Majority Logic Circuits Optimisation by Node Merging

Chun-Che Chung, Yung-Chih Chen[§], Chun-Yao Wang, Chia-Cheng Wu
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chungli, Taiwan, R.O.C.

***Abstract*—Quantum-dot Cellular Automata (QCA) has emerged as a new design paradigm for nanotechnologies. Since the operational logic in QCA is the majority logic, much research about the synthesis and optimisation of majority logic has been proposed recently. In this paper, we propose an optimisation method by merging nodes in the Majority-Inverter-Graph, which is the representation of majority logic circuits. Instead of using satisfiability solvers, our approach can identify the node mergers by using logic implications for circuit size reduction. The experimental results show that for a set of EPFL benchmarks, our approach can minimise the node count by 21% when integrated with the state-of-the-art on average.**

I. Introduction

Power consumption and thermal dissipation have been considered as the challenges that would limit the advances of designs with Complementary Metal-Oxide Semiconductor (CMOS) transistors. As a result, many emerging low-power nanotechnologies have been proposed[7][17][21]. Quantum-dot Cellular Automata (QCA) is one of such nanotechnologies that attracts much attention for its high device density and good switching speed [12][13][17][18].

In QCA, a quantum cell consists of four quantum dots and two free electrons. The electrons occupy diagonally opposite sites with Coulombic repulsion. These two electron configurations are denoted as cell polarizations, P = +1 and P = -1. Binary information is encoded by using the cell polarization: P = +1 represents logic "1" and P = -1 represents logic "0". Fig. 1 shows the two electron configurations of QCA cells. QCA also provides a new method of computation and information transfer that is achieved by Coulombic interactions among QCA cells. With this characteristic, no interconnections are needed between two cells, and no current flows are required in the QCA circuits such that its power dissipation is extremely lower than that of CMOS circuits [16].

Fig. 2 shows a QCA wire that is a chain of QCA cells. The binary information is propagated from the input to the output by Coulombic repulsive force between cells. In the QCA, the underlying logic operations are inverters and majority functions[17]. An inverter represented by QCA is shown in Fig. 3(a). The signal "1" comes from A, splits into two wires, and reaches at D with the value of "0". A majority gate is a three-input majority function as shown in Fig. 3(b). In Fig. 3(b), the device cell will obey the majority charge configuration of its input cells A, B, and C, and propagate the information to the output cell D. In the implementation of QCA circuit, to reduce the implementation cost, it is important to minimise the corresponding majority logic circuits with inverters and majority gates.

Recently, much research about the majority logic has been proposed [1][2][3][6][11][15][19][20]. The first Satisfiability (SAT) solver for majority logic was proposed in [6]. In [20], thirteen standard functions were proposed to represent all three-variable Boolean functions, and majority expressions corresponding to these standard functions were presented. However, the work only dealt with three-input Boolean functions. Then, the works [11][19] proposed methods focusing on synthesizing multi-input Boolean functions by decomposing a function into three-input functions. In [19], a Karnaugh map-based majority logic synthesis method was used to derive a majority logic circuit. In [11], the authors employed four decomposition methods to decompose a function into several three-input subfunctions and select the one having the minimum number of subfunctions as the best result. In [1], a two-level majority logic expression was presented as an alternative to the traditional Sum-Of-Product (SOP) or Product-Of-Sum (POS) forms for expressing Boolean functions. In[2], the Majority-Inverter-Graph (MIG), a new logic representation, and an axiomatic system for majority logic were presented to optimise the depth and the power consumption of logic circuits. Then, the work[3] proposed an MIG-based Boolean optimisation method and combined this method with the state-of-the-art MIG algebra techniques for further simplifying the logic circuits. Recently, another work[15] proposed an optimisation method on MIGs based on functional hashing. Although many optimisation methods on MIGs have been proposed, only few of them target on the area minimisation. Thus, in this paper, we propose another optimisation technique — node merging, to minimise the area of MIGs, such that the corresponding QCA circuits are minimised.

Node merging is a logic optimisation technique. It has been used in the logic circuits represented in And-Inverter-Graphs (AIGs)[8] for logic optimisation[4][5][14]. Given a target node n_t in an AIG, the node merging technique identifies substitute nodes n_s that can replace the target node. If a substitute node is found, the target node and the single-fanout nodes at the target node's transitive fanin cone can be removed after the

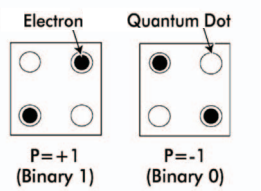

Fig. 1. Binary information of a QCA cell.

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3, NSC 102-2221-E-007-140-MY3, MOST 105-2221-E-155-068

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

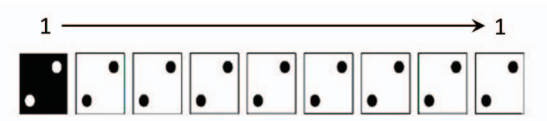

Fig. 2. QCA wire.

