Width Minimization in the Single-Electron Transistor Array Synthesis

Chian-Wei Liu, Chang-En Chiang, Ching-Yi Huang, Chun-Yao Wang,
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 to meet the Moore's law. For reducing power consumption, Single-Electron Transistor (SET) at room temperature has been demonstrated as a promising device for extending Moore's law due to its ultra-low power consumption during operation. Prior work has proposed an automated mapping approach for SET arrays which focuses on minimizing the number of hexagons in an SET array. However, the area of an SET array is more related to the width. Consequently, in this work, we propose an approach for width minimization of the SET arrays. The experimental results show that the proposed approach saves 26% of width compared with the state-of-the-art for a set of MCNC and IWLS 2005 benchmarks while spending similar CPU time.**

I. Introduction

Reducing power consumption has become one of the primary challenges in chip design to meet the Moore's law. To deal with this issue, many ultra-low power devices have been explored. Since the power consumption of Single-Electron Transistors (SETs) [9], which work with only one or few electrons during switching operations, is ultra-low, SETs are considered as a promising candidate that substitutes conventional Complementary Metal-Oxide-Semiconductor (CMOS) devices for future VLSI/SoC designs [5][15][17][20].

Although SETs are promising candidate devices that could substitute CMOSs, they have a poor driving capability and poor threshold control due to only one or few electrons involvement in the switching process. Therefore, the conduction mechanism of the conventional CMOS-based logic is not applicable to SETs. As a result, a binary decision diagram (BDD)-based [2] architecture was proposed as a suitable platform for implementing logic functions using SETs [1]. Therefore, a Boolean circuit can be implemented by mapping its BDD onto a BDD-based SET array which is represented as a hexagonal nanowire network controlled by schottky wrap gates [9][11].

In parallel with the advances of SET array realization, a reconfigurable version of SET that uses wrap gate tunable tunnel barriers was proposed in [8] and the first automatic synthesis method was proposed in [3]. Furthermore, [6][7] proposed mapping approaches to reduce the number of hexagons of the mapped SET arrays by using reordering techniques. Although [6][7] minimized the number of hexagons in SET arrays, the mapped area might not be significantly reduced. That is because the area of an SET array on a chip is the product of its bounded height and width. As a result, the area of an SET array is more related to the width, which is not considered in these works. Thus, in this work, we propose a synthesis algorithm for width minimization, which also reduces the length of routing wires and saves power in SET arrays.

This work is supported in part by the National Science Council of Taiwan under Grant NSC 102-2221-E-007-140-MY3, NSC 102-2221-E-155-087, NSC 101-2221-E-155-077, NSC 101-2628-E-007-005, and NSC 100-2628-E-007-031-MY3, and by the National Tsing Hua Unversity under Grant NTHU 102N2726E1.

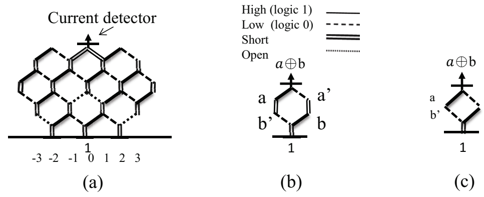


Fig. 1. (a) An SET array fabric. (b) An example of a⊕b. (c) A simplified diamond-shaped network of a⊕b [3].

We conducted the experiments on a set of MCNC [19] and IWLS 2005 [21] benchmarks. The mapped results were verified by an SET verification tool [5]. The experimental results show that our approach saves 26% of the width compared to [7] while spending similar CPU time.

II. Background

A. Reconfigurable BDD-based SET array

A reconfigurable BDD-based SET array can be represented as a hexagonal network as shown in Fig. 1(a). There is a current detector at the top and a current source, represented as 1, at the bottom. When the electrons are transported from the current source to the current detector through a conducting path, which is controlled by the input variables, the current is detected and the output value of the Boolean circuit is 1; otherwise, it is 0. All the sloping edges in the SET array can be configured as *active high*, *active low*, *short*, or *open*. An active high edge indicates that the corresponding node device operates in active mode controlled by a variable x. Conversely, an active low edge is controlled by x'. Furthermore, a short (open) edge is electrical short (open), where the corresponding SET device operates in short (open) mode. For example, Fig. 1(b) shows an implementation of a⊕b. The current detector detects the current and the function will be evaluated as 1 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 abstract graph. Fig. 1(c) is the corresponding diamond-shaped network of the hexagonal network in Fig. 1(b).

