Line Reduction in Reversible Circuits using KFDDs

Jayati J Law
Department of Mathematics and
Computer Science
University of Lethbridge
Alberta, Canada
Email: law@uleth.ca

Jacqueline E. Rice
Department of Mathematics and
Computer Science
University of Lethbridge
Alberta, Canada
Email: j.rice@uleth.ca

***Abstract*—Reversible computing has been theoretically shown to be an efficient approach over conventional computing. This is due to the property of virtually zero power dissipation in reversible circuits. A major concern in reversible circuits is the number of circuit lines which corresponds with qubits. Qubits are a limited resource. There are various reversible logic synthesis algorithms which require a significant number of additional constant lines. In this paper we explore the line reduction problem using a synthesis approach based on decision diagrams. We have added a sub-circuit for a positive Davio node structure to the existing node structures given in [1] with a shared node ordering in OKFDDs. OKFDDs are a combination of OBDDs and OFDDs, thus exhibiting the advantages of both. Our approach shows that the number of circuit lines and quantum cost can be reduced using OKFDDs wit our new sub-circuit and shared node ordering.**

***Keywords*—Kronecker Functional Decision Diagrams (KFDDs), reversible logic, Boolean function, logic synthesis, line reduction.**

I. Introduction

Reversible computing is emerging as a promising technology to reduce the energy consumption required to carry out a computation. A reversible system is bijective in nature, i.e. every input has a unique output which makes the circuit invertible. Therefore, there is no information loss. Since reversible circuits offer (theoretically) virtually zero power dissipation, they are preferable for low-power designs.

According to Launder's principle [2] $KTln2$ amount of energy is released every time an information bit is lost during a logical operation in an irreversible system. K is the Boltzmann constant and T is the room temperature (for T = 300 Kelvin this energy is about 2.9×10^{21} joules). The released energy increases with the increase in the number of transistors in an integrated chip. Since Moore's law predicts that the number of transistors in an integrated chip will double approximately every two years [3], power dissipation becomes a major concern for an irreversible system. Reversible computing also has connections to quantum computing [4]. In quantum computing, unitary matrices are the building blocks of a quantum circuit. These unitary matrices are invertible and hence, reversible circuits provide a suitable platform for quantum computing. Other areas of application of reversible circuits are cryptography [5], nano-computing technologies [6] and digital processing [7].

Design of a minimal reversible circuit is a complex challenge. In order to make a circuit reversible additional circuit lines are added to the circuit. These additional lines do not contribute to the output of the circuit and are also known as 'garbage lines'. Each line in a circuit represents a qubit which is still a limited resource. Therefore, the number of lines in a reversible circuit is an important specification to consider. Other parameters include gate count (number of gates in a reversible circuit) and quantum cost (cost of gates in a reversible circuit).

There are various techniques for synthesizing reversible circuits, including ESOP-based [8], Reed-Muller expansions [9], decision diagrams [10] and truth tables [11]. Besides synthesis methods, optimization [12], [13], testing [14] and verification [15] of reversible circuits has also been studied extensively. Every synthesis technique aims for an efficient algorithm to produce optimized circuits. A reversible circuit with minimal (or close to) number of lines, gates and quantum cost is considered to be an optimized circuit.

A. Related Works

Minimizing the number of lines in a circuit is an important issue for optimizing a reversible circuit. In order to convert an irreversible truth table to a reversible truth table extra qubits may be added to the table, hence adding extra lines in the circuit. Reducing circuit lines may cause a trade-off with the quantum cost [16] which makes this problem more challenging. Many heuristics have been introduced to reduce these additional lines in the circuit. Some of these reduce lines in the long term by adding lines in the short [17] and others reuse the input lines instead of additional lines [18]. In [19] a set of positive Davio decomposition templates are applied to the decision diagrams to optimize the circuit. The authors in [20] have introduced a lower bound on the minimum number of lines required by a reversible circuit to realize a Boolean function.

The rest of the paper is structured as follows. We first provide a brief background on reversible logic and decision diagrams. Section 3 explains our approach with the algorithm and an example. In section 4 we give the experimental results for our algorithm, followed by the conclusion in section 5.

