A Synthesis Flow for Sequential Reversible Circuits

Mathias Soeken* Robert Wille* Christian Otterstedt* Rolf Drechsler*†

*Institute of Computer Science
University of Bremen, 28359 Bremen, Germany

†Cyber-Physical Systems, DFKI GmbH
28359 Bremen, Germany

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

***Abstract*—In this paper, a synthesis flow for sequential reversible circuits is proposed. In particular, a methodology is introduced which transforms a finite state machine into a Boolean function representing the sequential behavior. Having that, any combinational synthesis approach can be exploited in order to perform the actual synthesis. Heuristics ensure that encodings for the states are applied which keep the costs of the resulting circuits low. Experiments show the applicability of the approach.**

I. Introduction

During the last decades, the number of transistors in integrated circuits has been doubled approximately every 18 months (also known as "Moore's Law"). Even if this development may continue in the upcoming years, physical limits will be reached in the future. In particular, power dissipation is going to become crucial. While nowadays power dissipation particularly is caused e.g. by non-ideal behavior of transistors and materials (where higher levels of integration and improved fabrication process may help), a more fundamental limit (namely "Landauers Barrier" [1]) will be approached. Landauer stated that at each time information is lost during computation (e.g. when an AND operation transforms two input signals into a single output signal), energy is dissipated. Extrapolating the recent achievements in the development of conventional CMOS technologies, this amount of power dissipation will halt further miniaturization in the future [2].

As a consequence, researchers intensely studied alternative technologies. In this context, reversible circuits [3] are promising. They perform reversible operations only, i.e. they do not lose information during the computation and, thus, can avoid power dissipation caused by information loss. Motivated by this, synthesis of reversible circuits became an active research area [4], [5], [6], [7].

However, most of the existing approaches address synthesis of combinational reversible circuits only, while the existing work on sequential reversible circuits mainly focuses on the realization of latches [8], [9], [10], [11]. As an exception, in [12] the conceptual design of sequential reversible circuits has been explored. However, no precise synthesis procedure was proposed. In contrast, a synthesis method based on finite state machines has been introduced in [13]. This approach, however, relies on a particular selected synthesis method which further requires the construction of reversible state machines.

In this paper, a synthesis flow is proposed which automatically realizes a sequential circuit for an arbitrary finite state machine. It generalizes the synthesis approach from [13] by allowing conventional finite state machines as well as an arbitrary synthesis method for the actual realization of the reversible circuit. This enables the possibility to choose a suitable approach in favor of a desired cost criteria.

The general idea is as follows: First, the given sequential behavior is transformed into a combinational description. Afterwards, state-of-the-art synthesis techniques (e.g. [4], [5]) are applied to generate the circuit structure. Having that, the specified behavior can sequentially be executed in reversible logic.

One important aspect is the precise encoding of the respective states. We observed, how the choice of a state encoding affects the costs of the resulting circuit. Motivated by this, heuristics are considered aiming for the determination of "good" encodings, i.e. encodings that lead to circuits with low costs.

The proposed approaches are evaluated using different (combinational) synthesis methods. This is the first case study investigating the applicability of an established sequential synthesis flow in the domain of reversible logic. Experiments show that the costs of the resulting circuits depend on the synthesis method applied in the second step.

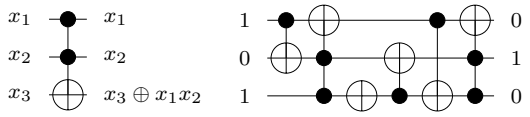
In the remainder of the paper, the proposed approaches are described using the following structure: The next section provides basics on reversible circuits as well as selected synthesis approaches and briefly introduces finite state machines. Section III discusses sequential reversible circuits in general, while Section IV presents the proposed synthesis approach. Section V deals with the optimization of the state encodings. Finally, Section VI provides the results of the experimental evaluation, while the paper is concluded in Section VII.

II. Preliminaries

To keep the paper self-contained, basics on reversible circuits, the respective synthesis approaches, and finite state machines are briefly reviewed in this section.

