Reversible Logic Synthesis via Biconditional Binary Decision Diagrams

Anupam Chattopadhyay*, Alessandro Littarru†, Luca Amarú‡, Pierre-Emmanuel Gaillardon‡, Giovanni De Micheli‡
*School of Computer Engineering, NTU, Singapore (*anupam@ntu.edu.sg*)
†MPSoC Architectures Research Group, RWTH Aachen, Germany (*aleciu@yahoo.it*)
‡Integrated Systems Laboratory, EPFL, Lausanne, Switzerland (*luca.amaru@epfl.ch*)

***Abstract*—Reversible logic synthesis is an emerging research area to aid the circuit implementation for multiple nano-scale technologies with bounded fan-out. Due to the inherent complexity of this problem, several heuristics are proposed in the literature. Among those, reversible logic synthesis using decision diagrams offers an attractive solution due to its scalability and performance. In this paper, we exploit a novel, canonical, Biconditional Binary Decision Diagram (BBDD) for reversible logic synthesis. Using BBDD, for multiple classes of Boolean functions, superior circuit performance is achievable due to its compact representation. We discuss theoretical and experimental studies in comparison with state-of-the-art reversible logic synthesis based on decision diagrams.**

I. Introduction

The study of reversible logic has received significant research attention over the last few decades since it was shown that asymptotic zero power dissipation can be achieved by reversible computation [1]. With diminishing returns in the current semiconductor technology and increasing focus on low-power computing, research on reversible logic synthesis is being pursued with more emphasis than ever. Reversible logic finds a wide range of applications in Quantum computing [2], optical computing [3] and nanotechnologies [4], where unbounded fan out is not supported.

An n-input n-output Boolean function is reversible if it is a bijection, that is, if each input vector can be mapped to a unique output vector. It can be expressed as an n-input, n-output bijection or alternatively, as a Boolean permutation function over the truth value set $\{0, 1, \ldots 2^{n-1}\}$. An *irreversible* Boolean function $f_{irr} : \{0,1\}^n \rightarrow \{0,1\}^m$ with $n \neq m$ can also be made reversible with the help of extra constant-initialized input lines termed as *ancilla*.

A. Reversible Logic Gates

Reversible Boolean logic synthesis is the mapping of a Boolean function to a set of reversible logic gates [9]. The gates are characterized by their implementation cost in quantum technologies, which is dubbed as Quantum Cost (QC). Prominent reversible logic gates are, NOT(A) = $\overline{a}$; CNOT(a, b) = $(a, a \oplus b)$, which can be generalized with Tof_n gate with first $n-1$ variables acting as control lines and Fred(a, b, c) = $(a, \overline{a}b + ac, \overline{a}c + ab)$, which can be generalized with $Fred_n$ gate ($n > 1$), with first $n-2$ variables as control lines. Multiple sets of reversible gates form **universal gate library** for realizing classical Boolean functions, such as, NCT (NOT, CNOT, Toffoli) and NCTSF (NOT, CNOT, Toffoli, SWAP, Fredkin).

B. Reversible Logic Synthesis

Several reversible logic synthesis methods have been proposed in the literature. Quantum computing being one of the prime target technologies for reversible circuit implementations, there have been several methods proposed for synthesizing Quantum logic circuits, i.e, unitary transformations on complex-valued circuits [19]. However, classical Boolean functions do also appear heavily in Quantum algorithms as well as in other applications of reversible logic. The focus of this paper is limited to the synthesis techniques of classical Boolean functions. Such techniques can be grossly classified based on the type of algorithm deployed (e.g., optimal, heuristic) or the type of intermediate representation (e.g., truth-table, decision diagram).

Optimal Methods: Due to the complexity of reversible logic synthesis [5], optimal synthesis methods fail to produce reversible circuits for Boolean functions with large number of variables [7]. Up to 6-variable, depth-limited, optimal reversible circuits could be synthesized using SAT-based approach [8].

Heuristic Methods Heuristic methods for reversible logic synthesis can be classified according to the internal data-structure of the Boolean function. Several methods rely on the Exclusive Sum-of-Product (ESOP) form of the Boolean function [10], while a set of methods use repeated transformations on the Boolean function truth-table for performing reversible logic synthesis [12].

