RevKit: A Toolkit for Reversible Circuit Design

Mathias Soeken Stefan Frehse Robert Wille Rolf Drechsler
Institute of Computer Science
University of Bremen, Bremen, Germany
revkit@informatik.uni-bremen.de – http://www.revkit.org

***Abstract*—In recent years, research in the domain of reversible circuit design has attracted significant attention leading to many different approaches for e.g. synthesis, optimization, simulation, verification, and test. However, most of the resulting tools are not publicly available. In this paper, we introduce RevKit, an open source toolkit that aims to make recent developments in reversible circuit design accessible to other researchers. Therefore, a modular and extendable framework is provided which easily enables the addition of new methods and tools. RevKit already provides some of the existing approaches for synthesis, optimization, and verification functionality.**

I. Introduction and Background

The development of computing machines has found great success in the last decades. Nowadays billions of components are built on a few square centimeters – and this increasing trend continues. The number of transistors in an integrated circuit doubles every 18 months (also known as *Moore's Law*). However, it is obvious that such an exponential growth must reach its limits in the future. Otherwise, the miniaturization would reach a level where transistors consist of only single atoms. Furthermore, power dissipation more and more becomes a crucial issue for designing high performance digital circuits.

To further satisfy the needs for more computational power, alternatives are needed that go beyond the scope of the "traditional" (CMOS) technologies. Reversible logic marks a promising new direction where all operations are performed in an invertible manner. That is, in contrast to traditional logic, only bijective operations are allowed implying a reversible computation (i.e. the inputs can be obtained from the outputs *and* vice versa). This reversibility builds the basis for emerging technologies that may replace, or at least enhance, traditional computer chips (e.g. in the domain of low-power design [11], [2], [4], quantum computation [21], [16], [27], optical computing [3], DNA computing [25], as well as nanotechnologies [14]).

The basic concepts of reversible logic are therefore not new and were already introduced in the 60's by Landauer [11] and further refined by Bennett [2] and Toffoli [26]. They observed that due to the reversibility fanouts and feedback are not directly allowed in reversible circuits. As a consequence new libraries of (reversible) gates have been introduced including e.g. Toffoli gates [26], Fredkin gates [7], and Peres gates [18]. Fig. 1 shows these gates in a cascade. Each gate consists of control lines (denoted by ●) and target lines (denoted by $\oplus$ except for the Fredkin gate where an $\times$ is used instead). For a Toffoli gate, the value of the target line becomes inverted, if all control lines are assigned to 1 while for the Fredkin gate the target lines are interchanged in this case. The Peres gate is a cascade of two Toffoli gates. The annotated values in Fig. 1 demonstrate the computation of the respective gates. As can be seen, the calculation can be done in both directions (i.e. it is reversible).

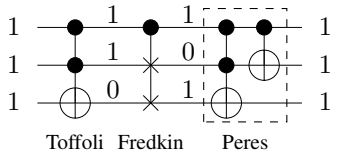

Fig. 1. Reversible gates

Even if this represents the basis for research in the area of reversible circuits, the topic was not intensively studied by computer scientists before the year 2000. The main reason for that may be due to the fact that applications of such circuits have been seen as "dreams of the future". However, this changed with recently made achievements. For example, in the domain of low-power design, first reversible circuits have been built which are powered only by their input signals and do not need additional power supplies (see e.g. [4]). In quantum computation, factorization has also been solved in polynomial time whereas only exponential solving methods are known for traditional circuits (see e.g. [21], [27]). These achievements (together with others) significantly moved the topic forward so that nowadays reversible logic is seen as a promising research area. As a consequence, in the last years computer scientists started to develop new methods for the design of reversible circuits. Among others, these include approaches for synthesis (see e.g. [20], [13], [30]), optimization (see e.g. [13]), simulation (e.g. [29]), verification (e.g. [28], [8], [32]), and test (e.g. [19], [17]).