A. Reversible Circuits

A Boolean function $f : \mathbb{B}^k \to \mathbb{B}^k$ is *reversible* if it is bijective, i.e. if the function maps each input pattern to a unique output pattern. Hence, for these functions it is possible to obtain the input pattern to a given output pattern. *Reversible circuits* are logical circuits that realize reversible functions. In order to preserve the reversibility, fan-out and feedback are not allowed directly [14]. Thus, a reversible circuit is realized as a cascade of reversible gates. The Toffoli gate [15] is the most widely used gate and is also universal, i.e. all reversible functions can be realized by means of gates of this type only. It consists of a set of *control lines* and a *target line*. The value



(a) Toffoli gate (b) Toffoli circuit

Fig. 1. Toffoli gate and Toffoli circuit

of the target lines is inverted, if and only if all control lines are assigned a logic value 1. The functionality can be obtained from Fig. 1(a), whereby the first two lines denote a control line and the third one denotes the target line.

Example 1: A reversible circuit composed of Toffoli gates is depicted in Fig. 1(b). This circuit maps e.g. the input pattern 101 to the output pattern 010. Note that due to the reversibility this computation can be performed in both directions on the circuit.

B. Synthesis of Reversible Circuits

Several approaches for the synthesis of (combinational) reversible circuits have been proposed in the past. They rely on different functional descriptions like truth tables or decision diagrams, and, accordingly, differ in their scalability. In this work, we apply the following two synthesis methods[1]:

(1) The *transformation-based synthesis* approach proposed in [4] works on a truth table description of the function to be synthesized. Traversing each row of the truth table, at every step gates are added to the circuit so that the identity between the inputs and outputs of the considered row is achieved. These gates are chosen so that they do not alter rows earlier in the table. After all rows are processed, the cascade of gates identified in reverse order is a circuit realizing the given reversible function. The transformation-based synthesis approach is applicable to reversible functions only.

(2) The *KFDD-based synthesis* approach proposed in [6] works on a *Kronecker Functional Decision Diagram* (KFDD) representing the function to be synthesized. The approach traverses each node of a KFDD in a depth-first manner and stores a cascade for each visited node. Afterwards, these cascades are merged leading to a circuit realizing the given function. This may result in *constant inputs* and *garbage outputs*, i.e. circuit lines with a fixed constant value assigned to the primary input and circuit lines whose primary output values are don't care, respectively. This approach is applicable not only to reversible functions, but also irreversible functions.

C. Finite State Machines

Finite state machines are used to describe the behavior of sequential circuits to be synthesized. One commonly used state machine is the Mealy machine [16], which provides an output function in addition to an input-sensitive transition relation.

Definition: A *Mealy machine* is a 5-tuple

$$\mathcal{M} = (S, s_o, \Sigma, \Lambda, T)$$

with a finite set of states S, an initial state $s_0 \in S$, a finite input alphabet Σ, a finite output alphabet Λ, and a transition function $T : S \times \Sigma \rightarrow S \times \Lambda$, which maps a source state together with a letter of the input alphabet to a next state and a *signal* of the output alphabet.

In the following, we consider a state machine to be deterministic. A finite state machine is *deterministic*, if for each state and for each letter in the input alphabet, there exists exactly one transition to a source state and a corresponding output signal, i.e.:

$$\forall s \in S : \forall i \in \Sigma : \exists!(s', o) \in S \times \Lambda : T(s, i) = (s', o).$$

A non-deterministic finite state machine can be transformed to a deterministic finite state machine using the powerset construction. An example of a deterministic finite state machine is given in Fig. 4(a) and considered later in detail.

[1]However, any other synthesis approach can easily be integrated in the proposed flow as well.

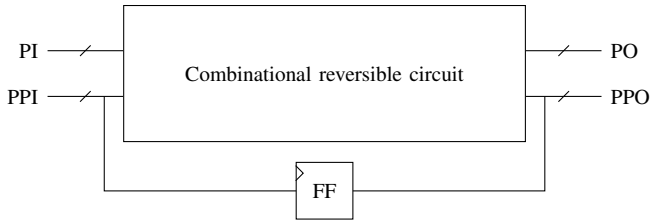


Fig. 2. Sequential reversible circuit

III. Sequential Reversible Circuits

While a significant number of approaches for the combinational synthesis of reversible circuits has been introduced in the past, research on design solutions for sequential logic elements is still at the beginning. One major issue is the treatment of feedback, which is not directly allowed in reversible circuits. Two different paradigms to overcome this restriction are currently under detailed consideration.