Methods based on Binary Decision Diagram Wille and Drechsler proposed reversible logic synthesis based on Binary Decision Diagrams (BDD) in [20]. The initial results are promising in terms of the large Boolean functions a BDD-based synthesis method could handle. The effects of BDD optimizations on the quality of synthesized reversible circuit are studied in [21]. The general observation was that BDD-based reversible logic synthesis method incurs a large number of ancilla lines compared to the other heuristic methods, while achieving low QC. A technique of isomorphic subgraph matching was presented to BDD-based reversible logic synthesis for reducing the ancilla count in [23]. Recently, Soeken *et al* [18] proposed ancilla-free reversible logic synthesis using BDDs. There, an earlier truth-table-based synthesis approach using Young subgroups is extended to use BDD as the representative structure. The key idea is to utilize characteristic function representation in BDD structures. The technique provides good scalability and reduced QC compared to [6]. However, the QC values still remain significantly higher than BDD-based synthesis techniques, where ancilla is supported [17]. Thus, ancilla-free BDD-based reversible logic synthesis remains scalable with very high QC and on the other hand, earlier BDD-based methods attempt to reduce ancilla without compromising QC. The work presented in this paper adds to the body of research on exploring new decision diagrams for reversible logic synthesis.

C. Motivation and Contribution

BDD-based reversible logic synthesis is clearly advantageous due to its scalability for large Boolean functions. Naturally, this scalability advantage is applicable to the general class of Decision Diagrams [14]. Despite that, little progress has been made w.r.t. the study of diverse types of Decision Diagrams towards their applicability in reversible logic synthesis, although the effect of optimizations and variable ordering in BDD-based synthesis methods have been explored [21], [11].

In this paper, we propose reversible logic synthesis based on another canonical representation of Boolean function, known as Biconditional Binary Decision Diagram (BBDD) [24]. This complements the state-of-the-art in the sense that, almost all BDD-based synthesis techniques can be ported to BBDDs with potential gain in implementation/runtime efficiency.

II. BBDD-based Synthesis Method

The biconditional expansion of a Boolean function is defined as following. Here, x_1 and x_2 are denoted as *Primary Variable (PV)* and *Secondary Variable (SV)* respectively.

$$f(x_1, x_2, \cdots, x_n) = (x_1 \oplus x_2) \cdot f(x_2', x_2, \cdots, x_n) + (x_1 \odot x_2) \cdot f(x_2, x_2, \cdots, x_n) \quad (1)$$

Based on this expansion[1], the BBDD is proposed as a Directed Acyclic Graph (DAG) that is uniquely identified by its root, the set of internal nodes, the set of edges and the 1/0-sink nodes. Each internal node in a BBDD is labeled by two Boolean variables: v, the Primary Variable (PV), and w, the Secondary Variable (SV), and has two outgoing edges labeled $PV \neq SV$ and $PV = SV$. In [24], several reductions of BBDD are shown. Under these reductions and a particular variable ordering, BBDD is canonical. By utilizing the reductions and multiple heuristic optimizations, BBDD manipulation package is able to achieve significantly reduced node count compared to BDD.

Synthesis of decision diagram representation to a reversible circuit essentially depends on two phases. First, the optimization of decision diagrams geared towards efficient reversible circuit generation [21], [11]. Second, node-wise mapping to a set of reversible gates. In this paper, we focus on the node-wise mapping strategies and rely on the default optimization techniques for node-count minimization.

[1] $\odot$ stands for exclusive-nor operator

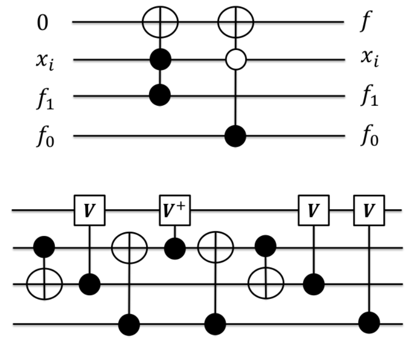

Fig. 1. Background: Mapping of BDD Node w/ Ancilla

A. Mapping of BDD Nodes

For a BDD node, different reversible circuit mapping with elementary 1-QC reversible gates are presented in [20]. It is important to note that, there are multiple possible mappings, in which the inputs are not necessarily preserved. Correspondingly, these result in different ancilla/gate-count/QC.

For a BDD with shared nodes, the reversible circuit realization in Figure 1 is required. This preserves all the initial nodes, while realizing the target function f in a separate ancilla line. Figure 1 shows a circuit based on Tof_n gates as well as a circuit using 1-QC reversible logic gates. The QC of the Toffoli circuit is 10 and the QC of the optimized circuit is 8. In [20], [23], [21], authors reported both the gate count and QC. Hereafter, we report only the QC as the implementations based on Toffoli networks are clearly inferior in terms of QC.

