Area-Aware Decomposition for Single-Electron Transistor Arrays

CHING-HSUAN HO, National Tsing Hua University
YUNG-CHIH CHEN, Yuan Ze University
CHUN-YAO WANG and CHING-YI HUANG, National Tsing Hua University
SUMAN DATTA and VIJAYKRISHNAN NARAYANAN, Pennsylvania State University

Single-electron transistor (SET) at room temperature has been demonstrated as a promising device for extending Moore's law due to its ultra-low power consumption. Existing SET synthesis methods synthesize a Boolean network into a large reconfigurable SET array where the height of SET array equals the number of primary inputs. However, recent experiments on device level have shown that this height is restricted to a small number, say, 10, rather than arbitrary value due to the ultra-low driving strength of SET devices. On the other hand, the width of an SET array is also suggested to be a small value. Consequently, it is necessary to decompose a large SET array into a set of small SET arrays where each of them realizes a sub-function of the original circuit with no more than 10 inputs. Thus, this article presents two techniques for achieving area-efficient SET array decomposition: One is a width minimization algorithm for reducing the area of a single SET array; the other is a depth-bounded mapping algorithm, which decomposes a Boolean network into many sub-functions such that the widths of the corresponding SET arrays are balanced. The width minimization algorithm leads to a 25%–41% improvement compared to the state of the art, and the mapping algorithm achieves a 60% reduction in total area compared to a naïve approach.

CCS Concepts: • **Hardware → Combinational circuits; Emerging tools and methodologies;**

Additional Key Words and Phrases: Circuit synthesis, low-power electronics, minimization methods, single-electron devices, single-electron transistors

ACM Reference Format:
Ching-Hsuan Ho, Yung-Chih Chen, Chun-Yao Wang, Ching-Yi Huang, Suman Datta, and Vijaykrishnan Narayanan. 2016. Area-aware decomposition for single-electron transistor arrays. ACM Trans. Des. Autom. Electron. Syst. 21, 4, Article 70 (September 2016), 20 pages.
DOI: http://dx.doi.org/10.1145/2898998

1. INTRODUCTION

As the integration density of a design increases with the advances of process technologies, high power consumption per chip has become one of the primary bottlenecks to extend Moore's law. To overcome this problem, a variety of ultra-low-power devices have been proposed in recent years. Among these devices, some demonstrations have shown

This work was supported by the Ministry of Science and Technology of Taiwan under Grant No. MOST 103-2221-E-007-125-MY3, Grant No. MOST 103-2221-E-155-069, Grant No. NSC 100-2628-E-007-031-MY3, Grant No. NSC 101-2221-E-155-077, Grant No. NSC 101-2628-E-007-005, Grant No. NSC 102-2221-E-007-140-MY3, and Grant No. NSC 102-2221-E-155-087, and by National Tsing Hua University under Grant No. NTHU 102N2726E1.
Authors' addresses: C.-H. Ho, C.-Y. Wang, and C.-Y. Huang, Dept. of Computer Science, National Tsing Hua University; emails: hchings@gmail.com, wcyao@cs.nthu.edu.tw, s9862516@m98.nthu.edu.tw; Y.-C. Chen, Dept. of Computer Science and Engineering, Yuan Ze University; email: ycchen.cse@saturn.yzu.edu.tw; S. Datta, Dept. of Electrical Engineering, Pennsylvania State University; email: sdatta@engr.psu.edu; V. Narayanan, Dept of Computer Science and Engineering, Pennsylvania State University; email: vijay@cse.psu.edu.

DOI: http://dx.doi.org/10.1145/2898998

that single-electron transistors (SETs) operating at room temperature is a promising candidate to extend Moore's law in the future [Liu et al. 2015; Postma et al. 2001; Tan et al. 2003; Zhuang et al. 1998]. Furthermore, the electrostatic properties of SET through in-depth device simulation were presented in Liu et al. [2011] and Saripalli et al. [2010], which demonstrated an order-of-magnitude reduction in energy-delay product compared to traditional Complementary Metal-Oxide-Semiconductor (CMOS) devices.

Since an SET involves only a few electrons in its switching operation, it suffers from low transconductance and degraded output resistance, making it essential to co-explore a non-CMOS logic architecture. To this end, a binary decision diagram-(BDD) based [Bryant 1986] logic architecture was proposed in Kasai et al. [2001] as a suitable architecture for implementing logic using this ultra-low-power nanodevice. Under this architecture, a Boolean circuit is implemented by mapping its BDD onto an SET array, which is a reconfigurable hexagonal nanowire network controlled by Schottky wrap gates [Eachempati et al. 2008; Hasegawa and Kasai 2001]. The latest simulation [Liu et al. 2015] further showed that the BDD architecture achieved a higher power efficiency than CMOS at the same throughput delay.

The great potential of SET has driven several synthesis research recently. The first automated product-term-based synthesis approach was proposed in Chen et al. [2011, 2013a], followed by a verification method proposed in Chen et al. [2013b]. The authors in Chiang et al. [2013] and Zhao et al. [2014] proposed synthesis methods to reduce the number of hexagons in the SET arrays. Nonetheless, the area of an SET array is the product of its width and bounded height. As a result, Chen et al. [2014, 2015] and Liu et al. [2014, 2015] aimed to reduce the width of the SET array.

However, the width minimization techniques proposed in Chen et al. [2014, 2015] and Liu et al. [2014, 2015] failed to explore all the factors that are in favor of width reduction simultaneously, for example, variable ordering and product term ordering. Furthermore, the existing synthesis methods mentioned above synthesized a Boolean network into an SET array with height equal to the number of primary inputs (PIs). In practice, however, an SET array with a large height may result in incorrect output due to its low transconductance. The size of an SET array is mainly limited by the process of fabricating the quantum dot and the three branches in its structure. Factors such as the lithography technology and process variations including the doping concentration variations also restrict its size [Liu et al. 2013, 2015]. Thus, SET array decomposition is an important issue from the realization viewpoint.

In this article, two algorithms are proposed to deal with the above issue: (1) A width minimization technique: It minimizes the width of an SET array by effectively exploring the combinations of all the favorable factors to width reduction simultaneously. (2) A depth-bounded mapping algorithm: It decomposes a Boolean circuit into an SET network under a height constraint and tries to balance the width of each SET array. Differing from the traditional Field-Programmable Gate Array (FPGA) mapping, each sub-function in the mapped network will be implemented by an SET array using the proposed width minimization synthesis algorithm, rather than a look-up table (LUT).

We conducted the experiments on a set of IWLS 2005 benchmarks [IWLS 2005]. The experimental results show that the proposed width minimization technique improves 29%, 41%, and 25% width compared to Chen et al. [2014], Liu et al. [2015], and Chen et al. [2015], respectively. Also, the SET network produced by the proposed mapping algorithm reduces 60% area compared to a naïve approach.

The main contributions of this work are twofold: (1) This is the first SET array synthesis work considering the height constraint of SET arrays. (2) An area-aware SET array decomposition algorithm composed of a width minimization algorithm and a depth-bounded mapping algorithm is proposed.

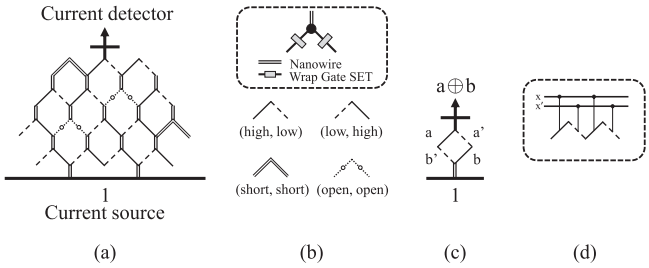

Fig. 1. (a) An SET array. (b) The structure of a node device and its symbols for configuration. (c) An example of $a \oplus b$. (d) An example of symmetric fabric.

The rest of the article is organized as follows. Section 2 describes the background of SET. Section 3 presents the proposed algorithm for SET array width minimization. Section 4 presents the proposed depth-bounded mapping algorithm for the SET network. Section 5 shows the experimental results. Section 6 concludes the article.

2. BACKGROUND

2.1. Reconfigurable SET Array

A reconfigurable SET array can be represented as a hexagonal network shown in Figure 1(a). There is a current source at the bottom and a current detector at the top. Each sloping edge pair in the SET array represents a node device. A node device has a wrap gate SET on the both of its edges as shown in Figure 1(b). The node devices on the same row of the SET array are controlled by the same variable.