II. Background

A. Reversible Logic

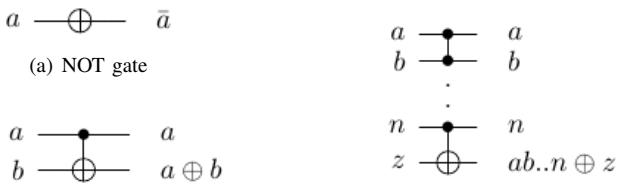
Definition 1. *A multi-output Boolean function* $f(x) : B^m \rightarrow B^n$ *is reversible if it is bijective.*

If B is a finite set and $f(x) : B^m \rightarrow B^n$ is a Boolean function which maps each input vector to a unique output

vector (bijection) then $f(x)$ is reversible. Each reversible function is mapped on to a reversible circuit using reversible gates.

Definition 2. *A reversible gate computes a reversible function.*

Let $X := \{x_1, ..., x_n\}$ be the set of Boolean variables. Then a reversible gate has the form $g(C, T)$, where $C = \{x_{i1}, ...x_{ik}\} \subset X$ is the set of control lines and $T = \{x_{j1}, ..., x_{jl}\} \subset X$ with $C \cap T = \phi$ is the set of target lines [21]. The most commonly used reversible gate is a Toffoli gate. A Toffoli gate with no controls is a NOT gate i.e. $g(0, x_{j1})$. Similarly, a Toffoli gate $g(x_{i1}, x_{j1})$ is a CNOT gate while $g(\{x_{i1}, ..x_{in}\}, x_{j1})$ is a n-bit Toffoli gate. Figure 1 shows the symbols and notation for several reversible gates. Each of these reversible gates have a specific quantum cost depending on the number of basic quantum gates they consist of [22]. For example the quantum cost of both the NOT and CNOT gates is 1 [23].



(a) NOT gate

(b) CNOT gate

(c) n-bit Toffoli gate

Fig. 1. Reversible gates.

Definition 3. *A reversible circuit is a network of reversible gates realizing a reversible function.*

A reversible circuit as shown in Figure 2, consists of reversible gates with controls shown as dots '•' while the target is represented by a '⊕'.

Definition 4. *Garbage outputs are additional outputs which do not produce any desired functionality.*

In [17] it is shown that at least $g = \lceil log_2(\mu) \rceil$ garbage outputs are required for converting an irreversible function to a reversible function, where μ is the maximum number of times a single output pattern is repeated in an irreversible truth table. Converting an irreversible function with n inputs and m outputs into a reversible function will require $m + g$ qubits. Since $m + g > n$, a line labeled c is added to make this function reversible. Thus, it becomes $n + c = m + g$, where c is a constant input. The process of adding extra 'c' qubits to the irreversible truth table to make it reversible is also known as embedding [17]. Thus, this causes a reversible circuit to have more lines than simply providing the input values for computation. In Figure 2 the garbage outputs are labeled as g_1 and g_2.

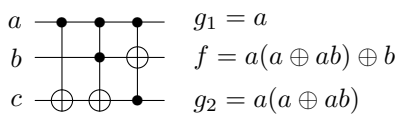


Fig. 2. Reversible circuit with garbage outputs labeled as g_1 and g_2.

The number of lines in a reversible circuit also depends on the synthesis algorithm being used. In this paper we discuss the problem of reducing these additional circuit lines that are added during the decision diagram based synthesis approach.

B. Decision Diagrams

Definition 5. *A decision diagram (DD) is an acyclic directed graph $G(V, E)$ representing a Boolean function f with $V \in \{v_1, v_2, ...v_n\}$ and $E \in \{$0-edge, 1-edge$\}$.*

In a DD each node $v \in V$ is labeled by a variable from the variable set $X \in \{x_1, x_2, ...x_n\}$. Nodes can be terminal (circle) or non-terminal (square). The 0-edge and 1-edge from the node v leads to $low(v)$ and $high(v)$ respectively. 0-edges and 1-edges are usually depicted by a dashed line and a regular line respectively.