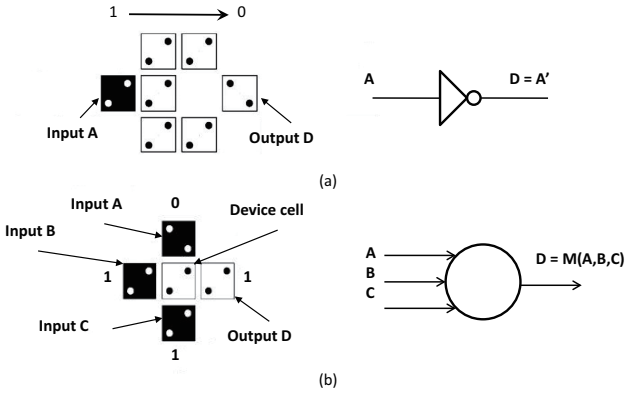

Fig. 3. Two basic QCA device. (a)Inverter. (b)Majority gate.

replacement. The substitute nodes of the target node are the functionally equivalent nodes or the functionally compatible nodes — the nodes that are not equivalent but their functional difference cannot be observed at the primary outputs (POs) of the circuits[4].

In this paper, we propose an optimisation approach for majority logic by merging nodes in the MIGs. Our approach exploits logic implications to identify the substitute nodes directly without collecting candidate nodes and running SAT solving processes. We conducted experiments on a set of benchmarks represented in MIG[22] from EPFL. The experimental results show that our approach can effectively minimise the gate count of the benchmarks. We also integrate our approach with an MIG online synthesis system [2][3]. The experimental results show that the area can be significantly reduced by 21% on average. These results demonstrate that the optimisation quality can be further elevated when different but independent approaches are integrated together.

The main contributions of this work are two-fold:

1) We propose the first node merging approach for the majority logic optimisation.
2) The logic implications in the majority logic are comprehensively discussed.

The rest of this paper is organized as follows. Section II gives the background of this work. Section III presents the proposed node merging algorithm. Section IV show shows the experimental results. Finally, the conclusion of this work is presented in Section V.

II. Preliminaries

In this section, we introduce the majority logic, related concepts of VLSI testing, and node merging.

A *majority function* is an odd-input function that has the output value of v if and only if more than half of the inputs are assigned the value of v. Majority-Inverter-Graph (MIG) is a directed, acyclic graph that represents a logic network with three-input nodes indicating the majority functions. A dot on an edge means that there exists an inverter between two nodes connected by the edge.

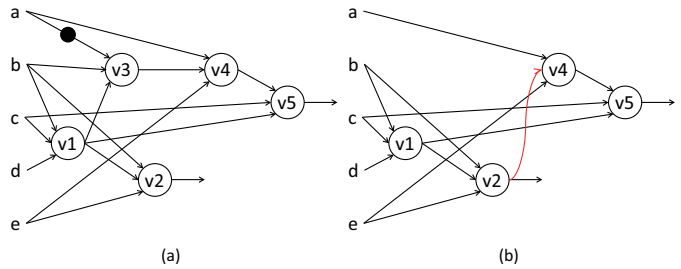

Fig. 4. Example of node merging performed on an MIG. (a) The original MIG. (b) The reduced MIG after merging v_3 with v_2.

The *input-controlling* value of a gate g is the value which can determine the output of g regardless of the other input values. The complement of the input-controlling value is *input-noncontrolling* value. For an AND gate, the input-controlling value is 0, the input-noncontrolling value is 1, and vice versa for an OR gate.

The *dominator*[9] of a gate g are a set of the gates D if and only if all paths from g to any POs have to pass through all gates in D. For all nodes on a path from gate g_i to gate g_j, the *side-inputs* of the path are the inputs of the nodes in the path that are not in the transitive fanout cone of g_i.

A stuck-at fault in VLSI testing is a model used to represent a manufacturing defect within a circuit. It assumes that the faulty wire (or the fanin gate connected to the wire) in a circuit is stuck at a constant value, either 0 or 1, referred to stuck-at 0 (sa0) or stuck-at 1 (sa1) fault, respectively. A stuck-at fault test is a process for finding an input pattern which can generate different PO values in the faulty and fault-free circuits. For the test pattern generation, it is required to activate the fault-effect on the fault site, and propagate the fault-effect to any POs in the circuit. If there does not exist input pattern that can simultaneously activate and propagate the fault-effect to a PO, the fault is an *untestable fault*.

An untestable fault is also a *redundancy* in the circuit because its fault-effect cannot be observed at any POs. Thus, an untestable fault can be replaced with a constant 0 or 1 with respect to the faulty value. *The mandatory assignments* (MAs) of a fault test are the unique value assignments on the circuit for detecting the fault. If the MAs of a fault test are inconsistent, it means that the test pattern for the fault does not exist and the fault is untestable.

