Exploiting Reversibility in the Complete Simulation of Reversible Circuits

Robert Wille*† Simon Stelter* Rolf Drechsler*†

*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany

{rwille,sstelter,drechsle}@informatik.uni-bremen.de

***Abstract*—Reversible circuits employ an alternative computation paradigm where all operations are performed in a reversible fashion only. Motivated by the promising applications, researchers started to developed corresponding design methods for this kind of circuits. In most of the resulting solutions, they try to address the restrictions and limitations that come with this alternative computation paradigm. In this work, we are instead showing possible advantages to be exploited. We present an alternative solution for complete simulation of reversible circuits which explicitly utilizes the reversibility of the underlying computation paradigm. By this, improvements of up to three orders of magnitude compared to the standard simulation can be achieved.**

I. Introduction

From the beginning, researchers and engineers narrowed the investigation of computing machines down to a preponderantly *irreversible* computing paradigm. In fact, most of the established computations are not invertible. A simple standard operation like the logical AND already illustrates that. Indeed, it is possible to obtain the inputs of an AND gate if the output is set to 1 (then, both inputs must be set to 1 as well). But, it is not possible to determine the input values if the AND outputs 0. While mainly relying on this *conventional* way of computation, alternative paradigms and their applications have hardly been considered and exploited yet.

A promising alternative is based on *reversible computation*, a computing paradigm which allows bijective operations only, i.e. reversible n-input n-output functions that map each possible input vector to a unique output vector. In circuits based on reversible logic, all computations can be reverted (i.e. the inputs can be obtained from the outputs *and* vice versa). For this purpose, established conventional gate libraries can obviously not be applied. As a consequence, new libraries of reversible gates have been introduced. Albeit not so well established yet, reversible computation enables several promising applications and, indeed, superiors conventional computation paradigms in many domains including but not limited to:

- Low Power Computation, where the fact that no information is lost in reversible computation can be exploited (see e.g. [1], [2], [3]),
- Adiabatic Circuits, a special low power technology where reversible circuits are particularly suited for (see e.g. [4]),
- Encoding and Decoding Devices, which always realize one-to-one mappings and, thus, inherently follow a reversible computing paradigm (see e.g. [5]),
- Quantum Computation, which enables to solve many relevant problems significantly faster than conventional circuits and inherently is reversible (see e.g. [6]), and
- Program Inversion (see e.g. [7]), as programs based on a reversible computation paradigm would allow an inherent and obvious program inversion.

Motivated by these applications, researchers started the investigation of corresponding methods for computer-aided design of this kind of circuits. This led to a variety of design solutions for a wide range of design tasks such as synthesis (see e.g. [8], [9], [10], [11]), optimization (see e.g. [12], [13]), verification (see e.g. [14], [15]), debugging (see e.g. [16]), and even automatic test pattern generation (see e.g. [17], [18]). Good overviews can be found e.g. in [19], [20], [21].

In most of these approaches, researchers try to address the restrictions and differences caused by reversible circuits in comparison to their conventional counterparts. For example, fanout and feedback are not directly allowed and, hence, the respective circuits have to be composed as cascades of special reversible gates. But beyond that, the reversible computing paradigm also employs several advantages that can be exploited in the design.

In this paper, we make use of one of these advantages in order to improve the complete simulation of reversible circuits. In fact, we present a revised simulation approach that does not consider each pattern to be simulated separately, but all possible patterns at once. We show that such a consideration is beneficial since, in reversible computation, (1) usually just a very small set of patterns is affected by a gate and (2) gates simply lead to an exchange of input patterns. The proposed solution significantly reduces the complexity of simulation and, hence, allows for speed-ups of up to three orders of magnitude.

In the remainder of this paper, the proposed simulation approach is described in detail. Section II briefly reviews the basics on reversible functions and circuits. Section III considers complete simulation and sketches the general idea of our contribution followed by Section IV describing the precise implementation. The complexity of the solutions are discussed in Section V and experimentally evaluated in Section VI. Finally, the paper is concluded in Section VII.

TABLE I
BOOLEAN FUNCTIONS

(a) Irreversible (Adder)

x_1	x_2	x_3	f_1	f_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) Irreversible

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

(c) Reversible

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	1

II. REVERSIBLE FUNCTIONS AND CIRCUITS

Logic computations can be defined as a function over Boolean variables. More precisely:

Definition 1. *A* Boolean function *is a mapping $f : \mathbb{B}^n \to \mathbb{B}$ with $n \in \mathbb{N}$. A function f is defined over its* primary input *variables $X = \{x_1, x_2, \ldots, x_n\}$ and hence is also denoted by $f(x_1, x_2, \ldots, x_n)$. The precise mapping is described in terms of Boolean expressions which are formed over the variables from X and operations like $\wedge$ (*AND*), $\vee$ (*OR*), or $\bar{}$ (*NOT*).*

