Abstract

Reversible circuits arise in bottlenecks of quantum algorithms, such as Shor’s and Grover’s. More generally, they offer a way to embed an arbitrary conventional computation into the quantum domain, where it can be performed on multiple input combinations at once. Performance optimizations modify reversible circuits for speed, depth and to abide by physical constraints, e.g., of spin-chain architectures. To verify such optimizations, we study equivalence-checking of reversible circuits and first observe that generic quantum techniques are not competitive in this case. After adapting the well-known SAT and BDD techniques from digital logic, we show how to exploit reversibility to speed up equivalence checking.

1. Introduction

Quantum circuits are considered among promising emerging technologies beyond CMOS. In the Winter 2008, EE Times mentioned quantum information processing on its list of “35 people, places and things that will shape the future” [14]. In the Spring 2008, quantum-optical satellite-to-Earth communications have been demonstrated [18] (sophisticated communications require circuits). Slow but steady progress in quantum devices has been made: computing devices with 16 and 28 superconducting qubits have been constructed [7].

Design automation techniques for quantum circuits exhibit several trends familiar from traditional EDA applications. Most relevant to this work are (i) the use of gate libraries, (ii) the classification of synthesis algorithms into logic decompositions (constructive) [12] and logic transformations (iterative improvement) [11, 4], (iii) circuit restructuring for performance optimizations and to accommodate physical constraints [5, 6], as well as (iv) important interactions between synthesis and verification. In particular, to verify the results of circuit synthesis and performance optimizations, one uses equivalence-checking, i.e., techniques to prove or disprove that for any input configuration two given circuits produce the same (or similar) output results.

Recent work on quantum equivalence-checking [16] points out that quantum circuits support several different notions of equivalence which can be checked by appropriate algorithms. However, generic operations on quantum circuits are resource-intensive, and our work focuses on components of quantum circuits that can be treated by non-quantum techniques, namely reversible circuits. These components typically represent conventional computations, e.g., arithmetics, embedded into the quantum domain so that each operation can be performed on multiple data at once. Aside from supporting quantum parallelism when applied to quantum states, reversible circuits can be viewed as conventional circuits, except that they avoid fanouts and never erase information. In particular, they support a single notion of equivalence.

In this work we point out that equivalence-checking methods developed for generic quantum circuits are too heavy-weight for reversible circuits and are outperformed by conventional techniques based on Boolean Satisfiability (SAT) and Binary Decision Diagrams (BDD) adapted to this application. Therefore, the possibility arises of developing specialized equivalence-checking techniques for reversible circuits. In a similar development, recent literature on ATPG [9, 10] shows that reversible circuits support very small sets of test patterns, which may be easy to find.

We develop several approaches to check equivalence of reversible circuits, including those based on SAT and ROBDDs, then compare them to state of the art in quantum equivalence-checking. To exploit reversibility, we develop reversible miters that are analogous to conventional miters and can be used in conjunction with conventional SAT and BDD techniques. Reversible miters can be constructed in several ways and reduce the complexity of equivalence-checking in some cases when used with existing techniques for iterative circuit simplification [3, 11, 4].
2. Reversible Circuits

Measurement-free quantum computation is reversible in nature, thus quantum circuits are reversible in the sense that (i) they map their input configurations to output configurations one-to-one, (ii) this property is also observed locally for every gate and subcircuit [8]. In order to embed conventional computation into the quantum domain, it must be made reversible, and standard procedures exist for such transformations. The resulting circuits do not use quantum properties, except that they can be applied to quantum data (superposition states), allowing them to perform conventional computation on many inputs at once. Leveraging this quantum parallelism in useful applications is far from easy, but can be illustrated by Shor’s algorithm for number-factoring in polynomial time [13]. This algorithm is dominated by a reversible module that performs modular exponentiation, applied after the Quantum Fourier Transform.

A reversible circuit consists of numbered bit-lines and a sequence of reversible gates, each applied to certain bits, e.g., the circuit in Fig. 1 includes three bit-lines and three gates. Note that the gate library used must be universal, i.e., able to express any reversible transformation by combining multiple copies of gates involved. In this work we explicitly consider only the most common gate library for reversible computation, specified below, but our techniques can be easily adapted to other universal gate libraries.