However, most of the resulting methods are not publicly available[1]. This often makes the development of new methods harder since e.g. previous approaches are not available for comparison. Furthermore, approaches have to be re-implemented from scratch in order to modify or improve them. The lack of tools for reversible hardware design makes it hard for beginners to get involved in the topic.

In this paper we introduce *RevKit*, an open source toolkit for reversible circuit design. The motivation behind RevKit is to make recent developments in the domain of reversible circuit design accessible to other researchers. Therefore, a modular and extendable framework is provided which easily enables the addition of new methods and tools. Besides basic functionality (like parser and export functions), RevKit already provides elaborated methods for synthesis, optimization, and verification. In this sense, RevKit addresses users who simply

[1]Exceptions are e.g. the RMRLS synthesis approach [10] which is available at http://www.princeton.edu/~cad/projects.html or the quantum simulator QuIDDPro [29] which is available at http://vlsicad.eecs.umich.edu/Quantum/qp/.

want to apply the framework and its tools as well as developers who actively want to develop further methods on top of the framework. For this purpose, RevKit is available online at http://www.revkit.org.

In the following, RevKit is introduced. Therefore, we describe the features of the framework from two different perspectives. First, the users' view is presented in Section II. This includes an overview of the currently provided tools and illustrates how to utilize them. Afterwards, some technical details are presented in Section III. This particularly addresses the needs of developers aiming to extend RevKit with their own tools and algorithms. Note that, in the following, the technical details are described in a very brief manner. A more detailed introduction in how to use and how to extend RevKit is available in a detailed documentation provided online at www.revkit.org. Finally, Section IV concludes the paper.

II. The Users' Perspective

In this section, we summarize the basic tools and algorithms of RevKit which are available so far. Afterwards, we sketch a possible application scenario.

A. Available Tools and Algorithms

RevKit provides core functionalities which are needed in order to work with reversible circuits. This includes read-in routines for functions and reversible circuits (based on the RevLib format introduced in [33]), several export functions (again into the RevLib format, but LaTeX and BLIF dumps are also available), cost calculations, etc. Besides that, the following approaches are available in RevKit.

Synthesis

- A transformation-based method inspired by the concepts of [15]
- The BDD-based synthesis method as introduced in [30]
- The KFDD-based synthesis method as introduced in [22]
- The heuristic synthesis with output permutation method as introduced in [31]
- The ESOP-based synthesis method inspired by the concepts of [6]
- The exact synthesis method as introduced in [9]

Optimization

- The window optimization method as introduced in [23]
- The circuit line reduction method as introduced in [34]

Verification

- The SAT-based equivalence checker as introduced in [32]

Further Methods

- A naïve method to embed irreversible functions into reversible ones (needed e.g. to synthesize irreversible functions using the transformation-based method)
- A simple simulation engine (for reversible circuits working on Boolean values)
- A simple decomposition method that maps a given reversible circuit (composed of Toffoli, Fredkin, and Peres gates) to its equivalent quantum circuit (composed of NOT, CNOT, V, and V+ gates) inspired by the concepts of [1] and [12].

All these tools and algorithms are written in C++ and directly accessible by an API. That is, they can be used in other C++ programs. Furthermore, to enable command line usage, all functions are also exposed as a Python library[2]. As a result, commands (or sequences of commands) can easily be specified and processed using a common Python interpreter. This is illustrated in the next section.

