On Reconfigurable Single-Electron Transistor Arrays Synthesis Using Reordering Techniques

Chang-En Chiang, Li-Fu Tang, Chun-Yao Wang, Ching-Yi Huang,
Yung-Chih Chen[§], Suman Datta[†], Vijaykrishnan Narayanan[‡]
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chung Li, Taiwan, R.O.C.
[†]Department of Electrical Engineering, The Pennsylvania State University, PA, U.S.
[‡]Department of Computer Science and Engineering, The Pennsylvania State University, PA, U.S.

***Abstract*—Power consumption has become one of the primary challenges in meeting Moore's law. Fortunately, 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 during operation. An automated mapping approach for the SET architecture has been proposed recently for facilitating design realization. In this paper, we propose an enhanced approach consisting of variable reordering, product term reordering, and mapping constraint relaxation techniques to minimizing the area of mapped SET arrays. The experimental results show that our enhanced approach, on average, saves 40% in area and 17% in mapping time compared to the state-of-the-art approach for a set of MCNC and IWLS 2005 benchmarks.**

I. INTRODUCTION

As technology scaling enables the packing of billions of transistors into a single chip, reducing power consumption has become one of the primary challenges in chip design. To this end, at the device level, many low power devices have been explored. Among these devices, Single-Electron Transistors (SETs) are particularly attractive when operating at room temperature [11][14][16].

Recent advances have resulted in the fabrication of SETs using conventional process steps akin to that used in multi-gate transistors. The use of wrap-gate structures akin to multi-gate Metal Oxide Semiconductor Field Effect Transistor (MOSFET) structures has been used to demonstrate SETs by two different groups [6][7]. A recent effort in [10] experimentally confirms a reconfigurable single-electron device that can be operated in short, Coulomb blockades and open mode operations. This work also projects that scaling of the device dimension beyond the proof-of-concept experimental device will enable room temperature operations. Such progress in small nano-islands has been demonstrated using focused ion-beam technology resulting in operational room-temperature 8nm nano-islands [6].

Since only a few electrons are involved in the switching process, SETs have poor driving capability and threshold control. Thus, a new binary decision diagram (BDD)-based [2] logic architecture was proposed as a suitable candidate for implementing logic functions using SETs [1]. Using this, a Boolean circuit can be implemented by mapping its BDD onto the BDD-based SET logic architecture, which is presented as a hexagonal nanowire network controlled by Schottky wrap gates [5][8].

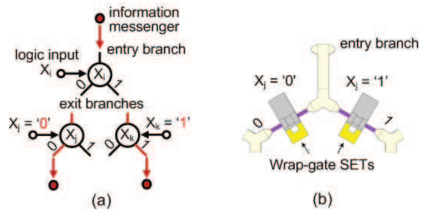


Fig. 1. (a) Node devices. (b) Node devices realized by controlling nanowires with wrap-gate SETs [9].

To implement a BDD, each BDD node corresponds to a node device in the hexagonal fabric. As shown in Fig. 1(a), a node device works like a switch that receives the messenger electrons from a preceding device through the entry branch and sends the electrons to a following device through either the left (0) or the right (1) exit branches according to the control variable. The node device can be realized by controlling nanowires with wrap-gate SETs as shown in Fig. 1(b) [9]. Each exit branch (left or right) corresponds to a nanowire and its conductivity is controlled by a wrap-gate SET that has two operating modes: active high and active low. Furthermore, all the node devices at the same row in the hexagonal fabric are controlled by a single input variable.

However, the realization of the BDD architecture in [8] is fixed and not amendable to functional reconfiguration. To increase the flexibility and reliability of the BDD-based SET array in [1][5][8], a reconfigurable version of SET using wrap gate tunable tunnel barriers was proposed in [4]. The electrostatic properties of this device were also presented in [9][12]. The results showed that this device can potentially provide an energy-delay product that is an order of magnitude smaller than the Complementary MOSFET-based devices [9][12].

However, although the feasibility of the BDD-based SET array had been demonstrated, the mapping process of a Boolean circuit that is represented as a BDD, onto the SET array was manual rather than automatic [4]. To this end, an automatic synthesis method was proposed [3].

This work is supported in part by the National Science Council of Taiwan under Grant NSC 101-2628-E-007-005, NSC 100-2628-E-007-031-MY3, and NSC 101-2221-E-033-073, and by the National Science Foundation of USA under grants 0916887 and 1147388.