Definition 6. *Each node v in a DD uses one of the following decomposition types for decomposition into its factors [24].*

- *Shannon (S) :* $f(x_i) = \overline{x_i} f_0 + x_i f_1$
- *Positive Davio (pD) :* $f(x_i) = f_0 \oplus x_i f_2$
- *Negative Davio (nD) :* $f(x_i) = f_1 \oplus \overline{x_i} f_2$

f_0 is the function f for $x_i = 0$, f_1 denotes f for $x_i = 1$ and f_2 denotes $f_0 \oplus f_1$. A Binary Decision Diagram (BDD) only implements the Shannon decomposition type for all the nodes while a Functional Decision Diagram (FDD) uses Davio decompositions. An ordered DD has a specific variable ordering such as $\phi = x_1 < x_2 < x_3 ... x_n$. An ordered BDD or FDD is denoted by OBDD or OFDD respectively.

Another type of DD is an Ordered Kronecker Functional Decision Diagram (OKFDD) which utilizes all three decomposition types. It exhibits the properties of an OBDD and OFDD. A Decomposition Type List (DTL) is defined for each node in a OKFDD such that the DTL $d : (d_1, d_2, ...d_n)$ where $d_i \in \{S, pD, nD\}$.

Example 1. *Let the OKFDD in Figure 3 represent a Boolean function $f = \overline{x_1} x_2 \oplus x_1 x_2 x_4 x_3 \oplus x_1 \overline{x_2} x_3$ with the variable order of $x_1 < x_2 < x_3 < x_4$. f first decomposes into $f_4 = x_2$ and $f_5 = x_2 x_4 x_3 \oplus \overline{x_2} x_3$ with a Shannon decomposition node. Then, f_4 decomposes into constants* 0 *and* 1 *while f_5 decomposes into $f_3 = x_3$ and $f_2 = x_4 x_3$ with a positive Davio decomposition node. Next, f_3 decomposes into terminal nodes* 0 *and* 1 *while f_2 decomposes into $f_1 = x_4$ with a negative Davio decomposition node. Finally, f_1 factors into terminal nodes* 0 *and* 1 *with a Shannon decomposition node.*

III. Algorithm

In this section we describe the algorithm used in [1]. This is based on synthesizing reversible circuits using the factors of a Boolean function through a decision diagram. Our addition to this work is a new sub-circuit for positive-Davio decompositions and an ordering in which the variables are to be addressed when a shared node is encountered in the OKFDD. Previously in [19] similar shared node ordering is used with only pD decomposition type nodes. Algorithm 1 explains the process of mapping an OKFDD to a reversible circuit. The input for the algorithm is the OKFDD or acyclic directed graph $G(V, E)$ generated by the algorithm in [1].

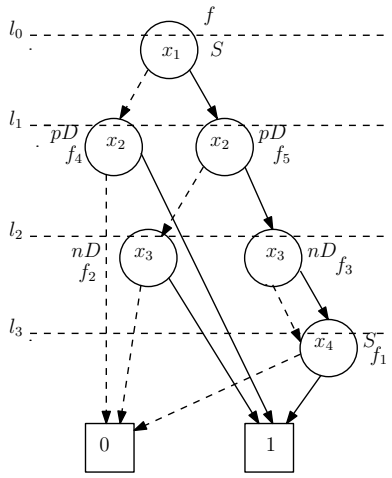


Fig. 3. OKFDD for the function $f = \overline{x_1}x_2 \oplus x_1x_2x_4x_3 \oplus x_1\overline{x_2}x_3$

The output of the algorithm is a reversible circuit termed as $rev\_cascade$. Initially, we take an empty circuit $rev\_cascade$ and add lines or gates when required. We start with an empty queue Q to store nodes in the order they have to be processed. The nodes are pushed into the Q at the front (push_front) and popped from the back (pop_back) when being processed. In step 2 we start traversing the graph bottom-up for each level l_i from the non-terminal nodes to the root node. In step 3 each node v_j of level l_i is scanned for the decomposition type. Step 4 defines the case for a shared node.

In Figure 4 the node labeled x_j is a shared node as it is shared by two x_i nodes. In the case of a shared node, if the function represented by x_j is to be implemented using no additional constant line then the node (v_{f_1}) with the 1-edge leading to the x_j is realized first and then the node (v_{f_2}) with the 0-edge to x_j. Figure 4 shows the circuit implementation of a shared node using a positive Davio decomposition [19]. In this case the circuit has no additional constant circuit lines but when synthesizing from a KFDD the shared node implementation depends on the decomposition type and the [10] gives a shared node implementation, but only for Shannon decomposition types, thus we are proposing our own approach for shared nodes that use pD and nD decompositions. The cost of the circuit depends on the decomposition type of the nodes.

