Please note: Methods introduced in this paper are availabe at www.revkit.org.

Equivalence Checking of Reversible Circuits

Robert Wille[1] Daniel Große[1] D. Michael Miller[2] Rolf Drechsler[1]

[1]Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{rwille,grosse,drechsle}@informatik.uni-bremen.de

[2]Department of Computer Science
University of Victoria
Victoria, BC, Canada V8W 3P6
mmiller@uvic.ca

Abstract

Determining the equivalence of reversible circuits designed to meet a common specification is considered. The circuits' primary inputs and outputs must be in pure logic states but the circuits may include elementary quantum gates in addition to reversible logic gates. The specification can include don't-cares arising from constant inputs, garbage outputs, and total or partial don't-cares in the underlying target function. The paper explores well-known techniques from irreversible equivalence checking and how they can be applied in the domain of reversible circuits. Two approaches are considered. The first employs decision diagram techniques and the second uses Boolean satisfiability. Experimental results show that for both methods, circuits with up to 27,000 gates, as well as adders with more than 100 inputs and outputs, are handled in under three minutes with reasonable memory requirements.

1. Introduction

Reversible, including quantum, circuits are constructed as a cascade of gates with no feedback or fanout [12]. These circuits usually realize a reversible function in which a desired irreversible target function has been embedded. The embedding is not unique and most often requires the addition of constant inputs and/or garbage outputs. These inputs and outputs lead to don't-care conditions in the reversible specification. In addition, there can be don't-care conditions in the original target function. The problem we address is whether two given circuits are equivalent with respect to the target functionality regardless of how they behave for the don't-care conditions.

In this paper, we present two approaches to this equivalence checking problem. The common specification can be completely or incompletely specified. The circuits can be composed of reversible gates and elementary quantum gates, and can thus assume multiple internal values, but we constrain the primary inputs and outputs of the circuits to assume pure (non-quantum) logic states.

Our research builds on well-known proof techniques for formal verification of irreversible circuits: decision diagrams and (Boolean) satisfiability. The first approach employs *Quantum Multiple-valued Decision Diagrams* (QMDDs) [8, 9] and involves the manipulation of the unitary matrices describing the circuits and additional matrices specifying the total or partial don't-cares.

The second approach is based on *Boolean satisfiability* (SAT). It is shown that the equivalence checking problem can be transformed to a SAT instance including constant inputs and garbage outputs. Additional constraints are added to deal with total and partial don't-cares.

Experiments on a large set of benchmarks show that both approaches are very efficient. Circuits with up to 27,000 gates, as well as adders with more than 100 inputs and outputs, are handled in less than three minutes with reasonable memory requirements.

For both methods, related work is discussed in the respective sections. Decision diagram approaches for the related problem of equivalence checking for quantum circuits can be found in [15, 20]. But that work does not address the incompletely-specied situation.

The remainder of this paper is structured as follows. Section 2 provides the necessary background. The circuit equivalence problem is defined in Section 3 and the QMDD and SAT based approaches are presented in Sections 4 and 5. Experimental results are given in Section 6 and the paper concludes with suggestions for further research.

2. Background

2.1. Reversible Functions and Circuits

A logic function is reversible if it maps each input assignment to a unique output assignment. Such a function must have the same number of input and output variables

$X := \{x_1, \ldots, x_n\}$. A circuit realizing a reversible function is a cascade of reversible gates.

Definition. *A reversible gate has the form $g(C, T)$, where $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ is the set of control lines, and $T = \{x_{j_1}, \ldots, x_{j_l}\} \subset X$ is the set of target lines. C may be empty. The gate operation is applied to target lines iff all control lines meet the required control conditions. Control lines and unconnected lines always pass through the gate unaltered.*

Common reversible gates include:

- A *multiple control Toffoli gate* (MCT) [14] with target line x_j maps $(x_1, x_2, \ldots, x_j, \ldots, x_n)$ to $(x_1, x_2, \ldots, x_{i_1} x_{i_2} \cdots x_{i_k} \oplus x_j, \ldots, x_n)$.
- A *multiple control Fredkin gate* (MCF) [4] has two target lines x_{j_1} and x_{j_2}. The gate interchanges the values of the target lines iff the conjunction of all control lines evaluates to 1.
- A *Peres gate* (P) [13] has a control line x_i, a target line x_{j_1} and a line x_{j_2} that serves as both a control and a target. It maps $(x_1, x_2, \ldots, x_{j_1}, \ldots, x_{j_2} \ldots, x_n)$ to $(x_1, x_2, \ldots, x_i x_{j_2} \oplus x_{j_1}, \ldots, x_i \oplus x_{j_2}, \ldots, x_n)$.