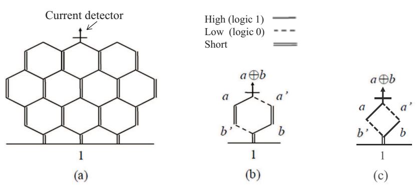


Fig. 2. (a) An SET array fabric. (b) An example of a EXOR b. (c)An abstract diamond fabric of a EXOR b [3].

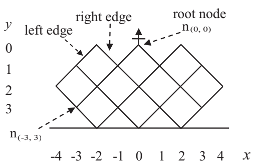


Fig. 3. An abstract diamond fabric [3].

Although [3] can reduce the effort required to map a Boolean function into an SET array, it does not consider the variable ordering of the mapped SET array, which significantly affects the mapped area. Furthermore, the mapping rules it referred to were stricter such that its results were not area-efficient. Thus, in this paper, we propose an enhanced synthesis method for minimizing the mapped area. The proposed approach consists of three techniques: $variable\ reordering$, $product\ term\ reordering$, and $mapping\ constraint\ relaxation$.

We conducted the experiments on a set of MCNC and IWLS 2005 benchmarks [15] [17]. The experimental results show that on average our enhanced approach saves 40% in area and 17% in mapping time compared to [3].

II. Background

An SET array can be presented as a graph composed of hexagons as shown in Fig. 2(a). In Fig. 2(a), there is a current detector at the top of the SET array that measures the current coming from the bottom of the hexagonal fabric. All the sloping edges in the SET array can be configured as $high$, low, $short$, or $open$ with respect to input variables. For example, a hexagonal implementation of an a⊕b is shown in Fig. 2(b), where the current is detected by the current detector when either (a=1, b=0)(left path) or (a=0, b=1)(right path). Since all the vertical edges of the hexagons are electrically short, for ease of discussion, only the sloping edges are preserved in the SET array and the new fabric is referred as the diamond fabric, as shown in Fig. 2(c) and the rest of the paper [3].

A. Notation

Fig. 3 shows the diamond fabric where each node n, i.e., the root of a pair of left and right edges, has a unique location (x, y). Based on the root node located at (0, 0), which is below the current detector, the y value increases from top to bottom, and the x value increases or decreases from center to right or left, respectively. The status of an edge can be $high$, low, $short$, or $open$, which indicates that the edge is configured as high, low, short, or open, respectively. Additionally, let $n_{(x,y)}$ denote the node located at (x, y).

B. Fabric constraint

A configuration circuitry, which involves metal wires, is used to program the SET array. To decrease the number

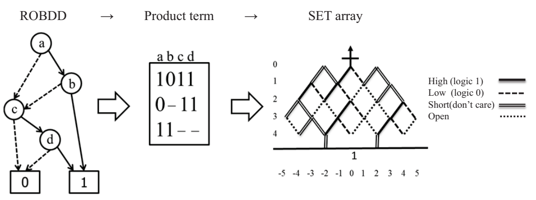


Fig. 4. The overview of the mapping method in the state-of-the-art [3].

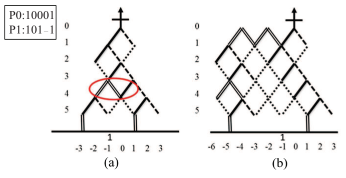


Fig. 5. The prevention of an invalid path by expansion.

of metal wires used for programming the SET array when supplying the input signals, a typical mapping constraint, called the fabric constraint [4][8], is imposed on the mapping process.

The fabric constraint allows the designers to use an input variable x to control all the left edges and x' to control all the right edges or vice versa for all cells in the same row of an SET array [4]. Thus, both $(high, low)$ and $(low, high)$ configurations cannot simultaneously appear in the same row of an SET array.

III. Discussion of the Previous work

[3] proposed an approach synthesizing a logic function by mapping all its product terms into the SET architecture. Its mapping method is illustrated in Fig. 4. First, the approach extracts all the product terms of the function which is represented by a reduced ordered BDD (ROBDD). These product terms are exactly the paths leading to the terminal 1 from the root of the function's ROBDD. Then, it sequentially maps these product terms into the SET array.

[3] attempted to configure a path in an SET array for each product term while avoiding creating invalid paths. An invalid path could be created from the short edges connecting to two conducting edges ($high$, low, or $short$) during the mapping process. For example, assume that Fig. 5(a) is the mapping result of a function containing only two product terms $P0$ and $P1$. However, when applying the input pattern 10011, which is not a product term of this function, the current can be detected at the top of this mapped SET array and causes incorrect behavior. This is because a partial conducting path is created in the fourth row of SET array, as looped. After discovering the phenomenon illustrated in Fig. 5(a), the authors in [3] found that the cause of the invalid path creation is usually that there exist two adjacent conducting edges involving one short edge, and these adjacent edges are from different conducting paths. Thus, their approach strictly prohibited the short edges connecting to any conducting edge from different conducting paths. As a result, in this example, they expanded the SET array at the first row to obtain a correct mapping result as shown in Fig. 5(b).

