EFFICIENT REALIZATION OF CONTROL LOGIC IN REVERSIBLE CIRCUITS

Sebastian Offermann *Robert Wille* *Rolf Drechsler*

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{offerman, rwille, drechsle}@informatik.uni-bremen.de

ABSTRACT

The development of design methods for reversible circuits found significant attention in the last years. Circuits are thereby considered which – in contrast to conventional circuits – are composed of reversible gates only. This enables promising applications, e.g. for quantum computation or low-power design. The recent achievements in this domain enabled the development of synthesis approaches based on high level description languages. This emerges new research problems.

In this paper, we address the problem of efficient realization of control logic in reversible circuits. So far, existing methods realize control logic with a significant amount of redundant circuit structures. An alternative is presented that avoids large parts of these redundancies by buffering the results of recurring computations in one additional circuit line. Accordingly, the proposed approach enables to realize control logic with significantly less circuit lines, while the increase of the circuit cost remains moderate – in some cases even reductions are possible. This conclusion is also confirmed by an experimental evaluation.

I. INTRODUCTION

Research in the domain of design automation for reversible circuits received significant attention in the last years. Circuits are thereby designed, which – in contrast to conventional circuits – realize bijections, i.e. one to one mappings of primary inputs and primary outputs, respectively. Mainly driven by the promising applications of reversible circuits, e.g. in quantum computation [1] or low-power design [2], [3], a wide range of approaches for synthesis [4]–[8], optimization [9], [10], verification [11]–[13], testing [14]–[16], and even debugging [17] have been introduced. These methods incorporate the special properties and design philosophies needed for reversible circuits. So far, they allow to automatically design this kind of circuits up to a certain degree of complexity.

However, even if first physical realizations exploiting reversible logic (e.g. quantum circuits solving the factorization problem in polynomial time [18] or adder circuits with low-power properties [19]) have been built yet for small instances only, larger machines are planned. Therefore, design methods are needed enabling the specification and the realization of more complex reversible systems. Accordingly, researchers started to lift the existing methods to higher levels of abstractions. First results obtained by replacing the existing low level synthesis methods with a reversible *High level Description Language* (HDL) recently have been presented [20].

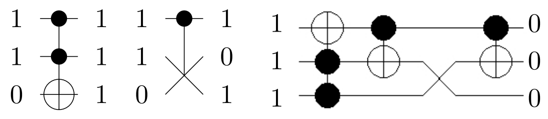
Following this direction, new research problems emerge that did not have intensely considered in the past. Using programming languages in order to design reversible circuits, the efficient realization of both, the data flow and the control flow, are crucial – respective building blocks are needed. While for the data-flow, some progress has been made e.g. in the realization of certain arithmetic structures like adders [21] or multipliers [22], [23], the development of efficient realizations for the control flow is just at the beginning. In particular, how to efficiently realize conditional statements is an open question. Existing methods introduced in [20] suffer either from a large number of additional circuit lines (a very limited resource) or from very large circuit cost (this is considered in more detail in Section III).

In this paper, a new method to realize conditional statements in reversible logic is presented which addresses these drawbacks. We exploit thereby the observation that existing realizations include a significant amount of redundancies. An alternative is presented that avoids large parts of these redundancies by buffering the results of recurring computations in one additional circuit line. Accordingly, the resulting realizations have significant less circuit cost, while the number of additional circuit lines is increased only by one.

Buffering results of recurring computations is thereby not completely new and already has been applied in [24]. There, a simple greedy search approach has been introduced to detect appropriate circuit structures. In this paper, we adapt and extend the idea of buffering explicitly for the purpose of control logic realization. Furthermore, while the approach introduced in [24] is applicable to low level circuits only, our method exploits all the information from the HDL and, thus, can be applied to complex designs as well.

Experimental results show that the proposed approach provides a trade-off between the existing methods. Both, the number of circuit lines as well as the respective quantum cost and transistor cost of the resulting circuits remain moderate, while previously introduced realizations suffer from either one of these criteria.

The remainder of this paper is structured as follows: The first two sections provides the basis for the rest of this work. More precisely, Section II introduces reversible circuits, while Section III briefly reviews and discusses existing realizations of control logic. Afterwards, the general idea and the main concepts of the proposed approach are introduced in Section IV. Section V describes the implementation. Finally, experimental results and conclusions are given in Section VI and Section VII, respectively.



