol. 13, no. 2–4, pp. 925–929, 2002.

[5] S. Kasai and H. Hasegawa, “A single electron binary-decision-diagram quantum logic circuit based on Schottky wrap gate control of a GaAs nanowire hexagon,” *IEEE Electron Device Lett.*, vol. 23, no. 8, pp. 446–448, Aug. 2002.

[6] T. Nakamura, Y. Abe, S. Kasai, H. Hasegawa, and T. Hashizume, “Properties of a GaAs single electron path switching node device using a single quantum dot for hexagonal BDD quantum circuits,” *J. Phys., Conf. Ser.*, vol. 38, no. 1, pp. 104–107, 2006.

[7] Y.-C. Chen, C.-Y. Wang, and C.-Y. Huang, “Verification of reconfigurable binary decision diagram-based single-electron transistor arrays,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 10, pp. 1473–1483, Oct. 2013.

[8] H. W. C. Postma, T. Teepen, Z. Yao, M. Grifoni, and C. Dekker, “Carbon nanotube single-electron transistors at room temperature,” *Science*, vol. 293, no. 5527, pp. 76–79, Jul. 2001.

[9] I. Tamai, T. Sato, and H. Hasegawa, “Formation of high-density GaAs hexagonal nano-wire networks by selective MBE growth on pre-patterned (001) substrates,” *Phys. E, Low-Dimensional Syst. Nanostruct.*, vol. 21, no. 2, pp. 521–526, 2004.

[10] Y. T. Tan, T. Kamiya, Z. A. K. Durrani, and H. Ahmed, “Room temperature nanocrystalline silicon single-electron transistors,” *J. Appl. Phys.*, vol. 94, no. 1, pp. 633–637, 2003.

[11] L. Zhuang, L. Guo, and S. Y. Chou, “Silicon single-electron quantum-dot transistor switch operating at room temperature,” *Appl. Phys. Lett.*, vol. 72, no. 10, pp. 1205–1207, 1998.

[12] J. Flak and M. Laiho, “Implementation aspects of fault-tolerant logic built with single-electron devices,” in *Proc. NORCHIP*, Trondheim, Norway, Nov. 2009, pp. 1–4.

[13] P. S. K. Karre, P. L. Bergstrom, G. Mallick, and S. P. Karna, “Room temperature operational single electron transistor fabricated by focused ion beam deposition,” *J. Appl. Phys.*, vol. 102, no. 2, pp. 024316-1–024316-4, 2007.

[14] S. Kasai, Y. Amemiya, and H. Hasegawa, “GaAs Schottky wrap-gate binary-decision-diagram devices for realization of novel single electron logic architecture,” in *Int. Electron Devices Meeting (IEDM) Tech. Dig.*, Dec. 2000, pp. 585–588.

[15] L. Liu, V. Saripalli, V. Narayanan, and S. Datta, “Device circuit co-design using classical and non-classical III–V multi-gate quantum-well FETs (MuQFETs),” in *Proc. Int. Electron Devices Meeting*, Dec. 2011, pp. 4.5.1–4.5.4.

[16] F. Nakajima, Y. Miyoshi, J. Motohisa, and T. Fukui, “Single-electron AND/NAND logic circuits based on a self-organized dot network,” *Appl. Phys. Lett.*, vol. 83, no. 13, pp. 2680–2682, 2003.

[17] Y. Shiratori, K. Miura, R. Jia, N.-J. Wu, and S. Kasai, “Compact reconfigurable binary-decision-diagram logic circuit on a GaAs nanowire network,” *Appl. Phys. Exp.*, vol. 3, no. 2, pp. 025002-1–025002-3, 2010.

[18] R. E. Bryant, “Graph-based algorithms for Boolean function manipulation,” *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[19] N. Asahi, M. Akazawa, and Y. Amemiya, “Single-electron logic device based on the binary decision diagram,” *IEEE Trans. Electron Devices*, vol. 44, no. 7, pp. 1109–1116, Jul. 1997.

[20] H. Hasegawa, "III–V nanoelectronics and related surface/interface issues," *Appl. Surf. Sci.*, vols. 212–213, pp. 311–318, May 2003.

[21] S. Kasai, M. Yumoto, and H. Hasegawa, "Fabrication of GaAs-based integrated 2-bit half and full adders by novel hexagonal BDD quantum circuit approach," in *Proc. Int. Semiconductor Device Res. Symp.*, Dec. 2001, pp. 622–625.

[22] S. Eachempati, V. Saripalli, N. Vijaykrishnan, and S. Datta, "Reconfigurable BDD based quantum circuits," in *Proc. Int. Symp. Nanosc. Archit.*, Jun. 2008, pp. 61–67.

[23] Y.-C. Chen, S. Eachempati, C.-Y. Wang, S. Datta, Y. Xie, and V. Narayanan, "Automated mapping for reconfigurable single-electron transistor arrays," in *Proc. Design, Autom. Conf.*, Jun. 2011, pp. 878–883.