Quantum logic is inherently reversible [12] and manipulates qubits rather than pure logic values. The state of a qubit for two pure logic states can be expressed as $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|0\rangle$ and $|1\rangle$ denote pure logic states 0 and 1, respectively, and α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$.

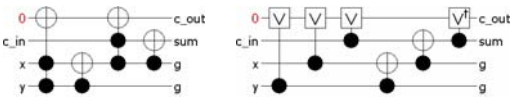
In this work, we consider the following *elementary quantum gates* [12]:

- Inverter (NOT): A single qubit is inverted.
- Controlled-NOT (CNOT): The target qubit is inverted if the control qubit is 1.
- Controlled-V gate: The operation of this gate is explained in Example 2. The V operation is also known as a square root of NOT, since two consecutive V operation are equivalent to an inversion.
- Controlled-V$^+$ gate: The V$^+$ gate performs the inverse operation of the V gate and is also a square root of NOT.

Example 1. *Figure 1 shows a Toffoli realization of a full adder and a second realization using elementary quantum gates.*

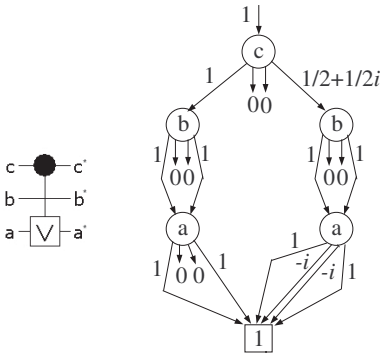
2.2. QMDDs

Quantum Multiple-valued Decision Diagrams (QMDD) [8, 9] provide for the representation and manipulation of



(a) Toffoli (b) Elementary Quantum

Figure 1. Two full adders circuits



(a) V gate (b) QMDD

Figure 2. QMDD for single V gate

$r^n \times r^n$ complex-valued matrices including the unitary matrices required to represent n-line quantum gates and circuits with r pure logic states. The QMDD structure is based on partitioning an $r^n \times r^n$ matrix M into r^2 submatrices, each of dimension $r^{n-1} \times r^{n-1}$ as shown in Equation 1. A formal definition of QMDD can be found in [8, 9]. In the following, we present the concepts by way of the following example of a single V gate.

$$M = \begin{bmatrix} M_0 & M_1 & \cdots & M_{r-1} \\ M_r & M_{r+1} & \cdots & M_{2r-2} \\ \vdots & \vdots & \ddots & \vdots \\ M_{r^2-r} & M_{r^2-r+1} & \cdots & M_{r^2-1} \end{bmatrix} \quad (1)$$

Example 2. *Figure 2(a) shows a V gate in a 3-line circuit. The unitary matrix describing the behaviour of this gate is given in Equation 2 where $v = \frac{1+i}{2}$ and $v^+ = \frac{1-i}{2}$. The QMDD for this matrix is given in Figure 2(b). The edges from each nonterminal vertex point to four submatrices indexed 0,1,2,3 from left to right. Each edge has a complex-valued weight. For clarity, edges with weight 0 are indicated as stubs. In fact, they point to the terminal vertex.*

$$\left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & v & v^+ & 0 & 0 \\ 0 & 0 & 0 & 0 & v^+ & v & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & v & v^+ \\ 0 & 0 & 0 & 0 & 0 & 0 & v^+ & v \end{array}\right] \quad (2)$$

The key features of QMDD are evident in this example. There is a single terminal vertex with value 1 and as noted

each edge has a complex-valued weight. Each nonterminal vertex implements a matrix partitioning. For example, the top vertex in Figure 2(b) implements the partitioning shown in Equation 2. The nonterminal vertices lower in the diagram represent similar partitioning of the resultant submatrices. Representation of common submatrices is shared. To ensure the uniqueness of the representation, edges with weight 0 must point to the terminal vertex and normalization is applied to nonterminal vertices so that the lowest indexed e_j with nonzero weight has weight 1. For more details, we refer the reader to [8, 9].