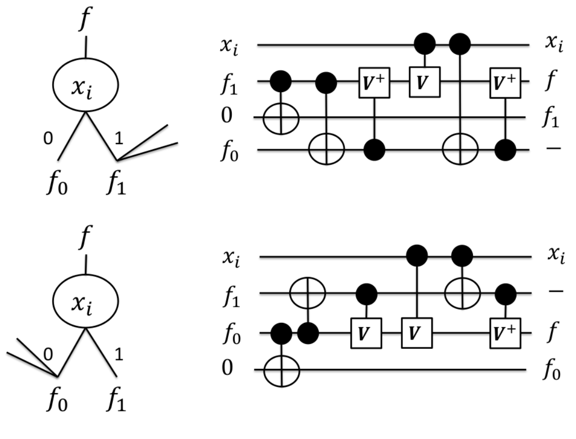

Fig. 2. Reversible Circuit for BDD Node: w/ Ancilla, One node shared

More efficient mapping to reversible circuit can be achieved in case of BDD nodes with children leading to a terminal value. Separate consideration has to be given for mapping BDD nodes with complemented edges. These cases are dealt with in [20], [21]. In Table I, the QC and ancilla count for these cases, as reported in the previous literature, are summarized. In addition, few new scenarios are mentioned in this table, which are not covered earlier. These are shown graphically in Figure 2. It is trivial to extend that with another ancilla line, resulting in a QC of 7 when both the children nodes are shared. The non-terminal values of f_0 and f_1 are indicated by g_0 and/or g_1 respectively.

Value		Shared		f_0 Uncomplemented		f_0 Complemented	
f_0	f_1	f_0	f_1	Ancilla	QC	Ancilla	QC
g_0	g_1	✓	✓	1	8	1	8
		✓	✓	2	7	-	-
		✓	×	1	6	1	7
		×	✓	1	6	1	7
		×	×	0	5	0	6
g_0	g_0	✓	✓	-	-	1	2
		×	×	-	-	0	2
0	g_1	×	✓	1	5	1	5
		×	×	1	5	1	5
1	g_1	×	✓	1	5	1	5
		×	×	1	5	1	5
g_0	0	✓	×	1	5	1	5
		×	×	1	5	1	5
g_0	1	✓	×	1	5	1	5
		×	×	1	5	1	5
0	1	×	×	1	1	-	-
1	0	×	×	1	1	-	-

TABLE I
QC, ANCILLA COUNT FOR MAPPING OF BDD NODES

B. Mapping of BBDD Nodes

On the basis of a range of mapping options for a BDD node, a simple extension is sufficient for mapping of a BBDD node. This is shown graphically in Figure 3. The additional CNOT gates used for this mapping are marked in gray. Essentially, the value of $(x_i \oplus x_j)$ is stored temporarily in the position of x_j, which is recovered at the end. Similarly, corresponding mappings for all variations of BDD node-wise mappings are possible with the introduction of additional 2 QC per node. Note that, both the BDD and BBDD node-wise mappings use the same number of ancilla lines.

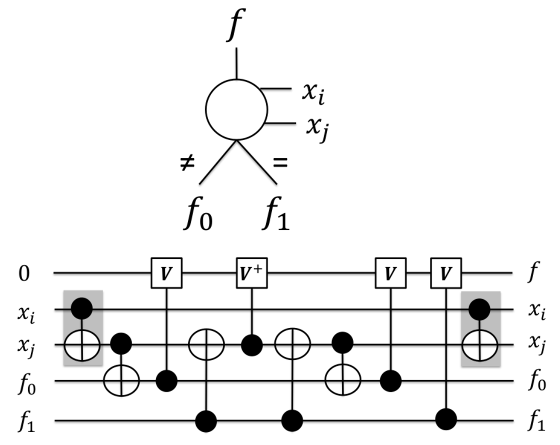

Fig. 3. Reversible Circuit for BBDD Node

Let us define the QC for a BDD node and BBDD as QC_{BDD} and QC_{BBDD} respectively. The ratio of these values denotes node-mapping efficiency. This can be expressed as the node-level Ratio of QC or RQC_{node}.

$$RQC_{node} = \frac{QC_{BDD}}{QC_{BBDD}} = \frac{QC_{BDD}}{QC_{BDD}+2} \quad (2)$$

Considering at least one non-terminal child node, this ratio ranges between $\frac{4}{5}$ and $\frac{5}{7}$, which corresponds to entries from row 1 and row 5 in Table I respectively. We will use RQC_{node} later in section III for deriving analytical performance advantage of BBDD.