Although the previous work successfully mapped the correct results, the mapping rules it employed for invalid conducting path prevention were too strict, leading to a large mapped area. Thus, in the proposed approach, we loosen the restriction on the mapping rules while continuing to prevent invalid conducting paths.

In addition to this invalid conducting path issue, we observe that changing the input variable orderings and product term orderings significantly affects the mapping results. Furthermore, we allow the expansion operation for configuring a new path at any level of the array based on the SET's characteristic.

IV. The Proposed Approach

The intention to the techniques of variable reordering and product term reordering is to create more *shared edges*, which are the edges shared among different product terms, in an SET array. This is because, in general, more shared edges result in a smaller SET array area.

For the ease of illustrating these techniques, the product term set derived from the ROBDD can be seen as a matrix as shown in Fig. 4. In this matrix, each column index and row index represents a variable and a product term, respectively. Therefore, the variable ordering and the product term ordering are the corresponding column ordering and row ordering in this matrix. Our objective is to rearrange the matrix such that the mapping results along with this matrix are minimized. In the following discussion, the terms variable reordering and product term reordering are referred as the *column reordering* and *row reordering*.

A. Column reordering

Since each product term corresponds to a path in an SET array, the order of variables in the product terms affects the structure and the size of the resultant array. Therefore, we propose a column reordering technique, which consists of two steps: *front-end column determination*, and *remaining column reordering*, to determine the column ordering for creating more shared edges among all the product terms.

1) Front-end column determination: A column ordering can be seen as a column index string where the leftmost column is the first column to map, and so on. In this step, we move the column having the same bit value B, where $B \in$ (0, 1, –), among all product terms to the front-end or the leftmost of the string. This is because these columns$'$ edges can be completely shared at the upper levels of the SET array. These columns are named $all\text{-}shared\ columns$.

Additionally, since the bit value – will be configured as short in the mapping process, which provides more flexibility to the succeeding mapping, we move the all-shared columns having this bit value, –, to the back-end of the all-shared column substring.

However, it is possible that there exists no all-shared column for a given set of product terms. If so, we will determine the first column of column string based on the method introduced in the next subsection.

2) Remaining column reordering: After determining the front-end columns, we start to reorder the remaining columns. We observed that there are some good share relationships between two product terms. For example, in Fig. 6(a), the product term, 0**01**001, has a good share relationship with another product term, 0**10**001, where only two variables configuring different types of edges in the rows of two and three. Because these two product terms branch in the second row and

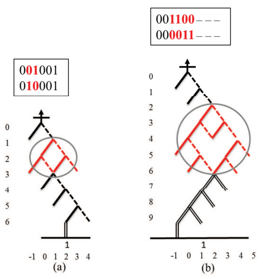
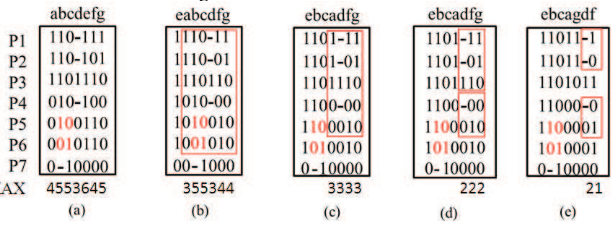


Fig. 6. Branch-then-Share.



	abcdefg	eabcdfg	ebcadfg	ebcadfg	ebcagdf
P1	110-111	1110-11	1101-11	1101-11	11011-1
P2	110-101	1110-01	1101-01	1101-01	11011-0
P3	1101110	1110110	1101110	1101110	1101011
P4	010-100	1010-00	1100-00	1100-00	11000-0
P5	0100110	1010010	1100010	1100010	1100001
P6	0010110	1001010	1010010	1010010	1010001
P7	0-10000	00-1000	0-10000	0-10000	0-10000
MAX	4553645	355344	3333	222	21
	(a)	(b)	(c)	(d)	(e)

Fig. 7. An example for demonstrating the column reordering.

merge in the third row such that the remaining edges are all shared, we name this case $Branch\text{-}then\text{-}Share$. Thus, we do not break down this share relationship during the process of remaining column reordering. Additionally, we only consider $two\text{-}bit\ Branch\text{-}then\text{-}Share$ cases like Fig. 6(a), and $four\text{-}bit\ Branch\text{-}then\text{-}Share$ cases like Fig. 6(b) in this work.