A sloping edge in the SET array can be configured as *active high*, *active low*, *short*, or *open*. For an active high edge, it is non-conducted and has no current if its control input is logic 0 and presence of current if the control input is logic 1 and vice versa for active low edge. A short (open) edge is electrical short (open). The functionality of an SET array is determined by path switching where electrons are transported from the current source to the current detector. The output value of an SET array is logic 1 if and only if a current is detected by the current detector.

An example of realizing a function $a \oplus b$ by an SET array is shown in Figure 1(c). The current will be detected at the top only when either $(a = 1, b = 0)$ (left path) or $(a = 0, b = 1)$ (right path). In the rest of this article, vertical edges in the hexagons will be omitted for brevity since they are electrically short.

2.2. Symmetric Fabric Constraint

To decrease the number of input metal wires used for programming an SET array, the symmetric fabric constraint introduced in Eachempati et al. [2008] is imposed to all the edge configurations. The constraint enforces the configuration of an edge pair of a node device to be one of *(high, low)*, *(low, high)*, *(short, short)*, and *(open, open)* as shown in Figure 1(b). Furthermore, the configurations of *(high, low)* and *(low, high)* are not allowed to appear in the same row of an SET array simultaneously. An example of symmetric fabric is shown in Figure 1(d), where both edge pairs are configured as (left, right) = (high, low).

2.3. Product-Term-Based SET Array Synthesis

The product-term-based approach [Chen et al. 2011, 2013a, 2014, 2015; Chiang et al. 2013; Liu et al. 2014, 2015] synthesized an SET array for each primary output (PO) of a

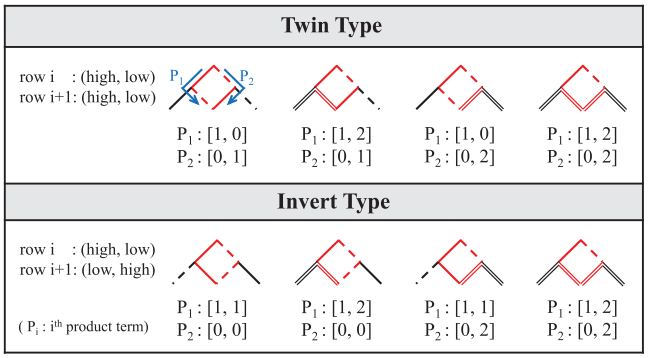

Fig. 2. Types of Branch-then-Share [Liu et al. 2014].

given Boolean network.[1] This approach can be summarized as two steps: first, deriving the *disjoint sum of product* (DSOP) of Boolean circuits and, second, configuring a path in the SET array for each disjoint product term such that no invalid paths are created. Here, DSOP instead of sum of product is used to avoid multiple path conduction simultaneously since an SET has a low driving strength. In this article, each variable in a product term is denoted as 0 (complemented), 1 (uncomplemented), or 2 (don't-care).

2.4. Branch-Then-Share

Branch-then-Share (BTS) [Chiang et al. 2013; Liu et al. 2014] is a relationship between two product terms under the symmetric fabric constraint. It indicates that two product terms branch in one row and merge in the next row and share the path in all the other rows. By exploring this sharing-relationship, the total width of a synthesized SET array can be significantly reduced.

BTS can be classified into two types according to the row configurations involved as shown in Figure 2. The *twin type* (*invert type*) indicates that a BTS occurs at two consecutive rows with the same (opposite) row configurations. Only half of all the BTSs are shown in Figure 2, and the rest of them can be derived by swapping the position of two product terms P_1 and P_2.

2.5. SET Network

In the realization of SET array, each SET array has a restricted height and width as mentioned. To implement a large Boolean function, decomposing a Boolean function into many sub-functions, and synthesizing each sub-function with a small SET array is necessary. These small SET arrays are then connected as an SET network. Note that the current at the current detector of a small SET array will charge the capacitance at the output. This output capacitance behaves as a charge integrator that converts the charging current to a voltage. Thus, the output of a small SET array can be considered as an input to the following SET arrays.

2.6. FPGA Mapping and Cut Enumeration

Conventional LUT-based FPGA mapping flows decompose a Boolean circuit/function through cut enumeration and map the sub-functions into LUTs. In recent years, many state-of-the-art FPGA mapping algorithms have been proposed to map Boolean functions into FPGA effectively [Mishchenko et al. 2007b; Heyse et al. 2012; Du et al. 2013;

[1]The Boolean functions realized by the SET arrays have to be single output since there is only one current detector at the top of an SET array.

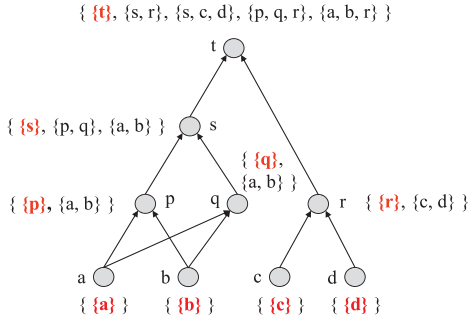

Fig. 3. Illustration of cut computation.

Souza and Silva-Filho 2013; Ghosh et al. 2014]. However, the mapping flow for SET arrays differs from the conventional FPGA mapping flows since each sub-function in the mapped network after decomposition will be implemented by an SET array, rather than an LUT. Thus, in this section, we only introduce the background of cut enumeration used in the proposed algorithm.

A network is *K-bounded* if and only if the number of fanins of any node in the network does not exceed K. Given a value K, the network used for mapping, called the *subject graph*, has to be K bounded. In this article, the subject graph is an And-Inverter Graph (AIG).

A *cut* c of a node n is a set of nodes in its transitive fanins such that any path from a PI to n passes through at least one node in the cut. The size of a cut refers to the number of nodes in that cut. A *trivial cut* of a node is the cut composed of the node itself only. A cut is said to be *K feasible* if the size of the cut is no more than K. A *representative cut* of a node is the best K-feasible cut currently considering the mapping goal. A cut c_1 is said to be *dominated* by another cut c_2 of the same node if and only if c_2 is the subset of c_1.

A *mapping pass* is a procedure of assigning one representative cut to each non-PI node according to a specific mapping goal in the bottom-up topological order. When a mapping pass is finished, each non-PI node will have a representative cut.

Cut enumeration [Cong et al. 1999] is the standard procedure for enumerating all K-feasible cuts of each node in the AIG. For the PI nodes, the K-feasible cut is the trivial cut. For the non-PI nodes, we need to perform the merging operation on the cut sets of its fanin nodes. The merging operation $\triangle$ of cut sets A and B of two nodes with respect to the K-feasible cut is as follows:

$$A \triangle B = \{u \cup v | u \in A, v \in B, |u \cup v| \leq K\}. \tag{1}$$

Let n_1 and n_2 denote the two fanin nodes of a non-PI node n, and $\Phi(n)$ denotes the set of all K-feasible cuts of node n. $\Phi(n)$ can be generated by merging $\Phi(n_1)$ and $\Phi(n_2)$ from Equation (1) and removing the dominated cuts. The trivial cut of node n is also added into $\Phi(n)$. Note that by recursively processing each node n in bottom-up topological order, $\Phi(n_1)$ and $\Phi(n_2)$ will be ready before computing $\Phi(n)$. An example of exhaustive cut enumeration process with $K = 3$ is shown in Figure 3. The cut $\{s\}$ is the trivial cut of node s. The cuts $\{p, a, b\}$ and $\{q, a, b\}$ of node s are removed since they are dominated by the cut $\{a, b\}$. Furthermore, considering the cut set of node t, the cuts $\{p, q, c, d\}$ and $\{a, b, c, d\}$ of node t are excluded as well since they are not 3-feasible cuts.

