RevKit: An Open Source Toolkit for the Design of Reversible Circuits

Mathias Soeken[1], Stefan Frehse[1], Robert Wille[1], and Rolf Drechsler[1,2]

[1] Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2] Cyber-Physical Systems, DFKI GmbH, 28359 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 e.g. for synthesis, optimization, simulation, verification, and test. The open source toolkit RevKit is an attempt to make these developments publicly available to other researchers. For this purpose, a modular and extendable framework has been provided which easily enables the addition of new methods and tools.

In this paper, we introduce the functionality as well as the internals of RevKit. We provide examples and use cases showing how to apply RevKit and its components in order to create and execute customized design flows. Furthermore, we demonstrate how the architecture and the design concepts of RevKit can be exploited to easily develop new or improved methods for reversible circuit design.

1 Introduction

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 need for more computational power, alternatives are required that go beyond the scope of the conventional (CMOS) technologies. Reversible logic marks a promising new direction where all operations are performed in an invertible manner. That is, in contrast to conventional 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, conventional computer chips, e.g. in the domain of low-power design [1,2,3], quantum computation [4,5,6], optical computing [7], DNA computing [8], as well as nanotechnologies [9].

A. De Vos and R. Wille (Eds.): RC 2011, LNCS 7165, pp. 64–76, 2012.

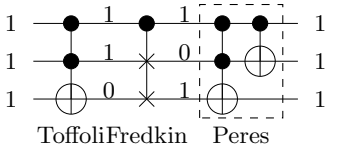

Fig. 1. Reversible gates

The basic concepts of reversible logic are not new and were already introduced in the 60's by Landauer [1] and further refined by Bennett [2] and Toffoli [10]. 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 [10], Fredkin gates [11], and Peres gates [12]. Figure 1 shows these gates in a cascade. Each gate consists of control lines (denoted by $\bullet$) 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 the logic value 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.

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 by their input signals only and do not need additional power supplies (see e.g. [3]). In quantum computation, factorization has been solved in polynomial time whereas only exponential solving methods are known for conventional circuits (see e.g. [4,6]). 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. [13,14,15]), optimization (see e.g. [14]), simulation (e.g. [16]), verification (e.g. [17,18,19]), and test (e.g. [20,21]).

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.

[1] Exceptions are e.g. the RMRLS synthesis approach [22] which is available at `http://www.princeton.edu/~cad/projects.html` or the quantum simulator QuIDDPro [16] which is available at `http://vlsicad.eecs.umich.edu/Quantum/qp/`

The open source toolkit RevKit is an attempt to make these developments publicly available to other researchers. For this purpose, a modular and extendable framework has been 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 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 this paper, we introduce the functionality as well as the internals of RevKit. We provide examples and use cases showing how to apply RevKit and its components in order to create and execute customized design flows. The paper is structured as follows. First, RevKit and the main approaches are briefly reviewed in the next section. Section 3 illustrates the application of RevKit by means of the Python interface and by means of a graphical user interface. Afterwards, the the architecture as well as the design concepts of RevKit are introduced in Sect. 4 enabling to easily extend or improve the framework with further functionality. Section 5 concludes the paper.

2 The RevKit Framework

RevKit is an open source toolkit available at www.revkit.org which aims to make recent developments in the domain of reversible circuit design accessible to other researchers. It provides core functionality like read-in routines for functions and reversible circuits (based on the RevLib format introduced in [23]), several export functions (again into the RevLib format, but LaTeX and BLIF dumps are also available), cost calculations, and more. Furthermore, more elaborated methods for synthesis, optimization, and verification of reversible (and quantum) circuits are available including:

Synthesis

- A transformation-based method inspired by the concepts of [24] and the extension based on the Reed Muller spectra [25]
- The BDD-based synthesis method as introduced in [15]
- The KFDD-based synthesis method as introduced in [26]
- The heuristic synthesis with output permutation method as introduced in [27]
- The ESOP-based synthesis method inspired by the concepts of [28]
- The exact synthesis method as introduced in [29]

Optimization

- The window optimization method as introduced in [30]
- The circuit line reduction method as introduced in [31]
- The adding lines optimization method as introduced in [32]

Verification

- The SAT-based equivalence checker as introduced in [19]

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 [33] and [34]
- Support of hierarchical circuitry (i.e. modules, flattening of circuits, etc.), sequential circuits, annotations, and more
- Visualization of circuits

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, all functions are also exposed as a Python library[2] as well as in a graphical user interface. This enables the user to create and execute customized design flows as illustrated in the next sections.