To recognize which pair of product terms is a Branch-then-Share, we simply scan all pairs of product terms. For example, in Fig. 6(a), we find that the first different bit between the two product terms is the second bit. Then, we check whether the combination of pairs of the second and third bits of the two product terms is one of [(1, 0), (0, 1)], [(1, –), (0, 1)], [(1, 0), (0, –)], or [(1, –), (0, –)]. If so, we then check whether the remaining bits of these two product terms, i.e., from the fourth bit to the last bit, are identical or not.

After identifying the columns that involve Branch-then-Share product terms, we start to reorder the remaining columns. We determine the remaining column ordering according to the quantity and the distribution of bit values, 0, 1, and –, in each remaining column among all the product terms.

We directly use Fig. 7 to demonstrate the proposed column ordering. In Fig. 7(a), there are seven product terms (rows) of seven columns with the initial column ordering from a to g. Since the largest Max (#0, #1, #–) among all columns is six, which occurs at the column e, we move this column e to the front-end of the column index string. After this movement as shown in Fig. 7(b), we repeat the ordering process for the remaining columns. However, a major difference is that we only focus on the partial product terms that are within the scope of one edge of the mapped SET, as squared in Fig. 7(b). This is because the column e has been mapped to two edges and each edge has its own scope for the succeeding mapping. Thus, the product terms without column e are divided into two scopes according to the value of the first column e, and the larger scope ($P1 \sim P6$) is first used for determining the order of the remaining columns.[1]

[1]These two scopes need to adopt the same column ordering when mapping. Thus, we first use the larger scope to effectively analyze the results. If the information in the larger scope is not enough to determine the result, we then refer to the other scope.

In the scope under consideration in Fig. 7(b), both columns b and c have the largest Max (#0, #1, #–) value of 5, thus, we move the column b instead of c to the front in accordance with the original column ordering. Furthermore, since the product terms, $P5$ and $P6$, belong to two-bit Branch-then-Share case involving the columns b and c, we also order the column c next to b. The result is shown in Fig. 7(c). Next, we continue calculating the Max (#0, #1, #–) for the remaining columns in the shrunk scope. We find that columns a, f, and g have the largest Max (#0, #1, #–) value, and a is listed in the front of the original order. Therefore, we set a as the fourth column as shown in Fig 7(d). In Fig. 7(d), for the larger scope, although we found that each remaining column has the same Max (#0, #1, #–) value and the same number of bit values, we first consider the columns f and g. This is because column d has a bit value of –, which could cause expansion. In this situation, we need to further consider the other smaller scope, where the column g has the largest Max (#0, #1, #–) value. Hence, we move the column g to the front as the fifth column. For the remaining columns d and f, because the column d has the largest Max (#0, #1, #–) value in the upper scope, we select the column d as the sixth column. The final result of this column ordering is shown in Fig. 7(e).

B. Row reordering

In the row reordering technique, the basic idea is to shape the mapping result like a ladder. This is because a ladder-like shape usually results in a smaller area.

In the following paragraphs, we introduce the proposed row reordering heuristic consisting of three steps for constructing a ladder-like mapping structure.

1) Grouping: Grouping is a step that explores the share relationship among all the rows by building a grouping tree. According to the grouping tree, we can realize which rows have good share relationships with other rows. We use an example whose column ordering has been determined, and hence neglected, to demonstrate the construction of this grouping tree. First, we scan all the rows column-wise from the left to the right until two or three different bit values occur in one column. As seen in Fig. 8, we found that three bit values, 0, 1, and – appear in the third column. Then, the rows with the same bit value are grouped into one group, and we obtain three groups of $P1{\sim}P6$, $P7$, and $P8$ accordingly. Then, we further separate a group having more than two rows until each group has either one or two rows. The group having either one or two rows is named $leaf group$. The separation of one group is based on the last different bit of scanned bit string from the left to the right. For example, using the next distinct bit value after the bit string 1–1, the $P1{\sim}P6$ group is further separated into two groups, $P1{\sim}P5$ and $P6$. The bit string that is shown on a leaf group is named the *label* of the leaf group. In this example, these rows are grouped into six leaf groups that are located at the leaf nodes of the grouping tree. Since two rows belonging to the same leaf group have the most bits in common consecutively, they are treated as having the best share relationship. For the sibling groups, they also have a good share relationship. For example, $P4$ and $P5$ group has the most bits in common consecutively, 1–1100. This

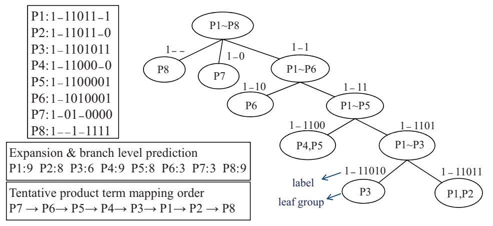


Fig. 8. An example of the grouping tree construction.