A node merging approach performing on AIGs was proposed in [4][5]. It models a node merger as a misplaced-wire error in a circuit. To detect the error of replacing a target node n_t with a substitute node n_s, an input pattern has to generate different values for n_t and n_s to activate and propagate the error-effect to any POs. If there does not exist an input pattern that can detect the error, the error is untestable and the replacement of n_t and n_s is safe. That work[4] proposed a sufficient condition to identify n_s for n_t:

Condition 1 [4]: Let f denote an error of replacing n_t with n_s. If n_s = 1 and n_s = 0 are MAs for stuck-at 0 and stuck-at 1 fault tests on n_t, respectively, f is untestable.

Furthermore, Condition 1 can be extended by reversing the value of n_s and for replacing n_t with an additional inverter. Hence, if there exist nodes having different values in the MAs of stuck-at 0 and stuck-at 1 fault test on n_t, these nodes are considered as n_s.

The approach performs a stuck-at 0 test and a stuck-at 1 test on n_t to compute their MAs, denoted as MAs(n_t = $sa0$)

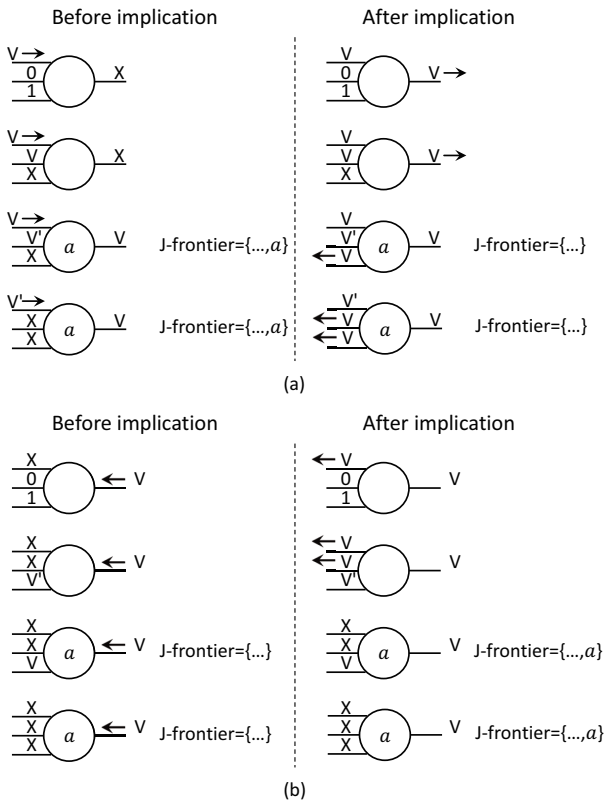

Fig. 5. Logic implications in majority gate. (a) Forward. (b) Backward.

and MAs(n_t = $sa1$), respectively. Then, the nodes that satisfy Condition 1 or its extension are the substitute nodes n_s for n_t.

III. The Proposed Node Merging Approach

In this section, we explain the logic implications for majority logic circuits. Then, we describe how to find the MAs of the stuck-at faults on n_t. At last, we demonstrate the overall flow of the proposed node merging approach.

A. Example

We use an example in Fig. 4 to demonstrate the node merging approach. Fig. 4 is a circuit represented in MIG. a, b, c, d, and e are the primary inputs (PIs); v_1, v_3, and v_4 are internal nodes; and v_2 and v_5 are the POs. For the nodes v_3 and v_2, their functionalities are different. Hence, replacing v_3 with v_2 may cause an error. However, we find that to observe this functional difference at the POs, a and e have to be assigned the same value[1]. For v_4, fortunately, when its two inputs a and e are assigned the same value v, the value of node v_4 is also v such that the value of v_3 cannot affect v_4 and even v_5. This means that the different values of v_3 and v_2 will not be observed at the POs. Therefore, replacing v_3 with v_2 still makes the functionality of the circuit intact. As a result, v_3 and the inverter at its fanin can be both removed to obtain the minimised circuit as shown in Fig. 4(b). In Fig. 4(b), we use v_2 to replace v_3 for driving v_4.

[1] v_3 and v_2 have two common fanins b and v_1, only the third fanin $\bar{a}$ and e is different.

TABLE I. Four sets of value assignments.

side-input pair		side-input pair		n_t						conflict
a	e	c	v_1	v_3	v_4	v_5	v_2	b	d	v_3
0	1	0	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	$\underline{0}$
1	0	0	1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	0	0	0	$\underline{0}$

B. Noncontrolling pair

To detect a fault, it is required to activate and propagate the fault-effect to any POs. Unlike two-input AND/OR gates, a three-input majority gate has two side-inputs in the fault propagation path. Furthermore, a majority gate does not have a noncontrolling value, which is used to propagate the fault-effect to the POs. For a three-input majority gate, the value of an input can be propagated to the output if and only if the two side-inputs are assigned to different values. For example in Fig. 3(b), if we would like to observe the change of the input A in the output D, the side-inputs B and C have to be assigned different values (e.g., (B, C) = (0, 1) or (B, C) = (1, 0)). We name these two side-inputs of a majority gate as a *side-input pair*, and name these different values of the side-input pair as a *noncontrolling pair* in this work.