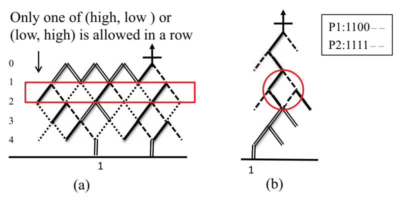


Fig. 2. (a) The fabric constraint. (b) 2-bit Branch-then-Share.

B. Fabric constraint

A typical mapping constraint, called the fabric constraint [8][11], is imposed on the SET arrays [3][6][7]. Let ($high$, low) and (low, $high$) denote the two configurations of edges in a node. According to the fabric constraint, except the short or open edges, both the (high, low) and (low, high) cannot simultaneously appear in the same row. Fig. 2(a) shows an SET array that meets the fabric constraint.

C. Branch-then-Share

Collected product terms are named $Branch\text{-}then\text{-}Share$ $product\ terms$ or $Branch\text{-}then\text{-}Shares$ in short, as they branch in one row and merge in the succeeding rows such that the remaining edges are all shared [7]. For example, in Fig. 2(b), the product terms, 11**00**– and 11**11**–, are Branch-then-Shares, since only two variables configure different types of edges. Therefore, the width of the resultant array is minimized. We only consider $two\text{-}bit\ Branch\text{-}then\text{-}Share$ like Fig. 2(b) in this work for simplicity.

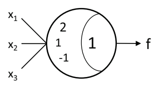
III. The Proposed Approach

A. Product term number minimization

In general, if the number of product terms of a circuit to be mapped is fewer, the mapped area in SET arrays would be smaller. Therefore, in addition to the BDD-based approach used in [3][6][7], we can use other methods to figure out the product terms of a circuit. Note that these product terms obtained have to be disjoint to each other mutually as obtained from its BDD. This is because only one path can be conducted at a time in an SET array from its physical characteristic [8]. As a result, we try to compute fewer disjoint product terms from a circuit's threshold network representation, which can be obtained from [4].

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1\ if\ \sum_{i=1}^{n} x_i w_i \geq T \\ 0\ if\ \sum_{i=1}^{n} x_i w_i < T \end{cases}$$

(a)



(b)

Fig. 3. (a) The definition of an LTG. (b) An LTG implementing the function $f = x_1 + x_2x_3'$.

A $threshold\ network$ is a network composed of $linear$ $threshold\ gates$ (LTG). The parameters of an LTG are weights w_i: $i = 1 \sim n$, which correspond to inputs x_i: i = $1 \sim n$, and a threshold value T. The output f of an LTG is evaluated by the equation in Fig. 3(a). For example in Fig. 3(b), the LTG generates 1 if $2x_1 + x_2 - x_3 \geq 1$, and generates 0 otherwise. Since $\{x_1 = 1\}$ or $\{x_2 = 1, x_3 = 0\}$ can uniquely make the LTG become 1, the Boolean function it represents is $f = x_1 + x_2x_3'$.

In this work, we assume the weights and the threshold value of an LTG are integers, and use a $weight\text{-}threshold$ $vector$ $\langle w_1, w_2, \ldots, w_n; T\rangle$ to represent an LTG. We also transform the negative weights of an LTG into positive ones by applying a Positive-Negative weight transformation procedure [12][14][18] for ease of analysis in this work.

Next, let us discuss how to derive the product terms from a threshold network. Given a single-output threshold network, we compute its disjoint product terms from the primary output (PO) towards the primary inputs (PIs) in a breadth-first search manner. Before this, we first compute the onset and offset of each LTG and disjointly decompose the input space to ensure that the product terms in both onset and offset are disjoint. The product terms (onset) of the whole threshold network then can be derived level by level.