**NOT gate** maps \( a \mapsto a' \).

**Controlled-NOT (CNOT) gate** maps \( (a, b) \mapsto (a, a \oplus b) \).

**Toffoli gate** maps \( (a, b, c) \mapsto (a, b, ab \oplus c) \).

In these gates we distinguish a single target bit whose value may change, and control bits which remain constant but may influence the target bit. As illustrated in Fig. 1, control bits are marked with \( \bullet \), and target bits are marked with \( \oplus \).

We now consider a circuit with \( n \) bit-lines and primary inputs. The logic function (with respect to \( x_1, x_2, \ldots, x_n \)) that is realized on the \( i \)-th bit-line right after the \( j \)-th gate is denoted by \( f_j^i(x_1, x_2, \ldots, x_n) \). For example, the \( i \)-th primary input function of a circuit is denoted by \( f_1^0(x_1, x_2, \ldots, x_n) \) which is actually \( x_i \). Assuming \( k \) gates in the circuit, we denote the \( i \)-th primary output function by \( f_k^i(x_1, x_2, \ldots, x_n) \).

As the functionality of our gates does not involve quantum phenomena, we can calculate the logic functions realized at each point in a circuit by a straightforward way as the case of conventional circuits. (A concrete calculation method is explained in Appendix.)

Equivalence-checking of reversible circuits can be formally stated as follows (cf. [16]).

**Fact 1** If \( C_1 \) and \( C_2 \) are reversible circuits, then the concatenation of \( C_1 \) and \( C_2 \) is also a reversible circuit, which we denote \( C_1 \cdot C_2 \).

Not only reversible circuits implement reversible transformations, but they can also be reversed themselves.

**Fact 2** Given a reversible circuit \( C \) using NOT, CNOT and Toffoli gates, its copy where all gates reversed implements the inverse transformation to what \( C \) implements. We therefore denote it by \( C^{-1} \).

In other words, \( C \cdot C^{-1} \) is a reversible circuit which consists of \( 2k \) gates, but “does nothing,” i.e., \( f_i^{2k} = x_i \) for all \( i \) (\( 1 \leq i \leq n \)).

For example, for a circuit, \( C \), as shown in Fig. 1, \( C \cdot C^{-1} \) is as shown in Fig. 2. As mentioned above, this circuit, \( C \cdot C^{-1} \), is indeed equivalent to an empty circuit which can also be confirmed by applying the following transformations: As shown in Fig. 3, two same adjacent reversible gates can be canceled out. Thus, in Fig. 2, the third and the fourth gates can be removed at first. Then, the second and the fifth gates, and then finally the first and the last gates are removed without changing the functionality. This observation leads to our new notion of “reversible miters” in the next section.

3. Verification of Reversible Circuits

Fig. 4 illustrates the construction of a conventional miter, which can also be applied to reversible circuits. In order
to more fully exploit reversible properties, we introduce the notion of a **reversible miter**, discuss its advantages and outline several equivalence-checking methods that use reversible miters.

### 3.1. Reversible Miter Circuits

**Definition 1** Given two reversible circuits $C_1$ and $C_2$ with $n$ inputs, their reversible miter is defined to be one of the following four circuits: $C_1 \cdot C_2^{-1}$, $C_2^{-1} \cdot C_1$, $C_2 \cdot C_1^{-1}$, $C_1^{-1} \cdot C_2$.

$C_1$ and $C_2$ are functionally equivalent if and only if all of their miters implement the identity transformation. In particular, if one miter implements identity, then so do the remaining miters. Moreover, if $C_1 = C_2$, then a straightforward circuit simplification (a series of transformations [3, 11, 4]) cancels out all gates, resulting in an empty circuit. Some of the four variant miters may lead to more cancellations than others. For example, if $C_1$ and $C_2$ differ only in their first segments, then $C_2 \cdot C_1^{-1}$ leads to many gate cancellations. In practice, of the four possible miters, two pairs facilitate very similar gate cancellations, as illustrated by empirical data as will be shown in Tables 5.

### 3.2. Formal Verification Techniques

To check the equivalence between $C_1$ and $C_2$, one can pursue two strategies.

