Synthesis of Approximate Coders for On-chip Interconnects Using Reversible Logic

Robert Wille[1,2] Oliver Keszocze[2,3] Stefan Hillmich[3] Marcel Walter[2,3] Alberto Garcia-Ortiz[4]
[1] Institute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria
[2]Cyber Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[4]ITEM, University of Bremen, 28359 Bremen, Germany
robert.wille@jku.at {keszocze,hillmich,mwalter}@informatik.uni-bremen.de agarcia@item.uni-bremen.de

***Abstract*—On-chip coding provides a remarkable potential to improve the energy efficiency of on-chip interconnects. However, the logic design of the encoder/decoder faces a main challenge: the area and power overhead should be minimal while, at the same time, decodability has to be guaranteed. To address these problems, we propose the concept of approximate coding, where the coding function is partially specified and the synthesis algorithm has a higher flexibility to simplify the circuit. Since conventional synthesis methods are unsuitable here, we propose an alternative synthesis approach based on reversible logic. Experimental evaluations confirm the benefits of both, the proposed concept of approximate codings as well as the proposed design method.**

I. Introduction

Low-power coding has a remarkable potential to decrease the energy consumption of the interconnects [1], [2], [3] and can be successfully used in NoCs [4]. However, the design of such encoder/decoder circuits faces a main challenge: the area and power overhead should be minimal while, at the same time, decodability has to be guaranteed. Typically, the coder/decoder overhead restrict the usability of low-power coding to long buses [1], [2] or NoC paths with a significant number of hops [4].

The traditional implementation of low-power coders uses the standard specification and synthesis tools employed for digital synchronous design. Remarkably, the use of reversible synthesis produces better results [5]. However, they require information on the desired mapping of *all* possible input patterns – an exponential number with respect to the bit-width of the interconnect. This leads to scalability issues.

In this work, we propose the concept of *approximate codings* in order to handle this problem. In contrast to a complete coding, approximate codings do not incorporate all possible information, but focus on the important transmission patterns. This avoids the consideration of an exponential number of patterns. However, this leads to a new design challenge: Although only a subset of patterns is relevant for the actual synthesis, a one-to-one mapping has to be guaranteed for all remaining patterns. Neither traditional synthesis approaches of partially specified functions, nor standard approximate synthesis approaches as [6] can work in this scenario since they cannot guaranty decodability.

To address this problem, we propose an alternative synthesis approach for the generation of an (approximate) coding which relies on the concept of reversible logic. Reversible logic inherently realizes bijections only, i.e. one-to-one mappings where each input pattern is mapped to a unique output pattern. This allows focusing on the subset of patterns for which information is available, while all remaining patterns are inherently covered by the reversible computing paradigm. Experimental evaluations confirm the benefits of both, the proposed concept of approximate codings as well as the proposed design method.

II. Considered Problem

Power consumption (and propagation delay) of on-chip interconnects are pattern dependent phenomena whose basics are covered e.g. in [3]. Low-power codings have the goal to reduce this power consumption by changing the actual values communicated through the interconnect. To this end, several models and frameworks have been proposed (see [1], [2], [3]). In this work, we consider a framework where the power consumption is proportional to the number of ones at its input (i.e. proportional to the Hamming weight). This motivated a *probability based mapping* (pbm) [1]. An example illustrates the general idea.

Example 1. *Tab. I(a) shows a complete set of data inputs (first column) with their probability of occurrence (second column). Based on that, a coding should map the most frequently occurring data input (i.e.* 0010*) to a bit-string with the lowest Hamming weight (i.e.* 0000*). 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 the third column. A precise coding satisfying this is given in the fourth column.*

However, a key problem of the design of codings for pbm-mappings is to provide information about the probabilities of occurrence for *all* possible input patterns to be communicated through the interconnect and to account for them during the definition of the coder. Since the number of possible patterns exponentially increases with respect to the bit-width of the interconnect, this quickly leads to scalability problems. Besides that, even for smaller bit-widths, the precise information on the probability of *all* patterns is frequently not available and/or hard to grasp. Alternatively, we can consider a reduced set of patterns as shown in the next example:

