# Reversible Computation
## An Alternative Computation Paradigm for Low Power Applications

Rolf Drechsler Robert Wille
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
Cyber Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{drechsle,rwille}@informatik.uni-bremen.de

***Abstract*—The development of low power systems gained significant importance and impressive progress has been made in this domain in the recent years. However, despite more efficient circuit technologies (made possible e.g. by the ongoing miniaturization of integrated circuits) as well as improvements in battery technology, also the way *how* computations are logically performed may have an effect on the required power consumption. In this invited paper, we consider reversible computation – an alternative computation paradigm which inherits certain characteristics and properties that may be of benefit for low power design. We review possible impacts to future developments and show how reversible computations can already been exploited today. Finally, we sketch design challenges which, besides other physical and electrical issues, still prevent the full exploitation of this computation paradigm.**

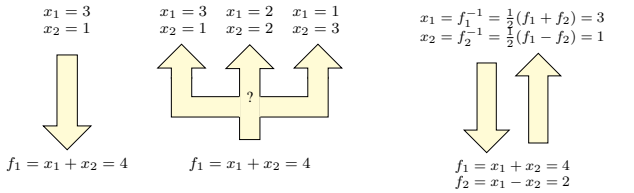
## I. Introduction

In the modern world, computation devices are found everywhere. Most visible are the ubiquitous desktop and laptop computers, but the vast majority of computation devices are actually embedded in everything from children's toys to smartphones. Common to many of these embedded devices is that (1) they rely on a battery or other low-power connections, (2) they are designed for a specialized application rather than for general computation, and (3) they operate with little-to-no external input for long periods of time.

A common challenge for all such computation devices (and computation devices in general) is, therefore, the power consumption associated with their use. This has two major impacts: Firstly, the total power consumption from areas related to information and communication technology amounts to a significant portion of the total power budget. Secondly, high power consumption greatly reduces usability and convenience in power-limited contexts, e.g. battery powered devices often need to be charged (e.g. smartphones almost daily) or require a change of batteries (e.g. hearing aids every week).

Recent years have shown tremendous improvements in this respect. The ongoing miniaturization of integrated circuits have led to a continual decrease in the power dissipation per computational step, while improvements in battery technology (e.g. the invention of the Lithium ion battery) have increased the power density of batteries. These complementary developments are the main reasons why devices today can perform many computation-intensive functions and still remain portable and convenient.

But despite that, also the fashion *how* we logically perform computations may have an effect on the required power consumption. In fact, researchers and engineers narrowed the investigation of computation machines down to a preponderantly *irreversible* computation paradigm: 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. This may lead to drawbacks when it comes to the development of (future) power-efficient computation devices.

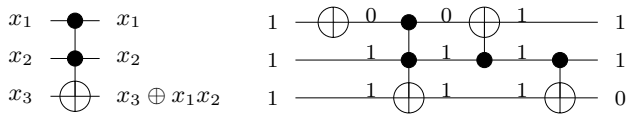



(a) Irreversible computation (b) Reversible computation

Fig. 1: Computation paradigms

In contrast, *reversible* computation is an alternative computation paradigm which only allows for bijective operations, i.e. reversible $n$-input $n$-output functions that map each possible input vector to a unique output vector. The underlying idea of reversible computation is exemplarily illustrated in Fig. 1a by means of a simple addition. Performing solely the addition leads to an information loss and makes it impossible to undo the calculation without knowing the original inputs. Instead, if the addition is realized as shown in Fig. 1b computations can be performed in a reversible fashion, i.e. from the inputs to the outputs and vice versa.

Albeit not so well established yet, this computation paradigm may allow for several promising applications in the domain of low power design. In this invited paper, we aim for providing an overview of these prospects. To this end, we first briefly review the basic principles of reversible computation by means of *reversible circuits*. Afterwards, we discuss in Section III the general impact reversible computation may have on the development of future low power circuits and systems. How to exploit reversible computations already today is discussed by means of the design of encoders for on-chip interconnects in Section IV. Finally, design questions raised by this unconventional computation paradigm are briefly discussed in Section V before the paper is concluded in Section VI.