1. Checking that the conventional miter or one of the reversible miters implements the identity transformation.
2. Representing the transformations performed by $C_1$ and $C_2$ in a canonical form which supports efficient equivalence-checking.

The latter strategy may use binary-decision diagrams (BDDs), such as ROBDDs [19] and QuIDDs [17]. The former can be implemented with either decision diagrams or Boolean Satisfiability solvers [20] by reducing Circuit-SAT to CNF-SAT. In particular, for conventional miters one needs to check that all output functions implement the constant 0 function, whereas for reversible miters one checks that each output bit is copy of a respective input bit.

We now distinguish possible approaches by the type of computational engine used.

**ROBDD** Calculate the output functions of conventional or reversible miter circuits, using ROBDD as the primary data structure.

**QuIDD** Use an equivalence-checking tool for quantum circuits since reversible circuits are a class of quantum circuits. In particular, QuIDDPro [17, 16] can build the QuIDD data structures for given circuits $C_1$ and $C_2$, and then check if the results are identical. This approach bears some resemblance to the use of ROBDDs, except that conventional miters are not reversible and cannot be used with QuIDDs. Additionally, QuIDDs support arbitrary quantum circuits, while ROBDDs do not.

**SAT** Construct a CNF-SAT formula that is satisfied only by those input combinations for which the corresponding outputs of two circuits produce different values, and then use a contemporary SAT solver [19] to check whether the formula is satisfied or not. How to construct such a formula is explained below.

For a SAT solver, we construct a CNF formula as follows. First we add a set of clauses for each gate in the miter circuit. The clauses should be satisfied only with the variable assignments that are consistent with the reversible gate. The readers familiar with SAT-based equivalence-checking can think of a CNOT gate as an XOR gate with a bypass wire, and of a Toffoli gate as an XOR, AND and a bypass. More efficient clause generation is illustrated below for a Toffoli gate whose control bits are $x_1$ and $x_2$, and target bit is $x_3$. Since the Toffoli gate does not modify two of its inputs, there is no need for separate output variables. We introduce only one new variable $y_1$ for the target bit. Then logical consistency is given by the condition $y_1 = (x_1 \cdot x_2) \oplus x_3$ which can be expressed by the following six clauses.

- **Case** $x_1 = 0$ or $x_2 = 0$.
  - Clauses: $(x_1 + x_3 + y_1) \cdot (x_1 + x_3 + y_1) \cdot (x_2 + x_3 + y_1) \cdot (x_2 + x_3 + y_1)$.

- **Case** $x_1 = x_2 = 1$.
  - Clauses: $(x_1 + x_2 + x_3 + y_1) \cdot (x_1 + x_2 + x_3 + y_1)$.

In the next step, we add a set of clauses that are satisfied only by those variable combinations where some circuit output differs from the respective circuit input.
Here we can reuse some of the \( y \) variables introduced earlier. Let such a new variable corresponding to the \( i \)-th primary output be \( y_{O_i} \). (If there is no target bit on the \( i \)-th bit-line, we do not introduce a new variable for the \( i \)-th primary output, i.e., it is obvious that the input and the output functions on the \( i \)-th bit-line are the same, and thus we do not add the following clauses.) We introduce a new variable \( z_i \) to express the functional consistency of the \( i \) bit-line. Namely, we consider that \( z_i \) becomes 1 only when \( x_i \neq y_{O_i} \). For this condition, we add the following clauses.

- Case \( z_i = 0 \). Clauses: \((z_i + x_i + \overline{y_{O_i}}) \cdot (z_i + x_i + y_{O_i})\).
- Case \( z_i = 1 \). Clauses: \((\overline{x_i} + x_i + y_{O_i}) \cdot (\overline{x_i} + x_i + y_{O_i})\).

Finally we add \((z_1 + z_2 + \cdots + z_n)\) where \( n \) is the number of bit-lines of the circuits. Since \( z_i = 1 \) means that the input and the output functions on the \( i \)-th bit-line are different, the two circuits are different when \((z_1 + z_2 + \cdots + z_n)\) is satisfied. Therefore, the above construction generates a SAT formula that is satisfied only by those input combinations for which the corresponding outputs of two circuits produce different values.

A CNF-SAT formula constructed for a miter grows linearly with the size of the miter. A key advantage of reversible miters is that they can be significantly smaller, due to gate cancellations and other circuit simplifications, which we will explain in the next section.