C. Optimizations

The two additional CNOT gates introduced for BBDD node mapping can be further optimized with two strategies. *First*, one may schedule the computations of $(x_i \oplus x_j)$ in synchronization with the mapping of BBDD nodes such that, the *uncomputing* of $x_i \oplus x_j$ can be deferred. This will allow saving of 1 QC for the BBDD nodes, which share the same input conditions. *Second*, it is also possible to compute all the conditions for BBDD nodes in advance. This will introduce an *xor-plane* driving the rest of the computation. With this, RQC_{node} becomes 1. However, this introduces ancilla lines and incurs a fixed QC overhead as shown below.

Lemma 1. *An n-variable* ***xor-plane*** *for $X = \{x_1, x_2, \cdots, x_n\}$ computing $Y = \{x_i \oplus x_j, \forall x_i, x_j \in X : i \neq j\}$ requires 1 ancilla/garbage line and a QC of $(n-1)^2$.*

Proof: Consider x_1. This needs to have $\oplus$ operation with $(n-1)$ elements. We create $(n-2)$ new copies of all the variables $\{x_3, \cdots, x_n\}$. These copy operations require $(n-2)$ 1-QC CNOT gates. Another $(n-2)$ CNOT gates with x_1 as control and the newly created lines as target generates $(n-2)$ linear functions. For the remaining function, dealing with x_1, i.e., $(x_1 \oplus x_2)$, x_1 is used as the target line. Hence, x_1 needs $(n-2)$ extra lines and $2(n-2)+1$ QC. This procedure can be repeated till x_{n-2}, which requires 1 additional line and 2 QC. Finally, x_{n-1} requires a QC of 1, due to $x_{n-1} \oplus x_n$ operated directly there. The total number of lines introduced is, therefore, $\sum_{i=1}^{n-2} i$, i.e. $\binom{n-1}{2}$. For each of these lines, a QC of 2 is required. For $n-1$ lines from $\{x_1, \cdots, x_{n-1}\}$, a QC of 1 is required. By summing this up, we get the QC result. Now considering the total number of $\binom{n}{2}$ functions that we need to generate, additional lines for storing the output required is $\binom{n}{2} - n = \frac{n^2-3n}{2}$. Considering the new lines introduced in the procedure, we have $\binom{n-1}{2} - \frac{n^2-3n}{2} = 1$ ancilla/garbage line. ■

The synthesis algorithm proceeds as following. First, the BBDD for a given function is generated. Then, from the root node, a *breadth-first* traversal is done. For each node, the corresponding mapping as outlined in the Table I is chosen with the goal of minimizing overall QC. At each step, sharing of control functions between neighboring nodes within a user-controllable edge distance is explored.

III. ANALYTICAL COMPARISON

In this section, we analytically compare the worst-case QC of a BBDD against that of a BDD. This is done using the following equation

$$RQC_{node} \times \frac{Node_{BDD}}{Node_{BBDD}} \qquad (3)$$

A general result for arbitrary Boolean function is out of scope for the current paper. We focus on the existing worst-case bounds and provide new worst-case bounds for symmetric Boolean functions in the following. For all the comparisons, under the same ancilla value for BDD and BBDD, two different values of RQC_{node} namely, $\frac{4}{5}$ and $\frac{5}{7}$ are considered.

A. Majority and Adder Functions

We present two theorems on BBDD node complexity (for proofs, please refer to [24]). The QC values as computed from Equation 3 are plotted against the increasing variable counts.

Theorem 1. *A BBDD for the majority function of n (odd) variables has $\frac{1}{4}(n^2+7)$ nodes [24].*

The corresponding BDD has $\lceil \frac{n}{2} \rceil (n - \lceil \frac{n}{2} \rceil + 1) + 1$ nodes [25]. Considering n odd, this bound is $\frac{1}{4}(n^2+2n+5)$.