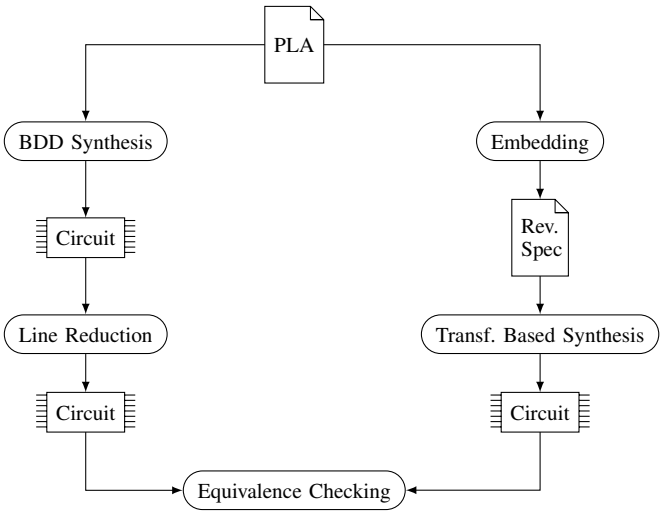

Fig. 2. An application scenario

B. Application Scenario

Fig. 2 shows a possible application scenario that can be processed by RevKit. An irreversible function (given in PLA-format) is synthesized as a reversible circuit. Therefore, two different approaches are applied, namely the BDD-based synthesis method as well as the transformation-based method, respectively. Since the latter approach only works for reversible functions, the irreversible function needs to be embedded first (leading to a reversible specification). In contrast, the BDD-based method tends to produce circuits with a large number of circuit lines. Thus, the respective optimization approach is to improve the result. Finally, both resulting circuits are checked for equivalence in order to verify the correctness of the results.

To process the respective steps in RevKit, the Python code depicted in Fig. 3 can be used. After importing the RevKit library (line 2), first an empty reversible circuit is initialized (line 5). Then, the BDD synthesis approach is called in order to realize a circuit representing the function specified in the given PLA-file (line 6). Afterwards, the resulting circuit is optimized (line 7). Similar steps are performed for the second flow (lines 9–14). Finally, both resulting circuits are checked for equivalence (line 17).

As can be seen, the respective commands can be invoked in a very intuitive manner. Furthermore, the user can decide either to separately enter all these commands into a shell or to merge them as a corresponding Python script.

III. The Developers' Perspective

Besides providing recent tools and algorithms, RevKit also aims to support researchers in the development of new or improved methods for reversible circuit design. To this end,

[2]Therefore, the Boost.Python library was utilized. For further information visit http://www.boost.org/doc/libs/1_42_0/libs/python/doc/index.html

```
#!/usr/bin/python
from revkit import *

# BDD based Synthesis
c_bdd = circuit()
bdd_synthesis(c_bdd, "function.pla")
line_reduction(c_bdd)

# Transformation based Synthesis
c_tbs = circuit()
tt = binary_truth_table()
read_pla(tt, "function.pla")
embed_truth_table(tt)
transformation_based_synthesis(c_tbs, tt)

# Check equivalence
print "Equal?", equivalence_check(c_bdd, c_tbs)
```

Fig. 3. Python code for the design flow depicted in Fig. 2

RevKit is based on a very modular and extendable framework which is introduced in more detail in this section. First, the architecture of RevKit is thereby described followed by a brief discussion of applied design concepts. Afterwards, it is illustrated, by means of an example, how new approaches can be added to the framework.

A. Architecture and Design Concepts

The architecture of RevKit is briefly illustrated in Fig. 4. As can be seen, RevKit consists of three main parts:

- the core, which provides data-structures (e.g. to store functions or circuits) and basic functionality (like parsing routines, export functions, cost calculations, circuit modifications) which can be used by every algorithm,
- the respective approaches and methods for reversible circuit design (e.g. synthesis, optimization, or verification), and
- the different applications built on top of the framework (e.g. the generic usage by means of the Python bindings or a concrete application that combines some algorithms in a certain way).

Additionally, RevKit makes use of third-party libraries like e.g. the *Colorado University Decision Diagram Package* (CUDD) [24], the SAT solver *MiniSAT* [5], and some C++-libraries.

The core and the corresponding algorithms form the main implementation of the framework. The respective algorithms are completely independent from each other, but rely on generic interfaces. In doing so, it is possible to utilize existing methods without a detailed treatment of them. For example, if a new optimization approach based on re-synthesis is added, the respective synthesis calls would be invoked by the generic interface. At run-time (or in a concrete application), the respective synthesis approach can then be chosen by parameters (denoted by the dashed arrow in Fig. 4). This enables a huge flexibility since the new optimization approach does not only rely on one single synthesis method, but can exploit all available ones. This also includes synthesis approaches that will be added in the future. Furthermore, this modular structure (together with the interfaces) has the great advantage that newly added methods do not affect already implemented functionality. In fact, even removing one approach will not affect the overall framework from compiling and operating.