### 3.3 Benefits of Reversible Miters

During equivalence-checking of large circuits, ROBDDs often run out of memory and SAT techniques may take very long time. Methods have been proposed to reduce problem sizes without changing the result of verification, but they require multiple SAT checks or multiple decision diagrams.

Reversible miters offer an elegant alternative and accelerate equivalence-checking by automatically exploiting similarities in the given circuits. This can be accomplished by two distinct mechanisms.

#### 3.3.1 Local Reduction of Reversible Miters

When two conventional circuits end with identical gate sequences, one cannot cancel out these sequences because of observability don’t-cares introduced by them. However, reversible circuits do not experience don’t-cares and identical suffixes do cancel out. Note that a reversible miter \( C_1 \cdot C_2 \) places the last gate of \( C_1 \) next to the last gate of \( C_2 \). If these two gates cancel out as in Fig. 3, the second-to-last gates from \( C_1 \) and \( C_2 \) become adjacent, etc. Thus, no search is required to identify these gate cancellations, and they can be performed one at a time. This is a special case of much more general local reductions discussed in [3, 11, 4]. If \( C_1 \) and \( C_2 \) are identical, an empty circuit will result, but this outcome is also possible when local reductions can prove equivalence of two structurally different circuits. A systematic procedure for applying reductions in the general case was first introduced in [3]. As later shown in [11, 4], local reductions in reversible circuits are particularly easy to perform, take little runtime and do not consume much memory.

As we show later, even the simplest reduction rules can dramatically simplify reversible miters. More sophisticated reduction techniques from [3, 11, 4] should provide an additional boost to reversible equivalence-checking.

#### 3.3.2 Automatic Reduction of Canonical Forms

Unfortunately, iterative circuit simplification is not guaranteed to reduce \( C_1 \cdot C_2^{-1} \) to the empty circuit in polynomial number of steps when some such reductions are possible. Even finding a sequence of reductions can take long time.

However, when constructing a canonical forms (ROBDDs or QuIDDs) of reversible miters, a different kind of reduction may occur. Suppose that \( C_1 \) and \( C_2 \) end with functionally-equivalent but structurally distinct suffixes that do not lead to any simplification by local reductions — an example of such circuits is given in [11]. In other words \( C_1 = A_1 \cdot B_1 \) and \( C_2 = A_2 \cdot B_2 \) where \( B_1 \approx B_2 \). Then \( C_1 \cdot C_2^{-1} = A_1 \cdot B_1 \cdot B_2^{-1} \cdot A_2^{-1} \approx A_1 \cdot A_2^{-1} \).

As we traverse the miter \( C_1 \cdot C_2^{-1} \) and add one gate at a time to the decision diagram, the size of the intermediate decision diagrams is only determined by the transformation implemented by the current circuit prefix, i.e., the functions of the intermediate output wires. In particular, the intermediate decision diagram for \( A_1 \cdot B_1 \cdot B_2^{-1} \) can be smaller than that for \( A_1 \cdot B_1 \) because the transformation implemented is the same as for \( A_1 \). This phenomenon is observed empirically in Sec. 4.

As we have four possible miters, some may be easier to construct than others, e.g., because the ROBDD for \( C_1 \) may be smaller than that for \( C_1^{-1} \), or because \( C_1 \) and \( C_2 \) have large functionally-equivalent prefixes rather than suffixes. In such cases, running four parallel threads to evaluate the four miters may help, and these threads may even be able to cooperate by sharing and reusing intermediate results.

### 4. Case Studies

We perform an empirical study on equivalence-checking of various reversible circuits and report collected results.

#### 4.1 Verifying CNOT Circuits and Adders

Several promising implementations of quantum computation rely on the linear-nearest-neighbor (LNN) architecture, also called the spin-chain architecture in the physics
literature. It arranges quantum bits (qubits) on a line, and allows only neighboring qubits to interact directly. Thus, we need to modify standard quantum circuits to use only LNN gates. Specific transformations and LNN circuits have been developed [2, 12, 6], and the overhead of the LNN architecture in terms of the number of gates is typically limited by a small factor (3-5). Such physical optimization motivates the need for equivalence-checking against original, non-LNN versions. In the first round of experiments we work with the following two cases.