(a) Reversible gates (b) Reversible circuit

Fig. 1. Reversible circuitry

II. REVERSIBLE CIRCUITS

Reversible circuits realize reversible functions, i.e. functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ where each input pattern uniquely maps to a certain output pattern. In other words, each reversible circuit realizes a bijection that perform permutations on the set of input patterns.

Definition 1. *A* reversible circuit *G over inputs X is a cascade of reversible gates g_i, i.e. $G = g_1 g_2 \cdots g_d$ where d is the number of gates. A* reversible gate *has the form $g(C, T)$, where $C \subset X$ is the set of* control lines *and $T \subset X$ with $C \cap T = \emptyset$ is the set of* target lines. *The gate operation is applied to the target lines iff all control lines meet the required control conditions. Control lines and unconnected lines always pass through the gate unaltered. The set of circuits over inputs X is denoted by $\mathcal{G}$.*

In the past, the *Toffoli gate* [25] and the *Fredkin gate* [26] established themselves as a universal gate library for reversible circuits. They are also considered in this paper.

Definition 2. *A (*multiple control) Toffoli gate *$t(C, \{x\})$ maps its single target line x to $\bigwedge_{c \in C} c \oplus x$. That is, a Toffoli gate inverts the target line iff all control lines are assigned to* 1.

*A (*multiple control) Fredkin gate *$f(C, \{x_1, x_2\})$ interchanges the values of the target lines x_1 and x_2 iff the conjunction of all control lines evaluates to* 1.

Example 1. *Fig. 1(a) shows a Toffoli gate with two control lines and a Fredkin gate with one control line, respectively, with possible input/output assignments. Fig. 1(b) shows different reversible gates in a cascade forming a reversible circuit.*

In the following, reversible gates and reversible circuits may be enriched with additional control lines.

Definition 3. *Let $g(C, T)$ be a reversible gate. To attach additional control lines $C' \subset X$ with $C \cap C' = \emptyset$ to g, the following notation is used:*

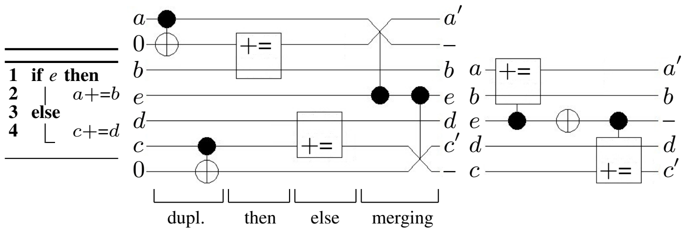
$$C' \mapsto g(C, T) := g'(C \cup C', T)$$

Accordingly, the same notation is used for a cascade G of gates.

The costs of a reversible circuit are measured by various cost metrics (sometimes depending on the addressed technology). In general, the number of circuit lines is an important criterion since it represents a very limited resource. Additionally, the costs of the respective gates themselves are important, too. Since simply counting the number of gates does not adequately reflect the effort to realize them, so called *quantum cost* [27] and *transistor cost* [28] have been introduced. These costs heavily depend on the number of control line connections in a gate. Table I lists the respective costs for the most common cases[1].

Table I. Costs of reversible gates

Control	Toffoli gate		Fredkin gate	
lines	Quant. cost	Trans. cost	Quant. cost	Trans. cost
0	1	0	3	0
1	1	8	7	8
2	5	16	15	16
3	13	24	28-31	24
4	26-29	32	40-63	32
5	38-61	40	52-127	40
6	50-125	48	64-255	48



```
if e then
| a+=b
else
└ c+=d
```

(a) Code (b) Using duplication (c) Using add. contr.

Fig. 2. Realization of an if-statement

Example 2. *The circuit in Fig. 1(b) is composed of 3 circuit lines as well as 4 gates and has quantum cost of 10 and transistor cost of 32, respectively.*

III. REALIZATION OF CONTROL LOGIC

How to efficiently design and synthesize reversible circuits is an active research area. First approaches are predominantly based e.g. on Boolean descriptions [4]–[8] and, thus, provide limited capacity only. Consequently, researchers started to investigate higher levels of abstraction raising the attention to hardware description languages for reversible circuits [20]. Here, the efficient realization of both, the data flow and the control flow, is crucial. While aspects of the data flow already are under detailed investigation (see e.g. [21], [23]), synthesis of reversible circuits for control logic was not investigated in much detail.