(a) Toffoli gate (b) Toffoli circuit

Fig. 2: Toffoli gate and Toffoli circuit

## II. Reversible Circuits

A Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ over the variables $X := \{x_1, \ldots, x_n\}$ is *reversible* iff (1) its number of inputs is equal to the number of outputs (i.e. $n = m$) and (2) it maps each input pattern to a unique output pattern. Reversible functions are realized by reversible circuits. A *reversible circuit* $G$ is a cascade of reversible gates, where fanout and feedback are not directly allowed [2]. Each variable of the function $f$ is thereby represented by a *circuit line*, i.e. a signal through the whole cascade structure on which the respective computation is performed. Computations are performed by *reversible gates*. In the literature, reversible circuits composed of *Toffoli gates* are frequently used. A Toffoli gate is composed of a (possibly empty) set of *control lines* $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ and a single *target line* $x_j \in X \setminus C$. The Toffoli gate inverts the value on the target line if all values on the control lines are assigned to 1 or if $C = \emptyset$, respectively. All remaining values are passed through unaltered.

Fig. 2a shows a Toffoli gate drawn in standard notation, i.e. control lines are denoted by ●, while the target line is denoted by $\oplus$. A circuit composed of several Toffoli gates is depicted in Fig. 2b. This circuit maps e.g. the input 111 to the output 110 and vice versa.

## III. Possible Impact of Reversible Computation

As mentioned above, the ongoing miniaturization of integrated circuits as well as improvements in battery technology are the main reasons why today's circuits and systems can perform many computation-intensive functions and still remain portable as well as convenient. However, these developments cannot continue indefinitely and, indeed, a particular limit of conventional computation technology will be reached in the near future.

In fact, all of today's computation devices inherit a non-technology specific and fundamental limit to power consumption – called the *Landauer limit* – common to all conceivable computation processes. Pioneering work by Landauer [3] showed that, regardless of the underlying technology, each "lost" bit of information causes a power dissipation amounting to at least $k \cdot T \cdot \log(2)$ Joules (where $k$ is the Boltzmann constant and $T$ is the temperature). In other words, whenever a device performs a computation that loses information, it *must* use some amount of power greater than this lower bound. Furthermore, all of the computation devices that surround us, regardless of their purpose, are built of elementary computation devices or *gates* (`AND`, `OR`, `NAND`, etc.). They all intrinsically lose information, and, thus, are *individually* subject to this principle. Each elementary gate in current computation devices must dissipate this power and a single computer chip can contain billions of such gates. This has been experimentally confirmed recently in [4].

Although the theoretical lower bound on power dissipation still does not constitute a significant fraction of the power consumption of current devices, it nonetheless poses both an obstacle and a possibility for the future. Fig. 3 illustrates the development of the power consumption of an elementary computational step in recent and expected future CMOS generations (based on values from the *International Technology Roadmap of Semiconductors* [1]). The figure shows that today's technology is still far away from the Landauer limit. However, a simple extrapolation also shows that the trend cannot continue with the current family of static CMOS gates as no amount of technological refinement can overcome the Landauer barrier.

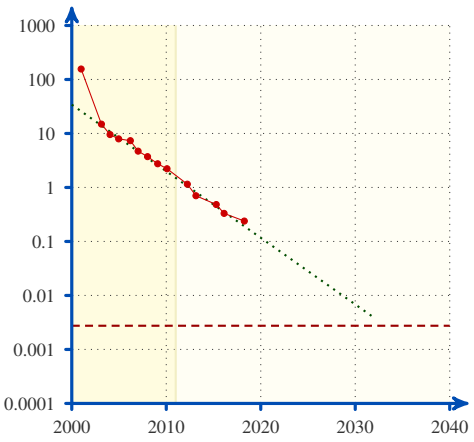
Additionally, the Landauer limit is only a *lower* bound on the dissipation. Gerschenfeld has shown that the actual power dissipation corresponds to the amount of power used to represent the signal [5]. The power used to represent a signal in CMOS is significantly higher than the lower limit. This in turn means that for conventional devices the Landauer barrier is closer than immediately implied by the extrapolation from Fig. 3.

Reversible circuits are a solution to avoid information loss in computation processes and, thus, to overcome this barrier. This computation model ensures that data is bijectively transformed at each computation step, which intrinsically avoids information loss. This circumvents Landauer's principle, removing the lower barrier to power dissipation if reversible elementary devices are used. Furthermore, the dissipation of signal power as a by-product of information loss should also be avoidable, meaning that there is a significant potential for power savings by going reversible, even when using the CMOS technology available today.