Since QMDD involve multiple edges from vertices and are applicable to both binary and multiple-valued problems, the QMDD package is not built using a standard decision diagram package, but the implementation employs well-known decision diagram techniques. A recent and very effective enhancement has been to add a computed table which stores the QMDD for recently handled gates. This avoids duplication of effort since the same gates often occur repeatedly in a circuit and quite often in close proximity in the cascade.

2.3. Boolean Satisfiability

The *Boolean satisfiability problem* (SAT problem) is to determine an assignment α to the variables of a Boolean function h such that h evaluates to true or to prove that no such assignment exists. Often, h is given in *Conjunctive Normal Form* (CNF). A CNF consists of a conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable or its negation.

Example 3. *Let* $h = (x_1 + \overline{x}_2 + x_3)(\overline{x}_1 + x_2)(\overline{x}_2 + \overline{x}_3)$. *Then* $x_1 = 1, x_2 = 1$ *and* $x_3 = 0$ *is a satisfying assignment for* h. *The value of* x_1 *ensures that the first clause becomes 1 while* x_2 *ensures this for the second and* x_3 *for the third clause.*

SAT is one of the central $\mathcal{NP}$-complete problems. In fact, it was the first known $\mathcal{NP}$-complete problem as was proven by Cook in 1971 [2]. Despite this proven complexity, there now exist SAT algorithms which solve many practical problem instances, i.e. a SAT instance can consist of hundreds of thousands of variables, millions of clauses, and tens of millions of literals. Most of today's SAT solvers are based on (backtracking) algorithms and use three essential procedures: (1) The decision heuristic assigns values to free variables, (2) the propagation procedure determines implications resulting from the last assignment(s) and (3) the conflict analysis tries to resolve conflicts by backtracking that occurs during the search. Advanced techniques such as *efficient Boolean constraint propagation* [11] and *conflict analysis* [6] are common in state-of-the-art SAT solvers (see e.g. [3]) and lead to their effectiveness.

3. Problem Formulation

The goal of this work is to determine the equivalence of two reversible circuits designed to realize the same target functionality. We assume the two circuits have the same labels for the primary inputs and outputs as the function specification. The problem of matching circuits with different labels has already been considered in [10] and that work can be applied to reversible circuits.

Consider a reversible circuit realizing an irreversible function f. Four types of don't-cares must be considered:

- *Constant input*: An input assigned to a fixed logic value. All outputs are don't-cares for all other assignments to the input.
- *Garbage output*: An output where the value is a don't-care for all input assignments.
- *Total don't-care condition*: The value of *all* outputs of f are unspecified for a given assignment to the inputs.
- *Partial don't-care conditions*: The value(s) of a *proper subset* of the outputs of f are unspecified for a given assignment to the inputs.

A specification with no don't-care conditions is *completely-specified*, otherwise it is *incompletely-specified*. Total and partial don't-cares are inherited from the irreversible function whereas constant input and garbage output don't-cares arise from embedding the irreversible function in a reversible specification.

4. QMDD-based Equivalence Checking

In this section the QMDD-based approach for equivalence checking of reversible circuits is presented. Details on related work are provided at the end of this section.

4.1. The Completely-specified Case

Given a reversible circuit with gates $G_0 G_1 \ldots G_{k-1}$, the matrix describing the circuit is given by $M = M_{k-1} \times \ldots \times M_1 \times M_0$ where M_i is the matrix for gate G_i. The construction of the M_i and the multiplication of matrices is described in [8, 9].

For the completely-specified case, two reversible circuits that realize the same function and adhere to the same variable ordering have the same matrix description. Because of the uniqueness of QMDD, it is sufficient to verify that the top edges of the two QMDD point to the same vertex with the same weight. A traversal of the QMDD is not required. Note that sorting is required when the inputs are not aligned in the two circuits and when the outputs are not aligned. In the latter case, swap gates must be added to the output side of one or both circuits, see [7, 16].