C. Logic implications in majority logic

The logic implications in a majority gate are summarised in Fig. 5. The J-frontier is a set of gates whose output values are known, but they are not implied from its input values. The forward (backward) implication is to the propagate the value from a wire to its immediate successor (predecessor). In Fig. 5(a), the first two rows (cases) show that the value V is propagated from an input of a majority gate to the output based on the majority logic (X denotes an unknown value). For the last two cases, the gate a is in the J-frontier with a known value V. Hence, the unknown inputs can be justified as having the value of V after the implication. After the implication, the gate a is removed from the J-frontier.

Fig. 5(b) summaries the cases in the backward implications. For the first two cases, the unknown input values are implied to V from the output value V. For the last two cases, since the information is not enough to infer the input values, we put the gate a into the J-frontier.

D. MA computation

The optimisation quality of our approach depends on the number of substitute nodes we found. The substitute node identification is based on the MA computations. The more MAs we compute, the more substitute nodes we can identify for the target node. However, computing all the MAs for a fault test is an NP-hard problem, which is equivalent to the problem that generates *all* the test patterns for a fault. Thus, we use the dominator-based MA computation method and a recursive learning technique[10] with the depth of one in this work to trade off the efficiency and effectiveness of the MA computation.

To identify the substitute node n_s for the target node n_t, we need to compute MAs(n_t = $sa0$) and MAs(n_t = $sa1$). In the circuits represented in AIGs, the MAs can be obtained by assigning the inverse value of the faulty value on n_t for fault activation and by assigning the noncontrolling values on the side-inputs for fault propagation. Furthermore, performing the forward and backward implications on the side-inputs can

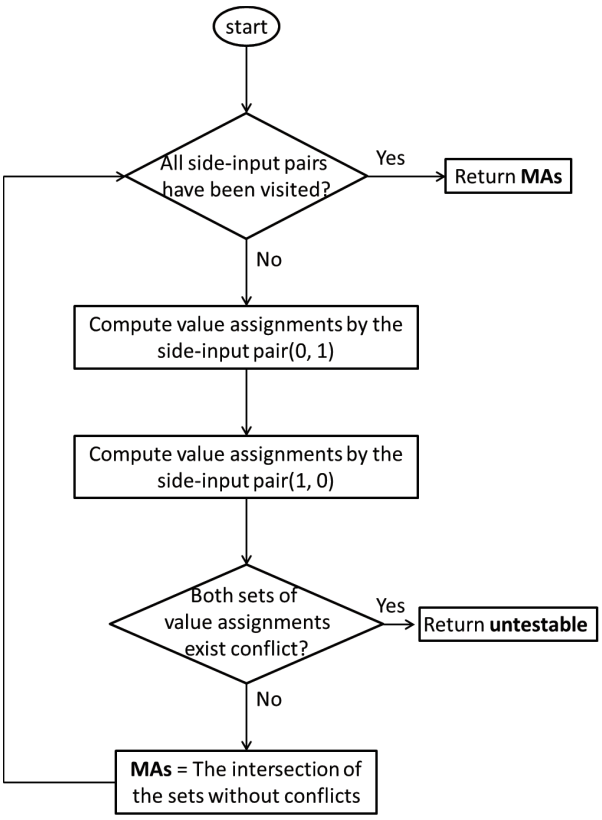

Fig. 6. The procedure of MA computation for single-fanout nodes.