We use an example, as shown in Fig. 4, to demonstrate the product term computation of a threshold network. Fig. 4(a) is the given threshold network, and both the onset and offset of each LTG have been computed. The onset of the LTG f $\langle 2, 1, 1; 2\rangle$ are $(n_1, n_2, n_3) = (1, -, -)$ and (0, 1, 1), where – represents don't-care. For the first onset $(n_1, n_2, n_3) =$ $(1, -, -)$, we assign 1 to n_1. Thus, we use the onset of n_1 to derive the input assignments, (a, b) = (1, 1). Then we also assign – to n_2 and n_3, and the corresponding input assignments are (b, c) = (–, –) and (c, d) = (–, –). Similarly, the other onset $(n_1, n_2, n_3) = (0, 1, 1)$ is also used to compute the product terms of LTGs as shown in the rest rows of Fig. 4(b).

After having the input assignments in the product term table of Fig. 4(b), we need to further refine these input assignments such that no conflict exists for a single input. For example, in the 4th row of Fig. 4(b), the input assignments of (a, $\mathbf{b}$), ($\mathbf{b}, c$) are (1, **1**), (**–** , –), respectively. Because – represents either 0 or 1, we can assign b as 1 without causing any conflicts. Thus, the input assignment of the 4th row can be refined as $(a, b, c, d) = (1, 1, -, -)$ as shown in the 3rd row of Fig. 4(c). For the product term that has conflict values, 0 and 1, in one variable, we discard this product term.

B. Architecture relaxation

The previous work [7] limited all the rows of an SET array to only (high, low) for simplicity. This limitation may lose the opportunities for achieving smaller SET arrays due to inflexibility. Thus, in this work, we adopt the hybrid architecture, i.e., allowing either (high, low) or (low, high) in the configuration of each row of the SET arrays, which is also compatible with the fabric constraint. Using the hybrid architecture, two issues have to be addressed.

1) Branch-then-Share collection: We allow different types of configurations on the rows to create more Branch-then-Shares, which are classified into two types: $twin\ type$ and $invert\ type$ as shown in Fig. 5(a). The twin type (invert type) represents that the Branch-then-Share occurs at two consecutive rows that have the same (opposite) configurations. The four diamonds in upper left of Fig. 5(a) are for (high, low) (high, low) configuration. Their reverses are for (low, high) (low, high) configuration. For simplicity, we only show the fifth diamond that is the reverse of the first one. We use subscripts t (for twin), and i (for invert) to denote the Branch-then-Share types, as shown in Fig. 5(b).

Figs. 5(c)$\sim$5(e) demonstrate the Branch-then-Shares collection. First, we mark Branch-then-Shares with the subscript, t or i as shown in Fig. 5(c). Second, in each product term, consecutive Branch-then-Shares form a $share\ group$, and the initial number of Branch-then-Shares (NBS) is the number of Branch-then-Share in this share group. Furthermore, for

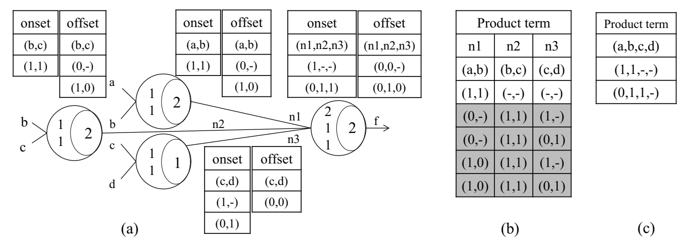


Fig. 4. An example for demonstrating the computation of product terms from a threshold network.