3 The Users' Perspective: Applying RevKit

Accessing the API of RevKit using C++ requires to write fully executable C++ programs which need to be compiled after every modification. In particular when using the toolkit for the purpose of evaluation and experimentation, this work flow is very inflexible.

To overcome this limitation, RevKit offers bindings of all functions and algorithms either to the Python language or to a graphical user interface. This allows to utilize RevKit without re-compilation. At the same time, the high performance of the algorithms is exploited since both, the Python binding as well as the graphical user interface, directly invoke the respective assembly code.

3.1 Using the Python Interface

In this section, the advantages of the Python bindings are demonstrated by means of two use cases. First, an interactive application of the Python shell is outlined. Afterwards, it is shown how to utilize the expressive Python syntax in order to create compact scripts defining a customized design flow.

Interactive Application in the Python Shell. A Python shell can be utilized enabling a dynamic interaction with the RevKit functions and algorithms. Furthermore, sophisticated Python shells such as *IPython* [35] additionally allow syntax highlighting, tab completion, UNIX shell interaction, and integrated documentation.

[2] For this purpose, the Boost.Python library was utilized. For further information visit http://www.boost.org/doc/libs/1_47_0/libs/python/doc/index.html

```
$ ipython
In [1]: from revkit import *
In [2]: circ = circuit(2)
In [3]: append_not(circ, 0)
Out[3]: <revkit_python.gate object at 0xb7348454>
In [4]: append_cnot(circ, 0, 1)
Out[4]: <revkit_python.gate object at 0xb734848c>
In [5]: circ
Out[5]:
0*
-0
In [6]: init_gui()
Out[6]: <PyQt4.QtGui.QApplication object at 0xb72be26c>
In [7]: w = display_circuit(circ)
In [8]: w.simulate([False, True])
```

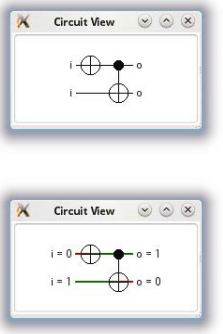

Fig. 2. Command line interface

As an example, consider the command line flow as outlined in Fig. 2. After the RevKit library is imported (see Command 1), a circuit consisting of a NOT and a CNOT gate is created (see Command 2 for the initialization of the circuit as well as Command 3 and Command 4 for the addition of the gates). Then, the resulting circuit is printed out on the console (Command 5), displayed in the GUI (Command 7), and simulated (Command 8). For this purpose, the last two commands open the GUI as shown on the right-hand side of Fig. 2.

Overall, using RevKit in the Python shell, the user directly gets feedback for the invoked actions. Thus, it is ideal e.g. for a first examination in order to observe the behavior of different design flows.

Python Scripts. An alternative to the interactive application is the use of scripts. They enable e.g. to define sequences of commands that, afterwards, can be executed on several instances, several times, or with different parameters.

As an example, Fig. 3 shows a Python script that creates an incrementer circuit and verifies it using exhaustive simulation. After importing the RevKit Python library (Line 3), a helper function is defined which maps a list of Boolean numbers to its natural representation (Line 5). The syntax can almost directly be mapped to the formula $\sum_{b_i} b_i \cdot 2^i$ for a $\boldsymbol{b} = (b_0 \dots b_{n-1})$. The size of the circuit is configurable by a program argument and defined in Line 7. The incrementer structure of the circuit is built in Lines 8 and 9 by prepending a gate with a target on line c and control lines on all preceding lines. In order to verify the correctness of this circuit, the truth table of it is created (Line 11/12). Then, for each line of the truth table it is checked whether it adheres the specification (Line 13). More precisely, it is checked whether adding 1 to each input value results in the desired output value. In case the script generates no output, the verification was successful. Otherwise, an assertion is thrown which can be further inspected, e.g. by checking the respective values for the variables `_in` and `_out`.