A multi-output Boolean function *is a mapping $f : \mathbb{B}^n \to \mathbb{B}^m$ with $n, m \in \mathbb{N}$. More precisely, it is a system of Boolean functions $f_i(x_1, x_2, \ldots, x_n)$ with $1 \leq i \leq m$. The respective functions f_i are also denoted as* primary outputs*.*

Multi-output functions are also denoted as n-input, m-output functions or $n \times m$ functions, respectively. In this work, realizations of reversible functions are considered. Reversible functions are a subset of multi-output functions and are defined as follows:

Definition 2. *A multi-output function $f : \mathbb{B}^n \to \mathbb{B}^m$ is* reversible *iff*

- *its number of inputs is equal to the number of outputs (i.e. $n = m$) and*
- *it maps each input pattern to a unique output pattern.*

In other words, each reversible function is a bijection that performs a permutation of the set of input patterns. A function that is not reversible is termed *irreversible*.

Example 1. *Table I(a) shows the truth table of a 3-input, 2-output function representing a 1-bit adder. This function is irreversible, since $n \neq m$. Also the function in Table I(b) is irreversible. Here, the number n of inputs indeed is equal to the number m of outputs, but there is no unique input-output mapping (e.g. both inputs 000 and 001 map to the output 000). In contrast, the 3×3 function shown in Table I(c) is reversible, since each input pattern maps to a unique output pattern.*

Reversible functions are realized through reversible circuits.

Definition 3. *A* reversible circuit *G is a cascade of reversible gates $G = g_1 \ldots g_d$, where fanout and feedback are not directly allowed [6] and d denotes the number of gates. 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 $T = \{x_{j_1}, \ldots, x_{j_l}\} \subset X$ with $C \cap T = \emptyset$ is the set of target lines. The set C may be empty.*

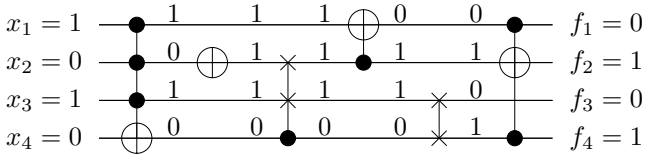

Fig. 1. A reversible circuit

Commonly used reversible gates are:

- *The* Toffoli gate $TOF(C, \{x_t\})$ *[22] which consists of a single target line $x_t \in X \setminus C$ whose value is inverted if all values on the control lines are set to 1 or if $C = \emptyset$. All remaining values are passed through the gate unaltered.*
- *The Fredkin gate $F(C, \{x_{t1}, x_{t2}\})$ [23] which consists of two target lines $x_{t1}, x_{t2} \in X \setminus C$ interchanges the values of these target lines if all values on the control lines are set to 1 or if $C = \emptyset$, respectively. Again, all remaining values are passed through the gate unaltered.*

Example 2. *Fig. 1 shows a reversible circuit composed of six reversible gates. Black circles denote the control lines, while the $\oplus$-symbol and the $\times$-symbol is used to denote the target line of the Toffoli gate and the Fredkin gate, respectively. The annotated values demonstrate the computation in the respective gates.*

III. COMPLETE SIMULATION OF REVERSIBLE CIRCUITS

In this work, we consider the complete simulation of reversible circuits, i.e. the determination of all possible output patterns obtained by applying all possible input patterns. For this purpose, we first discuss the straight-forward solution which probably is mostly used in the respective applications relying on simulation. Afterwards, we illustrate the general idea of our improved approach. The precise algorithm is then presented in the next section.

A. Standard Solution

The straight-forward solution for simulation is rather simple: All input patterns are separately applied to the circuit. Then, for each input pattern, the respective cascades of gates is successively applied eventually leading to the desired output pattern. This complies with the application of the pattern 1010 as illustrated above in Fig. 1.

TABLE II
ILLUSTRATION OF THE GENERAL IDEA

(a) Identity

Inp.	Out.
a b c d	a b c d
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 0
0 0 1 1	0 0 1 1
0 1 0 0	0 1 0 0
0 1 0 1	0 1 0 1
0 1 1 0	0 1 1 0
0 1 1 1	0 1 1 1
1 0 0 0	1 0 0 0
1 0 0 1	1 0 0 1
1 0 1 0	1 0 1 0
1 0 1 1	1 0 1 1
1 1 0 0	1 1 0 0
1 1 0 1	1 1 0 1
1 1 1 0	1 1 1 0
1 1 1 1	1 1 1 1