First implementations and fabrications of reversible logic in CMOS have indeed been accomplished (e.g. [6]). These exploit that reversible logic is particularly suited for both (1) when it comes to reuse of signal power (in contrast to static CMOS logic that dissipates the power in each gate) and (2) when using adiabatic switching [7], [6] to switch transistors in a more power efficient way. In fact, SPICE simulations of reversible circuits have shown that such implementations have the potential to reduce power consumption by a factor of ten [8]. However, a proof-of-concept in a "real", i.e. practical, context is still open. Besides that, it has to be considered that possible improvements in power consumptions are usually obtained at the expense of the execution speed.

## IV. Reversible Computation in the Design of On-Chip Interconnect Encoders

Even if the concepts reviewed above represent a possible motivation for exploiting reversible computation in the domain of low power design, practical applications in this direction are not to be expected in the near future. However, in other dedicated domains, the reversible computation paradigm may already find good use already today. This particular holds for the design of on-chip interconnect encoders which is discussed in this section (based on [9]). For this purpose, we briefly review the respective background first, before we illustrate how reversible computation may help in this domain.




The figure illustrates the development of the power consumption of an elementary computational step in recent CMOS generations (based on values from [1]). The power consumption is thereby determined by $CV_t^2$, where $V_t$ is the threshold voltage of the transistors and $C$ is the total capacitance of the capacitors in the logic gate. The capacitance $C$ is directly proportional to $\frac{LW}{t}$, i.e. to the length $L$ and the width $W$ of the transistors. Reducing these sizes of transistors enables significant reductions in the power consumption as shown in the extrapolation. However, this development will reach a fundamental limit when power consumption is reduced to $k \cdot T \cdot \log(2)$ Joule.

Fig. 3: Power consumption $Q$ in different CMOS generations.

### A. Background

With the rise of very deep sub-micron and nanometric technologies, interconnects are increasingly affecting the overall power consumption, performance, and reliability of a chip. As a result, interconnect-centric design [10], a paradigm which sets the *Interconnect Architecture* in the center of the design process, is gaining relevance. To address these issues, concepts from communication system engineering are getting introduced and adapted for on-chip communication architectures. Networks-on-Chip and coding applications are two examples of this trend.

An established methodology is thereby the application of coding strategies in order to improve the on-chip interconnections [11], [12], [13], [14], [15]. Fig. 4 illustrates the basic idea. Instead of simply transmitting the desired data (see top of Fig. 4), the architectures are extended by an encoder and a decoder (as shown in the bottom of Fig. 4). Thanks to the additional flexibility provided by the coding, it is possible to achieve important improvements on the overall communication architecture and additionally consider the trade-off between power consumption, delay, and reliability.

In the past, the first broad use of (on-chip) coding was aimed at reducing the power consumption [11], [12]. As technology improved, coupling capacitances as well as the related timing and noise issues had to be addressed. This led to *Coupling Aware Codes* [13], [16], [15]. Recently, reliability is emerging as an important issue where coding plays a major role [16], [17]. A *key issue* in any case is the reduction of the overhead caused by the additional hardware of the encoder and decoder. This poses an important design challenge.

### B. Exploitation of Reversible Computations

Thus far, all the encoders and decoders proposed in the past for the scenario sketched above have been determined by means of design methods following a conventional, i.e. irreversible, computation paradigm. Design methods for reversible computation may provide a promising alternative. In order to illustrate that, consider the following example in which a coding realizing a so-called *probability based mapping* (pbm, [11]) shall be derived, i.e. a coding which links the most frequently occurring data inputs to patterns with a low Hamming weight.

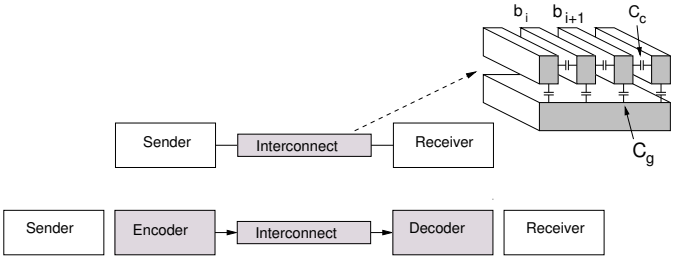



Fig. 4: Optimization of interconnect architect. through coding

TABLE I: Illustrating the objective of an encoder

(a) Pattern probability

| Inputs | Prob. |
|---|---|
| 000 | 8% |
| 001 | 8% |
| 010 | 10% |
| 011 | 10% |
| 100 | 40% |
| 101 | 10% |
| 110 | 8% |
| 111 | 6% |