Since enumerating all the K-feasible cuts is expensive, a priority cut heuristic was proposed in Mishchenko et al. [2007b] that computes only a small number, $L(4 \sim 8)$, of K-feasible cuts at each node. Note that the trivial cut of the current node will also be added into the cut set. Thus, this heuristic confines the cut enumeration process to

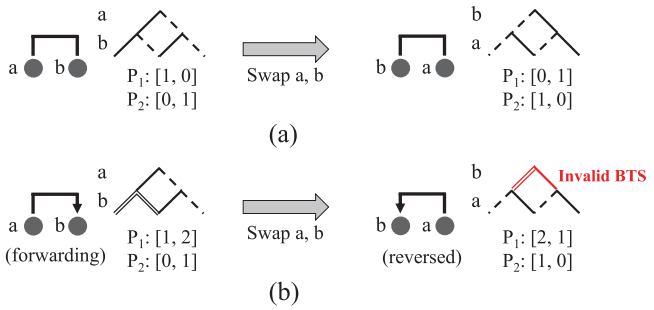

Fig. 4. Example of swapping the location of the connected vertices of (a) an undirected edge (b) a directed edge.

produce $(L+1)^2$ candidate cuts at most at each node and elevates the efficiency. To avoid missing good cuts, the best cut from the previous mapping pass is also added to the $(L+1)^2$ candidate cuts derived from the fanin nodes in the priority cut heuristic.

3. SET ARRAY WIDTH MINIMIZATION

This section introduces the proposed width minimization algorithm, which minimizes the width of SET array by maximally exploring BTS product terms among the disjoint product terms derived from the BDD. In addition to maximizing the number of BTS, minimizing the number of product terms before synthesis can reduce the width of the resultant SET array. This is because each product term of a Boolean function corresponds to a path in an SET array. Since minimizing the number of product terms is not the main objective of this work, the technique we applied to achieve this will be briefly described in the section of experimental results.

3.1. BTS Identification

Since BTS product terms share partial paths in the SET array, maximally identifying BTS is beneficial to width reduction in the synthesis process.

3.1.1. BTS Modeling. Given a DSOP of a Boolean function, the number of BTS in the synthesis process is affected by three factors: (1) *Variable order* in the product terms, (2) *Row configuration* of SET array, and (3) *Product term order* during synthesis. To consider these factors within an SET array simultaneously, we propose to model their relationship as a one-dimensional mixed graph [Weisstein 1999] where the edges are either directed or undirected.

In a one-dimensional mixed graph G, each vertex corresponds to an input variable of the Boolean circuit. The vertex order from left to right of G represents the top-down variable order in the SET array. Consecutive integers starting from 1 in ascending order are assigned to each vertex from left to right as its location. Each edge (a, b) connecting vertices a and b in G represents a possible BTS relationship between two product terms, where a and b correspond to branching and merging variables of the BTS, respectively.

An edge (a, b) is undirected if and only if the relative location of a and b will not affect the existence of the corresponding BTS. For example, in Figure 4(a), an undirected edge (a, b) indicates that the corresponding BTS will still exist if the vertices are swapped. On the contrary, an edge (a, b) is directed if and only if the relative location of a and b will affect the existence of the corresponding BTS. For example, a directed edge (a, b) in Figure 4(b) indicates that the corresponding BTS only exists when the edge is *forwarding* (pointing to the right). If the connected vertices of a directed edge are

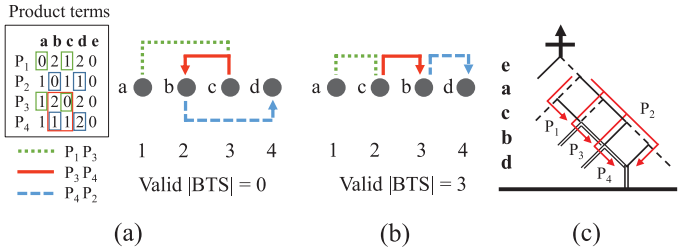

Fig. 5. (a) The initial mixed graph built from the product terms. (b) The mixed graph with an improved vertex order after the BTS exploration algorithm. (c) The final mapping of SET array with consecutive BTSs.

swapped, that is, the edge is *reversed*, then the BTS is invalid due to the symmetric fabric constraint violation.

For a BTS, the branching variable and the merging variable are adjacent, as shown in Figure 2. Thus, edges with length greater than 1 are invalid BTSs. In summary, a BTS represented by an edge e in the mixed graph is *valid* if and only if e satisfies (1) Length(e) = 1 and (2) e is either an *undirected* edge or a *forwarding* edge.

Figure 5(a) presents the mixed graph built from an example with four disjoint product terms. Variable e is omitted since it is not involved in any BTS. Specifically, by iteratively checking the variables within pairs of disjoint product terms, we can find that the disjoint product terms P_1 and P_3 form a BTS with variables a and c according to Figure 2. Besides, we can also identify that the relative locations of a and c do not affect the existence of the corresponding BTS. Thus, we use an undirected edge connecting them in the mixed graph to represent the potential BTS between P_1 and P_3. Similarly, we can also find that P_3 and P_4 form a BTS with the variables b and c. However this BTS can be formed if and only if the location of c is higher than that of b. Thus, we use a (reversed) directed edge connecting them in the mixed graph to indicate the corresponding BTS. There are three BTSs presented by three edges in the mixed graph, but none of them are valid.

The two product terms forming a BTS are labeled on the edge, and its order indicates the path configuration order in the synthesis process. For example, the BTS P_1P_3 in Figure 5(a) represents that P_1 will be configured before P_3. To maximally configure more consecutive BTSs, we prefer to have a chain of BTSs where the right product term of the ith BTS is the same as the left product term of the $(i + 1)$th BTS. Thus, we label a BTS P_4P_2 instead of P_2P_4 in Figure 5(a) since P_4P_2 forms a longer chain of consecutive BTSs, $(P_1P_3 \rightarrow P_3P_4 \rightarrow P_4P_2)$, which is more favorable to width reduction.

3.1.2. Conflicting BTSs Removal. Conflicting BTSs are BTSs that cannot exist simultaneously under any variable order, which need to be removed. There are two kinds of conflicts. The first one is *BTS type conflict*: In any vertex pair, only one type of BTS with more edges will be preserved in the mixed graph since the invert-type BTSs and twin-type BTSs are opposite in the row configuration.

The second kind of conflict is *product term conflict*. Since each product term can have a BTS relationship with at most two other product terms in the SET array, it is necessary to remove the extra ones. In the process of removing conflicting edges, an edge representing a BTS in a longer chain of consecutive BTSs will be kept.

In the example of Figure 5(a), if there is another BTS P_3P_5, it will be removed since the consecutive BTS $(P_1P_3 \rightarrow P_3P_5)$ is shorter than $(P_1P_3 \rightarrow P_3P_4 \rightarrow P_4P_2)$.

3.1.3. Exploring Maximal Number of BTS. We propose an algorithm to adjust the location of each vertex of a given mixed graph iteratively such that the number of valid BTSs

ALGORITHM 1: BTS Exploration Algorithm

Input: A mixed graph.
Output: A mixed graph with maximal valid |BTS|.

```
1: procedure ExploreBTS( )
2:   InitializeVertexOrder( );
3:   while (iterationCnt < iterationNum || valid |BTS| is increasing) do
4:     for (each edge e) do
5:       ComputeCOG(e);
6:       if (e is reversed) then
7:         ComputeShiftedCOG(e)
8:       end if
9:     end for
10:    for (each vertex v) do
11:      ComputeNewLocation(v);
12:    end for
13:    SortVertexLocation( );
14:    AssignIntegerLocation( );
15:    CalculateValidBTS( );
16:  end while
17:  return mixed graph;
18: end procedure
```

is maximized. That is, the connected vertices are placed as close as possible, and the reversed edges tend to be changed as forwarding ones.

The present location of a vertex v is denoted as l_v. The *center of gravity* (COG) of an edge e is computed as:

$$COG(e) = \left(\sum_{v \in e} l_v\right) \Big/ 2, \tag{2}$$

where 2 represents the number of vertices connected to an edge e. The COG acts as an attractive force between two connected vertices due to location average. For example, the COG of the edge $(b, c) = (2 + 3)/2 = 2.5$.

To change a reversed edge into a forwarding one, we also define *ShiftedCOG* for a reversed edge as follows:

$$ShiftedCOG(e) = 2l_{branch} - COG(e), \tag{3}$$

where l_{branch} is the location of the branching node of edge e. This is because for a reversed edge e, we hope the merging node will be placed on the right side of the branching node in order to become a forwarding edge. Equation (3) can achieve this by shifting the COG of e. Also, the longer the distance between the merging and the branching nodes, the stronger force this equation will create. In the example of Figure 5(a), for the reversed edge (b, c), its $ShiftedCOG = 2 \times 3 - 2.5 = 3.5$, which is at the right side of branching node c.