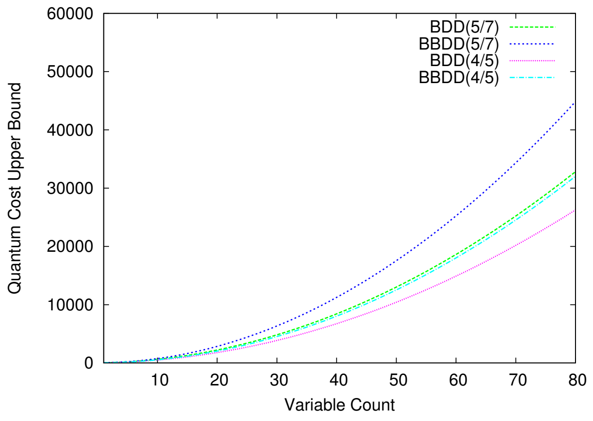

Fig. 4. QC Bound for n-bit Majority, n odd

Theorem 2. *A BBDD for the n-bit binary adder function has $3n+1$ nodes when the variable order $\pi = (a_{n-1}, b_{n-1}, a_{n-2}, b_{n-2}, \cdots, a_0, b_0)$ is imposed[24].*

Corresponding BDD has $5n+2$ nodes [25].

It can be observed from Figures 4 and 5 that, while for adder the BBDD representation provides an improved QC compared to the BDD, for the majority function the BDD representation fares better.

Remark: From the aforementioned analysis, it can be deduced that utilization of xor-plane (lemma 1) is useful only when the node count for BDD and BBDD is in the order of $n^k, k > 2$. In the experiments reported in the current paper, we have not used this lemma.

B. Symmetric Functions

Symmetric functions are an important class of Boolean functions, of which AND, OR, Majority and Threshold (a.k.a. Voting) functions are various subclasses. Symmetric functions are used in arithmetic and cryptographic applications [27].

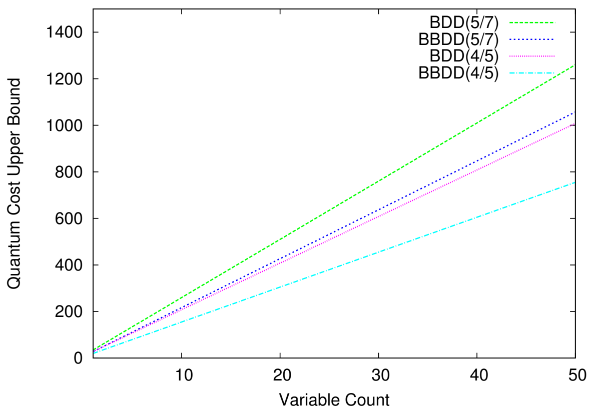

Fig. 5. QC Bound for n-bit Adder

An n-variable Boolean function $f\{x_1, x_2, \cdots, x_n\}$ is called *totally symmetric or symmetric* iff the function output is unchanged for any permutation of the input variables. This property leads to the characterization of a symmetric function only in terms of the Hamming weight of its input variables. In other words, an n-variable symmetric function can be described as a $(n+1)$-bit vector, where index of the bit-vector corresponds to a specific Hamming weight. If that index value is true, then the function evaluates to true for that particular Hamming weight. For example, a 7-input Majority function has corresponding representation as 00001111.

In order to determine the BBDD complexity for symmetric functions, we first introduce the definitions of *Hamming Weight Set* and *Hamming Weight Tree* for a BBDD.

Definition 3. *Every edge of a BDD or BBDD can be characterized with a set of possible Hamming weights, that the target child node can assume. This is called Hamming Weight Set (HWS).*

Definition 4. *A BDD or BBDD, when fully characterized with HWS for each of its edges, is called a Hamming Weight Tree (HWT).*

For an edge e_{ij}, connecting nodes $v_i \rightarrow v_j$, the HWS is denoted as HWS_{ij}. Here we list a few properties for HWS and HWT for BBDDs, without proofs, which are trivial.

- $1 \leq |HWS_{ij}| \leq 2$
- If v_j is a terminal node, $|HWS_{ij}| = 1$
- For continuously connected $\neq$-edges, $|HWS_{ij}| = 1$

A 2-member HWS for e_{ij} is always **ordered** as per the assumed node values of v_i, i.e., HWS(k) stores the value corresponding to $v_i = k, k \in 0, 1$. Note that, the value of the nodes in a HWT are dictated by the incident $=$ or $\neq$-edges.

Since each biconditional decomposition reduces the variable count by 1 and introducing 2 new BBDD nodes, worst-case BBDD node count is $O(2^n)$, which is of the same order of worst-case node complexity for BDD. However, the bi-variate decomposition pattern leads to more compact representation in some cases.