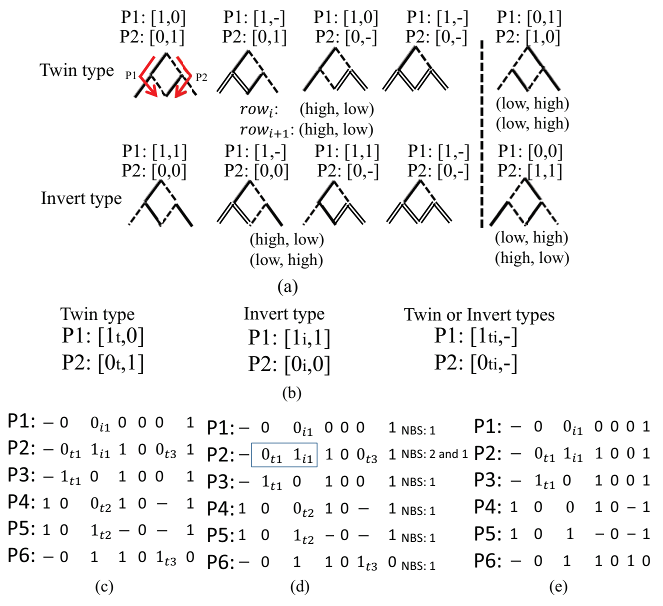


Fig. 5. (a) The types of Branch-then-Shares. (b) Notations of Branch-then-Shares. (c)∼(e) The demonstration of collecting Branch-then-Shares.

each variable within a share group, if there is another same type Branch-then-Share with respect to this variable in other product terms, we increase the NBS of this share group by one. Since the [1,–][0,–] can be considered either twin or invert type, we determine it by the majority of twin type and invert type. Fig. 5(d) shows the NBS for each share group.

Finally, to meet the fabric constraint, we eliminate the Branch-then-Shares that violate the constraint. The constraint limits that only one share group can be kept in a product term, and only one Branch-then-Share type can appear in one column. After calculating the NBSs, we select the share group with the maximal NBS. We then eliminate the violated Branch-then-Shares, and the result is as shown in Fig. 5(e).

After collecting Branch-then-Shares, we reorder the variables for creating more share edges by using the method in [7]. Figs. 6(a) and 6(b) show the results before and after the variable reordering.

2) Hybrid architecture determination: Without loss of generality, the first row is configured as (high, low). Then we examine the other variables sequentially. If there exits a Branch-then-Share, we determine the corresponding configuration according to its type. As shown in Fig. 5(a), the twin type Branch-then-Shares require that the configurations of two rows must be identical, and invert type Branch-then-Shares require that the configuration must be different. For the remaining variables, we set the row's configuration as same as the previous row if half or more than half of the product terms

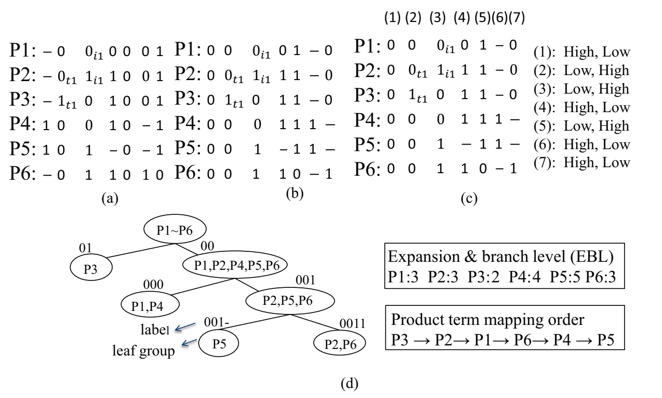


Fig. 6. An example. (a)(b) Variable reordering. (c) Architecture determination. (d) Grouping tree construction and EBL estimation.

change the variable's bit value. Otherwise, we change the row's configuration. Note that the don't-care bit value is ignored during the calculation. That is because the corresponding short edges can go either left or right, and do not influence the width of SET arrays.

For example, in Fig. 6(c), the 1st row's configuration is (high, low) as default. For the 2nd row, since less than half of the product terms change the bit values from the 1st row, we set the 2nd row as (low, high). Then, we determine the 3rd row's configuration as (low, high) since the 2nd row has the twin type Branch-then-Share with the 3rd row, and so on. Fig. 6(c) lists the determined configurations in the hybrid architecture of an SET array.