The first paradigm (suggested e.g. in [12]) assumes that a reversible circuit retains its state as long as its signal values remain unchanged. Then, a combinational circuit can be treated as a core component of a sequential device. More precisely, using e.g. a conventional (i.e. irreversible) controller, output values from one cycle are applied to the respective input signals of the next cycle. Therefore, the clocking as well as the feedback is handled by the controller, while the actual computation is performed on a combinational reversible circuit.

The second paradigm considers the realization of the sequential elements directly in reversible logic inhibiting the restrictions necessary for reversible circuits. For this purpose, several suggestions on how to realize the respective memory elements as flip-flops, latches, or registers have been made (see e.g. [8], [9], [10]). Using these basic sequential elements, more complex sequential components can be constructed.

In both cases, the resulting sequential circuit has a structure as depicted in Fig. 2. Besides *primary inputs* (PIs) and *primary outputs* (POs), the circuit has additional lines for the connection to the sequential element, referred to as *pseudo primary inputs* (PPIs) for the present state and *pseudo primary outputs* (PPOs) for the next state. For the first paradigm, a conventional irreversible flip-flop can be used. An external controller performs the computation and stores the signals of the PPOs in memory elements which serve the values for the

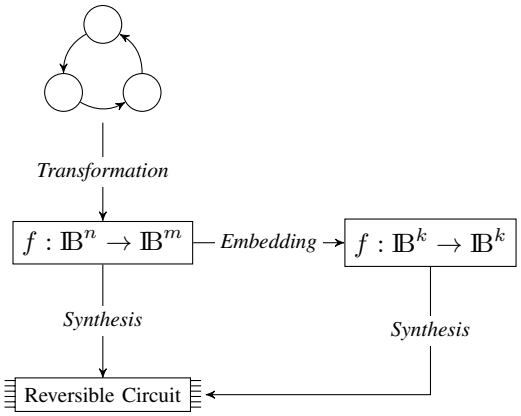


Fig. 3. Synthesis flow

PPIs in the next cycle. If instead a reversible memory element is used, the design corresponds to the second paradigm.

IV. Synthesis of Sequential Reversible Circuits

In this section, the proposed synthesis flow is described. First the main flow is outlined, followed by a more detailed description of the respective steps.

A. Main Flow

The main flow of the proposed synthesis approach is depicted in Fig. 3. Starting from a deterministic finite state machine, a Boolean function representing the sequential behavior is obtained. This function serves as input for the successive combinational synthesis approaches and may be irreversible. Thus, this function has to be *embedded* into a reversible one if the applied synthesis approach supports reversible function descriptions only. Afterwards, the circuit can combinationally be synthesized. Since the synthesized function included the sequential behavior of the finite state machine, a combinational circuit with pseudo primary inputs and pseudo primary outputs results which can be treated as a sequential circuit as described in Section III. In the following, the respective steps are described in detail.

B. Transforming the State Machine into a Boolean Function

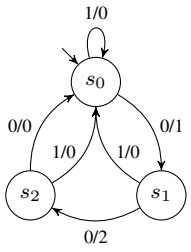
In order to transform the sequential behavior of a finite state machine $\mathcal{M} = (S, s_0, \Sigma, \Lambda, T)$ into a Boolean function which can serve as input for combinational synthesis approaches, the elements in the sets S, Σ, and Λ are encoded as bit-vectors. This can be done in two steps: First, the respective elements are transformed into a numerical representation, which afterwards can be represented as bit-vectors. Given that, the transition relation of the finite state machine can be expressed in terms of a bit-vector mapping.

More precisely, let $M \in \{S, \Sigma, \Lambda\}$. Since M is finite, the containing elements can be enumerated using the function

$$\text{enum} : M \to \{0, \ldots, |M| - 1\} \tag{1}$$

which assigns each element in M to a unique value from 0 to $|M| - 1$, where $|M|$ denotes the cardinal number of M. Having the numerical representation for the elements in M, they can be expressed as bit-vectors using the binary expansion

$$\text{bv} : \{0, 1, \ldots, |M| - 1\} \to \mathbb{B}^{\lceil \log_2 |M| \rceil}, \tag{2}$$