**LNN CNOTs** implement CNOT with target and control bits \( n \) bits apart. We consider two different implementations. The one illustrated in Fig. 5 (a) is built from bit-line swaps (each implemented as three CNOT gates), with subsequent cancellation of two adjacent CNOT gates. A more compact circuit illustrated in Fig. 5 (b) is based on the idea of [12, Fig. 3]. We verify the equivalence between these two circuits.

**Reversible Ripple-carry Adders** include the reversible ripple-carry adder circuit proposed in [1] (See Fig. 6 for \( n = 4 \)) and its LNN version (See Fig. 7 for \( n = 4 \)) obtained by adding several SWAP gates.

For the above two types of circuits, we studied the following three cases.

**Success** Consider two equivalent circuits.

**Failure 1** Add randomly Toffoli gates at the end of each of the two circuits, and then check equivalence.

**Failure 2** Add randomly Toffoli gates at the beginning of each of the two circuits, and then check equivalence.

Thus, we performed six sets of experiments, i.e., two types (CNOT or Adder) and three cases (Success, Failure 1 and Failure 2). User time on a Linux system (CPU: Intel(R) Xeon(TM) 2.40 GHz, Memory:1 Gbyte) is reported for various \( n \) in Tables 1 and 2.

For SAT, we used MiniSAT [20]. For QuIDD, we used QuIDDPro [17]. For BDD, we used CUDD [19]. For SAT and QuIDDPro, we first prepare relevant files and then run these tools. I/O time is not reported because (a) it was very similar for the cases we compared, (b) it was not significant in general, (c) I/O can be avoided if the tools in question are integrated using software libraries.

### 4.2 Verifying LNN Permutation Circuits

In quantum circuits on an LNN architecture, if we want to perform an operation, we always need to move together all qubits necessary for the operation. To do so, wire permutations are performed, e.g., by swapping adjacent wires in many steps (each implemented as three CNOTs). For example, in the LNN circuit for Shor’s algorithm [2], components called Mesh are used to swap many qubits in parallel as follows: A mesh component takes two \( n \)-bit inputs, \( a[n-1], a[n-2], \ldots, a[1], a[0] \) and \( b[n], b[n-1], \ldots, b[1], b[0] \) in this order, and arranges them into the interleaved order: \( a[n-1], b[n-1], a[n-2], b[n-2], \ldots, a[1], b[1], a[0], b[0] \).

As seen in Fig. 8 (\( n = 7 \)), fairly large circuits may be required for this operation.

Since bit-permutations are common in LNN quantum circuits, we use them as a benchmark. In practice bit-permutations are intermixed with other circuits, e.g., arithmetics, but we expect to see the same trends.

We empirically compare BDD and SAT techniques in three cases — Success, Failure 1 and Failure 2, — as the previous section. The results are shown in Table 3. The experiments indicate that BDD techniques outperform SAT

<table>
<thead>
<tr>
<th>Case</th>
<th>( n )</th>
<th>SAT</th>
<th>QuIDD</th>
<th>BDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>32</td>
<td>0.08</td>
<td>2.92</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.54</td>
<td>14.12</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>0.73</td>
<td>78.39</td>
<td>0.23</td>
</tr>
<tr>
<td>Failure 1</td>
<td>32</td>
<td>0.74</td>
<td>6.94</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>6.75</td>
<td>44.57</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>33.97</td>
<td>341.97</td>
<td>1.95</td>
</tr>
<tr>
<td>Failure 2</td>
<td>32</td>
<td>0.45</td>
<td>9.52</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>3.24</td>
<td>72.16</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>13.48</td>
<td>693.70</td>
<td>2.03</td>
</tr>
</tbody>
</table>
for this type of circuits. BDD techniques do so well because a Mesh circuit performs a bit-line permutation, thus each bit-line carries one of the initial inputs — a very simple function. The reason SAT does worse than BDD is that most SAT algorithms are sensitive to presence of XOR constraints, e.g., those generated for CNOT gates.