(b) 1^{st} Gate

Inp.	Out.
a b c d	a b c d
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 0
0 0 1 1	0 0 1 1
0 1 0 0	0 1 0 0
0 1 0 1	0 1 0 1
0 1 1 0	0 1 1 0
0 1 1 1	0 1 1 1
1 0 0 0	1 0 0 0
1 0 0 1	1 0 0 1
1 0 1 0	1 0 1 0
1 0 1 1	1 0 1 1
1 1 0 0	1 1 0 0
1 1 0 1	1 1 0 1
1 1 1 0	**1 1 1 1**
1 1 1 1	**1 1 1 0**

(c) 2^{nd} Gate

Inp.	Out.
a b c d	a b c d
0 0 0 0	**0 1 0 0**
0 0 0 1	**0 1 0 1**
0 0 1 0	**0 1 1 0**
0 0 1 1	**0 1 1 1**
0 1 0 0	**0 0 0 0**
0 1 0 1	**0 0 0 1**
0 1 1 0	**0 0 1 0**
0 1 1 1	**0 0 1 1**
1 0 0 0	**1 1 0 0**
1 0 0 1	**1 1 0 1**
1 0 1 0	**1 1 1 0**
1 0 1 1	**1 1 1 1**
1 1 0 0	**1 0 0 0**
1 1 0 1	**1 0 0 1**
1 1 1 0	**1 0 1 1**
1 1 1 1	**1 0 1 0**

(d) 3^{rd} Gate

Inp.	Out.
a b c d	a b c d
0 0 0 0	0 1 0 0
0 0 0 1	**0 0 1 1**
0 0 1 0	0 1 1 0
0 0 1 1	0 1 1 1
0 1 0 0	0 0 0 0
0 1 0 1	0 0 0 1
0 1 1 0	0 0 1 0
0 1 1 1	**0 1 0 1**
1 0 0 0	1 1 0 0
1 0 0 1	**1 0 1 1**
1 0 1 0	1 1 1 0
1 0 1 1	1 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	1 0 0 1
1 1 1 0	**1 1 0 1**
1 1 1 1	1 0 1 0

(e) 4^{th} Gate

Inp.	Out.
a b c d	a b c d
0 0 0 0	**1 1 0 0**
0 0 0 1	0 0 1 1
0 0 1 0	**1 1 1 0**
0 0 1 1	**1 1 1 1**
0 1 0 0	0 0 0 0
0 1 0 1	0 0 0 1
0 1 1 0	0 0 1 0
0 1 1 1	**1 1 0 1**
1 0 0 0	**0 1 0 0**
1 0 0 1	1 0 1 1
1 0 1 0	**0 1 1 0**
1 0 1 1	**0 1 1 1**
1 1 0 0	1 0 0 0
1 1 0 1	**0 1 0 1**
1 1 1 0	**0 1 0 1**
1 1 1 1	1 0 1 0

(f) 5^{th} Gate

Inp.	Out.
a b c d	a b c d
0 0 0 0	1 1 0 0
0 0 0 1	0 0 1 1
0 0 1 0	**1 1 0 1**
0 0 1 1	1 1 1 1
0 1 0 0	0 0 0 0
0 1 0 1	**0 0 1 0**
0 1 1 0	**0 0 0 1**
0 1 1 1	**1 1 1 0**
1 0 0 0	0 1 0 0
1 0 0 1	1 0 1 1
1 0 1 0	**0 1 0 1**
1 0 1 1	0 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	**1 0 1 0**
1 1 1 0	**0 1 1 0**
1 1 1 1	**1 0 0 1**

(g) 6^{th} Gate

Inp.	Out.
a b c d	a b c d
0 0 0 0	1 1 0 0
0 0 0 1	0 0 1 1
0 0 1 0	**1 0 0 1**
0 0 1 1	**1 0 1 1**
0 1 0 0	0 0 0 0
0 1 0 1	0 0 1 0
0 1 1 0	0 0 0 1
0 1 1 1	1 1 1 0
1 0 0 0	0 1 0 0
1 0 0 1	**1 1 1 1**
1 0 1 0	0 1 0 1
1 0 1 1	0 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	1 1 1 0
1 1 1 0	0 1 1 0
1 1 1 1	**1 1 0 1**

Because of this simplicity, this solution is heavily used in many applications such as validation, optimization, automatic test pattern generation, or debugging of reversible circuits. In total, this leads to a run-time complexity of $2^n \cdot d$ with n and d being the number of circuit lines and gates, respectively.