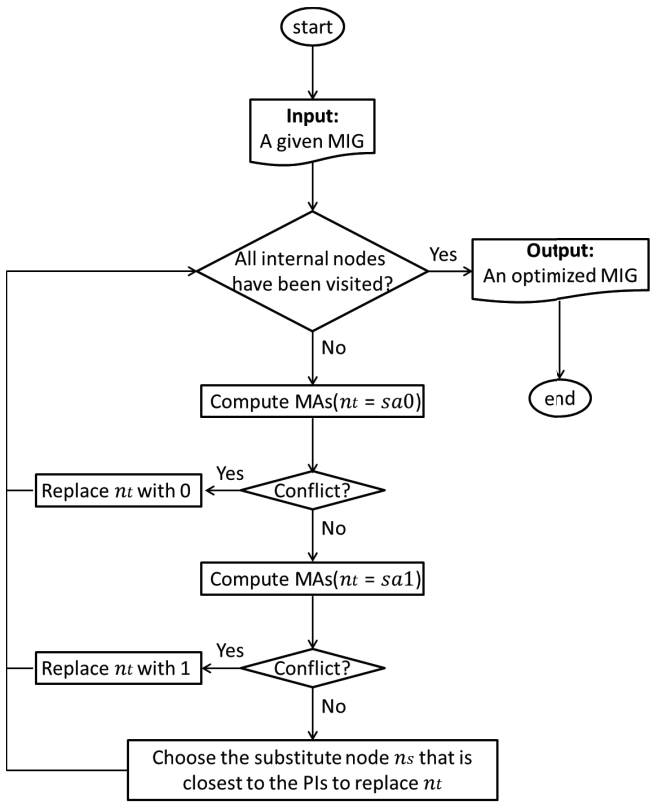

Fig. 7. The overall flow of the proposed approach.

infer more MAs. If the MAs are consistent in one fault test, and we obtain different values for one node in the MAs(n_t = $sa0$) and MAs(n_t = $sa1$), n_s are identified for n_t, where n_s are the nodes having different values in the MAs(n_t = $sa0$) and MAs(n_t = $sa1$). Otherwise, the fault is untestable and n_t can be replaced with the constant 0 or 1 with respect to the fault.

In the MIG, however, we use the side-input pair and noncontrolling pair as discussed in Section III.B to propagate the fault-effect to any POs. Since the noncontrolling pairs have two value assignments to the side-input pair (one input is assigned to v and the other is assigned to $\bar{v}$, where v is 0 or 1) and both of these value assignments could propagate the fault-effect successfully, the MA results are computed based on these two sets of consistent value assignments. According to the definition of MAs, the intersection of these two sets of value assignments is the resultant MA set.

For example in Fig. 4, we consider v_3 as n_t and then perform a stuck-at 0 fault test on it. Since our MA computation is dominator-based, only the side-input pairs of the dominators of v_3 are considered. The dominators of v_3 are v_4 and v_5, and the side-input pairs of v_4 and v_5 are (a, e) and (c, v_1), respectively. Hence, four combinations of noncontrolling pairs ($\{(a, e) = (0, 1), (c, v_1) = (0, 1)\}$, $\{(a, e) = (0, 1), (c, v_1) = (1, 0)\}$, $\{(a, e) = (1, 0), (c, v_1) = (0, 1)\}$, and $\{(a, e) = (1, 0), (c, v_1) = (1, 0)\}$) are considered for the fault test. The corresponding sets of value assignments are summarised in Table I. Since some assignments are self-conflict in a set, like the second and the fourth set, we discard them. For the remaining sets, we conduct the intersection operation on them. Then we can obtain the MAs(v_3 = $sa0$), which are $\{v_3 = 1, v_4 = 1, v_5 = 1, v_2 = 1, b = 1, d = 1, c = 0, v_1 = 1\}$. Similarly, we next conduct the stuck-at 1 fault test on n_t, and obtain the MAs(v_3 = $sa1$) = $\{v_3 = 0, v_4 = 0, v_5 = 0, v_2 = 0, b = 0, d = 0, c = 1, v_1 = 0\}$. As a result, v_2, b, d, c and v_1 have different values and they are the substitute nodes n_s for v_3. Note that here we do not consider v_4 and v_5 as the substitute nodes. This is because when choosing v_4 or v_5 as the substitute node for v_3, we will create a cyclic circuit. This is a situation that this paper does not discuss with.

For a stuck-at fault test on n_t, if there are n dominators on the path from n_t to the POs, 2^n different sets of noncontrolling pairs are assigned for the MA computation. Thus, the complexity for computing MAs in terms of the number of noncontrolling pairs assigned on the dominators is O(2^n). In fact, using this straightforward method, many logic implication operations are redundant and could be eliminated. Thus, we propose a more efficient method for the MA computation. Since the fanouts of a target node can be either single or multiple, we discuss them separately.

For a single-fanout node, the procedure of MA computation is as shown in Fig. 6. Instead of assigning noncontrolling pairs to all the side-input pairs of all the dominators, and then conducting intersection for MA computation, we consider one dominator a time gradually.

We also use the same example in Fig. 4 to demonstrate our method. We perform the stuck-at 0 fault test on v_3. The dominators are v_4 and v_5. We set the value 1 to v_3 and assign (a, e) = (0, 1) to the side-input pair of the first dominator v_4. We obtain the value assignments as $\{v_3 = 1, v_4 = 1, a = 0, e = 1, v_2 = 1\}$. Then we assign ($a$, e) = (1, 0) to the side-input pair of v_4 and obtain the other set of value assignments

TABLE II. THE EXPERIMENTAL RESULTS OF THE PROPOSED APPROACH ON THE WELL-OPTIMISED BENCHMARKS BY *MIGhty*[2][3].