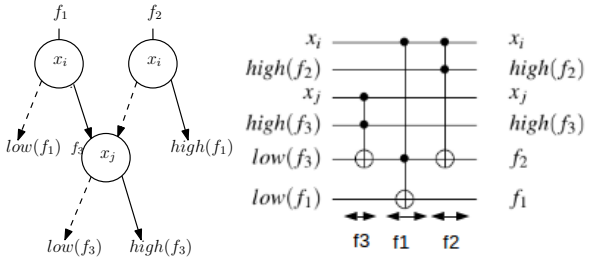


Fig. 4. Shared node and equivalent circuit

Step 4 in the algorithm checks if there is a case of shared node at a level l_i. If so then then f_1 is implemented before f_2, as explained above. The nodes are stored in the Q accordingly. Step 5 is the case where no shared node is involved, hence the algorithm simply stores the node in the Q by the order in which it occurs. In Step 6 the nodes are popped from the Q and implemented by selecting a sub-circuit from Table I depending on the node decomposition type and structure. The most frequent sub-circuits for each decomposition type shown in Table I are given in [1]. We have added a new sub-circuit for Davio decomposition of node structure 3 which requires no additional lines and only one CNOT gate. In Step 7 each sub-circuit selected by Step 6 is added to the main circuit $rev\_cascade$. If the $rev\_cascade$ has the input states required by the sub-circuit then they merge otherwise new constant additional lines are added for the required input states.

Input: A KFDD (directed acyclic graph) $G(V, E)$ where $V = \{v_1, v_2, ...v_n\}$ and $E = \{$0-edge, 1-edge$\}$.
Output: A reversible circuit $rev\_cascade$ with minimum lines.
Algorithm 1 $line\_reduction(G, v)$

1) Take an empty circuit $rev\_cascade \leftarrow \phi$ and an empty queue Q.
2) Traverse the OKFDD bottom-up for every level l_i where $i = (n, n-1, ...0)$.
3) Scan every unvisited non-terminal node v_j at level l_i for $j = (0, 1...k)$.
4) If node v_j represents function f_2 (0-edge of v_i leads to a shared node) such as given in Figure 2 then push_front v_{j+1} (f_1) in Q then v_j in Q. Similarly, if v_j represents the function f_1 (1-edge leads to shared node) then push_front v_j (f_1) in Q and then v_{j+1} (f_2) in Q.
5) Else if there is no shared node simply push_front the v_j in Q.
6) Pop_back the nodes from Q and implement the function represented by the nodes by selecting the sub-circuit from Table I according to the decomposition type of each node. Mark each implemented node as visited.
7) If $rev\_circuit$ has the required input states then merge the sub-circuit otherwise add extra constant lines and merge sub-circuit with $rev\_cascade$.

Example 2. *Figure 5 shows an OKFDD for the function* $\overline{x_1}(x_2 \oplus x_3) \oplus \overline{x_2}x_3$. *Firstly,* f *decomposes into* $f_3 = \overline{x_2}x_3 \oplus x_3$ *and* $f_2 = x_2 \oplus x_3$ *by nD decomposition type. Then,* f_3 *decomposes into* $f_1 = x_3$ *and terminal node* 1 *by pD type. Next,* f_2 *into* $f_1 = x_3$ *and* 0 *by pD type. Finally,* f_1 *decomposes into terminal nodes* 0 *and* 1 *by Shannon type.*

To generate the circuit using Algorithm 1, we traverse the OKFDD from node x_3 to the root node x_1 level wise starting from l_2. The suitable sub-circuit is selected from Table I for each node. Here node f_1 at l_2 is a shared node and thus, f_3 is implemented before f_2. For node f_1 at l_2 since the function is x_3 (a single variable) we require only a single line in the circuit. At l_1 for node f_2 the pD node structure 3 is used from the Table I. Similarly, for the node f_3, pD node structure 5 is used which is similar to S decomposition. Lastly, for node f at l_0, nD node structure 1 is used. Figures 5b and 5c show the equivalent circuit implementations using the previous algorithm [1] and our algorithm respectively. As illustrated our approach produces a smaller circuit as compared to the previous approach [1].