(a) Finite state machine

t_1	t_0	r	t'_1	t'_0	o_1	o_0
0	0	0	0	1	0	1
0	0	1	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0

(b) Resulting function

Fig. 4. Encoding of a state machine representing a Modulo-3 counter

which maps each natural number $\nu \in \{0, \ldots, |M| - 1\}$ to a bit-vector $\vec{b} = (b_0 \ \ldots \ b_{\lceil \log_2 |M| \rceil - 1}) = \text{bv}(\nu)$, such that

$$\nu = \sum_{i=0}^{\lceil \log_2 |M| \rceil - 1} 2^i b_i.$$

Given the mappings defined in (1) and (2), the binary encoding for an arbitrary element $\mu \in M$ is defined by

$$\text{enc}(\mu) = \text{bv}(\text{enum}(\mu)). \tag{3}$$

Using this encoding function, the finite state machine can be transformed into a Boolean function. Therefore, a function

$$f : \mathbb{B}^{\lceil \log_2 |S| \rceil} \times \mathbb{B}^{\lceil \log_2 |\Sigma| \rceil} \to \mathbb{B}^{\lceil \log_2 |S| \rceil} \times \mathbb{B}^{\lceil \log_2 |\Lambda| \rceil}$$

is created, which represents the transition relation $T : S \times \Sigma \to S \times \Lambda$, i.e. which is defined by

$$f(\text{enc}(s), \text{enc}(i)) = (\text{enc}(s'), \text{enc}(o)), \text{with } T(s, i) = (s', o).$$

Example 2: Consider the finite state machine given in Fig. 4(a) which represents a modulo-3 counter. The finite state machine consists of three states $S = \{s_0, s_1, s_2\}$ and has an input alphabet $\Sigma = \{0, 1\}$ denoting the states of a *reset* signal r as well as an output alphabet $\Lambda = \{0, 1, 2\}$ denoting the possible output values o, respectively. Applying the transformation described above, a function results whose truth table is shown in Fig. 4(b) and which represents the sequential behavior of the considered finite state machine. The encoded states are represented by the variables t_1, t_0 as well as t'_1, t'_0, while the encoded inputs and outputs are represented by r and o_1, o_0, respectively.

Having such a function description, existing synthesis approaches can be applied to generate the desired sequential circuit. However, the resulting function often is not reversible. Thus, if the applied synthesis approach supports reversible functions only, an additional embedding is required.

C. Embedding

Usually, k circuit lines are needed to realize a reversible function $f : \mathbb{B}^k \to \mathbb{B}^k$ in reversible logic. However, in order to embed an irreversible function into a reversible one, additional circuit lines with constant inputs or don't care outputs are needed [17]. This is illustrated in the following example.

Example 3: Consider again the finite state machine and the respective function from Fig. 4. It can be seen, that the output pattern 0000 occurs four times and, thus, the function is not reversible. Adding one additional garbage line (holding don't

t_1	t_0	r	t'_1	t'_0	o_1	o_0	-
0	0	0	0	1	0	1	0
0	0	1	**0**	**0**	**0**	**0**	0
0	1	0	1	0	1	0	0
0	1	1	**0**	**0**	**0**	**0**	1
1	0	0	**0**	**0**	**0**	**0**	?
1	0	1	**0**	**0**	**0**	**0**	?

(a) Adding one garbage line

0	0	0	t_1	t_0	r	t'_1	t'_0	o_1	o_0	-	-
0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	1	**0**	**0**	**0**	**0**	0	0
0	0	0	0	1	0	1	0	1	0	0	0
0	0	0	0	1	1	**0**	**0**	**0**	**0**	0	1
0	0	0	1	0	0	**0**	**0**	**0**	**0**	1	0
0	0	0	1	0	1	**0**	**0**	**0**	**0**	1	1

(b) Minimal number of garbage lines

Fig. 5. Embedding of an irreversible function

care values), the first two of these output patterns can be made unique as shown in Fig. 5(a). But, the two remaining 0000-outputs remain ambiguous. Thus, one more garbage output is needed. Furthermore, in order to adjust the number of inputs according to the number of outputs, three constant inputs have to be added. Then, a reversible function as (partially) shown in Fig. 5(b) results, which includes the representation of the considered finite state machine.