Example 2. *Consider again the set of data inputs and their probabilities as shown in Tab. I(a). Let's additionally assume that only information on the best and worst cases of pattern distribution is given (i.e. the distribution of patterns shown in the second column of Table I(b)). There exists one pattern,* 0010*, for which a relatively large probability is assumed and one pattern,* 0111*, which is assumed to never be communicated through the interconnect. Accordingly, Hamming weights are assigned for these patterns as shown in the third column of Tab. I(b). Similarly, Hamming weights are defined for 0000 and 1111. For all remaining patterns, the probabilities (and, hence, the Hamming weights) are undefined.*

Codings which have been derived from a subset of patterns constitute an approximation of the ideal mapping, i.e. an *approximate coding*. Approximate codings obviously do not incorporate all possible information about the communication on the interconnect and, hence, may lead to a slightly less efficient power performance. Nevertheless, they allow the application of a coding strategy at all, since they avoid the consideration of an exponential number of patterns. Furthermore, the higher degrees of freedom to synthesize an approximate coding imply a higher potential to minimize the coder.

Despite these promising trade-offs, approximate codings have not been considered in the bibliography. The reason is that conventional synthesis methods cannot easily guarantee a one-to-one mapping when only a subset of patterns is specified. The following example illustrates the problem.

TABLE I: Complete and approximate pbm-codings

(a) Complete

Inputs	Prob.	H	Code
0000	10%	1	0010
0001	4%	2	0011
0010	25%	0	0000
0011	4%	2	0101
0100	4%	2	0110
0101	3%	3	1101
0110	10%	1	0100
0111	0%	4	1111
1000	3%	2	1010
1001	4%	3	1011
1010	10%	1	1000
1011	4%	2	1001
1100	3%	3	1110
1101	10%	1	0001
1110	4%	2	1100
1111	2%	3	0111

(b) Approximate

Inputs	Prob.	H	Code
0000	10%	1	0010
0001	?%	–	–
0010	25%	0	0000
0011	?%	–	–
0100	?%	–	–
0101	?%	–	–
0110	?%	–	–
0111	0%	4	1111
1000	?%	–	–
1001	?%	–	–
1010	?%	–	–
1011	?%	–	–
1100	?%	–	–
1101	?%	–	–
1110	?%	–	–
1111	2%	3	0111

Example 3. *Consider again the (partial) information about the distribution of patterns and the resulting Hamming weights as shown in Tab. I(b). Using that, conventional design methods would derive an incompletely specified function description as e.g. shown in the fourth column of Tab. I(b) (providing outputs for the patterns with a defined Hamming weight only). Synthesizing a circuit from that specification will likely lead to a circuit where two different inputs map to the same output – making the coding invalid.*

III. Proposed Solution

To describe the proposed solution, we first review the basics on reversible logic and discuss how this paradigm helps in the design of approximate codings. Afterwards, we present details on the resulting methodology.

A. General Idea: Exploiting Reversible Logic

Reversible logic realizes reversible functions, i.e. logic functions $f : \mathbb{B}^m \rightarrow \mathbb{B}^{m'}$ over inputs $X = \{x_0, \ldots, x_{m-1}\}$, where (1) the number of inputs is equal to the number of outputs (i.e. $m = m'$) and (2) each input pattern maps to a unique output pattern. In other words, a reversible function represents a bijection and, hence, a one-to-one mapping – exactly what is needed in order to describe the codings considered in this work. Consequently, the application of design approaches for reversible logic is a reasonable choice for the design task sketched in the previous section.

In the past, a broad variety of different synthesis approaches for reversible logic has been proposed and is available (see e.g. overviews in [7], [8]). The resulting circuits differ from conventional circuit descriptions. More precisely, a *reversible circuit* is a cascade G of *reversible gates* g_i, i.e. $G = g_0 g_1 \ldots g_{d-1}$ with d being the number of gates. Fanouts and feedback are not directly allowed. The most frequently occurring reversible gate is the so-called *Toffoli gate* $T(C, t)$ which is composed of a set of *control lines* $C = \{x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}}\}$ with $C \subset X$ and a single *target line* $x_j \in X$ with $x_j \notin C$. This gate maps $(x_0, x_1, \ldots, x_j, \ldots, x_{m-1})$ to $(x_0, x_1, \ldots, (x_{i_0} x_{i_1} \ldots x_{i_{k-1}}) \oplus x_j, \ldots, x_{m-1})$, i.e. the target line is inverted if all control lines are set to 1; otherwise the value of the target line is passed through unchanged.