C. Product term reordering

1) Grouping: By building a grouping tree among all the product terms, we can realize which product terms have good share relationships with other product terms. With this information, a proper mapping order can be determined. First, we scan all the product terms from the first variable to the last one until at least one of the product terms has the different bit values on the variable. Then, the product terms with the same bit value are grouped into one group. We keep separating a group until each group contains only one or two product terms, which is called a *leaf group* and its common bits are called *label* as shown in Fig. 6(d).

2) Expansion and branch level estimation: A ladder-like shape usually results in a smaller width. To obtain a ladder-shape SET array, the expansion and branch level (EBL) information of each product term is needed before mapping. The initial EBL of a product term in a leaf group is the number of bits in its label. However, an expansion is to use two *short* edges to extend the mapping space. An expansion could create a pitfall of invalid path that maps illegal product terms of the circuit [7]. In order to avoid creating an invalid path, we have to decrease the EBL by one according to the bit value and the row's configuration. For example, consider the leaf group that contains $P2$ and $P6$ in Fig. 6(d). Since the last two bits in its label are identical (11) and these two rows' configurations (rows 3 and 4) are different, the EBL is estimated to be 3, which is the initial EBL 4 minus 1. Since the product terms with lower EBL provide more flexibility for expansion, we attempt to search the expanding location as lower as possible. Thus, we keep scanning the bits when the two consecutive bits are identical (different) and their rows' configurations are also identical (different). As a result, the EBL of $P1$, $P3$ and 4 is 3, 2 and 4 respectively.

A don't-care bit, –, which is configured as two short edges in the SET arrays, has two possible edges for conducting. Therefore, we propose a heuristic to predict the don't-care bits for EBL estimation. For example, for $P5$ in Fig. 6(d), the 4^{th} bit is a don't-care bit. Since the 3^{rd} and 4^{th} row are (low, high) and (high, low) configurations, we determine the don't-care bit as 0 which is different from the 3^{rd} bit. This is because we attempt to get the EBL as lower as possible. The EBL of all the product terms are shown in Fig. 6(d).

3) Product term order determination: First, we choose the product term with the smallest EBL to map. If it is a Branch-then-Share with other product terms, we put these product terms next to it. Otherwise, we put the product terms that are within the same leaf group next to it. Furthermore, if there are more than one product term having the same smallest EBL, we determine their ordering by their group levels in the grouping tree. For example, we first choose $P3$ to map, then $P2$ and $P1$ since they are Branch-then-Shares. Next, we map $P6$ followed by $P4$ and $P5$ according to the EBL ordering. The final product term ordering is shown in Fig. 6(d).

IV. Experimental Results

The proposed algorithm was implemented in C language. The experiments were conducted on a set of MCNC [19] and IWLS 2005[21] benchmarks in a 3.0 GHz Linux platform (CentOS 4.6), as was used in [3][7].

Table I summarizes the experimental results of [7] and ours. For example, the large benchmark $simple\_spi$ has 148 PIs and 144 POs. [7] cost 13.05 seconds to map 3065 product terms into an SET array with 129039 hexagons and 12483 widths while our approach reduced the number of product terms to 2000 and mapped an SET array with 85948 hexagons and 7379 widths in 6.66 seconds where 0.42 seconds is for the product term computation.

According to Table I, we find that our approach can reduce 21% of the hexagons and 26% of the width compared with [7] while spending similar CPU time for all the benchmarks. The reasons for the width-saving are that our approach created fewer product terms, more Branch-then-Shares, and more flexibility in the SET architecture. The CPU time overhead for some circuit, e.g., $i2c$, comes from the product term computation, don't-care prediction for EBL, and mapping.

V. Conclusion

In this paper, we propose an approach for reducing the width of an SET array. We first minimize the number of product terms. Then, we relax SET array architecture to allow more Branch-then-Share product terms. The experimental results show that our approach reduced 26% of the width of SET arrays for all the benchmarks compared with a previous work while spending similar CPU time.