B. General Idea for Improvement

Considering reversible circuits, complete simulation can significantly been improved by exploiting two certain characteristics of this type of computation:

1) *Functional Effect of Single Gates*
   For many input patterns, single gates often have no effect, i.e. often just very small sets of inputs are actually leading to different output patterns. In fact, e.g. a Toffoli gate $g(C, T)$ only has an effect for the $2^{n-|C|}$ input patterns where all control lines $x \in C$ are set to 1.
2) *Reversible Gates Realize Permutations*
   Reversible gates always realize a permutation of two input patterns, i.e. if e.g. an input pattern 0000 maps to an output pattern 0001, then the input pattern 0001 must map to an output pattern 0000, too. By this, each gate eventually leads to an exchange of input patterns.

Example 3. *Consider again the circuit from Fig. 1. Considering all $2^4 = 16$ possible input patterns for this circuit, then*

- *gate g_1 modifies only 2 of them,*
- *gate g_2 modifies all 16 of them,*
- *gate g_3 modifies only 4 of them,*
- *gate g_4 modifies only 8 of them,*
- *gate g_5 modifies all 8 of them, and*
- *gate g_6 modifies only 4 of them.*

Furthermore, e.g. the first gate just swaps the respective patterns 1110 and 1111. The same holds e.g. for the sixth gate where all patterns of the form $11-1$ are swapped with the corresponding pattern of the form $10-1$.

These characteristics enable a complete simulation without the need for the application of $2^n \cdot d$ steps. In fact, in order to simulate e.g. the first gate from Fig. 1, just two steps (for 1110 and 1111) are necessary. The remaining 14 simulations steps can be omitted because, due to the structure of the gate, they have no affect on the result. Moreover, this number of steps can be reduced further by exploiting the second characteristic: Since each gate has a unique input/output mapping, the corresponding computation just leads to an exchange of patterns. For example, the first gate from Fig. 1 simply swaps 1110 and 1110. Because of this, it is sufficient to only consider patterns with a target line set to 1 (in this case 1111)[1]. This reduces the number of patterns to be considered to just a single one (instead of 16 when the standard solution is applied).

This is exploited in the proposed simulation approach. Instead of separately applying all possible input patterns, we suggest a methodology that considers all patterns at once. Then, the application of a single gate does not consider each pattern separately, but just performs the respective swaps on exactly those patterns which are actually affected by the gate. The configuration of these affected patterns can easily be determined from the type and the set of control lines of the considered gate.

The application of this idea is illustrated by the following example.

Example 4. *Following the general idea from above, the circuit depicted in Fig. 1 shall completely been simulated. The respective steps are given in Table II. We start with a representation of all $2^4 = 16$ possible pattern. As long as no gate is applied, the identity results (see Table II(a)). Then, the respective gates are applied.*

As discussed above, the first gate in Fig. 1 only swaps the patterns 1110 and 1111. That is, only one step has to be performed to simulate this gate (leading to Table II(b)). The second gate is the worst case as all patterns are affected. Nevertheless, still only half the steps compared to standard simulation are necessary since each pattern of the form $-1--$ is simply swapped with the corresponding pattern of the form $-0--$ (leading to Table II(c)). The third gate represents a Fredkin gate. Because of the control line, this shall affect patterns of the form $---1$. However, patterns with both

[1] In a similar fashion, this also holds for the Fredkin gate.

Input : $G = g_1..g_d$ (circuit to be simulated)
Output: TT (truth table)

1 $TT \leftarrow id$
2 **foreach** $g(C,T)$ ***from*** G **do**
3 $P = \{\vec{p} \in \mathbb{B}^n \mid p_i = 1 \text{ for each } x_i \in C \cup \{t\} \text{ and } t \in T\}$
4 **foreach** $\vec{p} \in P$ **do**
5 **if** g *is a Toffoli gate* **then**
6 $\vec{q} = q_1 \dots q_n$ with $q_i = \begin{cases} 0, & \text{for } x_i \in T \\ p_i, & otherwise \end{cases}$
7 **if** g *is a Fredkin gate* **then**
8 // consider $x_i, x_j \in T$ with $x_i \neq x_j$
9 **if** $p_i = 1 \wedge p_j = 0$ **then**
10 $\vec{q} = q_1 \dots q_n$ with $q_i = \begin{cases} 0, & \text{for } x_i \\ 1, & \text{for } x_j \\ p_i, & otherw. \end{cases}$
11 swap($TT[\vec{p}]$,$TT[\vec{q}]$)
12 **return** TT

Fig. 2. Proposed simulation algorithm

target lines set to the same value are also not affected (as Fredkin gates swap the value of the target lines). Hence, for this gate only the patterns 0011 *and* 1011 *are swapped with* 0101 *and* 1101*, respectively (leading to Table II(d)). In a similar fashion all remaining gates are applied eventually leading to the results of the complete simulation as shown in Table II(g).*

IV. Implementation

Motivated by the general idea sketched above, this section presents the improved algorithm for complete simulation of reversible circuits. The algorithm does not simulate single patterns separately, but considers all of them at once. The execution of gates is conducted by swapping those patterns which are actually affected by the currently considered gate. The configuration of these affected patterns is determined from the type and the set of control lines of the considered gate. More precisely:

Definition 4. *Let $g(C,T)$ be a reversible gate in a circuit over n lines and the inputs $X = \{x_1 \dots x_n\}$. All patterns with the respective control line set to* 1 *are affected by this gate. Furthermore, due to the permutative nature of reversible gates, only those pattern with their first target line set to 1 have to be considered. Hence, the patterns to be considered can be summarized in a set P composed of* all *patterns of the form $\vec{p} \in \mathbb{B}^n$ with $p_i = 1$ for each $x_i \in C \cup \{t\}$ with $t \in T$ and $i \in \{1 \dots n\}$.*

Example 5. *Consider the last gate of the circuit from Fig. 1. The corresponding set M of patterns to be considered is $P = \{1101, 1111\}$.*

Having this definition, the proposed simulation approach follows the algorithm as sketched in Fig. 2. This algorithm gets a circuit G to be simulated and generates the complete function (in terms of a truth table) realized by G. In the following, the respective steps are discussed.

First, the truth table to be generated is initialized with the identity function (Line 1). Then, each gate $g(C,T)$ of the circuit G is traversed from the outputs to the inputs (Line 2). In each iteration, the set P of patterns to be considered for the current gate g is created according to Definition 4 (Line 3). For each of those patterns $\vec{p} \in P$ the corresponding counterpart $\vec{q}$, i.e. the pattern to be swapped with $\vec{p}$, is created next (Line 4-10). This depends on the type of the current gate g:

- In case of a Toffoli gate (Line 5/6), an inversion is conducted at the target line. Hence, $\vec{q}$ is equal to $\vec{p}$ except for the target line which is set to 0 (instead of 1).
- In case of a Fredkin gate (Line 7-10), the value of one of the target lines (1, in the currently considered pattern) is swapped with the value of the other target line. This only has an effect, if the other target line is 0. Hence, only patterns $\vec{p}$ with $p_i = 1$ and $p_j = 0$ for $x_i, x_j \in T$ and $x_i \neq x_j$ are considered. The counterpart q is then defined by the opposite assignment of the target lines.

By this, an affected pattern $\vec{p}$ and its corresponding counterpart $\vec{q}$ have been determined and, thus, only need to be swapped in the truth table representation (Line 11). If all affected patterns and all gates have been considered accordingly, the algorithm returns the resulting truth table and terminates (Line 12).

V. Discussion

Following the procedure described above leads to an improved run-time complexity in comparison to the standard simulation. In fact, we can distinguish between a best case and a worst case:

- The *best case* can be observed when the circuit is solely composed of Toffoli gates with the maximum number of control lines (i.e. of Toffoli gates $g(C,T)$ with $|C| = n-1$). Then, the simulation of each gate just requires one general step, namely swapping two patterns. This leads to d steps for the whole circuit. As the initial creation of all possible patterns requires 2^n steps, we eventually get $2^n + d$ steps in the best case. Compared to the standard approach with its $2^n \cdot d$ steps, this is considerably smaller. The number of gates hardly affects the total run-time anymore.
- The *worst case* can be observed when the circuit is solely composed of Toffoli gates with no control lines at all (i.e. of Toffoli gates $g(C,T)$ with $C = \emptyset$). Then, all patterns are affected. However, due to the exploitation of the permutation characteristic, still only half the swapping steps need to be considered. This eventually leads to $2^n + \frac{(2^n \cdot d)}{2}$ steps in the worst case. But since circuits solely composed of Toffoli gates with no control lines (i.e. basically NOT gates) hardly occur, this worst case does not occur frequently.

In both cases, the complexity is considerably less compared to the standard simulation. Of course, the major bottleneck remains the exponential complexity. However, as simulation still is heavily applied in various applications for small circuits

(where exponential complexity is feasible), the improvements enabled by the proposed simulation approach are beneficial. This is also confirmed by actual run-time measurements conducted in our experimental evaluation which is summarized in the next section.

VI. Experimental Evaluation

The approach introduced in Section IV has been implemented in C++ on top of *RevKit* [24]. In order to evaluate the efficiency of the approach, the proposed solution has been applied to completely simulate a wide range of benchmarks circuits taken from *RevLib* [25]. Then, the resulting run-time has been compared to the corresponding results obtained using a standard simulation[2]. All experiments have been conducted on an Intel Core i5-2430M machine with 2.4 GHz and 6 GB of memory running Linux.