				Depth Increase					Depth Preservation				
Benchmark	I/O	Size	Depth	Size	(%)	Depth	(%)	Time	Size	(%)	Depth	(%)	Time
usb_phy	113/111	484	8	469	3.10	8	0.00	0.51	470	2.89	8	0.00	0.51
ss_pcm	106/98	496	7	493	0.60	7	0.00	0.82	494	0.40	7	0.00	0.81
sasc	133/132	754	7	749	0.66	9	-28.57	1.15	751	0.40	7	0.00	1.14
simple_spi	148/147	985	9	962	2.34	11	-22.22	2.56	976	0.91	9	0.00	2.54
pci_spoci_ctrl	85/76	1009	12	863	14.47	15	-25.00	7.50	872	13.58	12	0.00	7.67
i2c	147/142	1114	9	1086	2.51	12	-33.33	2.68	1093	1.89	9	0.00	2.70
hamming	200/7	2709	62	1933	7.02	65	-4.83	11.88	1952	6.11	62	0.00	12.10
sqrt32	32/16	2173	165	2072	4.65	187	-13.33	42.46	2088	3.91	165	0.00	41.71
systemcdes	314/258	2712	20	2577	4.98	25	-25.00	34.40	2630	3.02	20	0.00	34.42
spi	274/276	3614	20	3361	7.00	20	0.00	75.81	3363	6.95	20	0.00	76.53
des_area	368/72	4259	23	4006	5.94	24	-4.34	136.53	4008	5.89	23	0.00	134.89
max	512/130	4341	30	4300	0.94	30	0.00	204.56	4300	0.94	30	0.00	204.80
div16	32/32	4407	103	3966	10.00	129	-25.24	141.08	4060	7.87	103	0.00	145.52
revx	20/25	7542	144	6989	7.33	174	-20.83	238.48	7180	4.80	144	0.00	252.21
tv80	373/404	7802	31	7553	3.19	38	-22.58	305.69	7592	2.69	31	0.00	306.78
mem_ctrl	1198/1225	8369	20	8256	1.35	30	-50.00	132.98	8278	1.09	20	0.00	134.49
MUL32	64/64	9161	37	8497	7.25	44	-18.97	117.07	8703	5.00	37	0.00	120.60
MAC32	96/65	9392	42	9144	2.64	58	-38.09	79.10	9182	2.24	42	0.00	78.95
systemcaes	930/819	10367	26	10229	1.33	29	-11.53	284.82	10272	0.93	26	0.00	282.89
ac97_ctrl	2255/2250	12996	9	12834	1.25	9	0.00	97.68	12979	0.13	9	0.00	100.35
usb_funct	1860/1846	14842	20	14636	1.39	25	-25.00	258.98	14704	0.93	20	0.00	259.17
square	64/127	18015	41	17562	2.51	81	-97.56	259.66	17937	0.43	41	0.00	263.63
diffeq1	354/289	18015	220	17162	4.73	244	-10.90	436.08	17358	3.65	220	0.00	446.67
comp	279/193	18687	78	18285	2.15	106	-26.31	1841.55	18388	1.60	78	0.00	1883.61
aes_core	789/668	21616	19	20402	5.62	24	-26.31	831.81	20555	4.91	19	0.00	841.73
pci_bridge32	3519/3528	22132	17	21955	0.80	21	-23.52	501.10	22022	0.50	17	0.00	503.10
mult64	128/128	25901	110	25403	1.92	113	-2.72	637.64	25410	1.90	110	0.00	636.07
log2	32/32	31359	202	30659	2.23	216	-6.93	5001.42	30715	2.05	202	0.00	5000.27
DSP	4223/3953	44166	35	43444	1.63	45	-28.57	1723.69	43614	1.25	35	0.00	1730.58
Average	—	—	—	—	3.85	—	-20.73	462.40	—	3.06	—	0.00	465.74

as $\{v_3 = 1, v_4 = 1, a = 1, e = 0, v_2 = 1, b = 1, v_1 = 1\}$. Then we conduct the intersection operation on these two sets of value assignments and obtain $\{v_3 = 1, v_4 = 1, v_2 = 1\}$. Note that this set of assignments is not MAs($v_3 = sa0$) since only the first dominator is considered. Next, we compute the value assignments for the second dominator v_5 in the same manner. The value assignments of $\{v_3 = 1, v_4 = 1, v_2 = 1\}$ from the first dominator are also assigned in this round to reduce the efforts in the MA computation. Finally, we obtain the MAs($v_3 = sa0$) as $\{v_3 = 1, v_4 = 1, v_5 = 1, v_2 = 1, b = 1, d = 1, c = 0, v_1 = 1\}$.

For a multiple-fanout node, we perform the stuck-at v fault test on each fanout w_i of n_t, where i is from 1 to the number of fanouts of n_t. The computation of MAs($w_i = sav$) is similar to that for a single-fanout node. After considering all the fanouts of n_t, the MAs($n_t = sav$) are the intersection of all the MAs($w_i = sav$), where the value assignments in the MAs($w_i = sav$) are not conflict.