(b) Desired encoding

| Inputs | Weight ($H$) |
|---|---|
| 000 | 2 |
| 001 | 2 |
| 010 | 1 |
| 011 | 1 |
| 100 | 0 |
| 101 | 1 |
| 110 | 2 |
| 111 | 3 |

(c) Possible encoding

| Inputs | Encoding |
|---|---|
| 000 | 101 |
| 001 | 011 |
| 010 | 010 |
| 011 | 001 |
| 100 | 000 |
| 101 | 100 |
| 110 | 110 |
| 111 | 111 |

**Example 1.** *Table Ia shows a set of data inputs with their corresponding probability of occurrence. Based on that, an encoder should map the most frequently occurring data input (i.e. 100) to a bit-string with the lowest Hamming weight (i.e. 000). Then, the second-most frequently occurring data inputs should be mapped to a bit-string with the second-lowest Hamming weight and so on. That is, a coding is desired which leads to patterns with Hamming weights as shown in Table Ib. A precise coding satisfying this property is given in Table Ic.*

However, determining the best possible or even just one "good" coding is a non-trivial task. Already for 3-bit data inputs as considered in Example 1, eight different codings are possible. In the general case with $m$-bit data inputs, this number significantly increases to $\prod_{i=0}^{m} \binom{m}{i}$. Motivated by this,

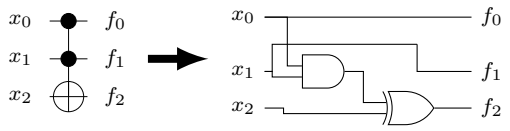



Fig. 5: Mapping Toffoli gate to conventional circuit

the problem remains how to automatically determine codings (as well as corresponding encoders and decoders) that map frequently occurring data inputs to patterns with low Hamming weight while keeping the hardware overhead as small as possible?

Since encoders realize reversible one-to-one mappings, the application of synthesis approaches for reversible logic is a reasonable choice. These methods realize circuits composed of Toffoli gates as introduced in Section II. But since Toffoli gates represent a logic description, they can easily be mapped to a conventional gate library. As an example, Toffoli gates with two control lines can be mapped to a netlist composed of one AND-gate and one XOR-gate as shown in Fig. 5. From such a netlist, the established optimization and technology mapping steps can be performed.

Following a reversible computation paradigm, several benefits can be exploited. In fact, encoders and decoders are realizing one-to-one mappings which are inherently reversible. Relying on design methods e.g. for reversible circuits avoids the explicit check whether a resulting design indeed realizes a one-to-one mapping – reversible circuits inherently guarantee this "feature", while conventional design methods always require an explicit check for that. Details how design methods for reversible circuits can be exploited for this purpose are provided in [9].

## V. Design Challenges

In order to exploit the potential of reversible computations in the domains sketched above, an efficient design flow must be available. For conventional computation, an elaborated design flow emerged over the last 20–30 years. Here, a hierarchical flow composed of several abstraction levels (e.g. specification level, electronic system level, register transfer level, and gate level) supported by a wide range of modeling languages, system description languages, and hardware description languages has been developed and is in (industrial) use. In contrast, the design of circuits and systems following the reversible computation paradigm is still in its infancy (overviews can be found in [18], [19]). Although the basic tasks, i.e. synthesis, verification, and debugging, have been considered by researchers[1], essential features and approaches of modern design flows are still missing. More precisely:

- Most of the existing approaches remain on the gate level. No real support of reversible circuits and systems on higher levels of abstractions are available.
- Most of the existing approaches for synthesis usually only accept specifications provided in terms of Boolean function descriptions like truth tables or Boolean decision diagrams (see e.g. [21], [22], [23], [24]). Recently, hardware description languages for reversible circuits have been introduced, but support a very basic set of operations only (see e.g. [25], [26]).
- For verification and validation simulation engines (e.g. [27]) and equivalence checkers (e.g. [28]) are available so far. But the most efficient methods can handle circuits composed of at most 20,000 gates only (while approaches for conventional circuits are able to handle hundreds of thousands of gates).
- Debugging has hardly been considered [29], [30].

[1] A variety of corresponding open source implementations are available in the tool *RevKit* [20].