In general, if the obtained function description is irreversible and not supported by the synthesis approach at hand, garbage outputs have to be added until all output patterns are unique. Afterwards, constant inputs have to be added accordingly. Then, a reversible function results, which can be passed to the synthesis engine.

V. Optimization of the State Encodings

In the previous section, the synthesis flow for sequential reversible circuits has been described. One important aspect is the precise encoding of the respective states in S. In contrast to the inputs Σ and the outputs Λ, which need to keep their functional values, the encoding of the states can arbitrarily be chosen as long as a unique encoding for each state $s \in S$ is ensured. In particular, the encoding function in Eq. (3) can be replaced by any function $\text{enc} : M \rightarrow \mathbb{B}^{\lceil \log_2 |M| \rceil}$. This enables significant reductions in the costs of the resulting circuits as illustrated in the following example.

Example 4: Consider again the finite state machine shown in Fig. 4(a). Using the binary expansion in order to encode the states, the function from Fig. 4(b) is obtained leading to a circuit with quantum costs of 41[2]. If instead all states $s_0, s_1, s_2 \in S$ are encoded by

$$\text{enc}(s_0) = 10,\ \text{enc}(s_1) = 00,\ \text{enc}(s_2) = 11,$$

the synthesis approach generates a circuit with quantum costs of 35 — a reduction by 15%.

As shown by the example, the choice of the encoding of the states has a significant impact on the cost of the resulting circuit. Motivated by this, further (heuristic) encodings are considered, namely:

Random Encoding. Here, randomly generated permutations of the binary expansions are generated and, afterwards, a corresponding circuit is synthesized. If the costs of the resulting circuit are lower than the costs of the currently best realization, the newly generated permutation is saved. This will be repeated for a fix number of iterations. Afterwards, the best permutation and therefore the best circuit is returned.

Encoding based on the Hamming-distance. For many synthesis approaches, it can be observed that functions with a low hamming-distance between their inputs and outputs can usually be realized with a fewer number of gates and, thus, smaller costs. Therefore, the Hamming distance is a proper criterion for the encoding of the states. A greedy scheme is thereby applied: A state is encoded by a pattern which has not already been applied for a previously considered state and, additionally, has the lowest hamming-distance between the current and the following state of each transition.

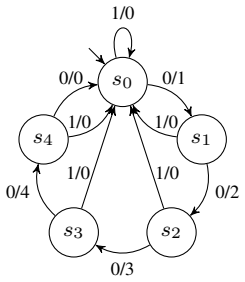


Fig. 6. Finite state machine representing a Modulo-5 counter

Encoding obtained by sifting. The best encoding can be obtained by considering the resulting costs of circuits generated from all possible permutations of the respective state encoding. However, this would lead to $\mathcal{O}(|S|!)$ possible permutations to be considered. Using sifting techniques [18] this complexity can be reduced to $\mathcal{O}(|S|^2)$ possible permutations to be considered, while, at the same time, often an acceptable approximation of the best result can be obtained.

Example 5: In order to illustrate the impact of the different state encodings, the considered heuristics have been applied to a somewhat larger example, i.e. a modulo-5 counter which is shown in Fig. 6. Encoding the states by means of the originally applied binary expansion, a circuit with

36 gates, 16 lines, and quantum costs of 80

results. If in contrast the heuristics outlined above are applied, circuits with

25 gates, 16 lines, and quantum costs of 73 (random),
32 gates, 14 lines, and quantum costs of 80 (Hamming), and
22 gates, 15 lines, and quantum costs of 66 (sifting)

can be obtained, respectively[3]. That is, the costs can be reduced significantly.

VI. Experimental Evaluation

In this section, experimental results obtained by the proposed approach are reported. The synthesis flow depicted in Fig. 3 has been implemented in C++ on top of RevKit [19] and applied to finite state machines taken from the LGSynth93 benchmark set. As (combinational) synthesis approaches both, the KFDD-based method [6] and the transformation-based method [4], have been applied. All experiments have been carried out on a 64-bit Intel Core 2 Duo with 2 GHz and 2 GB main memory running Linux 2.6. The timeout (denoted by TO) was set to 3 000 CPU seconds.

[2]Using the transformation-based approach [4] for the synthesis.

[3]Using the KFDD-based approach [6] for the synthesis.