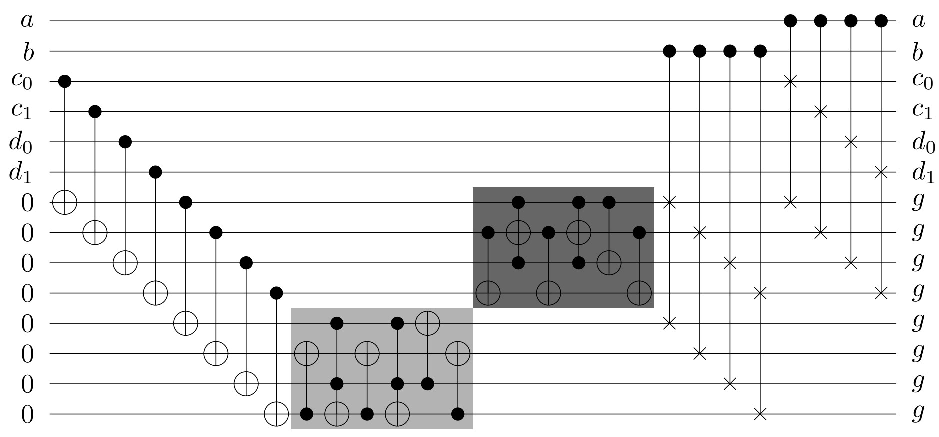
The realization of loops and procedure calls is thereby straightforward, since the respective instructions simply have to be cascaded together. In contrast, *conditional statements* require more elaborated methods. To the best of our knowledge, only the two approaches depicted in Fig. 2 are known so far [20]:

1) The first one (shown in Fig. 2(b)) relies on duplication. Here, the values of all signals that possibly might be affected in an if- or else-block are copied (using an additional circuit line with a constant input as shown by Signal a and Signal c in Fig. 2). Then, sub-circuits realizing the respective if-/else-block are added (denoted by the boxes). Finally, depending on the result of the conditional statement (Signal e in Fig. 2), the values of the duplicated lines and the

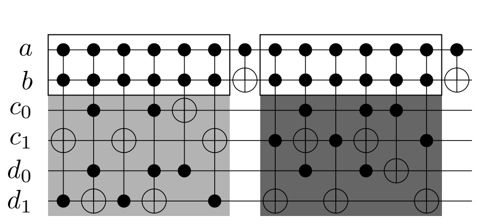
[1]Note that depending on the concrete configuration of a gate, improvements in the costs are possible. This is the reason why e.g. the quantum cost for a Toffoli gate with 4 control lines range from 26 to 29.

```
1 if $a$ then
2 |   if $b$ then
3 |   |   $c$ += $d$
4 |   else
5 |   |   $d$ += $c$
6 else
7 |   skip
```

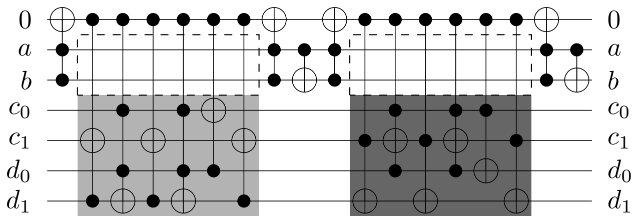
(a) Control logic



(b) Realization using duplication



(c) Realization using additional controls



(d) Proposed realization

Fig. 3. Different realizations of control logic illustrating the general idea

original lines are swapped leading to the desired result which can be used in the following.

2) The second realization (shown in Fig. 2(c)) makes intensive use of control connections. More precisely, control lines are added to all gates in the realization of the respective then- and else-block. Therewith, the gates in these blocks are only triggered iff the result of the conditional statement (i.e. signal e) is assigned to 1 or 0, respectively. A NOT gate (i.e. a Toffoli gate $t(\emptyset, \{e\})$ without control lines) is thereby applied to flip the value of e so that the gates of the else block can be "controlled" as well.

Both realizations have serious drawbacks. Obviously, the first approach makes extensive use of additional circuit lines. Since this is a very limited resource, researchers try to keep this number as small as possible. In contrast, the second one requires no additional circuit lines, but due to the added control connections the cost of a single gate increases significantly.