Algorithm 1 outlines the BTS exploration procedure. The initial vertex order can be arbitrary or given. In each iteration, the algorithm traverses all the edges and computes either the COG or the ShiftedCOG of each edge. Next, the algorithm computes the new location of each vertex v based on COG or ShiftedCOG of all its connected edges, E_v. If v is a merging node of a reversed edge e, then we add ShiftedCOG(e) into *ShiftedCOG_Sum*; otherwise, we add the COG of the connected edges of v into *COG_Sum*. The new location of a vertex v, denoted as l'_v, is computed as follows:

$$l'_v = (COG\_Sum + ShiftedCOG\_Sum)/|E_v|, \tag{4}$$

where $|E_v|$ is the degree of vertex v.

ALGORITHM 2: Synthesis for Width Minimization

Input: A single-output Boolean function f.
Output: SET arrays.

1: $dsop$ = CollectDSOP(f);
2: G = ModelBTS($dsop$);
3: ExploreBTS(G);
4: SynthesizeSETArray($dsop$, G);
5: **return** SET_Arrays;

By sorting the new locations of each vertex, we derive a new vertex order and reassign a new consecutive integer index to each vertex starting from 1. Finally, we calculate the number of valid BTS. The iteration continues until the number of valid BTS saturates or a given iteration limit is reached.

Take Figure 5(a) as an example, $l'_a = (2)/1 = 2$, $l'_b = (3.5 + 3)/2 = 3.25$, $l'_c = (2.5 + 2)/2 = 2.25$, and $l'_d = (3)/1 = 3$. With these new locations, the new vertex order of the mixed graph becomes $a \rightarrow c \rightarrow d \rightarrow b$ after the first iteration. Similarly, with one more iteration, the vertex order will be changed as $a \rightarrow c \rightarrow b \rightarrow d$ and the new mixed graph is shown as Figure 5(b). In the next iteration, the number of valid BTS saturates since the vertex order is frozen. Thus, the iteration terminates.

3.2. Synthesis for Width Minimization

Algorithm 2 outlines the SET array synthesis for width minimization, which minimally synthesizes one SET array for the sub-function of each PO[2].

First, the DSOP of a Boolean sub-function is collected. Next, we model all possible BTSs among the product terms into a mixed graph G and apply the proposed BTS exploration algorithm to explore a maximal number of BTSs.

Finally, we determine the variable order, row configurations, and product term order from G before synthesis as follows. First, variables having no BTS are moved to the top of the SET array by the original order, and the vertex order of G indicates the following top-down variable order. Second, the BTS type of the edges between each pair of adjacent vertices determines the row architecture of the SET array in the top-down order. For example, if the current row architecture is $(high, low)$, then the twin- (invert) type BTS indicates that the next row architecture must be $(high, low)$ $((low, high))$. Third, product terms are reordered such that the ones having a BTS relationship are put in consecutive order, and the ones involving no BTS remain in the original order. According to this information, we configure a path for each product term in the SET array. For example, based on the mixed graph in Figure 5(b), the synthesized SET array is shown in Figure 5(c).

4. MAPPING FOR SET NETWORK

In consideration of the height constraint [Liu et al. 2015] of the SET array, decomposing a large SET array into an SET network composed of small SET arrays becomes an important issue. The SET network can be realized by building an FPGA-like SET chip with interconnections fabricated by the CMOS process. In this work, the height of each SET array within the chip is a given constraint, while the width of each SET array is a variable to be determined by the mapping algorithm for total area minimization.

Since the structure of the SET array is based on the product terms of the corresponding Boolean function, it is complicated to decompose a large mapped SET array

[2]The proposed method for decomposing a Boolean function into smaller sub-functions will be introduced in Section 4.

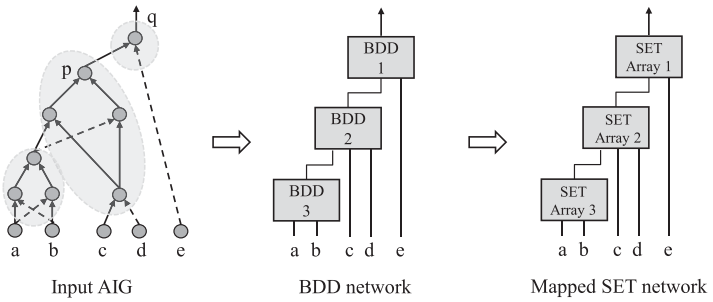

Fig. 6. Example of mapping an SET network.

into many smaller SET arrays directly. Instead, we move the decomposition task to the Boolean circuit level and customize the idea in the FPGA mapping [Mishchenko et al. 2007b] to this problem. An example of mapping an SET network from the Boolean circuit level is shown in Figure 6. After the mapping algorithm, each sub-function in the input network, represented by AND-INV graph (AIG), is transformed into a BDD. Then, for each BDD, we apply the SET array synthesis method proposed in Section 3 to derive the corresponding SET array and construct the final SET network.

In an SET network, the width of each SET array is relevant to the function it realizes under the height constraint K. Since the reserved width of each SET array is identical in an SET chip, an uneven width distribution among all the synthesized small SET arrays will lead to a poor resource utilization. Thus, we target at deriving a mapping that tries to balance the corresponding width of each sub-function. As a result, the objective function to be minimized in our work is as follows:

$$TotalArea = MaxWidth \times |SETArray|, \tag{5}$$

where $MaxWidth$ is the maximal width among all the synthesized SET arrays, and $|SETArray|$ is the number of decomposed SET arrays in the SET network. Note that both $MaxWidth$ and $|SETArray|$ are variables in the objective function and influence to each other, making this optimization problem more challenging. Since the depth in the SET network affects the operation speed and power consumption, we also set the depth as a given constraint, denoted as D_{bound}, to maintain the quality of the depth in this work. Since the structure of SET network is similar to that of FPGA, we take the optimal depth obtained by the depth-oriented FPGA mapping algorithm with the same K as an input parameter D_{bound} in our experiments. In general, a larger K results in a smaller depth, that is, D_{bound}, but results in a larger $MaxWidth$ due to a more complicated sub-function of a cut. The setting of D_{bound} in the experiments will be described in Section 5.2.

4.1. Overview of SET Network Mapping Algorithm

The proposed SET network mapping algorithm is shown in Algorithm 3. It starts by generating a small set of K-feasible cuts at each node using the priority cut heuristic [Mishchenko et al. 2007b] in the topological order. Next, it analyzes and assesses a potential width of the SET network to be mapped. Then, the depth-bounded mapping will be performed to produce a mapping that always satisfies D_{bound}. Next, the area recovery step refines the current mapping for reducing the number of SET arrays if applicable. At the end, the final mapped network is derived. The details of each step mentioned above will be discussed in the following subsections.

ALGORITHM 3: Mapping for SET Network

Input: AIG, K, D_{bound}.
Output: A mapped network.
/* *cut generation* */
1: PriorityCutEnumeration( );
/* *potential width analysis* */
2: pWidthBound = AnalyzepWidthBound( );
/* *depth-bounded mapping* */
3: SETmapDepthOriented( );
4: **if** (exists a PO with depth > D_{bound}) **then**
5: add cuts with pWidth > pWidthBound into CutSet(n) of each node n;
6: SETmapDepthRecovery( );
7: **end if**
/* *area recovery* */
8: SETmapAreaRecovery( );
/* *mapped network derivation* */
9: DeriveFinalMapping( );

4.2. Potential Width Analysis

4.2.1. Potential Width Associated with a Cut. In addition to the traditional costs of cuts such as depth, area flow [Manohararajah et al. 2006] and exact local area [Mishchenko et al. 2007a],[3] we introduce a new cost associated with a cut c, called *potential width*, denoted as *pWidth* in this work. pWidth is a positive value that estimates the potential width of an SET array when adopting cut c. Since our goal is to minimize the maximal width among the SET arrays in the SET network instead of the total width, knowing the relative width of the SET array is sufficient when adopting a cut. Let us denote the local function of a node in terms of variables in its cut c as f_c. For example, the f_c of the node q in Figure 6 is the function of variables p and e in the cut c. pWidth associated with c is estimated as the number of disjoint product terms in f_c. This is because the mapped width of an SET array has a positive correlation with the number of product terms in a function f_c as discussed in Section 3. Specifically, since each disjoint product term is implemented as a configured path in an SET array, the growth of the number of disjoint product terms will increase the SET width in general. Note that from the viewpoint of the SET array, pWidth metric just measures the width growth when you directly configure each conductive path for each disjoint product term in the SET array without considering any path sharing. However, the proposed technique in Section 3 explores the path sharing among all paths, which means that the resultant width will be much reduced. We adopt the pWidth metric as an efficient indicator in candidate cut sorting in the mapping algorithm while the proposed technique in Section 3 is adopted in synthesizing the final SET network.