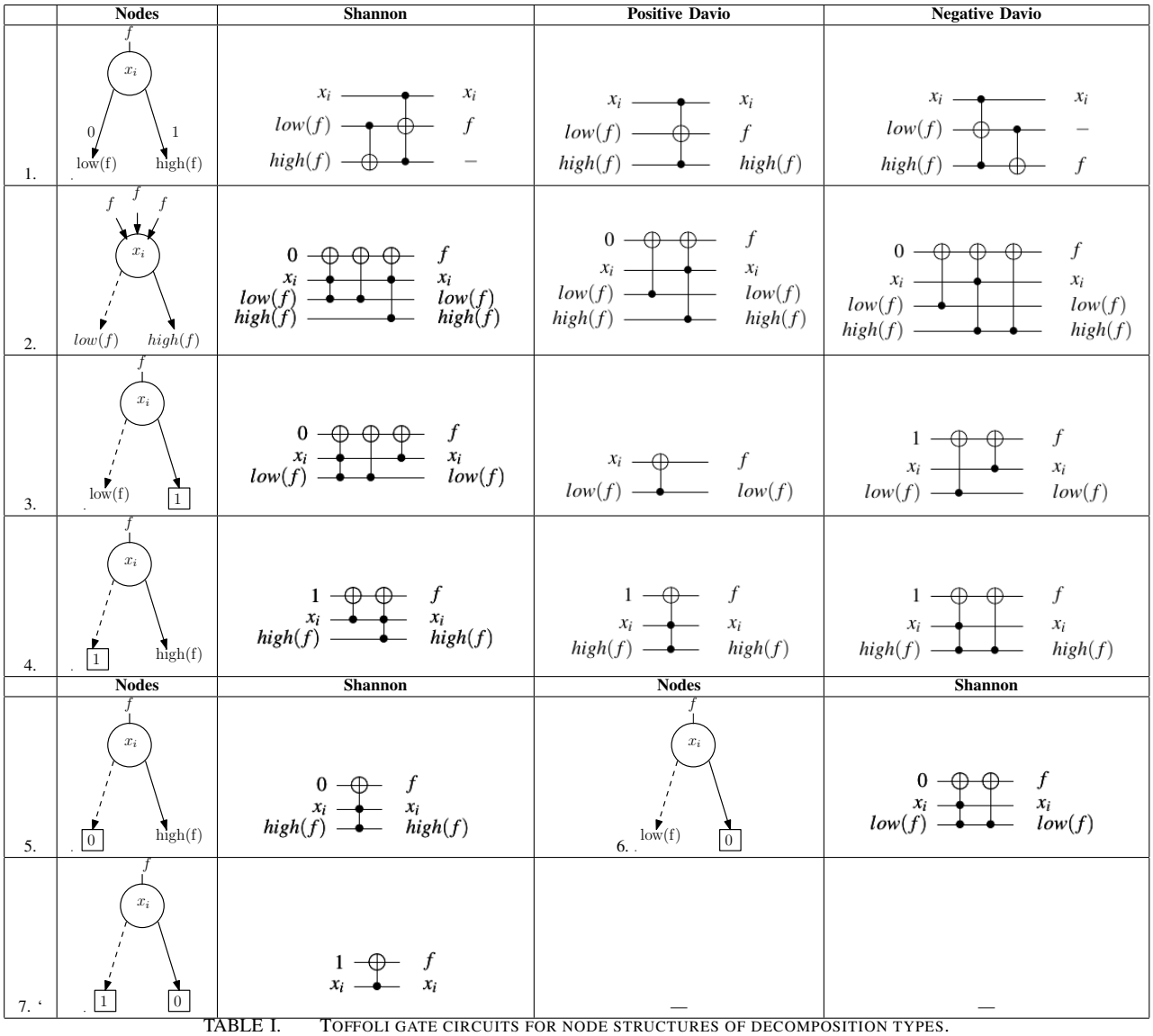


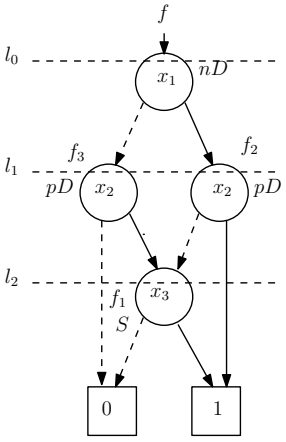
TABLE I. TOFFOLI GATE CIRCUITS FOR NODE STRUCTURES OF DECOMPOSITION TYPES.