### 4.3 Verifying Multipliers

One of the best known quantum algorithms is Shor’s number-factoring [13]. Its largest and important portion performs modular exponentiation, for which a number of quantum circuits have been proposed [2]. While some of them do not use multiplication directly, the functionality of the modular exponentiation is similar to that of multiplication. Thus, we consider reversible multipliers as proxies for modular multiplication and use them in our experiments, keeping in mind that multiplication is difficult for conventional equivalence-checking.

For experiments with multipliers, we again consider three cases — Success, Failure 1 and Failure 2. The results are shown in Tables 4. The reversible multiplier we implemented uses 5n bits, for two n-bit numbers, including 2n bits for its two inputs, 2n bits for the results, and n bits for auxiliary bits. Thus, for example, the line n = 6 in the tables deals with 30-bit circuits. Not surprisingly, all methods we tried timed out for n = 8, requiring more than 1,000 seconds.

#### 4.4 Utility of Reversible Miters

We compare the following reversible miter sets: $C_1^{-1} \cdot C_2$, $C_1 \cdot C_2^{-1}$, $C_2^{-1} \cdot C_1$ and $C_2 \cdot C_1^{-1}$. Results for hard instances based on multipliers are shown in Table 5. In the table, “w. simp.” means that we simplify the miter circuits by local reductions as mentioned in Sec. 3.3 before invoking the BDD-based verification. “w/o simp.” means that we do not perform such a simplification.

“Success 1” means that $C_1$ and $C_2$ are exactly the same, and so local reductions produce an empty circuit without invoking SAT or BDD.

“Success 2” means that $C_1$ and $C_2$ share the same functionality, but differ in circuit structure mainly in their last segments, as explained below. While simple local reduction considerably simplifies miters, it cannot produce empty circuits.

To create mutant circuits that cannot be simplified by local reduction we applied the following procedure. From the outputs of one circuit, we look for pairs of gates which can be swapped by one of the transformation rules from [3] illustrated in Fig. 10, and transform one tenth of the circuit. (We change the circuit from the left to the right in Fig. 10.) To put such two gates adjacently, we may need to move other gates (between the target two gates) in some cases. In these cases, we tried to apply a simple swap of gates (if the target bit of one gate is not one of the control bits of an-
other gate, the two gates can be simply swapped as shown in Fig. 9) as much as possible.

“Different” means that we add Toffoli gates randomly at the end of two circuits so that the functionality becomes different. (The number of added Toffoli gates is again 10 % of the total gates in the circuit.)

Since QulDDPro uses BDDs internally, we repeated with QulDDPro our experiment using multipliers without simplification. Results are shown in Table 6. Reversible miters do not achieve as great impact as in the case of BDDs, but we observe a similar trend.

4.5 Observations

We observed the following trends in experimental data.

- SAT and BDD-based techniques can handle large adder circuits and LNN-CNOT circuits very efficiently.

- QulDDs do not scale as well.

- Reversible multipliers are very difficult to verify, as expected.

- BDD is generally better than SAT, especially when SAT instances are unsatisfiable. LNN-permutation circuits offer an extreme case of this trend.

- Local reductions accelerate verification based on reversible miters.

To explain these trends, we point out that CNOT and Toffoli gates used in reversible circuits include the XOR functionality, which SAT solvers are known to handle poorly. On the other hand, QulDDs are designed to handle quantum correlations (so called entanglement) between multiple qubit lines. While (classical) reversible circuits do not generate such correlations, QulDDs are apparently unable to simplify their internal structure to reflect this circumstance.

5. Conclusion and Future Work

In this paper, we have discussed several techniques for equivalence-checking of reversible circuits, including the new concept of reversible miters. In particular, we observed that generic quantum techniques are not competitive with conventional equivalence-checking techniques based on BDDs and SAT, whereas BDD-based techniques adapted to reversible circuits usually outperform SAT-based techniques. As is the case with ATPG, reversibility can significantly simplify equivalence-checking.

Our preliminary experiments suggest that reversible miters can be very useful for the verification of reversible circuits. As we discussed in Sec. 3.3, reversible miters may be even more helpful in conjunction with transformation-based circuit simplifications. It is our future work to investigate such hybrid techniques.