Example 3. *Consider the nested if-statement shown in Fig. 3(a). In the respective blocks, an increase operation is applied to variables over two bits. Using the duplication method, the circuit depicted in Fig. 3(b) with 8 additional lines and quantum cost (transistor cost) of 92 (256) result (the realizations of the respective if-/else blocks are shadowed). Using the second approach, the circuit depicted in Fig. 3(c) with no additional lines, but quantum cost (transistor cost) of 222 (336) is generated. That is, using the existing approaches, the designer has to trade-off either to accept more than twice the number of circuit lines or twice the number of quantum cost (in case of transistor cost an increase by still 30%).*

IV. General Idea

In this section, we present the basic observations motivating an alternative realization of control logic in reversible circuits. The goal is to present a solution which provides a trade-off between the two complementary optimization goals. In fact, a realization is proposed which increases the number of circuit lines only by one, but on the other side reduces the gate costs significantly. The fact that control logic leads to redundant circuit structures is thereby exploited.

To illustrate this, consider again the nested if-statement from Fig. 3(a). The first block shall be executed only if the conditional statement a evaluates to true. Moreover, the second block shall be executed only if additionally the conditional statement b evaluates to true. As described in the previous section, this can be realized by either duplicating the respective signal values (as done in Fig. 3(b)) or by adding further control connections (as done in Fig. 3(c)) – both leading to redundancies.

Consider in more detail the latter method (see Fig. 3(c)). Here, a reversible cascade results including many similar gates (namely gates with control lines at a and b, tagged by a box around them). In the following, such cascades are called *$\hat{C}$-controlled cascades.*

Definition 4. *Let $G = g_1 g_2 \dots g_d$ be a cascade of reversible gates. If all these gates have a common subset $\hat{C}$ of control lines (i.e. if $\hat{C} \subset C_i$ holds for all gates $g_i(C_i, T_i)$ with $1 \leq i \leq d$), G is denoted by $\hat{C}$-*controlled cascade.

Note that each gate $g_i(C_i, T_i)$ of a $\hat{C}$-controlled cascade can be written as $\hat{C} \mapsto g_i(C'_i, T_i)$ with $C'_i = C_i \setminus \hat{C}$.

Example 4. *The first if-block is realized by an $\{a\}$-controlled cascade in Fig. 3(c), while the second if-block is realized by an $\{a, b\}$-controlled cascade.*

$\hat{C}$-controlled cascades represent the heart of the control logic, since they trigger whether a cascade (representing a block) is executed or not. But, these control connections include a significant amount of redundancies since identical control connections are permanently evaluated. This observation is exploited by our approach.

In order to remove the redundancies, we suggest to temporarily buffer the evaluated value of the common control lines within a $\hat{C}$-controlled cascade. Therefore, a single circuit line is added to the circuit to be synthesized. This line works as corresponding buffer which enables to significantly reduce the large amount of redundant control connections. More precisely:

Definition 5. *Let $G = g_1 g_2 \dots g_d$ be a $\hat{C}$-controlled cascade. In order to remove the redundancies, this cascade is replaced by an* optimized $\hat{C}$-controlled cascade *defined as follows:*

- *A new circuit line buf with a constant input 0 is added to the original cascade.*
- *A new Toffoli gate $t_{buf}(\hat{C}, \{buf\})$ is added at both ends of the original cascade, i.e. the cascade $t_{buf} g_1 g_2 \dots g_d t_{buf}$ is constructed. Due to the constant input and the first gate, the circuit line buf now buffers the evaluated value of the common control lines.*
- *Each gate $(\hat{C} \mapsto g_i(C'_i, T_i))$ with $1 \leq i \leq d$ is replaced by the gate $(\{buf\} \mapsto g_i(C'_i, T_i))$, i.e. a gate where the control connections given in $\hat{C}$ are removed and, instead, the control connection buf is used.*

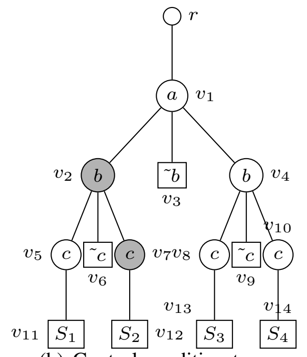
Note that the optimized cascades should be applied to $\hat{C}$-controlled cascades with $|\hat{C}| > 1$. Otherwise, the number of control connections does not change. Furthermore, adding the gate t_{buf} at both ends of the cascade first of all increases the cost of the circuit. But in particular for larger $\hat{C}$-controlled cascades, this increase can easily be compensated by the removal of the redundancies. Finally, note that the right most gate t_{buf} is not necessarily needed. However, it sets the line buf back to the constant value 0 so that this line can be used later for another $\hat{C}$-controlled cascade.