Lemma 5. *If a BBDD node v_i has an outgoing $\neq$-edge with $|HWS_{ij}| = 1$, v_j does not require the PV or SV from v_i for the HWT construction.*

Proof: SV of v_i is typically used as the PV of v_j during BBDD construction. If $|HWS_{ij}| = 1$ then, a $\neq$-edge starting from v_i indicates that the PV and SV of v_i differ by 1. This always increases the Hamming weight by 1. For HWT construction, it is not important to know which of PV and SV assumed which value. Hence, v_j can skip PV and SV. ■

Definition 6. *$HWS_{ij} = HWS_{mn}$ iff $|HWS_{ij}| = |HWS_{mn}|$ and $\forall t, HWS_{ij}(t) = HWS_{mn}(t), t = 0, \cdots, |HWS_{mn}|$*

Lemma 7. *Two BBDD nodes v_1 and v_2 are equivalent in a HWT, iff both the nodes have same PV, SV and $HWS_{i1} = HWS_{j2}$, $i \neq j$.*

Proof: For Hamming weight computation, two nodes are equivalent if those lead to the same sub-tree for Hamming weight computation under different variations of $=$ and $\neq$-edges. If two nodes share the same PV and SV and if they start from the same HWS for the incident edges, the sub-tree would be same. ■

It is evident that two BBDD nodes cannot be equivalent if they share the same immediate parent node.

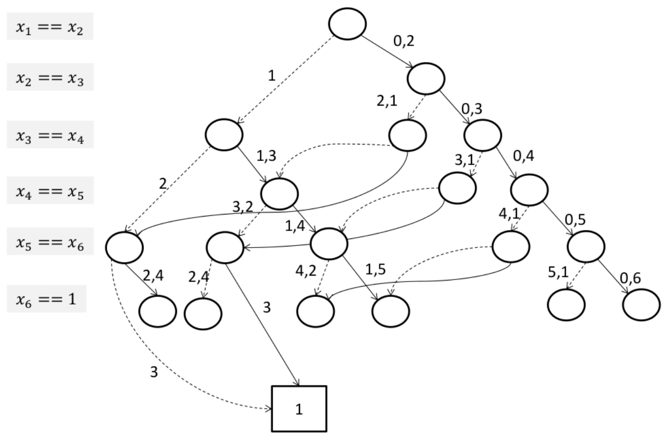

Fig. 6. HWT for Symm(6)

Theorem 8. *An n-variable symmetric function can be represented using BBDD with $\frac{1}{2}(n^2+3)$ nodes, when n is odd and with $\frac{1}{2}(n^2+1)$ nodes, when n is even.*

Proof: We show the proof for n even only. The HWT for a 6-variable symmetric function is shown in the Figure 6, where edges to the terminal node are skipped for simplicity. For the $=$-edges, at most after one child node, it is always possible to find an equivalent BBDD node, that is existent in the tree created by $\neq$-eges. Furthermore, one may skip nodes in the tree created by $\neq$-edges as per Lemma 5. This gives us a linear non-homogeneous recurrence relation as Equation 4.

$$Sym(n) = Sym(n-2) + n + n - 2 \tag{4}$$

where the initial conditions are $Sym(2) = 3$ and $Sym(3) = 6$. Deriving the closed form solution of the above equation proves the theorem. ■

BDD representation of a symmetric function has been studied in [26]. There, a symmetric function is defined as $Symm(n, k)$, where the function assumes the value of 1 iff the number of 1s in the input is exactly $k(k = 0, 1, 2, \cdots, n)$. The node count, considering one sink node is determined to be $\{\frac{(k+1)(k+2)}{2} + 1\}$. For $k = (n-1)$, this is $\frac{1}{2}(n^2+n+2)$.

For Symmetric functions, the relative order of complexity between BDD and BBDD is similar to that of Majority function. Thus, for reversible circuit implementation BBDD does not provide any assured performance edge over BDD, as far as the canonical structures are concerned.

IV. Experimental Results

For our experiments, the BBDD package, available online [22], is extended with optimization and backend flow for mapping to reversible logic circuits. In addition to the basic BBDD package, this constituted roughly 4K lines of C code. The reversible benchmark functions, obtained from RevLib [13], are processed with this flow to generate reversible circuits and associated performance values. Correctness of the generated BBDD circuits are verified via equivalence check. QC calculation follows the cost model from [15] to compare against the results of [20].

The experiments are done on a AMD Phenom™II X6 1055T 2800 MHz processor running a Scientific Linux Release 6.5 with 8 GB RAM. Table II reports the results in terms of QC, lines and runtime. The runtime of the synthesis for different benchmark shows that BBDD-based synthesis scales well for large benchmarks.