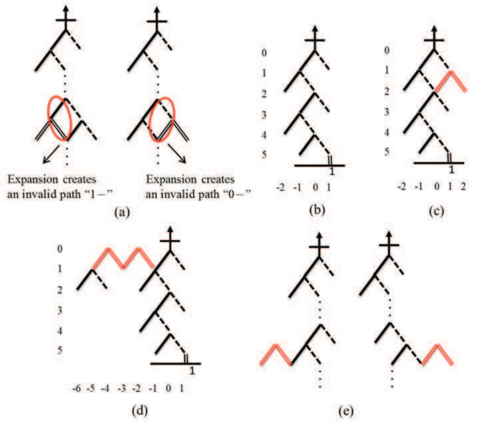

Expansion creates an invalid path "1–"
Expansion creates an invalid path "0–"
(a)
(b)
(c)
(d)
(e)

Fig. 9. (a)∼(c): Incorret expansion examples. (d)∼(e): Correct expansion examples.

group also has a good share relationship with its sibling group $P1{\sim}P3$. Therefore, according to the relationship information found in the grouping tree, a proper mapping order among all the rows can be determined.

2) Expansion and branch level predictions: To obtain a ladder-like mapping result, we need to know the expansion and branch level (EBL) of each product term before mapping. Thus, in this second step we predict the EBL for each product term. We first use an example in Fig. 9 to demonstrate the effect of expansion operation.

Since the fabric constraint has been employed in the mapping process, we can realize that the consecutive mapping of different bits, e.g., 01 (1 after 0) or 10 (0 after 1), creates the pitfall of invalid path creation during expansion. That is, if we expand, using *short* edge in this work, at the second level of the mapping of 01 (1 after 0) or 10 (0 after 1), an invalid path will occur. This situation is highlighted in a general mapping array of Fig. 9(a). For example, Fig. 9(b) is the mapping result of a product term: 10100. If there is another product term whose first two bits are 10 and expand at the second level, an invalid path 0–100 will be created, as shown in Fig. 9(c). Thus, the expansion level for this case should be upward by at least one level, as shown in Fig. 9(d) for successful mapping. On the contrary, if the expansion operation is performed at the end of two consecutive bits of the same value, e.g., 00 or 11, an invalid path will not be introduced, as shown in Fig. 9(e).

Since we have explored the relationships among all the product terms by building the grouping tree, we can figure out a tentative ordering for the product terms to obtain a ladder-like mapping result. We observed that for a leaf group in the grouping tree, if its label has more bits than that of other leaf groups, the product term that belongs to this leaf group would branch at lower levels. Thus, its priority in the mapping order should be lower. To reflect this observation in the mapping order, the EBL of a product term in a leaf group is determined by the number of bits in its label. For example, consider the $P3$ group in Fig. 8. Since the last two bits of its label are different, 10, the EBL is estimated to be 6, which is the number of bits in its label, 7, minus 1 to avoid creating an invalid path such as Fig. 9(a). Similarly, for $P6$ group, the EBL is estimated to be $(4 - 1) = 3$.

We know that an expansion operation at the end of two consecutive bits of the same value is safe. Thus, for a leaf group whose label′s last two bits are the same, an expansion at this location is safe. However, we attempt to search for a safe and even lower location for expansion. For example, consider $P5$ (1–1100001) in a leaf group with the label 1–1100 in Fig. 8. Here, the last two bits of the label are 00. Hence, we further scan the bits after the label until the bit is not 0 or is the last bit. As a result, the last bit we scan in $P5$ is 1. Similarly, if we expand at this level of 1, an invalid path 1–110001– will be created. Hence, the EBL of $P5$ is estimated to be 8, which is the summation of the number of bits in the label and the number of bit 0 after the label (6 + 2).