IV. EXPERIMENTAL EVALUATION

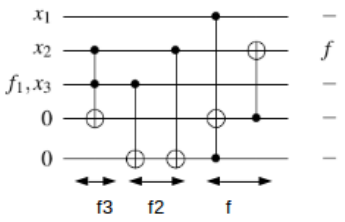
The algorithm is implemented in C++ in Revkit [25]. The reversible functions in Table II are from Revlib [26]. The algorithm to generate an OKFDD is given in Revkit under KFDD-based synthesis algorithms which includes the PUMA package for decompositions and optimizing algorithms. The sifting algorithm [27] is used by PUMA to find a variable ordering and DTL that results in the fewest nodes in the OKFDD. Verification of the results in Table II is done using Revkit's equivalence checker. The experiment was performed on a 1.9 GB, Intel Core 2 Duo processor Linux machine.

In Table II first column shows the functions. The next two columns consist of the number of inputs and outputs for corresponding functions. The results of our algorithm are compared with the results of the previous KFDD algorithm [1] and BDD approach from [10]. The notation 'L' denotes the number of lines in the circuit while 'QC' and 'GC' denote the quantum cost and gate count respectively. The changes in the metrics are shown by 'Δ L', 'Δ QC' and 'Δ GC'. The results show a significant decrease in the number of lines as well as in quantum cost and gate count. In the best case (e.g. $plus127$) the line reduction is 42% compared to KFDD approach and 29% (e.g. $tial$) compared to BDD approach. The average line reduction is approximately 10% in comparison to KFDD algorithm. Comparing the quantum cost values the average reduction is around 7% and 23% for KFDD and BDD techniques respectively.

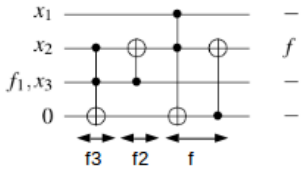
Since a KFDD uses all the decomposition types with the variable ordering, the decision diagram is more likely to generate a smaller realization compared to other DD based algorithms. Some of the functions show great improvement such as $plus127$ and $tial$ due to the frequent presence of node structure #3. We can see that if a pD decomposition type #3 is used then only one gate and no additional lines are required. Although there are a few functions that do not show any improvement compared to KFDD lines, they display QC or GC minimization such as $sqrt8$ and $ex2$. We hypothesize that when a node is shared with more than two nodes then an



(a) OKFDD for the function $\overline{x_1}(x_2 \oplus x_3) \oplus \overline{x_2}x_3$



(b) Previous approach



(c) Our approach

Fig. 5. OKFDD and its reversible circuit from different algorithms.

additional line is required to preserve the function for future use. This compensates the previous removal of lines.

V. Conclusion

In this paper we discuss improvements to the algorithm in [1] addition of a new circuit realization of node structure as well as node ordering for shared nodes. We conclude with the following observations:

- A single output n variable Boolean function can be represented by at most $(2^n - 1)$ nodes in a decision diagram. Thus, a circuit may have a maximum of $3(2^n - 1)$ Toffoli gates if every node requires 3 gates. This can be determined from the fact that we are using a maximum of 3 gates for any node implementation.
- Considering k levels for k variable Boolean function in a graph there are at least k lines in a circuit. Even if there are no additional circuit lines the circuit will consist of minimum number of lines to represent the variables.
- In a worst case a function would require $c + k$ lines for k levels in a graph. c is the number of additional lines added to the circuit. For n nodes in a graph, if each node requires an additional line then $c = n$.

The proposed approach utilizes the advantages of all the decomposition types to optimize the circuit from every dimension. It not only reduces lines but also quantum cost and gate count. If at least a single additional constant line is removed from every level then the lines can be greatly reduced. For future work we would introduce more circuit implementations for KFDD node structures. We are also interested in modifying the sifting algorithm [27] to generate more favorable node structures in OKFDDs and study the trade-offs associated with this. If this approach is combined with negative Toffoli gate implementations it could produce much cheaper circuits in terms of quantum cost and gate count.

Acknowledgment

This research was funded by NSERC (Natural Sciences and Engineering Research Council) of Canada.

References

[1] M. Soeken, R. Wille, and R. Drechsler, "Hierarchical Synthesis of Reversible Circuits using Positive and Negative Davio Decomposition," in *Design and Test Workshop (IDT), 2010 5th International*. IEEE, 2010, pp. 143–148.