TABLE II: Adder function and a possible embedding

(a) Adder function

| $c_{in}$ | x | y | $c_{out}$ | sum | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | **0** | **1** | **0** |
| 0 | 1 | 0 | **0** | **1** | **1** |
| 0 | 1 | 1 | *1* | *0* | *0* |
| 1 | 0 | 0 | **0** | **1** | **?** |
| 1 | 0 | 1 | *1* | *0* | *1* |
| 1 | 1 | 0 | *1* | *0* | *?* |
| 1 | 1 | 1 | 1 | 1 | 1 |

(b) Embedding

| 0 | $c_{in}$ | x | y | $c_{out}$ | **sum** | $g_1$ | $g_2$ |
|---|---|---|---|---|---|---|---|
| 0 | **0** | **0** | **0** | **0** | **0** | 0 | 0 |
| 0 | **0** | **0** | **1** | **0** | **1** | 1 | 1 |
| 0 | **0** | **1** | **0** | **0** | **1** | 1 | 0 |
| 0 | **0** | **1** | **1** | **1** | **0** | 0 | 1 |
| 0 | **1** | **0** | **0** | **0** | **1** | 0 | 0 |
| 0 | **1** | **0** | **1** | **1** | **0** | 1 | 1 |
| 0 | **1** | **1** | **0** | **1** | **0** | 1 | 0 |
| 0 | **1** | **1** | **1** | **1** | **1** | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| | | | | ... | | | |

A main reason for these open issues surely is the different computation paradigm itself which requires that even the simplest operation has to be reversible. As a representative of a problem to overcome when discussing reversible rather than conventional computation devices is illustrated in the following by means of the adder function shown in Table IIa.

This adder has three inputs (the carry-bit $c_{in}$ as well as the two summands $x$ and $y$) and two outputs (the carry $c_{out}$ and the *sum*). It surely belongs to one of the most important functions to be realized in terms of a circuit device. However, the adder obviously is not reversible (irreversible), since (1) the number of inputs differs from the number of outputs and (2) there is no unique input-output mapping. Even adding an additional output to the function (leading to the same number of input and outputs) would not make the function reversible. Then, the first four lines of the truth table can be embedded with respect to reversibility as shown in the rightmost column of Table IIa. However, since $c_{out} = 0$ and $sum = 1$ already appeared two times (marked bold), no unique embedding for the fifth truth table line is possible any longer. The same also holds for the lines marked italic.

This already has been observed in [31] and was further discussed in [32], [33]. There, the authors showed that at least $\lceil log(m) \rceil$ free outputs are required to make an irreversible function reversible, where $m$ is the maximum number of times an output pattern is repeated in the truth table. Since for the adder at most 3 output pattern are repeated, $\lceil log(3) \rceil = 2$ free outputs (and, hence, one additional circuit line) are required to make the function reversible.

Adding new lines causes constant inputs and garbage outputs. The value of the constant inputs can be chosen by the designer. Garbage outputs are by definition don't cares and thus can be left unspecified leading to an incompletely specified function. However, many synthesis approaches require a completely specified function so that all don't cares must be assigned with a concrete value.

As a result, the adder is embedded in a reversible function including four variables, one constant input, and two garbage outputs. A possible assignment to the constant as well as the don't care values is depicted in Table IIb. Note that the precise embedding may influence the respective synthesis results. Corresponding evaluations have been made e.g. in [34], [35]. Alternative approaches (e.g. [23]) address this embedding problem not explicitly, but implicitly and lead to a significant amount of additional circuit lines. As this may harm in turn the efficiency of the resulting circuit (last but not least with respect to the power consumption in a possible application), how to trade-off these issues remains a big design challenge in this domain (see e.g. [36]).

## VI. Conclusions

In this invited paper, we reviewed and discussed reversible computation as a promising alternative for low power applications. To this end, we reviewed the potential impact this computation paradigm may have to the development of future circuits and systems as well as sketched possible exploitations which could be used already today. However, besides the physical and electrical issues to be addressed, the non-availability of an elaborated design flow for the respective circuits and systems is one of the main obstacles of a full exploitation of reversible computation in the domain of low power design. The corresponding discussions from above and particularly in the cited references give an impression of issues left to be addressed in future work.

## Acknowledgments

The authors would like to thank all researchers and collaborators which, in the past years, worked with us on the development of design methods for reversible circuits and systems.

## References

[1] P. Zeitzoff and J. Chung. A perspective from the 2003 ITRS. *IEEE Circuits & Systems Magazine*, 21:4–15, 2005.

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

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

[4] A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483:187–189, 2012.