The results provide a complete perspective of the efficacy of BBDD against BDD. Out of 26 benchmarks functions studied, 20 reported improved QC and 13 reported improvement in QC as well as line count. As expected from the theoretical observations, for adder functionality, e.g., $mini-alu\_84$, BBDD provides better line count as well as QC. For Symmetric functions ($sym6\_63$, $sym9\_71$), the improvement for BBDD-based flow is more than expected, possibly due to the better heuristics for optimizing such functions in BBDD, compared to BDD. For several benchmark functions, BDDs provide significantly better results. A closer study reveals that these benchmark functions, e.g., $plus63mod4096\_79$ contain major contribution from non-linear sub-circuits, which are represented in more compact form in BDD. That translates to better performance of BDD-based synthesis. Interestingly, for functions computing Hamming distance, hidden-weighted Boolean functions and constant-operand modulo additions, the share of linear component diminishes with increasing variable count. This is also reflected in the performance figures. Note that, the improvement of synthesis results are irrespective of circuit size, e.g., $e64$ reports an improved QC and line count. Nevertheless, for large benchmarks, failure to report general improvements can be partly attributed to the immaturity of the BBDD package w.r.t. BDD.

Benchmark		BDD [20]			BBDD			Improvement (%)	
Name	I/O	Line	QC	Runtime (in seconds)	Line	QC	Runtime (in seconds)	Line	QC
4mod5_8	4/1	7	24	< 0.01	6	10	0.01	14.28	58.33
decod24_10	2/4	6	27	< 0.01	6	23	0.02	0	14.81
mini-alu_84	4/2	10	60	< 0.01	8	42	0.03	20.00	30.00
alu_9	5/1	7	29	0.01	7	25	0.02	0	13.79
rd53_68	5/3	13	98	< 0.01	13	81	0.03	0	17.34
mod5adder_66	6/6	32	292	< 0.01	32	269	0.05	0	7.76
rd73_69	7/3	13	217	< 0.01	15	117	0.04	-15.38	46.08
rd84_70	8/4	34	304	< 0.01	31	256	0.04	8.82	15.79
sym6_63	6/1	14	93	< 0.01	11	49	0.02	21.43	47.31
sym9_71	9/1	27	206	< 0.01	22	124	0.06	18.52	39.81
cycle10_2_61	12/12	39	202	0.09	25	183	0.03	35.89	9.41
cordic	23/2	52	325	0.06	50	222	0.02	3.84	31.69
bw	5/28	87	943	0.11	78	645	0.03	10.35	31.60
apex2	39/3	498	5922	0.24	744	5242	9.30	-49.39	11.48
seq	41/35	1617	19632	1.14	2440	18366	27.78	-50.89	6.45
spla	16/46	489	5925	0.10	788	5315	1.16	-61.15	10.30
ex5p	8/63	206	1843	0.24	251	1682	1.1	-21.85	8.74
e64	65/65	195	907	0.04	192	826	1.14	1.54	8.93
ham7_29	7/7	21	141	< 0.01	18	153	0.03	14.29	-8.51
ham15_30	15/15	45	309	0.25	43	573	0.06	4.44	-85.44
hwb5_13	5/5	28	276	0.01	30	238	0.02	-7.14	13.77
hwb6_14	6/6	46	507	< 0.01	49	488	0.06	-6.52	3.75
hwb7_15	7/7	73	909	< 0.01	102	978	0.12	-39.73	-7.59
hwb8_64	8/8	112	1461	0.01	189	1831	0.35	-68.75	-25.33
plus63mod4096_79	12/12	23	89	0.08	28	186	0.16	-21.74	-108.99
plus127mod8192_78	13/13	25	98	0.21	31	210	0.02	-24.00	-114.28

TABLE II
BENCHMARKING BBDD AGAINST BDD

V. SUMMARY AND OUTLOOK

In this paper, we proposed synthesis of reversible logic circuits via BBDD. The mapping is explored from a theoretical viewpoint, including the analysis of BBDD complexity for various Boolean functions. Clearly, the representation of Boolean functions play a major role in the achievable performance. While determining circuit-specific representation is hard, it is worthwhile to explore synthesis via a hybrid, adaptable data-structure. This is something that we plan to address in future.

REFERENCES

[1] C. H. Bennett, "Logical reversibility of computation," in *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[2] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2000.

[3] R. Cuykendall and D. R. Andersen, "Reversible optical computing circuits," in *Optics Letters*, vol. 12, no. 7, pp. 542–544, 1987.

[4] R. C. Merkle, "Reversible electronic logic using switches," in *Nanotechnology*, vol. 4, pp. 21–40, 1993.