[2] R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, 1961.

[3] R. R. Schaller, "Moore's Law: Past, Present and Future," *Spectrum, IEEE*, vol. 34, no. 6, pp. 52–59, 1997.

[4] T. Hey, "Quantum Computing: An Introduction," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 105–112, 1999.

[5] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Reversible Logic Circuit Synthesis," in *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*. ACM, 2002, pp. 353–360.

[6] R. C. Merkle, "Reversible Electronic Logic Using Switches," *Nanotechnology*, vol. 4, no. 1, p. 21, 1993.

[7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge university press, 2010.

[8] K. Fazel, M. Thornton, and J. Rice, "ESOP-based Toffoli Gate Cascade Generation," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. Citeseer, 2007, pp. 206–209.

[9] P. Gupta, A. Agrawal, and N. K. Jha, "An Algorithm for Synthesis of Reversible Logic Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 11, pp. 2317–2330, 2006.

[10] R. Wille and R. Drechsler, "BDD-based Synthesis of Reversible Logic for Large Functions," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 270–275.

[11] D. M. Miller, D. Maslov, and G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis," in *Design Automation Conference, 2003. Proceedings*. IEEE, 2003, pp. 318–323.

[12] M. Saeedi and I. L. Markov, "Synthesis and Optimization of Reversible Circuits — A Survey," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 21, 2013.

[13] M. Arabzadeh, M. Saeedi, and M. S. Zamani, "Rule-based Optimization of Reversible Circuits," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*. IEEE, 2010, pp. 849–854.

[14] K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault Testing for Reversible Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1220–1230, 2004.

[15] R. Wille, D. Große, D. M. Miller, and R. Drechsler, "Equivalence Checking of Reversible Circuits," in *Multiple-Valued Logic, 2009. ISMVL'09. 39th International Symposium on*. IEEE, 2009, pp. 324–330.

[16] R. Wille, M. Soeken, D. M. Miller, and R. Drechsler, "Trading Off Circuit Lines and Gate Costs in the Synthesis of Reversible Logic," *Integration, the VLSI Journal*, vol. 47, no. 2, pp. 284–294, 2014.

[17] D. M. Miller, R. Wille, and R. Drechsler, "Reducing Reversible Circuit Cost by Adding Lines." in *International Symposium on Multi-Valued Logic*, 2010, pp. 217–222.

[18] R. Wille, M. Soeken, and R. Drechsler, "Reducing the Number of Lines in Reversible Circuits," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 647–652.