TABLE I. The experimental results of [7] and ours.

bench.	PI	PO	[7]				ours				
			PT	N_{hex}	W	T_{tal}(s)	PT	N_{hex}	W	T_{pt}(s)	T_{tal}(s)
C17	5	2	8	54	31	0.09	8	57	28	<0.01	0.02
cm138a	6	8	48	460	199	0.10	48	244	152	<0.01	0.02
x2	10	7	33	397	134	0.10	30	330	106	<0.01	0.03
cm162a	14	5	37	578	156	0.09	84	1287	312	<0.01	0.03
cm163a	16	5	27	391	113	0.09	53	678	184	<0.01	0.02
cu	14	11	24	415	103	0.09	23	415	82	<0.01	0.06
cmb	16	4	26	376	122	0.09	26	334	93	<0.01	0.03
pm1	16	13	41	586	156	0.10	37	574	122	<0.01	0.02
pcle	19	9	45	751	183	0.10	46	675	179	<0.01	0.02
cc	21	20	57	1040	191	0.09	53	1006	181	<0.01	0.03
c8	28	18	94	2026	427	0.11	82	1968	369	<0.01	0.17
count	35	16	184	4590	755	0.15	184	4557	661	0.01	0.07
unreg	36	16	64	1515	257	0.11	49	1282	164	<0.01	0.03
cht	47	36	92	3556	349	0.13	91	3511	332	<0.01	0.05
cm85a	11	3	49	686	174	0.09	49	690	155	0.01	0.02
sct	19	15	142	3168	620	0.11	107	1880	395	0.01	0.03
i1	25	16	38	1190	159	0.09	31	934	99	<0.01	0.04
lal	26	19	160	3312	613	0.11	163	3159	577	<0.01	0.05
pcler8	27	17	68	1920	315	0.10	146	3235	584	0.01	0.04
b9	41	21	352	9112	1520	0.24	211	4764	808	0.01	0.09
apex7	49	37	1440	49004	5859	1.36	699	25256	2698	0.25	0.80
example2	85	66	430	14402	1517	0.50	469	14789	1514	0.02	0.81
stepper.	29	29	795	22994	3250	0.35	783	21532	2887	0.01	0.53
usb_phy	113	116	401	28960	1527	0.64	379	28536	1314	0.02	0.56
sasc	133	129	1407	54987	5883	4.01	1030	46131	3993	0.04	2.43
i2c	147	142	3187	115944	9756	12.10	2601	103706	9117	4.38	16.64
simple_spi	**148**	**144**	**3065**	**129039**	**12483**	**13.05**	**2000**	**85948**	**7379**	**0.42**	**6.66**
total			12314	454453	46852	34.19	9482	357478	34485	5.19	29.3
ratio			–	1	1	–	–	0.79	0.74	–	–

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 TC*, vol. 35, pp. 677-691, Aug. 1986.

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

[4] R. Zhang et al., "Synthesis and Optimization of Threshold Logic Networks with Application to Nanotechnologies," *in Proc. DATE*, pp. 904-909, 2004.

[5] Y. C. Chen et al., "Verification of Reconfigurable Binary Decision Diagram-based Single-Electron Transistor Arrays," *IEEE TCAD*, pp. 1473-1483, 2013.

[6] Y. C. Chen et al., "A Synthesis Algorithm for Reconfigurable Single-Electron Transistor Arrays," *ACM JETC*, Vol. 9, No. 1, Article 5, Feb. 2013.

[7] C. E. Chiang et al., "On Reconfigurable Single-Electron Transistor Arrays Synthesis Using Reordering Techniques," *in Proc. DATE*, pp. 1807-1812, 2013.

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

[9] 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, 2001.

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

[11] 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*, pp. 622-625, 2001.

[12] P. Y. Kuo et al., "On Rewiring and Simplification for Canonicity in Threshold Logic Circuits," *in Proc. ICCAD*, pp. 396-403, 2011.

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

[14] S. Muroga, *"Threshold Logic and its Applications."* John Wiley, 1971.

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

[16] 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.

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

[18] C. K. Tsai et al., "Sensitization Criterion for Threshold Logic Circuits and its Application," *in Proc. ICCAD*, pp. 226-233, Nov. 2013.

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

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

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