E. Overall algorithm

The overall flow of the algorithm is shown in Fig. 7. Given an MIG, each internal node n will be selected as a target node, and MAs($n = sa0$) and MAs($n = sa1$) are computed. The target node selection is in the ordering of depth-first-search from the POs to the PIs heuristically, which can obtain the better optimisation results than the topological ordering for most benchmarks. If there exist nodes having different values in the MAs($n = sa0$) and MAs($n = sa1$), they are substitute nodes. The substitute node that is the closest to the PIs is selected to replace n for minimising the depth increase in the MIG. Note that all the single-fanout nodes within the transitive fanin cone of n will be removed as well.

IV. EXPERIMENTAL RESULTS

We implemented our approach in C++ language. We conducted two experiments for a set of EPFL benchmarks[22] on an Intel Xeon® X5570 2.93GHz CentOS 5.11 platform with 48 GBytes memory.

The first experiment shows the results of our approach on the well-optimised benchmarks by[2][3]. We provide two versions of our results. One is to optimise the gate count without considering depth increase, the other is to optimise the gate count with depth preservation. The depth preservation version is that we only choose the substitute node whose depth is lower than that of the target node. Table II shows the experimental results of the first experiment. Columns 1–4 list the information of each benchmark represented in MIG. Columns 5–9 list the results of depth increase version. Columns 10–14 list the results of depth preservation version. The percentage of reduction is calculated as the (new - old) / old.

According to Table II, for these well-optimised benchmarks, our approach can further reduce the gate count by up to 14.47% with 25% depth increase in the depth increase version. On

TABLE III. THE RESULTS OF OUR APPROACH INTEGRATED WITH *MIGhty*[2][3].

				Optimisation			
Benchmark	I/O	Size	Depth	Size	(%)	Depth	(%)
ss_pcm	106/98	413	7	413	0.00	7	0.00
usb_phy	113/111	498	10	464	6.83	9	10.00
sasc	133/132	782	9	686	12.28	8	11.11
simple_spi	148/147	1068	12	877	17.88	16	-33.33
sqrt32	32/16	1113	495	1109	0.36	493	0.40
i2c	147/142	1181	14	1006	14.82	11	21.43
pci_spoci_ctrl	85/76	1402	18	749	46.58	13	27.78
max	512/130	2865	287	2865	0.00	287	0.00
systemcdes	314/258	3131	27	2640	15.69	24	11.11
hamming	200/7	3612	75	1790	50.44	69	8.00
spi	274/276	3847	32	3330	13.44	23	28.13
des_area	368/72	4865	34	4538	6.72	26	23.53
div16	32/32	7175	136	2467	65.62	134	1.47
tv80	373/404	9691	52	7098	26.76	34	34.62
MUL32	64/64	11613	43	8182	29.54	47	-9.30
MAC32	96/65	12063	70	8993	25.45	64	8.57
systemcaes	930/819	12533	46	9098	27.41	30	34.78
ac97_ctrl	2255/2250	14382	12	12444	13.48	9	25.00
mem_ctrl	1198/1225	15604	36	8145	47.80	25	30.56
usb_funct	1860/1846	16048	27	14681	8.52	25	7.41
revx	20/25	16164	192	6486	59.88	171	10.94
square	64/127	18485	250	17897	3.18	54	78.40
aes_core	789/668	21658	26	19645	9.29	27	-3.85
pci_bridge32	3519/3528	23215	30	20019	13.77	22	26.67
mult64	128/128	27062	274	26858	0.75	112	59.12
log2	32/32	32060	444	30639	4.43	229	48.42
diffeq1	354/289	33632	303	17962	46.59	247	18.48
Total		296162	2961	231081		2216	
Average					21.98		25.16

average, more 3.85% gate count reduction with 20.73% depth increase. However, for the depth preservation version, the gate count reduction is up to 13.58%, and 3.06% on average.

In the second experiment, we integrated our approach with the MIG online synthesis system *MIGhty*[2][3] to further minimise the gate count of original benchmarks. We optimised the benchmarks by repeatedly using our approach followed by[2][3] three times and the best results for gate count are kept. The experimental results are summarised in Table III. The gate count reduction and the depth reduction are up to about 65% and 78%, respectively. The average reduction for gate count and depth are 21% and 25%, respectively.

According to Table III, we know that our approach can be integrated with other optimisation techniques for reducing the size and the delay of majority logic circuits further.

V. CONCLUSIONS

In this paper, we propose a node merging algorithm targeting at gate count minimisation for majority logic circuits. Instead of collecting candidate nodes and running SAT solving processes, our approach exploits logic implications to identify the substitute nodes directly. According to the experimental results, our approach can effectively minimise the gate count of the majority logic circuit such that the implementation cost of the corresponding QCA circuit is also reduced.