A. Evaluation of the Generic Synthesis Flow

The results of the first evaluation applying the concepts from Section IV are listed in Table I. In the first 5 columns, the name of the finite state machine along with the number of primary inputs (#PI), number of primary outputs (#PO), number of states ($|S|$), and number of transitions ($|\operatorname{dom} T|$) are given. In the next columns, statistical data of the resulting circuits are outlined. More precisely, the number of lines, the number of gates, the quantum cost (QC) as well as the transistor cost (TC) are provided[4]. Furthermore, the run-time in CPU seconds needed to synthesize the respective circuits is given.

As can be seen, half the benchmark set cannot be synthesized using the transformation-based synthesis approach. This is because the method relies on truth table descriptions and, thus, is not applicable for larger functions. Nevertheless, for smaller finite state machines, reversible circuits with a small amount of circuit lines can be realized. In contrast, if the KFDD-based approach is applied, all circuits can be realized in nearly no time and with very moderate costs. Only the number of circuit lines is significantly higher. This fits with the experiences made in synthesis of combinational reversible circuits (see e.g. [6], [4]).

B. Evaluation of the Optimization of the State Encodings

In a second series of experiments, the impact of the state-encodings on the costs of the resulting circuits have been evaluated. The three additional encodings outlined in Section V are considered. The results are presented in Table II[5]. The denotation of the columns is similar to Table I, except that only the differences with respect to the results from the previous evaluation are reported.

The results clearly confirm the impact of the state encoding on the costs of the resulting circuits. The random heuristic as well as the sifting heuristic lead to significant improvements compared to the encoding based on the binary expansion. In contrast, the Hamming-distance heuristic does not perform that well. Except for the Hamming-distance heuristic (where just a single encoding is considered), all these optimizations require a larger run-time with increasing size of the finite state machines and, therefore, possible encodings. The best results can be achieved with the most elaborated heuristic, i.e. the sifting heurisitic which clearly outperforms the random encoding.

Overall, using the proposed flow as well as the considered optimizations, sequential reversible circuit can be realized exploiting combinational synthesis methods. While here results obtained by the transformation-based approach and the KFDD-based approach have been evaluated, any other synthesis method can be integrated into the flow as well.

VII. Conclusion

In this paper, a synthesis flow for sequential reversible circuits has been proposed. In particular, a finite state machine is transformed into a Boolean function representing the sequential behavior. Afterwards, this function is realized using combinational synthesis methods which may require an additional embedding step. The approach has been applied using different synthesis methods, therewith providing a case study investigating the applicability of an established sequential synthesis flow in the domain of reversible logic for the first time. Heuristics ensure that encodings for the states are applied which keep the costs of the resulting circuits low. The resulting circuits can be applied to different paradigms proposed for sequential reversible circuits, i.e. the respective pseudo primary inputs can be triggered by an external controller or by reversible memory elements.

[4]For quantum cost and transistor cost the metric provided in [20] is applied.

[5]Please note that, due to page limitations, only the results obtained by the KFDD-based approach are reported.

Acknowledgments

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

References

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, July 1961.

[2] V. V. Zhirnov, R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff, "Limits to binary logic switch scaling – a gedanken model," *Proc. of the IEEE*, vol. 91, no. 11, pp. 1934–1939, Nov. 2003.

[3] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, Nov. 1973.

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

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

[6] M. Soeken, R. Wille, and R. Drechsler, "Hierarchical synthesis of reversible circuits using positive and negative davio decomposition," in *Workshop on Reversible Computation*, July 2010, pp. 55–58.

[7] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler, "Synthesis of Reversible Circuits with Minimal Lines for Large Functions," in *Asia and South Pacific Design Automation Conference*, Jan. 2012.

[8] H. Thapliyal, M. Srinivas, and M. Zwolinski, "A beginning in the reversible logic synthesis of sequential circuits," in *Int'l Conf. on Military and Aerospace Programmable Logic Devices*, Sept. 2005.

[9] M.-L. Chuang and C.-Y. Wang, "Synthesis of reversible sequential elements," in *Asia and South Pacific Design Automation Conference*. IEEE, Jan. 2007, pp. 420–425.