4.2. Constant Inputs, Garbage Outputs

A constant input means the input space is restricted to those assignments containing that value, all others don't occur. This means the circuit's matrix description can be reduced. For clarity we consider $r = 2$, the extension to $r > 2$ is straightforward. Consider the case when the constant input is the top-level partition variable with constant value j. Equation 3 shows the transformation of the input space (denoted γ) to the output space (denoted δ) both partitioned to correspond to the matrix partitioning. Equation 4 shows the matrix and input vector reduced to account for the constant input. Here, ϕ denotes an empty submatrix or subvector of appropriate dimension. For QMDD, an empty submatrix is represented by a null edge pointer. Note the positioning of the j^{th} blocks in Equation 4 in the 0^{th} position is to account for the possibility that the two circuits use different fixed values for the constant input.

$$\begin{bmatrix} \delta_0 \\ \delta_1 \end{bmatrix} = \begin{bmatrix} M_0 & M_1 \\ M_2 & M_3 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} \delta_0 \\ \delta_1 \end{bmatrix} = \begin{bmatrix} M_j & \phi \\ M_{j+2} & \phi \end{bmatrix} \begin{bmatrix} \gamma_j \\ \phi \end{bmatrix} \quad (4)$$

Now suppose the top-level partition variable is a garbage output. In this case, we are interested in the output of the circuit regardless of the value of that variable. Again considering $r = 2$ for clarity, and starting from Equation 3, the matrix can be reduced as shown in Equation 5 where $\hat{\delta}$ denotes the output after removal of the garbage output. To explain the addition of submatrices in Equation 5, recall that we require the circuit inputs and outputs to be in pure logic states, so one element of γ is 1 and the others 0. The same is true for δ. Further, M is a permutation matrix (a special case of a unitary matrix).

$$\begin{bmatrix} \hat{\delta} \\ \phi \end{bmatrix} = \begin{bmatrix} M_0 + M_2 & M_1 + M_3 \\ \phi & \phi \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \end{bmatrix} \quad (5)$$

In general, constant inputs and garbage outputs can correspond to any variables in the circuit's QMDD. This can be handled by performing a depth-first traversal of the QMDD applying the above reductions to each vertex as it is encountered. In a depth-first traversal, the reductions are applied to a vertex's descendants before applying them to the vertex itself. Note that a variable can be both a constant input and a garbage output. The order of applying the two reductions is unimportant. This traversal reduces submatrices as required throughout the full matrix.

Wang et al. [16] have considered the circuit equivalency problem using their XQDD structure. Their approach requires a reordering of the circuit inputs and outputs that can be prohibitive for large functions and particularly so for functions where the decision diagram is subject to exponential growth depending on the variable order (e.g. adders). An earlier version of this work, [7], employed a similar reordering strategy and it is a major contribution here to show that such reordering can be avoided.

4.3. Don't-care Conditions

Let $\hat{M}$ denote the matrix for a circuit after the constant input and garbage output reductions are applied. To deal with total don't-cares in the target function, we construct a diagonal matrix D such that $D_{i,i} = 0$ if the corresponding output position is a total don't-care, and $D_{i,i} = 1$ otherwise. We then compute $\hat{M} \times D$. The effect is to force all total don't-care situations to 0 by ignoring the input states corresponding to don't-care output assignments. This ensures that when the reduced matrices (QMDD) are compared for two circuits, differences can not arise in total don't-care positions. Note that the easiest way to construct a QMDD for D is to start from a diagonal matrix and then use a depth-first traversal to zero the diagonal elements corresponding to total don't-cares.

Partial don't-care conditions can be handled in a similar fashion. The difference is that partial don't-care conditions apply only to a subset and not all outputs. The simplest approach is to treat the outputs for which a set of partial don't-cares does not apply as pseudo-garbage outputs and construct a new matrix for this situation by reducing the pseudo-garbage. A diagonal matrix is then constructed for those don't-cares and the equivalence check proceeds as above. This must be repeated for each subset of the outputs that have shared partial don't-cares.