Example 4. *Fig. 1 shows a reversible circuit. Control lines are indicated by black circles, while a target line is indicated by* $\oplus$.

Using reversible logic rather than conventional design methods makes the problem sketched in Section II and illustrated in Example 3 much easier. In fact, passing an incompletely specified function (as the one from Table I(b)) to a corresponding synthesis method yields a circuit in which the don't care outputs are also arbitrarily assigned. But because of the inherent reversibility of the circuit structure, each input pattern will always be mapped to a unique output pattern, i.e. an one-to-one mapping is implicitly guaranteed.

TABLE II: Transformation-based method

line (i)	input abcd	output abcd	1^{st} abcd	2^{nd} abcd	3^{rd} abcd	4^{th} abcd	5^{th} abcd	6^{th} abcd
0	0000	0010	00**1**0	0010	0010	0010	0010	0010
1	0001	0011	00**0**1	0001	0001	0001	0001	0001
2	0010	0000	00**0**0	0000	0000	0000	0000	0000
3	0011	0101	01**0**1	00**1**1	0011	0011	0011	0011
4	0100	0110	01**1**0	0100	0100	0100	0100	0100
5	0101	1101	11**0**1	10**1**1	11**1**1	11**0**1	**0**101	0101
6	0110	0100	01**0**0	0110	0110	0110	0110	0110
7	0111	1111	11**1**1	1101	1101	11**1**1	**0**111	0111
8	1000	1010	10**1**0	1000	1000	1000	1000	1000
9	1001	1011	10**1**1	1001	1001	1001	1001	1001
10	1010	1000	10**0**0	1010	1010	1010	1010	1010
11	1011	1001	10**0**1	11**1**1	10**1**1	1011	1011	1011
12	1100	1110	11**1**0	1100	1100	1100	1100	1100
13	1101	0001	00**0**1	01**1**1	0111	0111	**1**111	11**0**1
14	1110	1100	11**0**0	1110	1110	1110	1110	1110
15	1111	0111	01**1**1	0101	0101	0101	**1**101	11**1**1

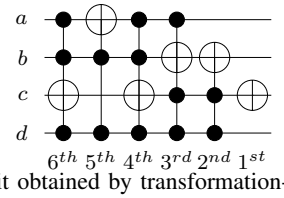

Fig. 1: Circuit obtained by transformation-based synthesis

However, synthesis approaches for pure reversible logic are not suited for the generation of circuits realizing (approximate) codings, i.e. there is neither a support for the Hamming weight objective nor for incompletely specified functions. As a consequence, we propose a new synthesis method which relies on reversible logic, but supports the design needs for the generation of (approximate) codings. To this end, we are re-using the general idea of the so-called *transformation-based synthesis* scheme which has been introduced in [9]. The next section reviews the main ideas of this scheme. Afterwards, we describe how this scheme can be extended for the purposes considered here.

B. Transformation-based Synthesis of Reversible Circuits

The transformation-based synthesis scheme is provided with a (completely specified) function to be synthesized in terms of a truth table. Then, the basic idea is to traverse each line of the truth table and to add (reversible) gates to the circuit until the output values match the input values (i.e. until the identity of both is achieved). Gates are chosen so that they do not alter already considered lines. Furthermore, gates are added starting at the output side of the circuit (this is, because *output* values are transformed until the identity is achieved).

In the following, the single steps are described by means of the (completely) specified function from Table I(a)[1]. Table II illustrates the single steps. The first column denotes the respective line numbers of the truth table, while the second and third column repeat the function specification. The inputs and outputs are denoted by a, b, c, d, respectively. The remaining columns provide the transformed output values for the respective steps.

The algorithm starts at truth table line 0. Here, the input/output mapping differs only for c. This can easily be equalized by appending the gate $T(\emptyset, c)$ to the circuit which modifies the output patterns as shown in the column for the 1^{st} step in Table II (changing bits are highlighted bold). Since the inputs/outputs of truth table lines 1 and 2 are now already equal, the approach continues with truth table line 3. Here, input b needs to be switched which can be accomplished using the gate $T(\{c, d\}, b)$. The control lines c and d ensure that none of the already traversed truth table lines is affected by this. This modifies the output patterns as shown in the column for the 2^{nd} step. Similarly, all remaining truth table lines are traversed; eventually leading to the circuit shown in Fig. 1.