Example 5. *Consider again the realization of the nested if-statement from Fig. 3(c). Applying the proposed method to both $\{a, b\}$-controlled cascades lead to the circuit shown in Fig. 3(d). Already in this simple example, this reduces the quantum cost from 222 to 114 and the transistor cost from 336 to 304. At the same time, the number of lines is increased by 1 only. In comparison to the realization using duplication (Fig. 3(b)), the number of additional lines is reduced from 8 to 1, while the cost increase is quite moderate (92 to 114 in case of quantum cost and 256 to 304 in case of transistor cost).*

In summary, $\hat{C}$-controlled cascades are an integral part of control logic and include a significant amount of redundancies which can be removed as illustrated above. Based on this general idea, the next section describes how these observations are exploited in order to efficiently realize control logic in reversible circuits.

```
if $a$ then
    if $b$ then
        if $c$ then
            $S_1$
        else
            $S_2$
    else
        if $c$ then
            $S_3$
        else
            $S_4$
```

(a) Conditional statement



(b) Control-condition tree

Fig. 4. If statement and its control-condition tree

V. IMPLEMENTATION

In this section, an algorithm is proposed that makes use of optimized $\hat{C}$-controlled cascades in order to synthesize the desired control logic with low cost and only a moderate increase in the number of circuit lines. The approach is thereby build on top of an extended HDL synthesizer which is briefly described in the first subsection. Having that, two new stages are applied: First, a structural analysis is performed determining whether an optimized cascade should be used or not. Then, the actual synthesis is executed.

V-A. HDL Synthesizer

In order to synthesize circuits specified in a hardware description language, we use a similar flow as recently presented in [20]: A hierarchical synthesis method is applied that traverses a given HDL program first. Afterwards, existing realizations of operations are applied as building blocks and combined together so that the desired circuit results. In order to efficiently realize control logic within this process, an additional data-structure defined as follows is joined up:

Definition 6. *Given an HDL program to be synthesized as reversible circuit. To represent the respective control-conditions, a* control-condition tree *$T = (V, s, c, g)$ is created, where*

- *$V = \{r\} \cup V_l \cup V_c$ is the set of nodes including r representing the root, the leaves V_l representing statement blocks, and the non-terminal nodes V_c representing controlled cascades,*
- *$s : V \to V^*$ is a function providing an ordered list of successors of a given node,*
- *$c : V_c \to X$ is a function providing the control connection for a given non-terminal node, and*
- *$g : V_l \to \mathcal{G}$ is a function providing a circuit realization of the statement block represented by the given leaf.*

Example 6. *Fig. 4(a) shows an HDL program. The corresponding control-condition tree is given in Fig. 4(b). Here, e.g. the application of $s(v_2)$ leads to $[v_5, v_6, v_7]$, $c(v_2)$ evaluates to b, and $g(v_{12})$ represents the circuit realization of the statement S_2.*

Using this data-structure, all information needed to synthesize the HDL program is available. In particular, the control connections are easily accessible: For any node $v \in V$, the set of lines $\hat{c}(v)$ controlling the represented cascade can be derived by recursively applying the function c.

In a similar, recursive manner, circuits are realized for a given $v \in V$. More precisely, a circuit realizing the cascade represented by v is generated by

$$\hat{g}(v) := \begin{cases} g(v), & \text{if } v \in V_l \\ \top_{u \in s(v)} \begin{cases} \{c(u)\} \mapsto \hat{g}(u), & \text{if } u \in V_c \\ \hat{g}(u), & \text{else} \end{cases}, & \text{else} \end{cases}$$

whereby $\top_{u \in s(v)}$ is an n-tupel representing the concatenation of the results of each successor $u \in s(v)$.

Example 7. *Applying the recursive functions to the nodes v_2 and v_7 of the control-condition tree in Fig. 4(b) provide the following values:*

$$\hat{c}(v_2) = \{a, b\}$$
$$\hat{g}(v_2) = (\{c\} \mapsto g(S_1), t(\emptyset, \{c\}), \{c\} \mapsto g(S_2))$$
$$\hat{c}(v_7) = \{a, b, c\}$$
$$\hat{g}(v_7) = g(S_2)$$

Note that using the abstraction of the control-condition tree, not only conditional statements, but also similar constructs (e.g. resulting from dynamic array realization, use of controlled functions as in synthesis of multiplication [23], and increment etc.) can be treated. Using the control-condition tree, the occurrences of $\hat{C}$-controlled cascades can easily be identified and analysized. As described in the next section, this is necessary in order to make the best use of optimized $\hat{C}$-controlled cascades.