Note that pWidth of a cut c cannot be recursively derived from that of its fanin cuts. This is because pWidth of a cut c is only relevant to f_c. In practice, we only calculate the pWidth for cuts whose sizes are close to K for elevating algorithm efficiency since the cut associated with the maximal pWidth is usually within these cuts.

4.2.2. The Determination of a Bound of Potential Width. The maximal width among SET arrays in the SET network is unknown before mapping the final network. Thus, the goal of the potential width analysis is to preview the width distribution of the SET network to derive a proper bound of potential width, called *pWidthBound*. With this

[3]The area flow and the exact local area are two heuristics for area recovery. The area flow chooses cuts with a better logic sharing, while the exact local area minimizes the number of sub-functions from a local view.

ALGORITHM 4: pWidth Bound Analysis

Input: subject graph, K.
Output: pWidthBound.

```
1:  procedure AnalyzepWidthBound( )
2:    TradMapDepthOriented( );
3:    Q = POs; A = ∅;
4:    while (Q has non-PI nodes) do
5:      n = PopNode(Q);
6:      mark n as processed;
7:      c = RepresentativeCut(n);
8:      pWidth = ComputepWidth( f_c );
9:      if (pWidth is not in A) then
10:       Push(A, pWidth);
11:     end if
12:     for (each node m ∈ leaves(c)) do
13:       if (m not processed) then
14:         PushNode(Q, m);
15:       end if
16:     end for
17:   end while
18:   Sort(A);
19:   pWidthBound = A[⌈ 0.7 × sizeof(A) ⌉];
20:   ClearCut( );
21:   return pWidthBound;
22: end procedure
```

bound, we can decide whether the pWidth of a cut is reasonable from a global view later.

Algorithm 4 outlines the procedure of determining the pWidthBound. First, we apply a traditional depth oriented mapping pass to obtain the optimum-depth cut for each node. Second, we derive a mapping from the POs to PIs based on the representative cuts of the nodes and record the pWidth of each used cut in an array A without repetition. Then, A is sorted in an increasing order, and the pWidth in the $\lceil 0.7 \times \text{sizeof}(A) \rceil$th entry of sorted A is selected as the pWidthBound heuristically. Using this value is because we observed that cuts with pWidth much larger than the average pWidth account for only a minority.

4.3. Depth-Bounded Mapping

The depth-bounded mapping will derive a mapping that satisfies the given D_{bound} while not incurs large pWidth. Thus, we give a higher priority to cuts whose pWidth do not exceed pWidthBound. However, there may exist some cuts with large pWidth but small depths, which means we may eliminate small depth cuts when pruning cuts with a large pWidth. Thus, an additional depth recovery mapping pass is performed to recover the depth.

In the depth-bounded mapping, at first, a depth oriented mapping pass is performed. In this mapping pass, a cut will be selected as the representative cut when its pWidth is less than pWidthBound, and its depth is the smallest among the candidate cuts with pWidth smaller than pWidthBound. Note that a cut c with pWidth exceeding pWidthBound can still be added into the cut set and be used in the cut merging operator of its predecessor later, the only constraint is that cut c cannot be the representative cut. During this depth oriented mapping pass, if a node in the subject graph whose representative cut has once been pruned, the node is marked as a critical node; otherwise, it is marked as a non-critical node. The critical nodes are the nodes that tend to generate

a representative cut with excessive pWidth in the depth oriented mapping, while the non-critical nodes are the nodes that naturally have representative cuts with pWidth smaller than pWidthBound.

Next, a depth recovery procedure is performed if needed as shown in the lines of 4-7 of Algorithm 3. It starts by checking the depth of each PO. If the depth of one PO is greater than D_{bound}, it implies that the chosen cuts whose pWidth are less than pWidthBound are not satisfactory, and some cuts with pWidth larger than pWidthBound are essential for meeting D_{bound}. Thus, we add those cuts back to the candidate cuts of each node. Next, a depth recovery mapping pass is performed. In the depth recovery mapping pass, we use two different criteria for sorting the candidate cuts at the non-critical and critical nodes in order to meet D_{bound} as well as maintain the quality of maximal width. The criteria adopted in the depth recovery mapping pass are shown as follows:

Node Category	1^{st} Criterion	2^{nd} Criterion	3^{rd} Criterion
non-critical	depth	area flow	cut size
critical	depth	pWidth	cut size

For the non-critical nodes, depth oriented sorting criteria are adopted to improve depth. Note that pWidth is not considered in the cut sorting criteria at the non-critical nodes since these nodes naturally have representative cuts with pWidth not exceeding pWidthBound as mentioned. For the critical nodes, cuts with the same depth are sorted by pWidth instead in an increasing order for reducing the maximal width. The third criterion for both the non-critical and critical nodes chooses cuts of smaller size heuristically. If there is still a tie, then the order of the cuts will be the same as their order in cut generation.

4.4. Area Recovery

With the mapping produced by the depth-bounded mapping where D_{bound} has been satisfied, the area recovery step further reduces area by performing other mapping passes to adjust the order of the candidate cuts in each node based on either the area flow or the exact local area heuristics. To avoid increasing maximal width, cuts with pWidth larger than pWidthBound will be discarded. However, there is an exception: If a cut c of a node n satisfies (1) $pWidth(c) \leq \alpha \times pWidthBound$ and (2) $AreaFlow(c) \leq \beta \times AreaFlow(RepresentativeCut(n))$, then it will be added into the cut set of node n. This criterion slightly loosens the estimated pWidthBound and reduces the area of the mapping result. Based on some preliminary experiments, α and β are set as 1.3 and 0.9, respectively.

4.5. Final Mapping Derivation

Having one representative cut in each node, the mapped SET network is derived as follows: Initially, all of the PO nodes are pushed into a queue Q. Then, a non-PI node n is extracted from Q and is added into the mapping M at each time, and each node belonging to the representative cut of node n is added into Q if it is not in the mapping M yet. This process terminates when Q contains no more non-PI nodes. Next, each node in the mapping M will be implemented as an SET array with the bounded height K using the SET array width minimization algorithm proposed in Section 3.

4.6. Overall Flow

Given an AIG, a height constraint K of the SET array and a user-specified D_{bound}, the flow of the area-aware decomposition for SET arrays is shown in Figure 7. First, a

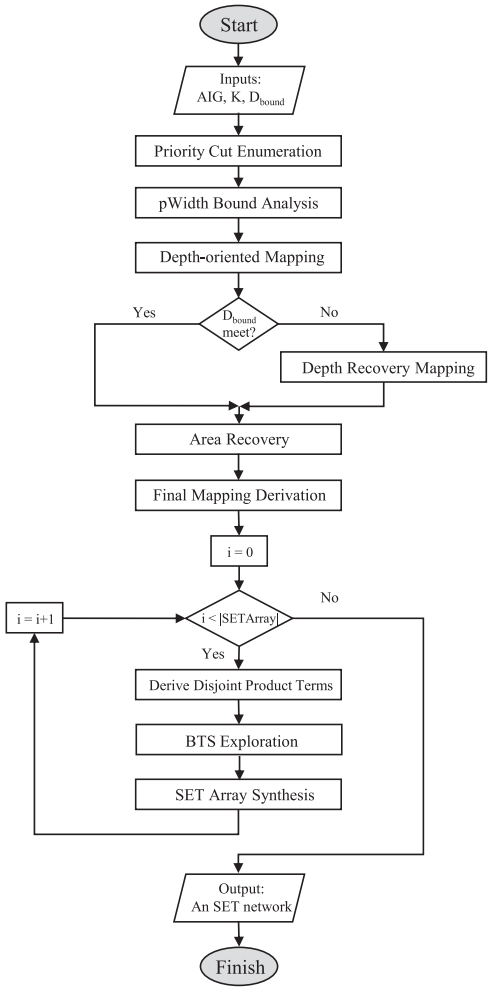

Fig. 7. The overall flow of area-aware decomposition for SET arrays.