[5] A. Chattopadhyay, C. Chandak and K. Chakraborty, "Complexity Analysis of Reversible Logic Synthesis," in *CoRR abs/1402.0491*, http://arxiv.org/abs/1402.0491, 2014.

[6] Miller, D. M., Thornton, M. A., "QMDD: A decision diagram structure for reversible and quantum circuits," in *International Symposium on Multiple-Valued Logic*, Vol. 36. p. 30., 2006

[7] O. Golubitsky, S. M. Falconer and D. Maslov, "Synthesis of the optimal 4-bit reversible circuits," in *Proceedings of DAC*, pp. 653–656, 2010.

[8] D. Grosse, R. Wille, G. W. Dueck and R. Drechsler, "Exact multiple-control Toffoli network synthesis with SAT techniques," in *IEEE TCAD*,vol. 28, issue 5,May 2009.

[9] A. Barenco *et al.*, "Elementary gates for Quantum Computation," in *Physical Review*, 1995.

[10] Y. Sanaee and G. W. Dueck, "ESOP-Based Toffoli Network Generation with Transformations," in *Proceedings of the ISMVL*, pp. 276–281, 2010.

[11] David Y. Feinstein and Mitchell A. Thornton, "On the Guidance of Reversible Logic Synthesis by Dynamic Variable Reordering," in *Proceedings of ISMVL*, 2009, doi=10.1109/ISMVL.2009.31

[12] D. Maslov, G. W. Dueck and D. M. Miller, "Toffoli network synthesis with templates," in *IEEE TCAD*, vol. 24, issue 6, pp. 807–817, 2005.

[13] R. Wille, D. Grosse, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An Online Resource for Reversible Functions and Reversible Circuits," in *38th International Symposium on Multiple Valued Logic*, pp. 220–225, 2008. RevLib is available at http://www.revlib.org.

[14] R. Drechsler and R. Wille, "Synthesis of Reversible Circuits Using Decision Diagrams," in *Electronic System Design (ISED), 2012 International Symposium on*, pp.1–5, 2012 doi: 10.1109/ISED.2012.37

[15] D. Maslov, "Reversible Benchmarks," *http://webhome.cs.uvic.ca/~dmaslov/definitions.html*, last accessed October, 2014.

[16] M. Soeken, S. Frehse, R. Wille and R. Drechsler, "RevKit: A Toolkit for Reversible Circuit Design," in *Workshop on Reversible Computation*, 2010.

[17] M. Soeken, R. Wille, C. Hilken, N. Przigoda and R. Drechsler, "Synthesis of reversible circuits with minimal lines for large functions," in *Design Automation Conference (ASP-DAC)*, pp.85–92, 2012, doi: 10.1109/ASPDAC.2012.6165069

[18] M. Soeken, L. Tague, G. W. Dueck and R. Drechsler, "Ancilla-free synthesis of large reversible functions using binary decision diagrams," in *CoRR abs/1408.3955*, http://arxiv.org/abs/1408.3955, 2014.

[19] M. Amy, D. Maslov, M. Mosca and M. Roetteler,"A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits," in *IEEE TCAD*, vol. 32, no. 6, pp. 818–830, June 2013, doi: 10.1109/TCAD.2013.2244643

[20] R. Wille and R. Drechsler, "BDD-based Synthesis of Reversible Logic for Large Functions," in *Proceedings of DAC*, pp. 270–275, 2009.

[21] R. Wille and R. Drechsler, "Effect of BDD Optimization on Synthesis of Reversible and Quantum Logic," in *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 6, pp. 57–70, March 2010.

[22] L. Amarú, "BBDD package," *http://lsi.epfl.ch/BBDD*, last accessed October, 2014.

[23] M. Krishna and A. Chattopadhyay, "Efficient Reversible Logic Synthesis via Isomorphic Subgraph Matching," in *Proceedings of the ISMVL*, 2014.

[24] L. Amarú, P.-E. Gaillardon and G. De Micheli, "Biconditional BDD: a novel canonical BDD for logic synthesis targeting XOR-rich circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '13)*, pp. 1014–1017, 2013.

[25] I. Wegener, "Branching Programs and Binary Decision Diagrams: Theory and Applications," SIAM, 2000.

[26] T. Sasao, "Optimization of Multiple-valued AND-EXOR Expressions using Multiple-Place Decision Diagrams," in *Proceedings of ISMVL*, (May 1992), pp. 451–458.

[27] A. Canteaut and M. Videau, "Symmetric Boolean functions," in *IEEE Transactions on Information Theory*, 2004, pp. 2791–2811.