REFERENCES

[1] L. Amarú, P.-E. Gaillardon, and G. De. Micheli, "Majority Logic Representation and Satisfiability," *in Proc. International Workshop on Logic & Synthesis*, 2014.

[2] L. Amarú, P.-E. Gaillardon, and G. De. Micheli, "Majority-Inverter Graph: A Novel Data-Structure and Algorithm for Efficient Logic Optimization," *in Proc. Des. Autom. Conf.*, pp. 1-6, 2014.

[3] L. Amarú, P.-E. Gaillardon, and G. De. Micheli, "Boolean Logic Optimization in Majority-Inverter Graphs," *in Proc. Des. Autom. Conf.*, pp. 1-6, 2015.

[4] Y.-C. Chen and C.-Y. Wang, "Fast Detection of Node Mergers Using Logic Implications," *in Proc. Int. Conf. Conput.-Aided Des.*, pp. 785-788, 2009.

[5] Y.-C. Chen and C.-Y. Wang, "Fast Node Merging with Don't Cares Using Logic Implications," *IEEE Trans. Comput.-Aided Design*, vol. 29, no. 11, pp. 1827-1832, Nov. 2010.

[6] Y.-M. Chou, Y.-C. Chen, C.-Y. Wang, and C.Y. Huang, "MajorSat: A SAT Solver to Majority Logic," *in Proc. Asia South Pacific Des. Autom. Conf.*, pp. 480-485, 2016.

[7] H. A. Fahmy and R. A. Kiehl, "Complete Logic Family Using Tunneling-Phase-Logic Devices, *in Proc. Asia South Pacific Des. Autom. Conf.*, pp. 22-24, Nov. 1999.

[8] L. Hellerman, "A Catalog of Three-Variable Or-Invert and And-Invert Logical Circuits," *IEEE Trans. Electron. Comput.*, vol. EC-12, no. 3, pp. 198-223, 1963.

[9] T. Kirkland and M. R. Mercer, "A Topological Search Algorithm for ATPG," *in Proc. Des. Autom. Conf.*, pp. 502-508, 1987.

[10] W. Kunz and D. K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems-Test, Verification, and Optimization," *IEEE Trans. Comput.-Aided Des.*, vol. 13, no. 9, pp. 1143-1158, Sep. 1994.

[11] K. Kong, Y. Shang, and R. Lu, "An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata," *IEEE Trans. on Nanotechnology*, vol. 9, no. 2, pp. 170-183, Mar. 2010.

[12] C. S. Lent, P. D. Tougaw, and W. Porod, "Bistable Saturation in Coupled Quantum Dots for Quantum Cellular Automata," *Applied Physics Letters*, vol. 62, no. 7, pp. 714-716, 1993.

[13] C. S. Lent and P. D. Tougaw, "A Device Architecture for Computing with Quantum Dots," *in Proc. IEEE*, vol. 85, no. 4, pp. 541-557, Apr. 1993.

[14] S. M. Plaza, K.-H. Chang, I. L. Markov, and V. Bertacco, "Node Mergers in the Presence of Don't Cares," *in Proc. Asia South Pacific Des. Autom. Conf.*, pp. 414-419, 2007.

[15] M. Soeken, L. Amaru, P.-E. Gaillardon, and G. De. Micheli, "Optimizing Majority-Inverter Graphs with Functional Hashing," *in Proc. Des., Autom. and Test in Europe*, pp. 1030-1035, 2016.

[16] J. Timler and C. S. Lent, "Power Gain and Dissipation in Quantum-Dot Cellular Automata," *Journal of Applied Physics*, vol. 91, no. 2, pp. 823-831, 2002.

[17] P. D. Tougaw and C. S. Lent, "Logical Devices Implemented Using Quantum Cellular Automata," *Journal of Applied Physics*, vol. 75, no. 3, pp. 1818-1825, 1994.

[18] M. Wilson, K. Kannangara, G. Smith, M. Simmons, and B. Raguse, Nanotechnology: Basic Science and Emerging Technologies, Chapman & Hall/CRC, 2002.

[19] R. Zhang, P. Gupta, and N. K. Jha, "Majority and Minority Network Synthesis With Application to QCA-, SET-, and TPL-Based Nanotechnologies," *IEEE Trans. on Comput.-Aided Des.*, vol. 26, no. 7, pp. 1233-1245, July 2007.

[20] R. Zhang, K. Walus, W. Wang, and G. A. Jullien, "A Method of Majority Logic Reduction for Quantum Cellular Automata," *IEEE Trans. on Nanotechnology*, vol. 3, no. 4, pp. 443-450, Dec. 2004.

[21] L. Zhuang, L. Guo, and S. Y. Chou, "Silicon Single-Electron QuantumDot Transistor Switch Operating at Room Temperature," *Applied Physics Letters*, pp. 1205-1207, 1998.

[22] http://lsi.epfl.ch/MIG