[5] N. Gershenfeld. Signal entropy and the thermodynamics of computation. *IBM Systems Journal*, 35(3-4):577–586, 1996.

[6] P. Patra and D. Fussell. On efficient adiabatic design of mos circuits. In *Proceedings of the 4 th Workshop on Physics and Computation*, pages 260–269, Boston, 1996.

[7] J.G. Koller and W.C. Athas. Adiabatic switching, low energy computing, and the physics of storing and erasing information. In *Workshop on Physics and Computation. PhysComp '92.*, pages 267–270, 1992.

[8] A. De Vos and Y. Van Rentergem. Energy dissipation in reversible logic addressed by a ramp voltage. In *Proceedings of the 15 th International Workshop PATMOS*, pages 207–216, Leuven, 2005.

[9] R. Wille, R. Drechsler, C. Osewold, and A. García Ortiz. Automatic design of low-power encoders using reversible circuit synthesis. In *Design, Automation and Test in Europe*, pages 1036–1041, 2012.

[10] S. Pasricha and N. Dutt. *On-Chip Communication Architectures*. Systems on Chip. Morgan Kaufman, 2008.

[11] S. Ramprasad, N.R. Shanbhag, and I.N. Hajj. A Coding Framework for Low-Power Address and Data Busses. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(2):212–220, June 1999.

[12] L. Benini, A. Macii, E. Macii, M. Poncino, and R. Scarsi. Architectures and synthesis algorithms for power-efficient bus interfaces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19:969–980, September 2000.

[13] S. R. Sridhara and N. R. Shanbhag. Coding for System-on-Chip Networks: A Unified Framework. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(6):655 – 667, June 2005.

[14] A. García Ortiz, L. S. Indrusiak, T. Murgan, and M. Glesner. Low-Power Coding for Networks-on-Chip with Virtual Channels. *Journal of Low-Power Electronics*, 5:1–8, 2009.

[15] J. Seo, H. Kaul, R. Krishnamurthy, D. Sylvester, and D. Blaauw. A Robust Edge Encoding Technique for Energy-Efficient Multi-Cycle Interconnect. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(2):264–273, 2011.

[16] A. Ganguly, P. P. Pande, and B. Belzer. Crosstalk-Aware Channel Coding Schemes for Energy Efficient and Reliable NOC Interconnects. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(11):1626–1639, 2009.

[17] A. Ejlali, B. M. Al-Hashimi, P. Rosinger, S. G. Miremadi, and L. Benini. Performability/energy tradeoff in error-control schemes for on-chip networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(1):1–14, 2010.

[18] 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*, pages 78–85, 2011.

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

[20] M. Soeken, S. Frehse, R. Wille, and R. Drechsler. RevKit: an open source toolkit for the design of reversible circuits. In *Reversible Computation 2011*, volume 7165 of *Lecture Notes in Computer Science*, pages 64–76, 2012. RevKit is available at www.revkit.org.

[21] 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.

[22] 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.

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

[24] 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.

[25] R. Wille, S. Offermann, and R. Drechsler. SyReC: A programming language for synthesis of reversible circuits. In *Forum on Specification and Design Languages*, pages 184–189, 2010.

[26] M. K. Thomsen. A functional language for describing reversible logic. In *Forum on Specification and Design Languages*, pages 135–142, 2012.

[27] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes. Gate-level simulation of quantum circuits. In *ASP Design Automation Conf.*, pages 295–301, 2003.

[28] 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.

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

[30] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler. Debugging reversible circuits. *Integration*, 44(1):51–61, 2011.

[31] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Trans. on CAD*, 23(11):1497–1509, 2004.

[32] R. Wille, O. Keszöcze, and R. Drechsler. Determining the minimal number of lines for large reversible circuits. In *Design, Automation and Test in Europe*, 2011.

[33] M. Soeken, R. Wille, O. Keszocze, D. M. Miller, and R. Drechsler. Embedding of large Boolean functions for reversible logic. *J. Emerg. Technol. Comput. Syst.*, 2015.

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

[35] D. M. Miller, R. Wille, and G.W. Dueck. Synthesizing reversible circuits for irreversible functions. In *EUROMICRO Symp. on Digital System Design*, pages 749–756, 2009.

[36] R. Wille, M. Soeken, D. Michael Miller, and R. Drechsler. Trading off circuit lines and gate costs in the synthesis of reversible logic. *Integration*, 47(2):284–294, 2014.