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

def b2d(bits): return sum([b * 2**i for i, b in enumerate(bits)])

n = int(sys.argv[1])
circ = circuit(n)
for c in range(n): prepend_toffoli(circ, range(c), c)

spec = binary_truth_table()
circuit_to_truth_table(circ, spec)
for [_in, _out] in spec.entries: assert((b2d(_in) + 1) % 2**n == b2d(_out))
```

Fig. 3. Python script

Overall, using the RevKit bindings and the syntactical features of Python, scripts also for complex tasks can be written within few lines of code.

3.2 Using the Graphical User Interface

Besides the Python library, also a *graphical user interface* (GUI) is available in RevKit. This enables the creation and execution of customized design flows without writing any line of code. Instead, the respective steps of a design flow to be executed can easily put together by means of blocks to be connected by a graph. Each block performs an operation and may have ports for the respective input parameters and output results. Input ports can be connected to output ports forming a channel when they support the same data types.

As an example, a *Circuit from file* block reads a circuit from a given file-name and passes the resulting data-structure to its single output port of type *Circuit*. Then, this block can be connected to a *Line Reduction* block which takes this circuit as a parameter and performs the line reduction approach [31]. Afterwards, the result is provided at the output port of this block. In this manner, more complex scenarios can be set up.

When executing the design flow, the graph is sorted in a topologically order and is executed level wise. Visual feedback provides the user with current progress information, i.e. which steps have already been performed and which step is currently being executed. In the following, two use cases illustrating possible applications of the RevKit GUI are presented.

Building Custom Design Flows. An example flow is given in Fig. 4. Here, a reversible function given as truth table is synthesized utilizing a heuristic [24] as well as an exact [29] approach. Afterwards, the resulting circuits are checked for equivalence. Besides that, the results of each synthesis run are passed to a statistics element which provides information e.g. about the circuit cost and also visualizes the circuit.

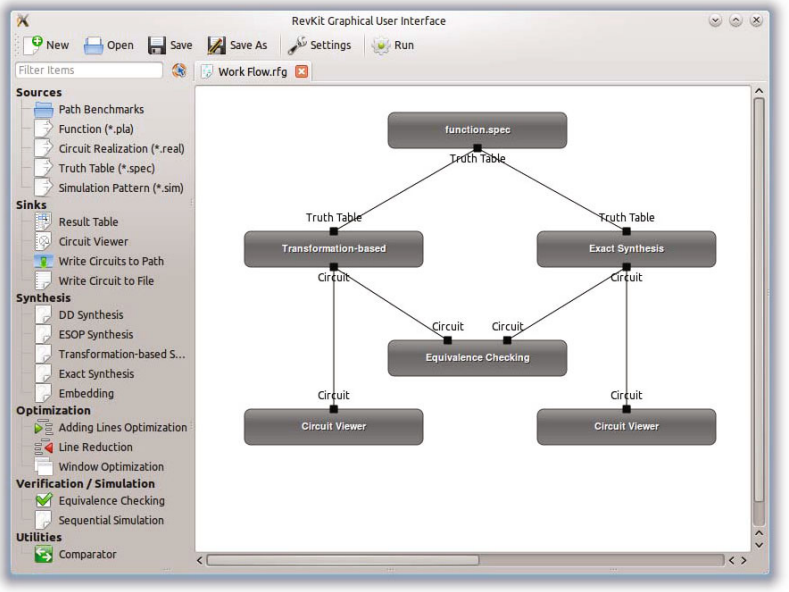

Fig. 4. Example GUI execution

Benchmarking. The elementary blocks in the RevKit GUI are of different complexity. While some provide very basic operations such as parsing files, more powerful blocks exist. As an example, the block *RevLib Functions* provides access to the RevLib [23] database. A respective block is depicted in its expanded form in Fig. 5. The table on the left-hand side lists all benchmarks that meet certain criteria specified on the right-hand side, i.e. functions with more than 5 but less than 8 inputs as well as functions with more than 3 outputs. When executing this block, all these functions can be passed to the successive blocks. Therewith, a whole set of functions can sequentially be applied e.g. to a synthesis approach. The results of such a process can afterwards be collected in another block which enables to export a result table in terms of a CSV or LaTeX file.

4 The Developers' Perspective: Extending RevKit

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

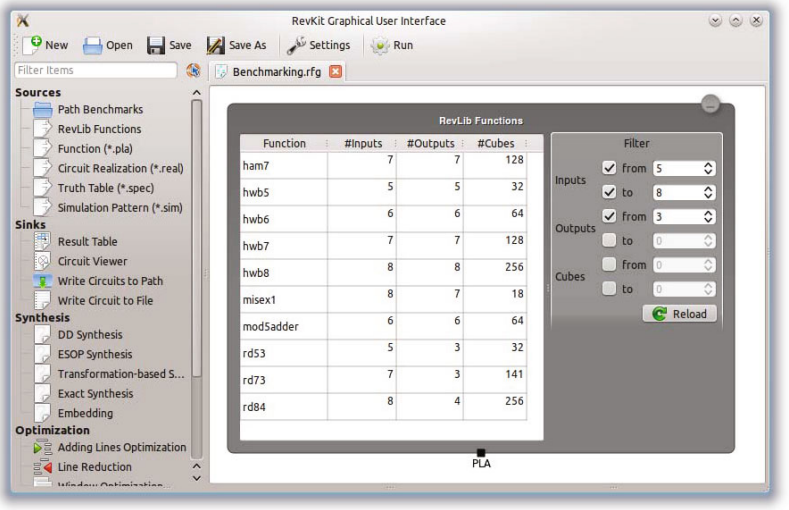

Fig. 5. Benchmarking example

4.1 Architecture and Design Concepts

The architecture of RevKit is briefly illustrated in Fig. 6. 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 precise 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) [36], the metaSMT framework [37], 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 precise application), the respective synthesis approach can then be chosen by parameters (denoted by the dashed arrow in Fig. 6). 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

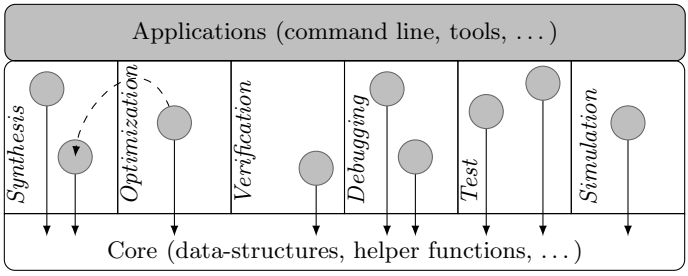

Fig. 6. Architecture of RevKit

will be added in the future. Furthermore, this modular structure (together with the interfaces) has the 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.

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.

4.2 Adding a New Approach to RevKit

Figure 7 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 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. 7, 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