References


Table 5. Multiplier verification by BDD

<table>
<thead>
<tr>
<th></th>
<th>Conventional Miter</th>
<th>$C_1^{-1} \cdot C_2$</th>
<th>$C_1 \cdot C_2^{-1}$</th>
<th>$C_2^{-1} \cdot C_1$</th>
<th>$C_2 \cdot C_1^{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/o simp.</td>
<td>w. simp.</td>
<td>w/o simp.</td>
<td>w. simp.</td>
<td>w/o simp.</td>
</tr>
<tr>
<td>Success 1</td>
<td>4</td>
<td>0.07</td>
<td>0.06</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.3</td>
<td>0.51</td>
<td>0.00</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>36.5</td>
<td>7.87</td>
<td>0.00</td>
<td>32.36</td>
</tr>
<tr>
<td>Success 2</td>
<td>4</td>
<td>0.11</td>
<td>0.06</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.59</td>
<td>0.64</td>
<td>0.01</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>38.14</td>
<td>12.69</td>
<td>0.05</td>
<td>37.4</td>
</tr>
<tr>
<td>Different</td>
<td>4</td>
<td>0.09</td>
<td>0.06</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.5</td>
<td>0.61</td>
<td>0.00</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>41.75</td>
<td>6.65</td>
<td>0.02</td>
<td>41.24</td>
</tr>
</tbody>
</table>

Table 6. Multiplier verification by QuIDD (Success)

<table>
<thead>
<tr>
<th></th>
<th>Conventional Miter</th>
<th>$C_1^{-1} \cdot C_2$</th>
<th>$C_1 \cdot C_2^{-1}$</th>
<th>$C_2^{-1} \cdot C_1$</th>
<th>$C_2 \cdot C_1^{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/o simp.</td>
<td>w. simp.</td>
<td>w/o simp.</td>
<td>w. simp.</td>
<td>w/o simp.</td>
</tr>
<tr>
<td>4</td>
<td>48.23</td>
<td>42.33</td>
<td>45.71</td>
<td>42.26</td>
<td>45.65</td>
</tr>
<tr>
<td>5</td>
<td>766.21</td>
<td>672.56</td>
<td>743.20</td>
<td>667.63</td>
<td>738.24</td>
</tr>
<tr>
<td>6</td>
<td>13001.39</td>
<td>10599.96</td>
<td>12387.79</td>
<td>10591.33</td>
<td>12443.68</td>
</tr>
</tbody>
</table>


Appendix: Calculation of Functions

We can calculate the logic functions realized at each point in a circuit by the following straightforward algorithm, where we abbreviate $f^j_i(x_1, x_2, \cdots, x_n)$ as $f_i^j$ and leave the choice of data structures to applications.

Initialization.

Let $f^0_i = x_i$ for $1 \leq i \leq n$.

Calculation of functions right after the $j$-th gate.

- If the target bit of the $j$-th gate is not on the $i$-th bit-line, then $f^j_i = f^{j-1}_i$, i.e., the function on the $j$-th bit-line is unchanged.
- If the target bit of the $j$-th gate is on the $i$-th bit-line,
  - If the gate is a NOT gate, then $f^j_i = f^{j-1}_i$.
  - If the gate is a CNOT gate whose control bit is on the $c_1$-th bit-line, then $f^j_i = f^{j-1}_i \oplus f^{-1}_{i_1}$.
  - If the gate is a Toffoli gate whose control bits are on the $c_1$-th and the $c_2$-th bit-lines, then $f^j_i = f^{j-1}_i \oplus f^{-1}_{i_1} \oplus f^{-1}_{i_2}$.

For example, we can calculate the functions in the circuit shown in Fig. 1 as follows. Since there is no target bit on the first and the second bit-lines, $f^0_1 = f^1_1 = f^2_1 = f^3_1 = x_1$ and $f^0_2 = f^1_2 = f^2_2 = f^3_2 = x_2$. The functions realized on the third bit-line can be calculated as:

- $f^3_3 = x_3$.
- $f^1_3 = f^0_3 \oplus x_1 \cdot x_2 = x_3 \oplus x_1 \cdot x_2$.
- $f^2_3 = f^1_3 \oplus x_3 = x_3 \oplus x_1 \cdot x_2 \cdot x_1$.
- $f^3_3 = \overline{f^3_3} = x_3 \oplus x_1 \cdot x_2 \oplus x_1 \oplus 1$. 