Table III summarizes the results. The first columns provide the name (denoted by *Circuit*), the number of lines (denoted by n), and the number of gates (denoted by d) of the respectively considered benchmark circuits. The run-time (in milliseconds) required for standard simulation and the proposed simulation scheme are given in the columns denoted by *Standard* and *Proposed*, respectively. The improvement of the proposed approach, i.e. the run-time of the standard approach divided by the run-time of the proposed simulation, is provided in Column *Impr.*

The results clearly confirm the benefits of the proposed approach. For *all* benchmark circuits, significant reductions in the needed run-time can be observed. In particular for circuits with a large number of gates, the increase in the performance is evident. In these cases improvements of more than three orders of magnitude can be observed. Instead of waiting several minutes for a result, the proposed approach delivers the results in a fraction of a second. In particular, for approaches which rely on frequent complete simulation runs (such as window optimization [13]), this speed-up is essential.

Besides that, Table III also provides some numbers concerning the complexity discussion from Section V. As discussed there, the proposed simulation approach has a worst case and best case complexity of $2^n + \frac{(2^n \cdot d)}{2}$ and $2^n + d$, respectively, compared to the complexity of the standard simulation (which is $2^n \cdot d$). Applied to precise benchmark circuits, Columns *Complexity Impr.* of Table III show the improvements with respect to the worst case (denoted by *wc*) and the best case (denoted by *bc*). Also these numbers confirm the discussions from above. For almost all benchmark circuits, the actual run-time improvement (Column *Impr.*) indeed is between the estimated worst and best case; in most of the cases a bit closer to the worst case.

Overall, the proposed algorithm provides an alternative simulation method that is as easy to implement as the standard simulation, but has significant benefits with respect to the resulting complexity which has also been confirmed by the actual application to a large set of circuits.

[2]In fact, we used the open source simulation engine provided in *RevKit* [24].

VII. Conclusion

In this paper, we introduced an approach for complete simulation of reversible circuits which explicitly exploited the reversibility of the underlying computation paradigm. In particular, we utilized the fact that (1) usually just a very small set of patterns is affected by a gate and (2) that gates simply lead to an exchange of input patterns. Although the proposed solution still remains exponential in the complexity, improvements of up to three orders of magnitude can be achieved compared to the standard simulation. Nevertheless, due to the complexity the approach still is restricted to circuits with a small number of inputs/outputs only. Furthermore, the approach only supports complete simulation thus far. However, for all applications which rely on frequent complete simulation runs (such as window optimization [13]) the proposed methods offers a simple but effective alternative.

References

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Dev.*, vol. 5, p. 183, 1961.

[2] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev*, vol. 17, no. 6, pp. 525–532, 1973.

[3] B. Desoete and A. D. Vos, "A reversible carry-look-ahead adder using control gates," *INTEGRATION, the VLSI Jour.*, vol. 33, no. 1-2, pp. 89–104, 2002.

[4] P. Patra and D. Fussell, "On efficient adiabatic design of MOS circuits," in *Workshop on Physics and Computation*, Boston, 1996, pp. 260–269.

[5] R. Wille, R. Drechsler, C. Oswald, and A. Garcia-Ortiz, "Automatic design of low-power encoders using reversible circuit synthesis," in *Design, Automation and Test in Europe*, 2012, pp. 1036–1041.

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

[7] R. Glück and M. Kawabe, "A method for automatic program inversion based on LR(0) parsing," *Fundamenta Informaticae*, vol. 66, no. 4, pp. 367–395, 01 2005.

[8] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conf.*, 2003, pp. 318–323.

[9] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 22, no. 6, pp. 710–722, 2003.

[10] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *Design Automation Conf.*, 2009, pp. 270–275.

[11] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler, "Synthesis of reversible circuits with minimal lines for large functions," in *ASP Design Automation Conf.*, 2012, pp. 85–92.

[12] D. Y. Feinstein, M. A. Thornton, and D. M. Miller, "Partially redundant logic detection using symbolic equivalence checking in reversible and irreversible logic circuits," in *Design, Automation and Test in Europe*, 2008, pp. 1378–1381.

[13] M. Soeken, R. Wille, G. W. Dueck, and R. Drechsler, "Window optimization of reversible and quantum circuits," in *Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 341–345.

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

[15] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits," *IEICE Transactions*, vol. 91-A, no. 2, pp. 584–594, 2008.

[16] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler, "Debugging of Toffoli networks," in *Design, Automation and Test in Europe*, 2009, pp. 1284–1289.

[17] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes, "A family of logical fault models for reversible circuits," in *Asian Test Symp.*, 2005, pp. 422–427.

[18] R. Wille, H. Zhang, and R. Drechsler, "ATPG for reversible circuits using simulation, Boolean satisfiability, and pseudo Boolean optimization," in *IEEE Annual Symposium on VLSI*, 2011, pp. 120–125.