set of K-feasible cuts at each node is generated by using the priority cut heuristic. Second, the pWidthBound is derived through the analysis. Then, a depth oriented mapping pass will be performed. If the resultant depth after the mapping is larger than the D_{bound}, then an additional depth recovery mapping pass will be performed to produce a mapping satisfying D_{bound}. Next, the area recovery step further reduces the number of SET arrays and the final mapping is derived. With this final mapping, we obtain the number of SET arrays in the SET network, denoted as $|SETArray|$, which is equal to the number of logic nodes in the final mapping. After the decomposition, we derive the disjoint product terms of the sub-function of each logic node in the final mapping by building the corresponding BDD. Then, the BTS exploration algorithm and synthesis technique proposed in Section 3 are applied to derive the corresponding SET array with bounded height K. The final SET network is derived by connecting these synthesized SET arrays based on the connection in the mapped network.

5. EXPERIMENTAL RESULTS

We implemented the SET array width minimization algorithm and the SET network mapping algorithm in C language and conducted the experiments on a Linux platform with 64GB memory and 2.40GHz CPU. Each benchmark to be mapped was transformed into an AIG by ABC [Group 2007]. We also used CUDD package [Somenzi 2009] to build and manipulate BDDs. The experimental results on the proposed algorithms are shown in Sections 5.1 and 5.2.

5.1. Comparison of Width for SET Arrays without Decomposition

We compared our SET array width minimization algorithm against the state of the art [Chen et al. 2014, 2015; Liu et al. 2015] on a set of IWLS [2005] benchmarks. Since these state-of-the-art algorithms did not consider the height constraint of the SET array, the decomposition technique proposed in Section 4 was not applied in the experiments of this subsection. Instead, we synthesized a large SET array for each PO of the given Boolean circuits as the previous works did for a fair comparison. Thus, the resultant width of a benchmark refers to the summation of widths of SET array from each PO.

5.1.1. Comparison of BTS Exploration Algorithms. The first experiment was designed to show the effectiveness of the BTS exploration algorithm in width reduction. For a fair comparison, we applied the proposed width minimization algorithm and the one in Liu et al. [2015] over the same set of product terms produced by BDD using a BDD variable reordering heuristic CUDD_REORDER_SYMM_SIFT in the CUDD package [Somenzi 2009].

The experimental results are shown in Table I. In Table I, Column 1 lists the name of each benchmark. Column 2 presents the number of PIs and POs of the benchmark. Column 3 shows the number of product terms in the benchmarks using CUDD_REORDER_SYMM_SIFT heuristic. The number of BTSs and the resultant width of a benchmark are shown in Columns 4 and 5, respectively. The last column presents the CPU time of BTS exploration algorithms. The last row shows the ratios of BTS numbers and the resultant width between different approaches.

According to Table I, our algorithm explored 23% more BTSs on average compared to Liu et al. [2015] for the same set of product terms. Our approach also has a 35% reduction in width on average compared to Liu et al. [2015]. This indicates that exploring more BTSs can reduce the width significantly. Also, the required CPU time for both approaches is very close. In summary, our algorithm provides a better combination of variable order, row configuration, and product term order to SET array synthesis compared to Liu et al. [2015].

5.1.2. Comparison of Overall SET Array Synthesis Flows. In addition to the proposed width minimization technique, we also integrated the BDD variable reordering heuristic [Fey and Drechsler 2006] into our synthesis flow to further reduce the resultant width by minimizing the derived number of product terms in the circuits. Thus, we can compare the results of the overall synthesis flows among the state of the art [Chen et al. 2014, 2015; Liu et al. 2015] and ours.

To obtain a smaller number of product terms in the product term derivation stage, Liu et al. [2015] used both the CUDD_REORDER_SYMM_SIFT heuristic [Liu et al. 2015] and a linear threshold logic gate approach, while Chen et al. [2015] used a dynamic shifting based BDD variable ordering algorithm. Chen et al. [2014] derived the product terms from the BDD without mentioning further information.

The experimental results and comparisons are shown in Table II. The meaning of columns and rows are the same as Table I. According to Table II, the proposed synthesis

Table I. Comparison of BTS Exploration Algorithms for the Same Set of Product Terms

	Statistics		|PT|	|BTS|		Width		Time(s)	
Bench.	PI	PO	SymmSift	[Liu et al. 2015]	Ours	[Liu et al. 2015]	Ours	[Liu et al. 2015]	Ours
C17	5	2	8	3	4	23	19	<0.01	<0.01
cm138a	6	8	48	40	40	64	64	<0.01	<0.01
cm151a	12	2	17	0	0	79	68	<0.01	<0.01
i1	25	16	38	13	18	115	79	<0.01	0.01
cm85a	11	3	76	22	38	168	162	0.01	0.01
cm163a	16	5	49	28	32	130	94	<0.01	<0.01
cm162a	14	5	73	36	36	205	197	<0.01	<0.01
cmb	16	4	26	22	22	35	34	<0.01	<0.01
x2	10	7	30	13	13	88	79	<0.01	<0.01
cu	14	11	26	4	4	91	84	<0.01	<0.01
pm1	16	13	54	30	39	137	97	<0.01	<0.01
pcle	19	9	45	28	28	91	92	<0.01	<0.01
cc	21	20	55	25	26	130	124	<0.01	<0.01
pcler8	27	17	152	80	108	461	241	0.01	<0.01
unreg	36	16	64	16	16	211	194	<0.01	<0.01
b9	41	21	262	135	152	808	597	0.01	<0.01
count	35	16	184	121	136	403	322	0.01	0.01
lal	26	19	181	84	114	591	336	<0.01	<0.01
sct	19	15	110	58	67	282	222	0.01	<0.01
stepper.	29	29	698	346	502	2175	1224	0.01	0.01
cht	47	36	90	17	17	335	323	<0.01	<0.01
apex7	49	37	1049	430	589	3620	2173	0.16	0.04
c8	28	18	86	33	40	258	217	<0.01	0.01
example2	85	66	537	230	254	1812	1409	0.05	0.01
usb_phy	113	116	392	148	155	1171	1047	0.02	<0.01
sasc	133	129	1405	627	700	4681	3681	0.04	0.01
simple_spi	148	144	3015	1655	2312	8816	4315	0.71	0.06
i2c	147	142	2942	1969	2165	6891	4489	2.05	0.02
Ratio	**-**	**-**	**-**	**1**	**1.23**	**1**	**0.65**	**-**	**-**

flow can lead to 29%, 41%, and 25% reductions in width on average compared to Chen et al. [2014], Liu et al. [2015], and Chen et al. [2015], respectively. These reductions in width are contributed by exploring more BTSs or collecting fewer number of product terms. The required CPU time in our approach is shown in the last column of Table II.

Note that the comparison with Chen et al. [2015] shows that even if our approach has more product terms in the first place, it can still have 25% reduction in width, which demonstrates that the proposed BTS exploring algorithm is effective.