Function Name	#in	#out	Our approach			KFDD approach [1]						BDD approach [10]					
			L	QC	GC	L	QC	GC	Δ L	Δ QC	Δ GC	L	QC	GC	Δ L	Δ QC	Δ GC
Rd_32	3	2	6	16	8	8	20	12	-2	-4	-4	6	22	10	0	-6	-2
mod10	4	4	12	60	24	14	64	28	-2	-4	-4	13	80	28	-1	-20	-4
one_two_three	3	3	9	35	15	10	37	17	-1	-2	-2	9	44	16	0	-9	-1
plus127	13	13	26	85	41	37	97	53	-11	-12	-12	25	98	54	1	-13	-13
plus63	12	12	24	78	38	34	89	49	-10	-12	-11	23	89	49	1	-11	-11
radd	8	5	17	59	27	21	68	36	-4	-9	-9	28	217	73	-11	-158	-46
rd53	5	3	14	64	32	15	69	37	-1	-5	-5	-	-	-	-	-	-
rd73	7	3	23	107	51	25	115	59	-2	-8	-8	25	217	73	-2	-110	-22
rd84	8	4	30	153	69	33	161	77	-3	-8	-8	34	304	104	-4	-151	-35
root	8	5	51	413	153	52	415	159	-1	-2	-6	45	444	140	6	-31	13
sqr6	6	12	48	304	112	50	315	123	-2	-11	-11	49	486	154	-1	-182	-42
sqrt8	8	4	28	166	62	28	170	66	0	-4	-4	-	-	-	-	-	-
z4	7	4	14	46	22	20	56	32	-6	-10	-10	14	66	30	0	-20	-8
z4ml	7	4	14	46	22	20	56	32	-6	-10	-10	14	66	30	0	-20	-8
add6	12	7	40	183	79	39	184	80	1	-1	-1	54	499	159	-14	-316	-80
adr4	8	5	16	54	26	23	65	37	-7	-11	-11	16	74	34	0	-20	-8
bw	5	28	78	581	237	81	593	249	-3	-12	-12	87	943	307	-9	-362	-70
cm82a	5	3	10	31	15	12	36	20	-2	-5	-5	13	82	30	-3	-51	-15
con1	7	2	15	94	30	17	100	36	-2	-6	-6	-	-	-	-	-	-
cycle	12	12	28	97	49	34	104	56	-6	-7	-7	39	202	78	-11	-105	-29
dc1	4	7	21	141	45	22	146	50	-1	-5	-5	20	160	56	1	-19	-11
inc	7	9	56	442	158	58	447	171	-2	-5	-13	53	579	187	3	-137	-29
xor195	5	1	6	6	6	10	10	10	-4	-4	-4	6	8	8	0	-2	-2
sym9	9	1	30	150	70	28	154	74	2	-4	-4	27	206	62	3	-56	8
ex1	5	1	6	6	6	10	10	10	-4	-4	-4	6	8	8	0	-2	-2
ex2	5	1	11	50	18	11	48	20	0	2	-2	11	73	25	0	-23	-7
max46	9	1	62	664	216	67	684	228	-5	-20	-12	54	598	190	8	66	26
sym10	10	1	38	259	103	40	266	110	-2	-7	-7	32	253	77	6	6	26
life175	9	1	26	159	67	31	168	76	-5	-9	-9	27	204	64	-1	-45	3
9syml	9	1	30	150	70	28	154	74	2	-4	-4	27	206	62	3	-56	8
sao	10	4	74	562	186	73	568	192	1	-6	-6	74	667	211	0	-105	-25
tial	14	8	410	4185	1681	419	4179	1703	-9	6	-22	578	7609	2253	-168	-3424	-572
urf1	9	9	384	4320	1628	390	4372	1672	-6	-52	-44	374	6080	1848	10	-1760	-220
urf2	8	8	206	2276	920	209	2304	948	-3	-28	-28	209	3187	983	-3	-911	-63
wim	4	7	18	93	37	19	95	39	-1	-2	-2	18	107	39	0	-14	-2
mjority	5	1	10	37	17	10	38	18	0	-1	-1	10	41	13	0	-4	4
sym6	6	1	16	76	28	16	77	29	0	-1	-1	14	93	29	2	-17	-1
cordic	23	2	52	264	104	53	264	108	-1	0	-4	-	-	-	-	-	-
cm85a	11	3	35	161	61	35	164	64	0	-3	-3	36	275	87	-1	-114	-26
clip	9	5	66	546	214	69	566	226	-3	-20	-12	66	704	228	0	-158	-14
e64	64	64	195	897	385	195	899	387	0	-2	-2	-	-	-	-	-	-
cps	24	109	619	5677	2305	625	5668	2332	-6	-9	-27	-	-	-	-	-	-

TABLE II. EXPERIMENTAL RESULTS FOR THE ALGORITHM

[19] Y. Pang, S. Wang, Z. He, J. Lin, S. Sultana, and K. Radecka, "Positive Davio-based Synthesis Algorithm for Reversible Logic," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*. IEEE, 2011, pp. 212–218.

[20] D. Maslov and G. W. Dueck, "Reversible Cascades with Minimal Garbage," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 11, pp. 1497–1509, 2004.

[21] T. Toffoli, *Reversible Computing*. Springer, 1980.

[22] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary Gates for Quantum Computation," *Physical Review A*, vol. 52, no. 5, p. 3457, 1995.

[23] "Table with Quantum Cost Calculation," http://webhome.cs.uvic.ca/ dmaslov/definitions.html, accessed: 2015-05-03.

[24] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD-Foundations and Applications*. Springer Science & Business Media, 1998.

[25] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "RevKit: An open source toolkit for the design of reversible circuits," in *Reversible Computation 2011*, ser. Lecture Notes in Computer Science, vol. 7165, 2012, pp. 64–76, RevKit is available at www.revkit.org.

[26] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.

[27] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," in *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Computer Society Press, 1993, pp. 42–47.