5. SAT-based Equivalence Checking

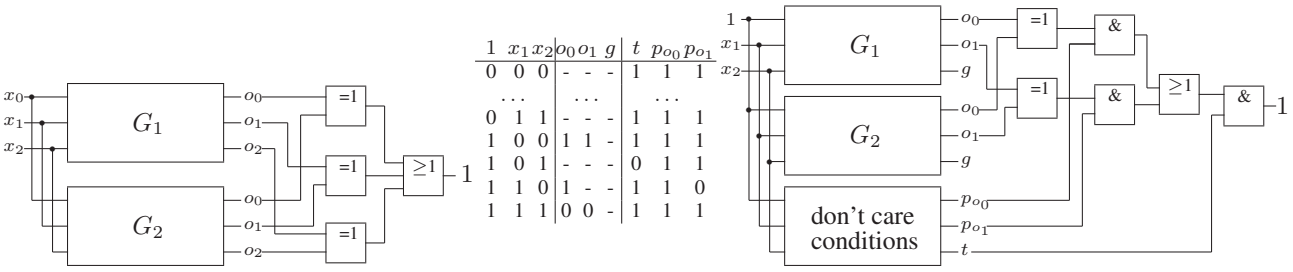
In this section the SAT-based equivalence checker for reversible logic is described. While SAT has been used for synthesis [5, 17] as well as for debugging [18] of reversible circuits, to the best of our knowledge it has not been used for checking the equivalence of reversible or quantum circuits.

The general idea of the SAT-based equivalence checker is to encode the problem as an instance of Boolean satisfiability to be solved by a SAT solver. If the SAT solver returns *unsatisfiable*, then the checked circuits are *equivalent*. Otherwise, a counter-example can be extracted from the satisfying assignment of the instance.

5.1. The Completely-specified Case

To formulate the problem, a so-called *miter structure* as proposed in [1] for traditional (irreversible) circuits is built. By applying the input assignments in turn to both circuits C_1 and C_2, differences at corresponding outputs are observed by XOR operations. If at least one XOR evaluates to 1 (determined by an additional OR operation), the two circuits are not equivalent.

Example 4. *The miter structure for two circuits containing three lines is shown in Figure 3(a). Note that the added XOR and OR operations are only used in formulating circuit equivalence checking as a SAT instance. They are not actually added to the circuits.*



(a) For completely specified functions

(b) For incompletely specified functions

Figure 3. SAT Formulation

In the encoding of this structure into a SAT instance, a new free variable is introduced for each signal while each of the reversible gates and the added XOR and OR operations are represented by a set of clauses. Since V and V^+ gates may produce non-Boolean values, the variables for the associated signals employ a multiple-valued rather than a Boolean encoding. The specific encoding is:

- 00 represents the Boolean value 0,
- 01 represents the non-Boolean value v,
- 10 represents the Boolean value 1, and
- 11 represents the non-Boolean value v^+.

To complete the SAT instance, the output of the OR gate is constrained to value 1. Clearly, a satisfying assignment can be found, iff there exists an input assignment to the circuits where at least one pair of corresponding outputs assumes different values. Such a satisfying assignment is a counter-example to the proposition that the two circuits are equivalent.

5.2. Constant Inputs, Garbage Outputs, and Don't-care Conditions

In order to handle constant inputs, garbage outputs, and total and partial don't-cares, the SAT formulation introduced in the last section is extended as follows:

- *Constant Inputs*: The associated SAT variables are restricted to the appropriate constant values.
- *Garbage Outputs*: Garbage outputs are by definition don't-cares and can be ignored in the SAT miter structure.
- *Total and Partial Don't-care Conditions*: In these cases, new variables t, $p_{o_0}, \ldots, p_{o_{n-1}}$ and additional AND operations are added to the miter structure. The variable t evaluates to 0, iff an input assignment leading to a total don't-care condition is applied. The variable p_{o_i} $(0 \leq i < n)$ evaluates to 0, iff an input assignment leading to a partial don't-care condition at output o_i is applied. The AND operations ensure, that if t (p_{o_i}) is assigned to 0, all outputs (the respective output) are ignored by the miter. Hence, only differences in output values without don't-care conditions are detected.

Example 5. *Figure 3(b) shows the extended miter for two circuits realizing an incompletely-specified function. A truth table showing the garbage output, the don't-care conditions, as well as the resulting values for* t *and* p_i *is given in the left part of Figure 3(b). Note that the first half of the truth table includes don't-cares due to the constant input.*

6. Experimental Results

This section provides experimental results. QMDD V3.1 [9] was used with the default sizes for the computational tables, a garbage collection limit of 250,000 and a maximum of 200 circuit lines. For the SAT-based approach we use the SAT solver *MiniSat 2* [3]. The experiments used an AMD Athlon 3500+ with 1 GB of memory with a timeout of 500 CPU seconds. All benchmarks were taken from RevLib [19].