5.2. Comparison of Total Area in SET Networks

Since this is the first work for SET network mapping, we design a naïve SET network mapping algorithm as the basis for comparison. This naïve mapping algorithm also adopts priority cuts and involves a depth-oriented mapping pass and area recovery step. However, compared with the proposed SET network mapping algorithm, the pWdith bound analysis stage described in Algorithm 4 is totally removed, and the candidate cuts at each node in the subject graph are sorted in a simple way. Specifically, in the depth-bounded (area recovery) mapping pass of the naïve algorithm, the candidate cuts are first sorted by depth (area flow or exact local area) and then by pWidth (the cut with a smaller pWidth will be chosen). Although this naïve scheme can also reduce the maximal width of the resultant SET network, it may sacrifice some good candidate

Table II. Comparison of the Overall SET Array Synthesis Flows

	Statistics		|PT|				Width				Time (s)
Bench.	PI	PO	[Chen'14]	[Liu'15]	[Chen'15]	Ours	[Chen'14]	[Liu'15]	[Chen'15]	Ours	Ours
C17	5	2	8	8	7	7	22	26	25	23	<0.01
cm138a	6	8	48	48	48	48	129	64	134	64	<0.01
cm151a	12	2	17	25	-	17	75	108	-	70	<0.01
i1	25	16	38	31	30	31	97	76	95	79	0.02
cm85a	11	3	-	49	98	48	-	168	239	118	<0.01
cm163a	16	5	31	27	27	27	114	77	100	62	0.01
cm162a	14	5	41	37	37	37	152	120	131	116	0.01
cmb	16	4	26	26	26	26	55	35	51	34	<0.01
x2	10	7	40	33	28	30	122	103	98	89	<0.01
cu	14	11	23	22	22	23	95	78	95	77	0.01
pm1	16	13	49	37	37	40	136	81	102	82	0.01
pcle	19	9	45	45	45	45	116	91	112	92	0.01
cc	21	20	53	54	53	47	193	129	179	131	0.01
pcler8	27	17	67	68	61	61	234	208	198	153	0.03
unreg	36	16	48	49	48	48	194	209	195	193	0.03
b9	41	21	376	200	177	245	1190	649	566	594	0.13
count	35	16	200	184	184	184	444	403	394	322	0.08
lal	26	19	171	160	168	169	543	407	433	328	0.05
sct	19	15	153	134	103	103	439	448	354	270	0.02
stepper.	29	29	-	667	661	669	-	2106	1539	1101	0.25
cht	47	36	81	91	81	81	340	336	344	323	0.09
apex7	49	37	-	1440	497	513	-	4576	1967	1163	0.61
c8	28	18	85	88	79	80	249	231	254	206	0.04
example2	85	66	447	403	367	383	1144	1393	979	1016	1.13
usb_phy	113	116	-	389	335	364	-	1161	1265	968	1.21
sasc	133	129	-	794	798	727	-	2284	2903	1876	2.19
simple_spi	148	144	-	1959	-	1595	-	5872	-	3336	3.35
i2c	147	142	-	1858	964	1060	-	4610	3228	2551	5.44
	-	**-**	**1**	**-**	**-**	**0.84**	**1**	**-**	**-**	**0.71**	**-**
Ratio	**-**	**-**	**-**	**1**	**-**	**0.76**	**-**	**1**	**-**	**0.59**	**-**
	-	**-**	**-**	**-**	**1**	**1.02**	**-**	**-**	**1**	**0.75**	**-**

cuts during the cut sorting without the pWdith bound analysis stage and other schemes proposed in Section 4.

The experiments were conducted on the combinational part of a set of IWLS 2005 benchmarks. Note that we chose the benchmarks with larger node size than the benchmarks used in the experiments of Section 5.1 in order to demonstrate the effectiveness of the proposed mapping algorithm. The parameter K in the mapping was set to the height constraint 10 of the SET array [Liu et al. 2015], and the maximal number of priority cuts stored at each node was set to 8. Furthermore, we set the depth bound, D_{bound}, to the optimal depth of each benchmark produced by the command *If* in the ABC package, which performs optimum-depth FPGA mapping. The area recovery step in the both algorithms involved one pass of area flow oriented mapping and two passes of exact local area oriented mapping.

Table III presents the experimental result of the naïve algorithm and the proposed SET network mapping algorithm. In Table III, Column 3 shows the resultant depth of the mapped SET network, which equals to the D_{bound}, that is, the optimal depth of each benchmark. Column 4 shows the pWidthBound derived from the pWidth bound analysis step. Furthermore, the number of SET arrays in the resultant SET network

Table III. Comparison between the Naïve Mapping Approach and the Proposed Mapping Approach

Bench.	Statistics		Depth	pWidthBound	|SETArray|		MaxWidth		Total Area		Time(s)	
	PI	PO			Naïve	Ours	Naïve	Ours	Naïve	Ours	Naïve	Ours
ss_pcm	106	96	2	5	53	54	9	8	477	432	<0.01	0.01
example2	85	66	2	9	88	88	39	32	3432	2816	0.03	0.09
usb_phy	113	116	2	8	83	82	29	12	2407	984	0.03	0.06
x4	94	71	2	8	118	99	25	24	2950	2376	0.09	0.19
sasc	133	129	2	8	133	137	61	29	8113	3973	0.05	0.11
simple_spi	148	144	2	8	181	178	39	29	7059	5162	0.13	0.32
i2c	147	142	3	10	250	230	57	22	14250	5060	0.10	0.28
systemcdes	322	255	4	18	455	604	147	78	66885	47112	0.44	0.74
spi	276	274	5	28	792	595	226	40	178992	23800	0.39	0.62
wb_dma	217	215	5	13	1013	844	56	27	56728	22788	0.19	0.40
des_area	368	192	4	52	1085	706	224	224	243040	158144	0.72	1.11
systemcaes	260	129	5	36	1820	1370	496	236	902720	323320	1.09	3.88
ac97_ctrl	2283	2247	3	12	2608	2667	94	44	245152	117348	0.75	1.66
mem_ctrl	1198	1235	6	22	3720	3258	415	101	1543800	329058	1.61	2.27
usb_funct	128	121	4	43	2623	2989	682	97	1788886	289933	1.02	1.46
aes_core	259	129	2	86	807	692	1412	345	1139484	238740	1.10	1.86
pci_bridge32	162	207	5	30	4606	4517	513	101	2362878	456217	1.50	2.96
wb_conmax	1130	1416	5	18	8585	8171	58	58	497930	473918	2.79	6.76
ethernet	98	115	6	30	12952	11793	144	65	1865088	766545	2.90	4.58
des_perf	234	64	3	52	4305	7843	776	246	3340680	1929378	5.71	12.2
vga_lcd	89	109	5	19	32360	28984	68	47	2200480	1362248	4.17	6.2
Ratio	**-**	**-**	**-**	**-**	**1**	**0.97**	**1**	**0.33**	**1**	**0.40**	**-**	**-**
Total	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**24.8**	**47.8**

and the maximal width among these SET arrays are shown in Columns 5 and 6, respectively. Column 7 shows the total area in the mapped SET network, which is calculated by the product of the numbers in Columns 5 and 6. The last column shows the CPU time of the mapping algorithms for SET network measured in seconds.

According to Table III, our approach has a 3% reduction in the number of SET arrays and a 67% reduction in the maximal width of the SET network compared to the naïve algorithm on average. As a result, the proposed mapping leads to a 60% reduction in the total area of the SET network on average. The proposed mapping algorithm consumed 23 more seconds for all the benchmarks since more calculations of pWidth are involved than the naïve approach.

6. CONCLUSION

This article presents two algorithms for area-aware decomposition of an SET array to accommodate the device-level height constraint. The first one is a width minimization algorithm that leads to a larger reduction in width compared to the state of the art. The second one is a depth-bounded mapping for an SET network that balances the width of each SET array in the SET network. The combination of these two algorithms effectively reduces the total area in the SET network.

REFERENCES

Randal E. Bryant. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* C-35, 8 (Aug. 1986), 677–691. DOI:http://dx.doi.org/10.1109/TC.1986.1676819

Yung-Chih Chen, Soumya Eachempati, Chun-Yao Wang, Suman Datta, Yuan Xie, and Vijaykrishnan Narayanan. 2011. Automated mapping for reconfigurable single-electron transistor arrays. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. 878–883.