[19] R. Drechsler and R. Wille, "From truth tables to programming languages: Progress in the design of reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2011, pp. 78–85.

TABLE III
EXPERIMENTAL EVALUATION

Circuit	n	d	Simulation Run-time			Complexity Impr.	
			Standard	Proposed	Impr.	wc	bc
urf6_160	15	10740	663459	1639	404.8	2.0	8089.6
urf6_281	15	5088	349400	2021	172.9	2.0	4405.0
urf4_187	11	32004	152443	279	545.4	2.0	1924.9
alu3_200	18	94	63286	12458	5.1	2.0	95.0
urf3_155	10	26468	52481	142	369.9	2.0	985.9
sqr6_259	18	81	52346	13335	3.9	2.0	82.0
5xp1_194	17	85	28241	5998	4.7	2.0	85.9
dk27_225	18	24	24266	12470	1.9	1.9	25.0
urf3_279	10	14075	24174	95	254.4	2.0	954.6
example2_231	16	157	23937	2586	9.3	2.0	157.6
alu2_199	16	157	23682	2594	9.1	2.0	157.6
ryy6_256	17	44	23389	5898	4.0	2.0	45.0
mlp4_245	16	131	21064	2687	7.8	2.0	131.7
x2_267	17	38	16692	6970	2.4	2.0	39.0
inc_237	16	93	15635	2598	6.0	2.0	93.9
parity_247	17	32	13148	5991	2.2	1.9	33.0
urf1_149	9	11554	11520	58	197.1	2.0	490.3
t481_263	17	21	11406	6001	1.9	1.9	22.0
ham15_107	15	132	9991	1242	8.0	2.0	132.5
urf5_158	9	10276	9882	53	186.9	2.0	487.7
urf3_156	10	2732	7826	41	191.2	2.0	745.1
clip_206	14	174	7603	587	12.9	2.0	173.2
ham15_109	15	109	7551	1482	5.1	2.0	109.6
dc2_222	15	75	7129	1240	5.7	2.0	75.8
cnt3-5_179	16	25	6697	2608	2.6	1.9	26.0
ham15_108	15	70	6617	1601	4.1	2.0	70.8
urf3_157	10	2674	6613	39	168.8	2.0	740.7
urf1_278	9	6761	5967	40	147.8	2.0	476.0
cnt3-5_180	16	20	5480	2622	2.1	1.9	21.0
misex1_241	15	55	5110	1219	4.2	2.0	55.9
co14_215	15	30	4692	1307	3.6	1.9	31.0
urf5_280	9	5097	4638	39	120.1	2.0	465.4
sao2_257	14	88	4076	546	7.5	2.0	88.5
rd84_142	15	28	3571	1332	2.7	1.9	29.0
dist_223	13	185	3535	248	14.3	2.0	181.9
cm85a_209	14	69	3407	784	4.3	2.0	69.7
urf2_152	8	5030	2440	22	109.5	2.0	243.7
cm42a_207	14	35	2070	712	2.9	1.9	35.9
pm1_249	14	35	2053	698	2.9	1.9	35.9
hwb9_123	9	1959	1975	26	77.1	2.0	406.1
root_255	13	99	1960	258	7.6	2.0	98.8
0410184_169	14	46	1893	559	3.4	2.0	46.9
hwb9_121	9	1541	1800	23	79.6	2.0	384.6
urf1_150	9	1517	1797	19	92.7	2.0	383.1
hwb9_119	9	1544	1785	23	78.4	2.0	384.7
urf1_151	9	1487	1763	21	83.9	2.0	381.1
sym6_316	14	29	1722	576	3.0	1.9	29.9
urf2_161	8	3250	1639	17	99.3	2.0	237.4
urf2_277	8	3144	1590	21	75.3	2.0	236.8
radd_250	13	48	1245	261	4.8	2.0	48.7
adr4_197	13	55	1133	261	4.3	2.0	55.6
squar5_261	13	43	1043	254	4.1	2.0	43.8
rd84_253	12	111	995	121	8.3	2.0	109.0
sym10_262	11	194	922	59	15.7	2.0	178.1
rd53_311	13	34	778	259	3.0	1.9	34.9
urf5_159	9	499	631	15	43.4	2.0	253.2
sym9_148	10	210	513	29	17.5	2.0	175.1
sqrt8_260	12	40	476	150	3.2	2.0	40.6
dc1_220	11	39	439	91	4.8	2.0	39.3
dc1_221	11	39	402	93	4.3	2.0	39.3
urf2_153	8	638	374	8	46.8	2.0	183.0
hwb8_116	8	749	364	8	45.4	2.0	191.0
hwb8_113	8	637	363	11	32.7	2.0	182.9
urf2_154	8	620	354	8	45.2	2.0	181.5
sym9_146	12	28	353	121	2.9	1.9	28.8
cycle10_2_110	12	19	349	131	2.7	1.9	19.9
hwb8_114	8	614	348	8	45.7	2.0	181.0
9symml_195	10	129	303	29	10.3	2.0	115.5