A short configuration has two edges, left and right, for conducting. However, only one direction will be used in the actual operation. Hence, we have to interpret the bit value – in a product term as either 0 or 1 for EBL prediction. This interpretation determines the direction that a product term might expand. A bit value of – in a product term might occur in the label of its leaf group or not. First, we interpret the bit value of – in the label. We scan the bits in the label from the last bit to the bit which is either 0 or 1, then all the – in the label, either scanned or non-scanned, are interpreted as this bit value. For example, for the leaf group $P8$ in Fig. 8, we scan its label from the last bit to the bit that is either 0 or 1, and we reach the first bit. Hence, the scanned bit value – in the second and third bits are interpreted to be 1. For $P4$, since its last bit in the label is 0, the non-scanned bit value – in the second bit is interpreted to be 0. The labels in the $P7$ group, and the $P1$ and $P2$ group are interpreted to be 100, and 1111011, respectively.

Next, we interpret the other bit value – of a product term that is not in the label. We calculate the number of 0s and the number of 1s in the label, and use the bit value with a larger number to interpret the – not in the label. This is because the bit value with a larger number in the label indicates the direction of this product term most likely expands (1: left, 0: right). If there is a tie, we use the last bit value of the label to interpret the – not in the label. In the last example, the label of $P8$ has three 1s and no 0s after the interpretation of –. Hence, the bit value – of $P8$ in the fifth bit is then interpreted as 1. Since the last two bits of $P8$'s label are now 11, we further scan the succeeding bits until reaching 0 or the last bit. As a result, we reach the last bit. Since the last two bits of $P8$, 11, are the same bit value, the EBL for $P8$ is predicted to be 9, which is the summation of 3 and 6. For $P4$, we interpret the bit value – in the eighth bit as 0, and we also scan the succeeding bits to the last bit. Hence, the EBL for $P4$ is also predicted to be 9 (6 + 3). The EBLs of all the product terms are summarized in Fig. 8.

3) Row order determination: In this subsection, we figure out a tentative row mapping order based on the estimated EBL, Branch-then-Share occurrences, and group relationships in the grouping tree.

First, we choose the row that has the smallest estimated EBL to map due to a ladder-like shape consideration. If there is more than one row having the same smallest estimated EBL, we determine their ordering by their positions in the grouping tree. That is, we first find these rows' nearest common ancestor group. Then, the row in the smallest child group is first selected to map. As shown in Fig. 8, the $P6$ group and $P7$ group have the same EBL. Since the number of rows in the left child group ($P7$) is 1 and is less than that in the right child group ($P1 \sim P6$), 6, we first choose $P7$ to map. This is because $P7$ is less related to ($P1 \sim P6$) and $P7$ needs to expand more if we map ($P1 \sim P6$) first. As a result, we always choose the product term in the smaller child group to map first.

Since Branch-then-Share represents a good share relationship between two rows, we sequentially map them. For the other rows, when we have mapped one row, we look for the product term that is within the same leaf group. If there is no such row, we go back to its parent group and map the row with the smallest EBL.

C. Mapping constraint relaxation

During the mapping process, if we want to keep the mapping results like a ladder, the expansion level or the branch level of current product term cannot be smaller than that of the previous one. Although we can guarantee the correctness of mapping results by prohibiting configuring partial conducting paths, we could not always construct the mapping structure like a ladder using the proposed algorithm. Thus, if there is a product term chosen for mapping whose expansion level is smaller than that of the previous one, we prohibit this product term from connecting to the array from anywhere other than this short edge of the expansion. As a result, we can ensure that no partial conducting path exists.

V. Experimental results

We implemented the proposed algorithm in C language. We set the fabric constraint ($high$, low) for every row of the SET array for simplicity. The experiments were conducted on a 3.0 GHz Linux platform (CentOS 4.6), as was used in [3]. The benchmarks are from the MCNC and IWLS 2005 benchmark suites [15] [17]. For each benchmark, we separately mapped the Boolean function of each primary output (PO) due to the physical constraint in SET arrays, and measured the total number of configured hexagons, the width of the mapping area, and the total CPU time.