Yung-Chih Chen, Soumya Eachempati, Chun-Yao Wang, Suman Datta, Yuan Xie, and Vijaykrishnan Narayanan. 2013a. A synthesis algorithm for reconfigurable single-electron transistor arrays. *J. Emerg. Technol. Comput. Syst.* 9, 1, Article 5 (Feb. 2013). DOI:http://dx.doi.org/10.1145/2422094.2422099

Yung-Chih Chen, Chun-Yao Wang, and Ching-Yi Huang. 2013b. Verification of reconfigurable binary decision diagram-based single-electron transistor arrays. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 32, 10 (Oct. 2013), 1473–1483. DOI:http://dx.doi.org/10.1109/TCAD.2013.2267453

Yi-Hang Chen, Jian-Yu Chen, and Juinn-Dar Huang. 2014. Area minimization synthesis for reconfigurable single-electron transistor arrays with fabrication constraints. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. 1–4. DOI:http://dx.doi.org/10.7873/DATE.2014.136

Yi-Hang Chen, Yang Chen, and Juinn-Dar Huang. 2015. ROBDD-based area minimization synthesis for reconfigurable single-electron transistor arrays. In *2015 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 1–4. DOI:http://dx.doi.org/10.1109/VLSI-DAT.2015.7114494

Chang-En Chiang, Li-Fu Tang, Chun-Yao Wang, Ching-Yi Huang, Yung-Chih Chen, Suman Datta, and Vijaykrishnan Narayanan. 2013. On reconfigurable single-electron transistor arrays synthesis using reordering techniques. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 1807–1812. DOI:http://dx.doi.org/10.7873/DATE.2013.362

Jason Cong, Chang Wu, and Yuzheng Ding. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays (FPGA'99)*. ACM, New York, NY, 29–35. DOI:http://dx.doi.org/10.1145/296399.296425

Fangqing Du, Colin Yu Lin, Xiuhai Cui, Jiabin Sun, Feng Liu, Fei Liu, and Haigang Yang. 2013. Timing-constrained minimum area/power FPGA memory mapping. In *2013 23rd International Conference on Field Programmable Logic and Applications (FPL)*. 1–4. DOI:http://dx.doi.org/10.1109/FPL.2013.6645565

Soumya Eachempati, Vinay Saripalli, Vijaykrishnan Narayanan, and Suman. Datta. 2008. Reconfigurable BDD based quantum circuits. In *IEEE International Symposium on Nanoscale Architectures, 2008 (NANOARCH 2008)*. 61–67. DOI:http://dx.doi.org/10.1109/NANOARCH.2008.4585793

Görschwin Fey and Rolf Drechsler. 2006. Minimizing the number of paths in BDDs: Theory and algorithm. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 25, 1 (Jan. 2006), 4–11. DOI:http://dx.doi.org/10.1109/TCAD.2005.852662

Anandaroop Ghosh, Somnath Paul, Jongsun Park, and Swarup Bhunia. 2014. Improving energy efficiency in FPGA through judicious mapping of computation to embedded memory blocks. *IEEE Trans. VLSI Syst.* 22, 6 (Jun. 2014), 1314–1327. DOI:http://dx.doi.org/10.1109/TVLSI.2013.2271696

Berkeley LSV Group. 2007. ABC: A System for Sequential Synthesis and Verification. (Sep. 2007). Retrieved May 7, 2015 from http://www.eecs.berkeley.edu/~alanmi/abc/.

Hideki Hasegawa and Seiya Kasai. 2001. Hexagonal binary decision diagram quantum logic circuits using schottky in-plane and wrap-gate control of GaAs and InGaAs nanowires. *Physica Es* 11, 2–3 (2001), 149–154. DOI:http://dx.doi.org/10.1016/S1386-9477(01)00193-X

Karel Heyse, Karel Bruneel, and Dirk Stroobandt. 2012. Mapping logic to reconfigurable FPGA routing. In *2012 22nd International Conference on Field Programmable Logic and Applications (FPL)*. 315–321. DOI:http://dx.doi.org/10.1109/FPL.2012.6339224.

IWLS. 2005. IWLS 2005 Benchmarks. (June 2005). Retrieved March, 2015 from http://iwls.org/iwls2005/benchmarks.html.

Seiya Kasai, M. Yumoto, and Hideki Hasegawa. 2001. Fabrication of GaAs-based integrated 2-bit half and full adders by novel hexagonal BDD quantum circuit approach. In *2001 International Semiconductor Device Research Symposium*. 622–625. DOI:http://dx.doi.org/10.1109/ISDRS.2001.984596

Chian-Wei Liu, Chang-En Chiang, Ching-Yi Huang, Yung-Chih Chen, Chun-Yao Wang, Suman Datta, and Vijaykrishnan Narayanan. 2015. Synthesis for width minimization in the single-electron transistor array. *IEEE Trans. VLSI Syst.* 23, 12 (Dec. 2015), 2862–2875. DOI:http://dx.doi.org/10.1109/TVLSI.2014.2386331

Chian-Wei Liu, Chang-En Chiang, Ching-Yi Huang, Chun-Yao Wang, Yung-Chih Chen, Suman Datta, and Vijaykrishnan Narayanan. 2014. Width minimization in the single-electron transistor array synthesis. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. 1–4. DOI:http://dx.doi.org/10.7873/DATE.2014.135

Lu Liu, Xueqing Li, V. Narayanan, and S. Datta. 2015. A reconfigurable low-power BDD logic architecture using ferroelectric single-electron transistors. *IEEE Trans. Electron. Device.* 62, 3 (Mar. 2015), 1052–1057. DOI:http://dx.doi.org/10.1109/TED.2015.2395252

Lu Liu, Vijay Narayanan, and Suman Datta. 2013. A programmable ferroelectric single electron transistor. *Appl. Phys. Lett.* 102, 053505 (2013), 1–4. DOI:http://dx.doi.org/10.1063/1.4791601

Lu Liu, Vinay Saripalli, Euichul Hwang, Vijaykrishnan Narayanan, and Suman Datta. 2011. Multi-gate modulation doped In0.7Ga0.3 as quantum well FET for ultra low power digital logic. *ECS Trans.* 35, 3 (2011), 311–317. DOI:http://dx.doi.org/10.1149/1.3569923

Valavan Manohararajah, Stephen D. Brown, and Zvonko G. Vranesic. 2006. Heuristics for area minimization in LUT-based FPGA technology mapping. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 25, 11 (Nov. 2006), 2331–2340. DOI:http://dx.doi.org/10.1109/TCAD.2006.882119

Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. 2007a. Improvements to technology mapping for LUT-based FPGAs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 26, 2 (Feb. 2007), 240–253. DOI:http://dx.doi.org/10.1109/TCAD.2006.887925

Alan Mishchenko, Sungmin Cho, Satrajit Chatterjee, and Robert Brayton. 2007b. Combinational and sequential mapping with priority cuts. In *IEEE/ACM International Conference on Computer-Aided Design, 2007 (ICCAD 2007)*. 354–361. DOI:http://dx.doi.org/10.1109/ICCAD.2007.4397290

Henk W. Ch. Postma, Tijs Teepen, Zhen Yao, Milena Grifoni, and Cees Dekker. 2001. Carbon nanotube single-electron transistors at room temperature. *Science* 293, 5527 (2001), 76–79. DOI:http://dx.doi.org/10.1126/science.1061797

Vinay Saripalli, Lu Liu, Suman Datta, and Vijaykrishnan Narayanan. 2010. Energy-delay performance of nanoscale transistors exhibiting single electron behavior and associated logic circuits. *J. Low Power Electron.* 6, 3 (2010), 415–428.

Fabio Somenzi. 2009. CUDD: CU decision diagram package - release 2.4.2. (2009). Retrieved May 25, 2015 from http://vlsi.colorado.edu/~fabio/CUDD/.

V. L. Souza and A. G. Silva-Filho. 2013. MogaMap: An application of multi-objective genetic algorithm for LUT-based FPGA technology mapping. In *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, Washington, DC, 485–488. DOI:http://dx.doi.org/10.1109/ICECS.2013.6815459

Y. T. Tan, T. Kamiya, Z. A. K. Durrani, and H. Ahmed. 2003. Room temperature nanocrystalline silicon single-electron transistors. *J. Appl. Phys.* 94, 1 (2003).

Eric W. Weisstein. 1999. Mixed Graph. (1999). Retrieved June 2, 2015 from http://mathworld.wolfram.com/MixedGraph.html From MathWorld–A Wolfram Web Resource.

Zheng Zhao, Chian-Wei Liu, Chun-Yao Wang, and Weikang Qian. 2014. BDD-based synthesis of reconfigurable single-electron transistor arrays. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, New York, NY, 47–54. DOI:http://dx.doi.org/10.1109/ICCAD.2014.7001328

Lei Zhuang, Lingjie Guo, and Stephen Y. Chou. 1998. Silicon single-electron quantum-dot transistor switch operating at room temperature. *Appl. Phys. Lett.* 72, 10 (1998).

Received September 2015; revised December 2015; accepted February 2016