Table 1 shows the results. We have conducted two kinds of experiments (typically using different gate types): equivalent circuits and non-equivalent circuits (see upper and lower part of the table). The first column gives the name of the circuit. For equivalent circuits, two numbers following the name give the unique identifier of the circuit realizations in RevLib. For non-equivalent circuits, only one number is given which identifies the circuit from the corresponding equivalent test with the larger number of gates. That circuit is used as given in RevLib and in a modified form we constructed by arbitrarily altering, adding or deleting gates. Column *DC* shows the types of don't-cares (see table note *a* for the coding). In column *GT* the gate types used in each circuit are provided (see table note *b* for the coding). Column *n* presents the number of inputs and column *Gates* gives the number of gates for the first and second circuit, respectively.

In the next three columns the data for the QMDD-based approach is shown, i.e. peak number of QMDD nodes, runtime in CPU seconds and memory in MByte. The peak

number of nodes is the maximum number of active nodes at any point in building the circuit QMDDs and checking for equivalence. Finally, the last four columns provide the data for the SAT-based method. First, the number of variables and the number of clauses of the SAT instance are shown. Then, the run-time as well as the required memory are given.

Both approaches prove or disprove the equivalence for all benchmarks (except one for the SAT-based approach) very quickly. The maximum run-time observed was less then three minutes. Several experiments with more than 10,000 gates are included. The largest has nearly 27,000 gates. Even for these cases the proof times are very fast. The largest circuit in terms of the number of inputs and outputs, add-64, with $n = 193$ is a 64 bit ripple carry adder which requires 64 constant inputs and 128 garbage outputs to achieve reversibility. The two versions were constructed by concatenating 64 instances of the circuits in Figures 1.

Comparing the run-times of the two approaches, the QMDD method appears to be faster in the case of equivalent circuits, while the opposite is true in the non-equivalent case. Regarding memory usage it can be seen that the QMDD approach does not blow up even for the large problems presented. Since a fixed number of clauses is generated for each gate, the memory consumption of the SAT approach increases linearly with the number of gates.

7. Conclusions

We have presented two novel approaches for equivalence checking of reversible circuits which have been designed to meet a common specification. The specification can contain don't-care conditions resulting from constant inputs, garbage outputs, total or partial explicit definitions. Both approaches, the QMDD-based and the SAT-based methods, give very good results regarding run-time and memory consumption. For many different benchmarks including very large circuits (in terms of inputs and gates) the results were obtained in less than three CPU minutes.

In future work, we plan to extend the approaches to handle circuits with common target functionality but different reversible specifications. Lastly, we are examining the extension of the approaches by allowing a greater variety of quantum gates and allowing quantum values at the circuits' inputs and outputs.

8. Acknowledgments

This work was supported in part by a research grant from the Natural Sciences and Engineering Council of Canada and by the German Academic Exchange Service (DAAD). The work was undertaken while D. Michael Miller was on sabbatical leave and visiting the University of Bremen.

References

[1] D. Brand. Verification of large synthesized designs. In *Int'l Conf. on CAD*, pages 534–537, 1993.

[2] S. Cook. The complexity of theorem proving procedures. In *3. ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[3] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[4] E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[5] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares. In *Int'l Symp. on Multi-Valued Logic*, pages 214–219, 2008.

[6] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[7] D. M. Miller. Decision diagram techniques for reversible and quantum circuits. In *8th International Boolean Problems Workshop*, pages 1–15, 2008.

[8] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *Proc. Int'l Symp. on Multiple-valued Logic*, 2006.

[9] D. M. Miller and M. A. Thornton. *Multiple-Valued Logic: Concepts and Representations*. Morgan and Claypool, 2008.

[10] J. Mohnke, P. Molitor, and S. Malik. Limits of using signatures for permutation independent Boolean comparison. In *ASP Design Automation Conf.*, pages 459–464, 1995.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

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

[13] A. Peres. Reversible logic and quantum computers. *Phys. Rev. A*, (32):3266–3276, 1985.