[1]Note that this neither considers the actual Hamming weights nor supports a partial consideration of patterns.

TABLE III: Synth. for appr. codings

input abcd	Hamming weight	output abcd	1^{st} abcd	2^{nd} abcd	3^{rd} abcd
0000	1	0010	00**1**0	0000	0000
0010	0	0000	00**1**0	0010	0010
0111	4	1111	11**0**1	**0**101	01**1**1
1111	3	0111	01**0**1	**1**101	11**1**1

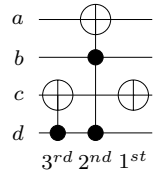

Fig. 2: Res. circuit

C. Transformation-based Synthesis for Approximate Codings

In order to generate an approximate coding, transformation based synthesis is utilized as follows: Instead of traversing the entire truth table (which is not available anyway), only the available subset of patterns is traversed. For each input pattern, the corresponding Hamming weight is instantly converted to an appropriate output pattern which (1) satisfies the Hamming weight and (2) has the smallest Hamming distance to the currently considered input pattern. If all patterns are traversed, a reversible circuit results that realizes the desired mappings for all given patterns. Moreover, due to the reversibility, the circuit also implicitly provides a (non-exponential) description of the one-to-one mapping for all remaining patterns. That is, a symbolic description of the desired approximate coding is derived.

In the following, the single steps are described by means of the partial information about the distribution of patterns and the resulting Hamming weights (as shown in Table I(b), respectively) which have already been discussed before in Example 3. Table III illustrates the single steps (using a similar notation as before in Table II, where, instead of the precise output patterns, the desired Hamming weights are listed in the second column).

The algorithm, again, starts with the first line of Table III. Here the input pattern 0000 shall be mapped to an output pattern with Hamming weight 1, e.g. 0010. This can be realized by the gate $T(\emptyset, c)$ (1^{st} step). The next input pattern is already mapped to a correspondingly desired output pattern (additionally incorporating the first gate). For input pattern 0111 an output with Hamming weight 4, i.e. 1111, is desired. Incorporating the effect from the previously added gate (leading to the pattern 1101), this requires two variables to be changed – accomplished by the gates $T(\{b, d\}, a)$ and $T(\{d\}, c)$ (2^{nd} step and 3^{rd} step). This also leads to the desired mapping for the final pattern and, hence, a circuit as shown in Fig. 2 is determined. This circuit represents a symbolic representation of the desired (approximate) coding.

D. Handling Corner Cases

Following the synthesis scheme presented in the previous section allows for the automatic generation of approximate codings and, hence, solves the problems discussed in Section II. Unfortunately, the scheme is not applicable for arbitrary use cases. In fact, the (revised) transformation-based synthesis requires a *cyclic* input/output mapping to be realized, i.e. the set of output patterns O to be obtained from the given Hamming weights has to be equal to the set of given input patterns I. An example illustrates the problem:

Example 5. *Consider the synthesis problem shown in the first two columns of Tab. IV. These Hamming weights do not allow for a* cyclic *input/output mapping: the three instances of Hamming weight 2 can be realized with patterns 011, 101, 110 and the one instance of Hamming weight 3 can be realized with the pattern 111 (all patterns included in set I). However, the instance with Hamming weight 0 can only be realized with pattern 000, which is not included in I. Accordingly, the set of output patterns O to be obtained from the given Hamming weights cannot be equal to I. This causes problems as illustrated in the remaining columns of Tab. IV: The first four patterns can be handled similarly to the previous example from Tab. III (again, changing bits are highlighted bold). However, after the 3^{rd} step, the last output pattern 000 has to be mapped to the input pattern 111. Obviously, this is only possible with three gates $T(\emptyset, a)$, $T(\emptyset, b)$, $T(\emptyset, c)$ which, however, would all modify previously traversed patterns. Hence, the transformation-based synthesis does not terminate with a useful result.*

TABLE IV: Synthesis with a non-cyclic mapping

input abc	Hamming weight	output abc	1^{st} abc	2^{nd} abc	3^{rd} abc	4^{th} abc
011	2	011	011	011	011	011
100	3	111	**1**00	100	100	100
101	2	101	**1**10	**1**01	101	101
110	2	110	**1**01	**1**11	**1**10	110
111	0	000	000	000	000	**???**