```
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. 7. Sources for a simple optimization approach

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 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.

5 Conclusions

In this paper, we reviewed the functionality as well as the internals of RevKit and provided examples and use cases showing how to apply RevKit and its components in order to design reversible circuits. For this purpose, several interfaces provided by RevKit (Python bindings, the graphical user interface, or the C++-API) can be exploited. RevKit itself as well as further documentation is available at www.revkit.org.

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

References

1. Landauer, R.: Irreversibility and Heat Generation in the Computing Process. IBM Journal of Research and Development 5(3), 183–191 (1961)
2. Bennett, C.H.: Logical Reversibility of Computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
3. Desoete, B., DeVos, A.: A reversible carry-look-ahead adder using control gates. Integration 33(1-2), 89–104 (2002)
4. Shor, P.W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: Symp. on Foundations of Computer Science, pp. 124–134. IEEE Computer Society (November 1994)
5. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, New York (2000)
6. Vandersypen, L.M.K., Steffen, M., Breyta, G., Yannoni, C.S., Sherwood, M.H., Chuang, I.L.: Nature 414, 883–887 (2001)
7. Cuykendall, R., Andersen, D.R.: Reversible optical computing circuits. Optical Letters 12(7), 542–544 (1987)
8. Thapliyal, H., Srinivas, M.B.: The need of DNA computing: Reversible designs of adders and multipliers using Fredkin gate. In: Proceedings of SPIE, vol. 6050 (December 2005)
9. Merkle, R.C.: Reversible electronic logic using switches. Nanotechnology 4(1), 21–40 (1993)
10. Toffoli, T.: Reversible Computing. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 632–644. Springer, Heidelberg (1980)
11. Fredkin, E., Toffoli, T.: Conservative logic. Int'l. Journal of Theoretical Physics 21(3), 219–253 (1982)
12. Peres, A.: Reversible logic and quantum computers. Phys. Rev. A 32(6), 3266–3276 (1985)
13. Shende, V., Prasad, A., Markov, I., Hayes, J.: Synthesis of reversible logic circuits. IEEE Trans. on CAD 22(6), 710–722 (2003)