Table I summarizes the experimental results of the previous approach [3] and our approach. Column 1 lists the benchmarks. Columns 2 and 3 show the number of primary inputs (PIs) and POs in each benchmark. The next column lists the number of computed product terms (PT) obtained from its ROBDD [13]. Columns 5 to 7 list the number of hexagons, width, and CPU time obtained in the previous approach [3], respectively. Columns 8 to 10 list the corresponding results

TABLE I
THE EXPERIMENTAL RESULTS OF [3], [3] WITH EDL, AND OURS.

benchmark	PI	PO	PT	[3]			[3] with EDL			ours			ratio (%)		
				N_{hex}	W	T(s)	N_{hex}	W	T(s)	N_{hex}	W	T(s)	N_{hex}	W	T
C17	5	2	8	66	31	0.09	68	33	0.10	54	31	0.09	81.81	100.00	100.00
cm138a	6	8	48	438	217	0.09	530	224	0.10	460	199	0.10	105.02	91.70	111.11
x2	10	7	33	790	188	0.10	663	193	0.10	397	134	0.10	59.87	71.27	100.00
cm85a	11	3	49	528	123	0.10	669	171	0.10	686	174	0.09	129.92	141.46	90.00
cm151a	12	2	25	1045	195	0.10	1032	195	0.10	521	109	0.09	50.48	55.90	90.00
cm162a	14	5	37	1163	188	0.09	1098	180	0.10	578	156	0.09	52.64	86.66	100.00
cu	14	11	24	662	116	0.10	659	118	0.10	415	103	0.09	62.97	88.79	90.00
cmb	16	4	26	855	146	0.10	704	144	0.10	376	122	0.09	53.40	84.72	90.00
cm163a	16	5	27	1029	145	0.09	940	140	0.09	391	113	0.09	41.59	80.71	100.00
pm1	16	13	41	1239	180	0.10	1064	176	0.10	586	156	0.10	55.07	88.63	100.00
pcle	19	9	45	1775	212	0.10	1201	212	0.10	751	183	0.10	62.53	86.32	100.00
sct	19	15	142	5186	606	0.11	5228	632	0.10	3168	620	0.11	61.08	102.31	110.00
cc	21	20	57	2306	242	0.10	1823	225	0.10	1040	191	0.09	57.04	84.88	90.00
i1	25	16	38	1920	173	0.10	1757	166	0.10	1190	159	0.09	67.72	95.78	90.00
lal	26	19	160	8684	735	0.15	7438	726	0.33	3312	613	0.11	44.52	84.43	73.33
pcler8	27	17	68	3435	324	0.12	3011	336	0.14	1920	315	0.10	63.76	97.22	83.33
frg1	28	3	399	13731	1027	0.33	21453	1597	0.44	13962	1224	0.22	101.68	119.18	66.66
c8	28	18	94	4869	416	0.12	3623	385	0.11	2026	427	0.11	55.92	110.90	100.00
term1	34	10	1246	80293	4923	7.88	80806	5367	160.70	35975	4622	1.16	44.80	93.88	14.72
count	35	16	184	14678	987	0.29	10088	956	0.44	4590	755	0.15	45.49	78.97	51.72
unreg	36	16	64	4632	292	0.15	3566	281	0.11	1515	257	0.11	42.48	91.45	100.00
b9	41	21	352	22089	1200	2.08	22210	1405	695.97	9112	1520	0.24	41.25	126.66	11.53
cht	47	36	92	7934	394	0.11	7497	396	0.11	3556	349	0.13	47.43	88.57	118.18
apex7	49	37	1440	135543	5798	11.39	–	–	>36000	49004	5859	1.36	36.15	101.05	10.65
example2	85	66	430	50471	1447	0.72	–	–	>36000	14402	1517	0.50	28.53	104.83	69.44
steppermotordrive	29	29	795	–	–	–	38961	3186	2.99	22994	3250	0.35	–	–	–
usb_phy	113	116	401	–	–	–	64067	1618	0.77	28960	1527	0.64	–	–	–
sasc	133	129	1407	–	–	–	–	–	>36000	54987	5883	4.01	–	–	–
i2c	147	142	3187	–	–	–	–	–	>36000	115944	9756	12.10	–	–	–
simple_spi	148	144	3065	–	–	–	–	–	>36000	129039	12483	13.05	–	–	–
i8	133	81	5316	–	–	–	–	–	>36000	111992	14777	11.00	–	–	–
total				365361	20305	26.08	–	–	–	149987	19908	5.51	–	–	–
ratio				1	1	1	–	–	–	0.41	0.98	0.21	–	–	–
average				–	–	–	–	–	–	–	–	–	59.73	94.25	82.42

of the algorithm in [3] with expansion at different levels (EDL). Columns 11 to 13 list the corresponding results of our approach. Columns 14 to 16 list the ratios of results between our approach and the best of [3] as well as [3] with EDL. For example, the lal benchmark has 26 PIs, 19 POs, and 160 product terms. [3] requires 0.15 seconds to map all the product terms into an SET array of 8684 hexagons and 735 units in width. [3] with EDL approach requires 0.33 seconds to map the resultant SET array of 7438 hexagons and 726 units in width, while our approach costs 0.11 seconds to get an SET array with only 3312 hexagons and 613 units in width. Thus, the ratios are 44.52% (3312/7438) for hexagon count, 84.43% (613/726) for width, and 73.33% (0.11/0.15) for CPU time, respectively.