[14] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, page 632. Springer, 1980. Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[15] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Checking equivalence of quantum circuits and states. In *Int'l Conf. on CAD*, pages 69–74, 2007.

[16] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo. An xqdd-based verification method for quantum circuits. *IEICE Transactions*, 91-A(2):584–594, 2008.

[17] R. Wille and D. Große. Fast exact Toffoli network synthesis of reversible logic. In *Int'l Conf. on CAD*, pages 60–64, 2007.

[18] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler. Debugging of toffoli networks. In *Design, Automation and Test in Europe*, 2009.

[19] 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*, pages 220–225. RevLib is available at http://www.revlib.org.

[20] S. Yamashita, S. ichi Minato, and D. M. Miller. An efficient verification of quantum circuits under a practical restriction. In *Proc IEEE Int'l Conf. on Computer and Information Technology*, pages 873–879, 2008.

Table 1. Experimental Results

Circuits					QMDD-based			SAT-based			
Name	DC[a]	GT[b]	n	Gates	Nodes	Time	Mem.	Vars	Clses	Time	Mem.
Equivalent Circuits											
0410184 (170,169)	none	NCV / NCT	14	74 / 46	2924	0.01	12.54	2843	4640	0.05	3.70
add16 (175,174)	CG	CV / CT	49	96 / 64	7841	0.02	12.72	12788	18356	0.08	6.31
add32 (185,183)	CG	CV / CT	97	192 / 128	28761	0.03	13.82	50148	70500	0.29	15.14
add64 (186,184)	CG	CV / CT	193	384 / 256	109769	0.12	17.53	198596	276164	1.12	48.10
alu-v0 (26,27)	G	NCT / NCT	5	6 / 6	171	0.01	12.53	72	159	0.00	2.70
alu-v1 (28,29)	G	NCT / NCT	5	7 / 7	188	0.00	12.53	82	180	0.00	2.70
alu-v2 (30,33)	G	NCT / NCT	5	18 / 7	295	0.00	12.53	145	340	0.00	2.70
alu-v3 (34,35)	G	NCT / NCT	5	7 / 7	177	0.00	12.53	83	185	0.00	2.70
alu-v4 (36,37)	G	NCT / NCT	5	7 / 7	223	0.00	12.53	84	189	0.00	2.70
c2 (182,181)	none	NCV / NCT	35	305 / 116	40767	0.08	14.36	26018	40774	0.52	10.67
ckt1 (149,151)	none	T / CT	9	11554 / 1487	250311	5.05	23.91	130419	302871	32.10	55.24
ckt2 (152,154)	none	T / CT	8	5030 / 620	250020	1.09	23.89	50849	119592	3.82	23.86
ckt3 (155,157)	none	T / CT	10	26468 / 2674	250441	22.71	23.91	320578	735809	86.81	134.67
ckt5 (158,159)	none	T / CT	9	10276 / 499	250061	3.32	23.91	107764	249191	19.15	48.18
ckt6 (160,160)	none	T / T	15	10740 / 10740	258140	97.95	24.25	343712	751876	TO	–
cnt3-5 (179,180)	CGT	CT / CT	16	25 / 20	204874	0.64	21.95	2338	21976	0.43	9.56
decod24-v2 (43,44)	CT	NCT / NCV	4	6 / 9	192	0.00	12.52	115	214	0.00	2.70
hwb4 (52,49)	none	CT / CT	4	11 / 17	208	0.00	12.52	133	336	0.01	2.82
hwb5 (55,53)	none	NCT / CT	5	24 / 55	972	0.00	12.52	448	1111	0.02	2.94
hwb6 (56,58)	none	CT / CT	6	126 / 42	3588	0.01	12.53	1146	2846	0.03	3.21