[24] Y.-C. Chen, S. Eachempati, C.-Y. Wang, S. Datta, Y. Xie, and V. Narayanan, "A synthesis algorithm for reconfigurable single-electron transistor arrays," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 1, Feb. 2013, Art. ID 5.

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

[26] S. Muroga, *Threshold Logic and Its Applications*. New York, NY, USA: Wiley, 1971.

[27] T. Gowda, S. Vrudhula, and G. Konjevod, "Combinational equivalence checking for threshold logic circuits," in *Proc. Great Lake Symp. VLSI*, 2007, pp. 102–107.

[28] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 5, pp. 665–677, May 2011.

[29] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold logic circuits," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2011, pp. 396–403.

[30] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for threshold logic circuit minimization," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2014, pp. 1–6.

[31] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronics Center of North Carolina, Research Triangle Park, NC, USA, Tech. Rep., 1991, pp. 502–508.

[32] *IWLS 2005 Benchmarks*. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html, accessed May 22, 2014.

Chian-Wei Liu received the B.S. degree from the Department of Computer Science and Information Engineering, Chung-Cheng University, Chiayi, Taiwan, in 2012, and the M.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2014.

Her current research interests include logic synthesis, optimization, and verification for very large-scale integrated, system-on-chip designs, and automation in single-electron transistor.

Chang-En Chiang received the B.S. degree from the Department of Computer Science, National Taipei University of Education, Taipei, Taiwan, in 2010, and the M.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2012.

His current research interests include logic synthesis for emerging technologies and design verification.

Ching-Yi Huang received the B.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2009, where he is currently pursuing the Ph.D. degree with the Department of Computer Science.

His current research interests include logic synthesis, optimization, verification for VLSI designs, and automation for emerging technologies.

Yung-Chih Chen received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2003, 2005, and 2011, respectively.

He was a Visiting Student with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, from 2010 to 2011. He joined the Department of Electronic Engineering, Chung Yuan Christian University, Taoyuan, Taiwan, where he was an Assistant Professor from 2011 to 2012. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Yuan Ze University, Chungli, Taiwan. His current research interests include logic synthesis, design verification, and design automation for emerging technologies.

Chun-Yao Wang (S'00–M'03) received the B.S. degree from the Department of Electronics Engineering, National Taipei University of Technology, Taipei, Taiwan, in 1994, and the Ph.D. degree from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2002.

He has been an Assistant Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, since 2003, where he is currently a Professor. He has authored over 50 technical papers and is an Inventor of 8 patents. His current research interests include logic synthesis, optimization, and verification for VLSI/system-on-chip designs and emerging technologies.

Prof. Wang's two research results were nominated as Best Papers in the 2009 IEEE Asia and South Pacific Design Automation Conference and the 2010 IEEE/Association for Computing Machinery Design Automation Conference, respectively.

Suman Datta (F'13) received the B.S. degree in electrical engineering from IIT Kanpur, Kanpur, India, in 1995, and the Ph.D. degree in electrical and computer engineering from the University of Cincinnati, Cincinnati, OH, USA, in 1999.

He was a member with the Logic Technology Development Group, Intel Corporation, Santa Clara, CA, USA, from 1999 to 2007. He was instrumental in the demonstration of indiumantimonide-based quantum-well transistors operating at room temperature with a record energy delay product, the first experimental demonstration of metal gate plasmon screening and channel strain engineering in high-*k*/metal gate CMOS transistors, and the investigation of the transport properties in nonplanar trigate transistors. He was the Joseph Monkowski Associate Professor of Electrical Engineering with Pennsylvania State University, University Park, PA, USA, in 2007, where he is currently a Professor of Electrical Engineering. He holds over 160 U.S. patents. His group is exploring new materials and novel device architecture for CMOS enhancement and replacement for future energy-efficient computing applications.

Dr. Datta is a Distinguished Lecturer of the IEEE Electron Devices Society.

Vijaykrishnan Narayanan (F'11) received the B.S. degree in computer science and engineering from the University of Madras, Chennai, India, in 1993, and the Ph.D. degree in computer science and engineering from the University of South Florida, Tampa, FL, USA, in 1998.

He is currently a Professor of Computer Science and Engineering and Electrical Engineering with Pennsylvania State University, University Park, PA, USA. His current research interests include power-aware and reliable systems, embedded systems, nanoscale devices and interactions with system architectures, reconfigurable systems, computer architectures, network-on-chips, and domain-specific computing.

Dr. Narayanan has received several awards, including the Penn State Engineering Society Outstanding Research Award in 2006, the IEEE Circuits and Systems Society VLSI Transactions Best Paper Award in 2002, the Penn State CSE Faculty Teaching Award in 2002, the Association for Computing Machinery (ACM) Special Interest Group for Design Automation Outstanding New Faculty Award in 2000, Upsilon Pi Epsilon Award for Academic Excellence in 1997, the IEEE Computer Society Richard E. Merwin Award in 1996, and the first rank in Computer Science and Engineering from the University of Madras, in 1993. He is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He has received several certificates of appreciation for outstanding service from ACM and the IEEE Computer Society.