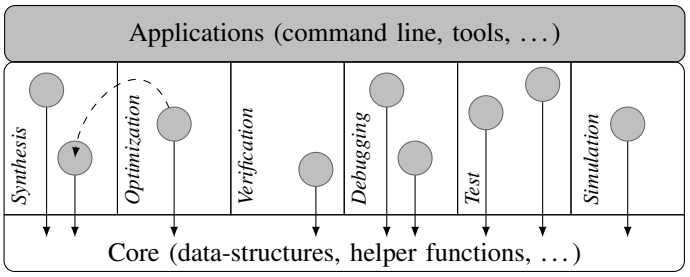

Fig. 4. Architecture of RevKit

Besides that, being prepared for future developments was an important design criterion during the implementation of RevKit. This can be illustrated very well by the support of the respective gate libraries for the considered circuits. So far, RevKit supports the established Toffoli gate, the Fredkin gate, and the Peres gate as well as the quantum V gate and the quantum V+ gate. However, in the future other gate types may be used. This would not only affect the data-structures of RevKit, but would also have implications for many approaches like simulation or verification. In order to keep RevKit flexible, generic structures are applied as well. More precisely, a generic data-structure including a so called target tag is used. These target tags can be defined separately without modifying the core of the framework. Having these target tags, new gate types can be easily supported by extending or overriding the concerned methods. For example, in the case of simulation, only the treatment of a single gate has to be extended while the overall simulation engine can remain unaltered.

The usage of these design concepts ensures a high extendability of the framework. Furthermore, several scripts are provided to aid developers in creating new algorithms from scratch. In particular, they generate basic code skeletons in order to allow an easy integration of new approaches and to make existing algorithms accessible. The next section illustrates this by means of an example.

B. Adding a New Approach to RevKit

Fig. 5 shows the complete source code of an optimization approach that can be added to RevKit in this form. In fact, a window optimization approach is thereby realized, where sub-circuits are considered from left to right. In each iteration, the currently considered sub-circuit is re-synthesized. If a sub-circuit with smaller cost results, the newly generated sub-circuit is substituted with the original one.

In the first lines of Fig. 5, the respective parameters are given, i.e. the resulting circuit (*circ*), the original circuit (*base*), and some settings (*settings*), which are parsed into local variables in lines 4–7. As can be seen, the simulation and the synthesis approach are passed by settings and stored in respective variables. As discussed above, this employs a generic interface, i.e. no concrete simulation or synthesis approach is invoked but defined from outside when calling the window optimization algorithm. Then, the original circuit is traversed from left to right (line 10) and a sub-circuit of a certain size (defined in the settings) is extracted and stored in *s* (lines 12–16). Afterwards, the function of the considered sub-circuit is extracted (lines 18–19) and passed to the synthesis approach (lines 21–22). Finally, the costs of the sub-circuits

```
1  bool window_optimization(circuit& circ, const circuit& base,
2      properties::ptr settings)
3  {
4    unsigned window_length = get(settings, "window_length");
5    simulation_func simulation = get(settings, "simulation");
6    truth_table_synthesis_func synthesis = get(settings, "synthesis");
7    cost_function cf = get(settings, "cost_function");
8
9    unsigned pos = 0u;
10   while (pos < base.num_gates())
11   {
12     unsigned length = std::min(
13         window_length, base.num_gates() − pos);
14     unsigned to = pos + length;
15
16     subcircuit s(base, pos, to);
17
18     binary_truth_table spec;
19     circuit_to_truth_table(s, spec, simulation);
20
21     circuit new_part;
22     bool ok = synthesis(new_part, spec);
23
24     bool cheaper = ok && costs(new_part, cf) < costs(s, cf);
25
26     append_circuit(circ, cheaper ? new_part : s);
27
28     pos = to;
29   }
30
31   return true;
32 }
```

Fig. 5. Sources for a simple optimization approach

are compared in line 24 (using a cost function again specified in the settings). If the newly synthesized sub-circuit is cheaper than the original one, then this new one is appended to the resulting circuit. Otherwise, the original sub-circuit is used (line 26).