[10] N. M. Nayeem, M. A. Hossain, L. Jamal, and H. M. H. Babu, "Efficient design of shift registers using reversible logic," in *Int'l Conf. on Signal Processing Systems*. IEEE, May 2009, pp. 474–478.

[11] H. Himanshu and N. Ranganathan, "Design of reversible sequential circuits optimizing quantum cost, delay, and garbage outputs," *J. Emerg. Technol. Comput. Syst.*, vol. 6, pp. 14:1–14:31, 2010.

[12] M. Lukac and M. A. Perkowski, "Quantum finite state machines as sequential quantum circuits," in *Int'l Symp. on Multiple-Valued Logic*. IEEE Computer Society, May 2009, pp. 92–97.

[13] L.-K. Chang and F.-C. Cheng, "Automatic synthesis of composable sequential quantum boolean circuits," in *Int'l Conf. on Computer Design*. IEEE Computer Society, Oct. 2005, pp. 289–296.

[14] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. New York, NY, USA: Cambridge University Press, Oct. 2000.

[15] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, J. W. de Bakker and J. van Leeuwen, Eds., vol. 85. Springer, July 1980, pp. 632–644.

[16] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Systems Technical Journal*, vol. 34, Sept. 1955.

[17] R. Wille, O. Keszöcze, and R. Drechsler, "Determining the Minimal Number of Lines for Large Reversible Circuits," in *Design, Automation and Test in Europe*. IEEE Computer Society, Mar. 2011.

[18] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *ICCAD*, M. R. Lightner and J. A. G. Jess, Eds. IEEE Computer Society, 1993, pp. 42–47.

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

[20] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. Dordrecht, Heidelberg, London, New York: Springer, July 2010.

TABLE I
EXPERIMENTAL EVALUATION OF THE GENERIC SYNTHESIS FLOW

					Transformation-based Synthesis [4]					KFDD-based Synthesis [6]				
Function	#PI	#PO	$\|S\|$	$\|\operatorname{dom} T\|$	Lines	Gates	QC	TC	Run-time	Lines	Gates	QC	TC	Run-time
lion	2	1	4	11	8	455	27 950	16 944	0.29	19	42	118	472	0.01
dk15	3	5	4	32	-	-	-	-	TO	38	108	300	1 200	0.00
tav	4	4	4	49	13	34 555	6 389 160	2 185 520	2 139.85	26	56	164	608	0.00
s8	4	1	5	20	11	6 124	827 909	324 320	52.11	36	101	281	1 160	0.00
bbtas	2	2	6	24	8	613	34 557	22 736	0.44	21	49	137	552	0.01
dk27	1	2	7	14	13	34 555	6 389 160	2 185 520	2 157.78	16	45	117	464	0.00
beecount	3	4	7	28	12	16 618	2 591 761	960 456	433.88	38	115	351	1 352	0.01
dk14	3	5	7	56	11	6 780	781 658	356 160	68.01	47	158	498	1 896	0.01
shiftreg	1	1	8	16	9	1 289	93 980	55 152	1.78	18	41	113	456	0.00
ex6	5	8	8	34	-	-	-	-	TO	72	228	624	2 544	0.01
bbara	4	2	10	60	13	34 770	6 515 290	2 219 744	2 079.11	53	149	449	1 760	0.01
modulo12	1	1	12	24	8	518	30 244	19 552	0.35	17	42	118	456	0.01
ex4	6	9	14	21	-	-	-	-	TO	55	142	398	1 560	0.02
mark1	5	16	15	22	-	-	-	-	TO	65	187	515	2 016	0.02
dk512	1	3	15	30	9	1 108	84 788	46 368	1.42	34	110	286	1 176	0.00
sse	7	7	16	56	-	-	-	-	TO	87	252	724	2 880	0.00
bbsse	7	7	16	56	-	-	-	-	TO	87	252	724	2 880	0.02
cse	7	7	16	91	-	-	-	-	TO	148	469	1 401	5 520	0.02
keyb	7	2	19	170	-	-	-	-	TO	87	274	794	3 168	0.03
s1	8	6	20	107	-	-	-	-	TO	182	601	1 845	7 224	0.04
s1a	8	6	20	107	-	-	-	-	TO	126	387	1 199	4 632	0.04
ex1	9	19	20	138	-	-	-	-	TO	191	603	1 683	6 816	0.06
pma	8	8	24	73	-	-	-	-	TO	168	574	1 782	6 928	0.07
donfile	2	1	24	96	11	6 669	816 380	350 176	60.77	67	224	644	2 616	0.00
dk16	2	3	27	108	13	40 145	6 739 746	2 481 920	2 744.57	96	388	1 160	4 600	0.01
styr	9	10	30	166	-	-	-	-	TO	235	873	2 497	10 032	0.06
sand	11	9	32	184	-	-	-	-	TO	288	949	2 793	11 144	0.24
tbk	6	3	32	1 569	-	-	-	-	TO	211	758	2 034	8 312	0.07
planet	7	19	48	115	-	-	-	-	TO	268	949	2 789	11 096	0.04
planet1	7	19	48	115	-	-	-	-	TO	268	949	2 789	11 096	0.05