TABLE V: Cyclic mapping

input abc	Hamming weight	output abc
011	2	011
100	3	111
101	2	101
110	2	110
111	**0**	**000**
000	**–**	**100**

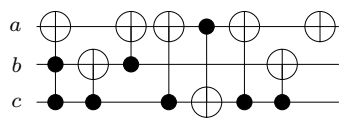

Fig. 3: Resulting circuit

The problem is that deriving a set O from the Hamming weights that is equal to the given set I is not always possible. In order to address this problem, the given set of Hamming weights or, more precisely, input/output mappings has to be made cyclic before the transformation-based synthesis previously presented, is applied. To this end, all given input patterns are traversed in a pre-synthesis step. For each input pattern $i \in I$, it is checked whether another input pattern $i' \in I \setminus \{i\}$ exists which (1) satisfies the corresponding Hamming weight and (2) has not already been utilized as output pattern. If this is the case, the one with the smallest Hamming distance to i is chosen as output pattern and added to O. Otherwise, another input $i'' \notin I$ with the correct Hamming weight is added to I. All additional patterns are assumed to have an arbitrary Hamming weight (i.e. any remaining input may be chosen as a mapping) and are processed analogously to the originally provided input patterns[2].

Example 6. *Applying the pre-synthesis step to the problem discussed in Example 5 leads to the following result: The first input pattern 011 shall be mapped to an output with Hamming weight 2. Here, several options are available in I, namely 011, 101, 110. Obviously, 011 has the smallest Hamming distance to the considered input pattern and, hence, is chosen (see first line of Table V). Similarly, corresponding output patterns are determined for the following input patterns 100, 101, and 110 (see following lines in Table V). In contrast, for the input pattern 111, no unused pattern with Hamming weight 0 can be found in I. Hence, another input pattern 000 is added to I. This allows for a cyclic mapping from 111 to 000. Since, at the same time, the mapping from the newly added 000 to an output may be arbitrary, the only pattern from I which has not been used yet, namely 100, can be chosen (eventually leading to the cyclic mapping shown in Table V).*

The resulting input/output mappings are eventually cyclic and, hence, can be realized to an approximated coding using the transformation based synthesis approach presented in the previous section (note that, for this purpose, the patterns are ordered e.g. from 000 to 111 again). For Example 6, this leads to the circuit shown in Fig. 3.

[2]Note that, in the worst case, the number of input/output patterns is doubled.

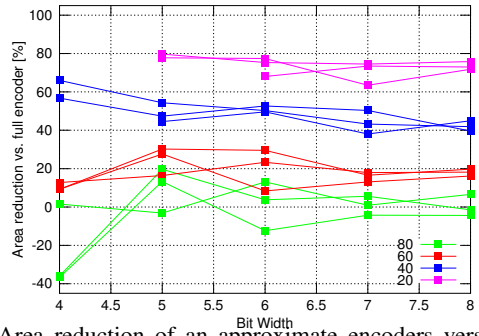

Fig. 4: Area reduction of an approximate encoders versus a fully specified one as a function of the bus-width. Results when considering 20, 40, 60, and 80 percent of the patterns for the approximate encoder.

IV. Experimental Evaluation

We have evaluated the proposed approach using a broad set of input signals: a Gaussian PDF as typically found in DSP applications, a monotonically decreasing PDF with a linear shape and a monotonically decreasing PDF with a Gaussian shape (e.g., as it would appear calculating the absolute value of a typical DSP signal). We also considered inverted versions of this PDFs, and added "random" PDFs. This set covers the most practical relevant cases and allows a direct comparison with previous approaches [5]. When possible, the PDF was parameterized modifying the variance. For each definition of the input signal, a coder was created. The circuits were mapped in a commercial 65nm technology using Synopsys `design_vision` to precisely evaluate the area, delay, dynamic power consumption, and static power consumption.

Firstly, we analyzed the overhead of the encoder using a full description. We observed an exponential increase in the complexity of the encoder as the bit-width of the bus increases. These results agree with those reported in [5] for reversible synthesis and those of [2]. The overhead of exact coders for large bit-withs is unacceptable.