hwb7 (62,60)	none	NCT / F	7	331 / 166	22010	0.04	13.44	3730	9249	0.14	4.49
hwb8 (116,115)	none	NCT / CTP	8	749 / 614	82942	0.26	16.19	11599	27786	1.02	7.42
hwb9 (123,122)	none	NCT / CTP	9	1959 / 1541	250162	1.89	23.88	33454	79798	6.28	16.73
mod10 (171,176)	T	NCT / NCT	4	10 / 7	122	0.00	12.53	97	265	0.00	2.70
mod8-10 (178,177)	GTP	NCT / NCT	5	9 / 14	235	0.00	12.53	161	491	0.01	2.70
rd53 (135,134)	CGT	CT / TP	7	16 / 12	593	0.00	12.53	224	523	0.01	2.83
sym9 (146,147)	CGT	CT / CTP	12	28 / 28	8825	0.02	12.89	727	1582	0.02	3.37
Non-equivalent Circuits											
0410184 (170)	none	NCV / NCV	14	74 / 73	2550	0.00	12.59	4318	6385	0.04	4.16
add16 (175)	CG	CV / NCV	49	96 / 97	9516	0.02	13.22	19270	23732	0.10	7.23
add32 (185)	CG	CV / NCV	97	192 / 193	21892	0.04	14.11	75398	90522	0.38	19.31
add64 (186)	CG	CV / NCV	193	384 / 386	91417	0.15	17.93	298632	353478	1.53	63.91
alu-v0 (26)	G	NCT / NCT	5	6 / 5	116	0.00	12.52	67	148	0.00	2.70
alu-v1 (28)	G	NCT / CT	5	7 / 6	119	0.00	12.52	77	172	0.00	2.70
alu-v2 (30)	G	NCT / NCT	5	18 / 16	346	0.00	12.52	198	480	0.01	2.70
alu-v3 (34)	G	NCT / CT	5	7 / 6	202	0.00	12.52	79	178	0.00	2.70
alu-v4 (36)	G	NCT / CT	5	7 / 6	204	0.01	12.52	81	186	0.00	2.70
c2 (182)	none	NCV / NCV	35	305 / 304	34107	0.15	14.21	43648	64262	0.32	15.84
ckt1 (149)	none	T / CT	9	11554 / 11554	250311	9.53	23.91	231099	531527	4.58	97.09
ckt2 (152)	none	T / T	8	5030 / 5029	229154	2.00	22.97	90549	211280	2.12	39.08
ckt3 (155)	none	T / T	10	26468 / 26464	250441	44.14	23.91	582274	1323351	13.72	243.19
ckt5 (158)	none	T / CT	9	10276 / 10276	250163	6.91	23.90	205539	472739	4.85	84.53
ckt6 (160)	none	T / CT	15	10740 / 10740	260196	144.93	24.50	343711	751873	16.48	143.23
cnt3-5 (179)	CGT	CT / NCT	16	25 / 26	204745	0.63	21.94	2429	22158	0.43	9.74
decod24-v2 (43)	CT	NCT / NCT	4	6 / 5	89	0.00	12.52	60	143	0.00	2.70
hwb (4)	none	CT / NCT	4	11 / 18	216	0.00	12.52	137	344	0.01	2.83
hwb5 (55)	none	NCT / CT	5	24 / 54	961	0.00	12.53	442	1094	0.01	2.94
hwb6 (56)	none	CT / CT	6	126 / 125	3922	0.01	12.53	1733	4357	0.03	3.50
hwb7 (62)	none	NCT / NCT	7	331 / 331	22541	0.06	13.45	4865	11553	0.08	4.94
hwb8 (116)	none	NCT / NCT	8	749 / 750	61990	0.31	15.28	12438	28977	0.16	8.05
hwb9 (123)	none	NCT / NCT	9	1959 / 1958	250162	2.55	23.91	36244	83568	0.57	17.39
mod (10)	T	NCT / NCT	4	10 / 10	107	0.00	12.53	109	296	0.00	2.70
mod8 (10)	GTP	NCT / NCT	5	9 / 15	269	0.00	12.52	166	501	0.01	2.70
rd53 (135)	CGT	CT / CT	7	16 / 16	611	0.00	12.54	252	582	0.01	2.82
sym9 (146)	CGT	CT / CT	12	28 / 27	10045	0.02	12.93	714	1553	0.01	3.36

[a] Don't-care: none = completely-specified, C = constant input, G = garbage output, T = total don't-care, P = partial don't-care
[b] Gate type: N = NOT, C = controlled-NOT, F = multiple control Fredkin, P = Peres, T = multiple control Toffoli, V = V or V+