TABLE II
EXPERIMENTAL EVALUATION OF THE OPTIMIZATION OF THE STATE ENCODINGS

					Sequential Synthesis Random (QC)				Sequential Synthesis Hamming (QC)				Sequential Synthesis Sifting (QC)			
Function	#PI	#PO	$\|S\|$	$\|\operatorname{dom} T\|$	Lines	Gates	QC	Run-time	Lines	Gates	QC	Run-time	Lines	Gates	QC	Run-time
lion	2	1	4	11	-4	-6	-18	1.17	0	6	6	0.00	-4	-6	-18	0.02
dk15	3	5	4	32	0	1	-3	2.11	0	0	0	0.00	0	1	-3	0.09
s8	4	1	5	20	-3	-17	-61	2.66	2	25	33	-0.01	-5	-11	-43	0.18
bbtas	2	2	6	24	-2	-14	-26	1.28	-2	-9	-5	0.00	-1	-12	-24	0.12
dk27	1	2	7	14	-4	-13	-29	0.99	-4	-11	-31	0.01	-3	-12	-24	0.13
beecount	3	4	7	28	-1	-23	-71	1.92	-1	-5	-13	0.00	-3	-11	-43	0.29
dk14	3	5	7	56	-1	-4	-16	2.58	-1	30	14	-0.01	0	0	0	0.40
shiftreg	1	1	8	16	0	0	0	1.38	3	0	8	0.00	-5	-25	-53	0.24
ex6	5	8	8	34	-3	-37	-57	4.08	0	-26	-38	0.00	-11	-52	-120	0.74
bbara	4	2	10	60	-3	-23	-31	2.97	-6	-10	-14	-0.01	-1	-11	-23	0.91
modulo12	1	1	12	24	0	0	0	1.45	8	25	57	0.00	0	0	0	0.50
ex4	6	9	14	21	-15	-25	-73	3.84	-8	-3	-15	0.00	-14	-8	-52	2.24
mark1	5	16	15	22	-10	-16	-48	3.82	-7	-9	19	0.00	-11	-30	-58	2.84
bbsse	7	7	16	56	0	0	0	8.02	10	34	78	0.01	-8	-45	-121	5.44
cse	7	7	16	91	-1	11	-5	11.70	-1	2	-78	0.03	-26	-34	-166	8.73
keyb	7	2	19	170	0	0	0	11.43	36	155	331	0.01	-9	-44	-144	11.48
s1	8	6	20	107	15	42	-58	21.10	3	46	-6	0.01	-26	-127	-483	23.02
s1a	8	6	20	107	0	0	0	15.59	17	38	118	0.02	-21	-128	-396	16.43
ex1	9	19	20	138	0	0	0	24.50	31	92	276	0.02	-3	-53	-137	30.76
pma	8	8	24	73	0	0	0	26.15	-7	5	17	0.02	-15	-51	-167	50.24
donfile	2	1	24	96	0	0	0	4.07	7	38	98	0.00	-13	-50	-162	6.88
dk16	2	3	27	108	-6	-28	-48	5.09	3	31	59	-0.02	-19	-49	-153	12.45
styr	9	10	30	166	0	0	0	31.90	-12	8	56	0.03	-23	-90	-190	53.61
planet	7	19	48	115	-9	-43	-55	25.09	-11	8	48	0.01	-37	-102	-262	71.73
$\sum$					-47	-195	-599	214.89	60	470	1.018	0.12	-258	-950	-2.842	299.47