Next, we analyzed the overhead of approximate encoders, i.e. the reduction in the area of the coder versus the full implementation for different PDFs, parameter selection strategies, etc. was considered. We used the methods of [5] as a baseline (versus traditional approaches our gains are even larger). Fig. 4 provides the key results. Area reduction is almost independent of the bit-width and the shape of the PDF, the main dependency comes from the relative number of considered patters. If 40% of the patters are considered, around 40% in area can be saved; for 20% of the patters, around 80% of the area can be saved. For very small bit-widths and high number of patterns (80%) some circuits do not provide any improvement. Overall, if less than 60% of the patters are considered, an improvement in area overhead is observed. Similar results were obtained for delay and power.

Finally, we analyzed the loss of efficiency of the coder as it becomes "approximate". This analysis is delicate since there is a strong effect of the PDF. We focused first on the worst scenario: We considered a monotonic decreasing PDF with a parameterizable variance. This PDF appears, for example, in the sign-magnitude representation of DSP signals and is one of the few cases where no coding in the interface-layer is adequate. We measured the additional improvement in transition activity achieved by the approximate coders; for illustration purposes we also represent the maximum improvement achievable by an ideal coder. The results for bit-widths 8 and 12 are reported in Fig. 5, where we normalized the variance of the signal with the total number of patters in the bus to facilitate the comparison of the two graphs. Remarkably, even when only 40% of the patters where considered, the performance of the code is very close to the ideal one; moreover, the larger the bus is, the more efficient it gets. The smallest number of patters which provides a sensible selection is around 20-30%; less patters degrade the coder too much. As

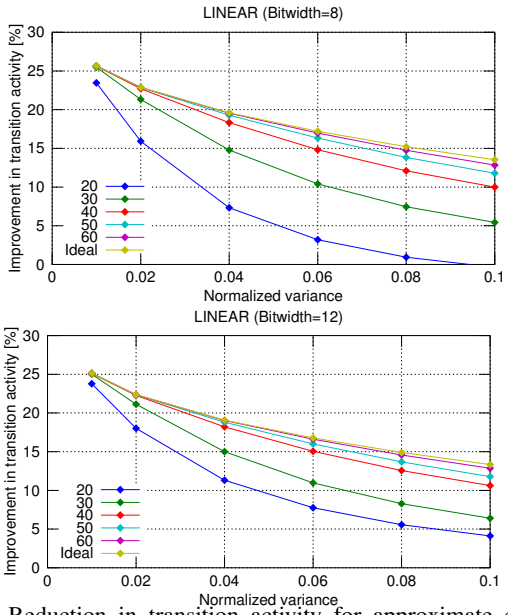

Fig. 5: Reduction in transition activity for approximate encoders using between 20% and 60% of the patterns. Reduction analysed versus the statistical characteristics of the input PDF

expected, the improvement decreases as the variance increases (the signal has less spatial redundancy), but the trend is similar to that of the ideal coder. In summary, a number of patters around 40% is a good trade off between area reduction (40%) and non-ideality of the code. It is worth to mention that for other PDFs, the improvements in transition activity are even larger. For example, if the PDF increases monotonically, the improvements are more than two times larger.

V. Conclusions

Data representation plays a major role in the synthesis tools. This work shows that a representation using reversible concepts is a more natural, efficient, and powerful approach to synthesize coders than traditional ones; furthermore, it allows the synthesis of partially specified coders or *approximate coders*. Compared with industry-standard and state-of-the art reversible approaches, our solution provides significant advantages in terms of scalability and efficiency. Specifying only 40% of the codes almost halves the area of the coder without significantly degrading the optimality of the coder.

References

[1] 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.

[2] 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.

[3] A. Garcia-Ortiz, D. Gregorek, and C. Osewold. Optimization of interconnect architectures through coding: A review. In *Electronics, Communications and Photonics Conference (SIECPC), 2011 S. Int.*, pages 1–6, April 2011.

[4] J.C.S. Palma, L.S. Indrusiak, F.G. Moraes, A. Garcia Ortiz, M. Glesner, and R.A.L. Reis. Inserting data encoding techniques into noc-based systems. In *IEEE Computer Society Annual Symp. on VSLI*, pages 299–304, March 2007.

[5] 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.

[6] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 796–801, June 2012.

[7] Rolf Drechsler and Robert 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.

[8] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuitsa survey. *ACM Computing Surveys*, 45(2):21, 2013.

[9] 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.