sym9_193	10	129	295	36	8.3	2.0	115.5
cm152a_212	12	16	261	131	2.0	1.9	16.9
z4ml_269	11	48	258	59	4.4	2.0	47.9
z4_268	11	48	255	54	4.8	2.0	47.9
life_238	10	107	254	27	9.5	2.0	97.8
max46_240	10	107	249	35	7.1	2.0	97.8
sqn_258	10	76	190	27	7.1	2.0	71.7
rd73_252	10	80	185	29	6.4	2.0	75.1
hwb7_59	7	289	170	5	33.3	2.0	89.0
wim_266	11	25	155	60	2.6	1.9	25.7
rd73_140	10	20	100	30	3.4	1.9	20.6
hwb7_62	7	331	84	4	23.7	2.0	92.6
sys6-v0_111	10	20	75	28	2.7	1.9	20.6
hwb7_61	7	236	71	4	19.2	2.0	83.3
mini_alu_305	10	20	69	27	2.5	1.9	20.6
hwb7_60	7	166	51	4	14.0	2.0	72.7
con1_216	9	21	38	12	3.2	1.9	21.1
rd53_251	8	27	21	5	3.8	1.9	25.3
hwb6_56	6	126	19	2	10.6	2.0	42.8
f2_232	8	19	18	5	3.8	1.9	18.6
cm82a_208	8	22	18	5	3.3	1.9	21.2
rd53_130	7	30	15	4	4.2	1.9	25.1
rd53_138	8	12	15	5	2.7	1.9	12.4
sym6_145	7	36	14	3	4.5	1.9	28.9
rd53_131	7	28	14	5	2.9	1.9	23.8
ham7_104	7	23	10	3	3.4	1.9	20.3
hwb6_57	6	65	9	2	5.6	2.0	32.7
ham7_106	7	25	9	3	3.3	1.9	21.8
ham7_105	7	21	9	3	3.2	1.9	18.9
majority_239	6	8	8	3	2.9	1.8	8.0
rd53_135	7	16	7	3	2.6	1.9	15.1
hwb6_58	6	42	7	2	4.5	2.0	26.0
rd53_137	7	16	7	3	2.5	1.9	15.1
rd53_133	7	12	6	3	2.4	1.9	11.9
C17_204	7	9	6	3	2.3	1.8	9.3
alu-bdd_288	7	9	5	3	1.6	1.8	9.3
4mod5-bdd_287	7	8	5	3	1.9	1.8	8.5
mod5adder_128	6	15	4	2	2.5	1.9	13.0
decod24-bdd_294	6	11	4	2	2.6	1.8	10.2
graycode6_47	6	5	4	2	1.9	1.7	5.6
mod5adder_127	6	21	4	2	2.6	1.9	16.6
graycode6_48	6	5	4	2	1.9	1.7	5.6
mod5adder_129	6	17	4	1	2.4	1.9	14.2
decod24-enable_126	6	14	4	1	2.4	1.9	12.3
ex3_229	6	7	4	1	2.4	1.8	7.2
ex3_228	6	13	3	1	2.3	1.9	11.6
ex2_227	6	13	3	1	2.2	1.9	11.6
xor5_254	6	7	3	2	1.9	1.8	7.2
ex1_226	6	7	3	1	2.2	1.8	7.2
decod24-enable_125	6	9	3	1	1.9	1.8	8.8

Circuit: Name of the benchmark circuit n: Number of lines d: Number of gates
Standard: Run-time of the standard simulation Proposed: Run-time of the proposed simulation Impr.: Improvement of the proposed simulation (factor)
Complexity Impr.: Improvements in the complexity with respect to the worst case (*wc*) and the best case (*bc*) compared to the complexity of the standard simulation

[20] M. Saeedi and I. L. Markov, "Synthesis and optimization of reversible circuits - a survey," *ACM Computing Surveys*, 2011.

[21] R. Drechsler and R. Wille, "Reversible circuits: Recent accomplishments and future challenges for an emerging technology," in *Int'l Symp. on VLSI Design and Test*, 2012, pp. 383–392.

[22] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, W. de Bakker and J. van Leeuwen, Eds. Springer, 1980, p. 632, technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[23] E. F. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3/4, pp. 219–253, 1982.

[24] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "RevKit: An Open Source Toolkit for the Design of Reversible Circuits," in *Reversible Computation 2011*, ser. Lecture Notes in Computer Science, vol. 7165, 2012, pp. 64–76, RevKit is available at www.revkit.org.

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