14. Maslov, D., Dueck, G.W., Miller, D.M.: Toffoli network synthesis with templates. IEEE Trans. on CAD 24(6), 807–817 (2005)
15. Wille, R., Drechsler, R.: BDD-based synthesis of reversible logic for large functions. In: Design Automation Conference, pp. 270–275. ACM (July 2009)
16. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Quantum Circuit Simulation. Springer, Heidelberg (2009)
17. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Checking equivalence of quantum circuits and states. In: Int'l Conf. on Computer-Aided Design, pp. 69–74. IEEE (November 2007)
18. Gay, S.J., Nagarajan, R., Papanikolaou, N.: QMC: A Model Checker for Quantum Systems. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 543–547. Springer, Heidelberg (2008)
19. Wille, R., Große, D., Miller, D.M., Drechsler, R.: Equivalence Checking of Reversible Circuits. In: Int'l. Symp. on Multiple-Valued Logic, 324–330. IEEE Computer Society (May 2009)
20. Polian, I., Fiehn, T., Becker, B., Hayes, J.P.: A Family of Logical Fault Models for Reversible Circuits. In: Asian Test Symposium, pp. 422–427. IEEE Computer Society (December 2005)
21. Patel, K.N., Hayes, J.P., Markov, I.L.: Fault testing for reversible circuits. IEEE Trans. on CAD 23(8), 1220–1230 (2004)
22. Gupta, P., Agrawal, A., Jha, N.K.: An Algorithm for Synthesis of Reversible Logic Circuits. IEEE Trans. on CAD of Integrated Circuits and Systems 25(11), 2317–2330 (2006)
23. Wille, R., Große, D., Teuber, L., Dueck, G.W., Drechsler, R.: RevLib: An Online Resource for Reversible Functions and Reversible Circuits. In: Int'l Symp. on Multiple-Valued Logic, pp. 220–225 (May 2008)
24. Miller, D.M., Maslov, D., Dueck, G.W.: A transformation based algorithm for reversible logic synthesis. In: Design Automation Conference, pp. 318–323. ACM (June 2003)
25. Maslov, D., Dueck, G.W., Miller, D.M.: Techniques for the synthesis of reversible Toffoli networks. ACM Trans. Design Autom. Electr. Syst. 12(4), 42:1–42:28 (2007)
26. Soeken, M., Wille, R., Drechsler, R.: Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition. In: Int'l Design and Test Workshop, pp. 143–148 (December 2010)
27. Wille, R., Große, D., Dueck, G.W., Drechsler, R.: Reversible Logic Synthesis with Output Permutation. In: Int'l Conf. on VLSI Design, pp. 189–194. IEEE (January 2009)
28. Fazel, K., Thornton, M., Rice, J.: ESOP-based Toffoli Gate Cascade Generation. In: IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, pp. 206–209. IEEE (August 2007)
29. Große, D., Wille, R., Dueck, G.W., Drechsler, R.: Exact Multiple-Control Toffoli Network Synthesis With SAT Techniques. IEEE Trans. on CAD 28(5), 703–715 (2009)
30. Soeken, M., Wille, R., Dueck, G.W., Drechsler, R.: Window optimization of reversible and quantum circuits. In: Int'l Symp. on Design and Diagnostics of Electronic Circuits and Systems, pp. 341–345 (April 2010)
31. Wille, R., Soeken, M., Drechsler, R.: Reducing the number of lines in reversible circuits. In: Design Automation Conference, pp. 647–652. ACM (June 2010)
32. Miller, D.M., Wille, R., Drechsler, R.: Reducing reversible circuit cost by adding lines. In: Int'l Symp. on Multiple-Valued Logic, pp. 217–222 (May 2010)

33. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Phys. Rev. A 52(5), 3457–3467 (1995)
34. Maslov, D., Dueck, G.: Improved quantum cost for n-bit toffoli gates. Electronics Letters 39(25), 1790–1791 (2003)
35. Pérez, F., Granger, B.E.: Ipython: A system for interactive scientific computing. Computing in Science and Engineering 9(3), 21–29 (2007)
36. Somenzi, F.: CUDD: CU Decision Diagram Package Release 2.3.1. University of Colorado at Boulder (2001), CUDD, `vlsi.colorado.edu/~fabio/CUDD/`
37. Haedicke, F., Frehse, S., Fey, G., Große, D., Drechsler, R.: metaSMT: Focus on Your Application not on Solver Integration. In: Int'l Workshop on Design and Implementation of Formal Tools and Systems (November 2011)