As can be seen, using RevKit this approach can be implemented in a very compact and straight-forward way. Existing approaches (in this case synthesis methods) are utilized. Furthermore, the resulting approach is very flexible since both, the synthesis method and the considered cost function, can be arbitrarily selected.

IV. Conclusions

In this paper, we introduced RevKit, a toolkit for reversible circuit design. The aim of RevKit is to make recent approaches in this domain publicly available as well as to provide a flexible and easily extendable framework for future developments. RevKit is online available at http://www.revkit.org. Any feedback is welcome via revkit@informatik.uni-bremen.de.

Acknowledgments

This work was supported by the German Research Foundation (DFG) (DR 287/20-1).

References

[1] A. Barenco, C. H. Bennett, R. Cleve, D. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[2] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev*, 17(6):525–532, 1973.

[3] R. Cuykendall and D. R. Andersen. Reversible optical computing circuits. *Optics Letters*, 12(7):542–544, 1987.

[4] B. Desoete and A. D. Vos. A reversible carry-look-ahead adder using control gates. *INTEGRATION, the VLSI Jour.*, 33(1-2):89–104, 2002.

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

[6] K. Fazel, M. Thornton, and J. Rice. ESOP-based Toffoli gate cascade generation. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 206 –209, 2007.

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

[8] S. Gay, R. Nagarajan, and N. Papanikolaou. QMC: A model checker for quantum systems. In *Computer Aided Verification*, pages 543–547, 2008.

[9] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple control Toffoli network synthesis with SAT techniques. *IEEE Trans. on CAD*, 28(5):703–715, 2009.

[10] P. Gupta, A. Agrawal, and N. K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 25(11):2317–2330, 2006.

[11] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183, 1961.

[12] D. Maslov and G. Dueck. Improved quantum cost for n-bit Toffoli gates. *Electronics Letters*, 39(25):1790 – 1791, 11 2003.

[13] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD*, 24(6):807–817, 2005.

[14] R. C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4:21–40, 1993.

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

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

[17] K. N. Patel, J. P. Hayes, and I. L. Markov. Fault testing for reversible circuits. *IEEE Trans. on CAD*, 23(8):1220–1230, 2004.

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

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

[20] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 22(6):710–722, 2003.

[21] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Foundations of Computer Science*, pages 124–134, 1994.

[22] M. Soeken, R. Wille, and R. Drechsler. Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition. In *Workshop on Reversible Computation*, 2010.

[23] M. Soeken, R. Wille, G. W. Dueck, and R. Drechsler. Window optimization of reversible and quantum circuits. In *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010.

[24] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.

[25] H. Thapliyal and M. B. Srinivas. The need of DNA computing: reversible designs of adders and multipliers using Fredkin gate. In *SPIE*, 2005.

[26] 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.

[27] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414:883, 2001.

[28] 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.

[29] G. F. Viamontes, I. L. Markov, and J. P. Hayes. *Quantum Circuit Simulation*. Springer, 2009.

[30] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Automation Conf.*, pages 270–275, 2009.

[31] R. Wille, D. Große, G. Dueck, and R. Drechsler. Reversible logic synthesis with output permutation. In *VLSI Design*, pages 189–194, 2009.

[32] R. Wille, D. Große, D. M. Miller, and R. Drechsler. Equivalence checking of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 324–330, 2009.

[33] 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, 2008. RevLib is available at http://www.revlib.org.

[34] R. Wille, M. Soeken, and R. Drechsler. Reducing the number of lines in reversible circuits. In *Design Automation Conf.*, 2010.