According to the last row of Table I, our approach saves about 40% of the hexagon count, 5% of the width, and 17% of the CPU time on average compared with the best of [3] and [3] with EDL. The CPU time-saving comes from the fact that the number of required configurations is greatly reduced due to the ladder-like shape in the mapping result. Table I also shows some larger benchmarks that cannot be successfully mapped by [3] with EDL. These benchmarks are excluded from the comparison with our approach.

VI. CONCLUSION

Reconfigurable SETs have attracted researchers′ attention due to their ultra low power consumption during operations at room temperature. In this paper, we propose an approach consisting of variable reordering, product term reordering, and mapping constraint relaxation techniques to efficiently mapping reconfigurable SET arrays. The experimental results show that the proposed approach accelerates the mapping process and scales to larger benchmarks while configuring much less hexagons, compared with the prior art.

REFERENCES

[1] N. Asahi, et al., "Single-Electron Logic Device Based on the Binary Dicision Diagram," *IEEE Trans. Electron Devices*, vol. 44, pp. 1109-1116. Jul. 1997.

[2] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. 35, pp. 677-691, Aug. 1986.

[3] Y. C. Chen, et al., "Automated Mapping for Reconfigurable Single-Electron Transistor Arrays," *in Proc. Design Automation Conf.*, 2011, pp. 878-883.

[4] S. Eachempati, et al., "Reconfigurable Bdd-based Quantum Circuits," *in Proc. Int. Symp. on Nanoscale Architectures*, 2008, pp. 61-67.

[5] H. Hasegawa, et al., "Hexagonal Binary Decision Diagram Quantum Logic Circuits Using Schottky In-Plane and Wrap Gate Control of GaAs and InGaAs Nanowires," *Physica E: Low-dimensional Systems and Nanostructures*, vol. 11, pp. 149-154, Oct. 2001.

[6] P. Santosh Kumar Karrea, et al., "Room Temperature Single Electron Transistor Fabricated by Focused Ion Beam Deposition," *Journal of Applied Physics*, vol. 102, pp. 034316-024316-4, 2007.

[7] S. Kasai, et al., "GaAs Schottky Wrap-Gate Binary-Decision-Diagram Devices for Realization of Novel Single Electron Logic Architecture," *in Proc. IEEE International Electron Devices Meeting*, 2000, pp. 585-588.

[8] S. Kasai, et al., "Fabrication of GaAs-based Integrated 2-bit Half and Full Adders by Novel Hexagonal BDD Quantum Circuit Approach," *in Proc. Int. Symp. on Semiconductor Device Research*, 2001, pp. 622-625.

[9] L. Liu, et al., "Multi-Gate Modulation Doped In0.7Ga0.3As Quantum Well FET for Ultra Low Power Digital Logic," *Electro Chemical Society Transactions*, vol. 35, issue 3, pp. 311-317, 2011.

[10] L. Liu, et al., "Device Circuit Co-Design Using Classical and Non-Classical III-V Multi-Gate Quantum-Well FETs (MuQFETs)," *in Proc. IEEE International Electron Devices Meeting*, 2011 pp. 83-86.

[11] H. W. Ch. Postma, et al., "Carbon Nanotube Single-Electron Transistors at Room Temperature," *Science*, vol. 293, pp. 76-79, Jul. 2001.

[12] V. Saripalli, et al., "Energy-Delay Performance of Nanoscale Transistors Exhibiting Single Electron Behavior and Associated Logic Circuits," *Journal of Low Power Electronics*, vol. 6, pp. 415-428, 2010.

[13] F. Somenzi, *CUDD: CU decision diagram package - release 2.4.2*, 2009. http://vlsi.colorado.edu/∼fabio/CUDD/

[14] Y. T. Tan, et al., "Room Temperature Nanocrystalline Silicon Single-Electron Transistors," *Journal of Applied Physics*, vol. 94, pp. 633-637, Jul. 2003.

[15] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report, Microelectronics Center of North Carolina*, 1991.

[16] L. Zhuang, et al., "Silicon Single-Electron Quantum-Dot Transistor Switch Operating at Room Temperature," *Applied Physics Letters*, vol. 72, pp. 1205-1207, Mar. 1998.

[17] http://iwls.org/iwls2005/benchmarks.html