V-B. Structural Analysis

Usually, every description of a non-trivial reversible circuit includes many different conditional statements nested into each other. This leads to a significant number of different and, in particular, overlapping $\hat{C}$-controlled cascades. Since not all of them can be replaced by an optimized $\hat{C}$-controlled cascade simultaneously, a structural analysis is performed first. This is motivated by the following example.

Example 8. *Consider the control-condition tree as shown in Fig. 4(b). Synthesizing this control logic according to the additional control line scheme lead to a circuit with an $\{a\}$-controlled cascade, two $\{a, b\}$-controlled cascades, and four $\{a, b, c\}$-controlled cascades. However, optimized $\hat{C}$-controlled cascades cannot be applied to all of them simultaneously. In fact, with a single additional line buf, only the value of the common control connections of one cascade can be buffered at one time. Hence, optimized $\hat{C}$-controlled cascades can be applied either*

- *to the whole $\{a\}$-controlled cascade,*
- *to both $\{a, b\}$-controlled cascades,*
- *to the four $\{a, b, c\}$-controlled cascades,*
- *to the first $\{a, b\}$-controlled cascade and the last two $\{a, b, c\}$-controlled cascades, or*
- *to the first two $\{a, b, c\}$-controlled cascades and the second $\{a, b\}$-controlled cascade.*

Obviously, which of these alternatives should be selected is crucial to the resulting cost of the circuit. While replacing the whole $\{a\}$-controlled cascade with an optimized one

```
Input  : $v$ controlled cascade Tree
Input  : $buf$ (additional circuit line with const. inp. 0)
Output: $bool$ whether to apply an optimized version of
        the cascade represented by $v$
1  decide( $v$, $buf$ ):
2  return ( optCost( $v$, $buf$ ) $\leq$ bestCost( $v$, $buf$ ) )
3  bestCost( $v$, $buf$ ):
4  $stdCost$ = cost( $\hat{c}(v) \mapsto \hat{g}(v)$ )
5  $optCost$ = cost( $\{buf\} \mapsto \hat{g}(v)$ )
6            + 2 $\cdot$ cost( $t(\hat{c}(v), \{buf\})$ )
7  if $v \in V_l$ then
8      return min( $stdCost$, $optCost$ )
9  $sucCost$ = 0
10 foreach $u \in s(v)$ do
11     $sucCost$ += bestCost( $u$, $buf$ )
12 return min( $stdCost$, $optCost$, $sucCost$ )
```

Fig. 5. Algorithm for cascade optimization decision

does not lead to any reductions at all (see the discussion in the previous section), the effect of choosing one of the remaining possibilities varies significantly depending on the concrete nature of the respective if- and else-blocks.

Thus, to determine the best improvement, a structural analysis is performed first. That is, for each node of the control-condition tree to be realized, it is checked whether an optimized version of the represented cascade should be applied or not. This is determined by the function *decide* shown in Fig. 5. This function applies *bestCost* (also shown in Fig. 5) to compute the costs of the following possible realizations:

- the cost *stdCost* of the cascade represented by v without the proposed optimization (Line 4),
- the cost *optCost* of the cascade represented by v with the proposed optimization (Line 5), and
- the best cost *sucCost* obtained by applying the proposed optimization to any of the succeeding cascades represented by $u \in s(v)$ independently (recursively computed in Line 9 to Line 11).

Afterwards, the resulting costs are compared (Line 2). If and only if the application of the proposed optimization at the cascade represented by v leads to the smallest cost, the respective optimized $\hat{C}$-controlled cascade is applied. Therewith, the proposed optimization is applied to the best possible cascade.

Example 9. *Consider again the control-condition tree depicted in Fig. 4(b). The application of the* decide*-function is illustrated by means of Node v_2, i.e. it is checked whether an optimized $\{a, b\}$-controlled cascade should be applied or not. Both statements S_1 and S_2 are thereby assumed to be 2-bit additions, i.e. statements which are realized by 2 Toffoli gates including 2 control lines and 4 Toffoli gates including 1 control line. Having this, the respective costs are as follows:*

- *If the cascade represented by v_2 is realized without optimization, three additional control lines (namely a, b, c) are applied to the realization of the additions and two*

```
Input  : $v$ controlled cascade Tree
Input  : $buf$ (additional circuit line with const. inp. 0)
Output: $G$ circuit realizing $v$ (initially empty)
1  optCascade( $v$, $G$, $buf$ ):
2  if ( decide( v ) ) then
3      $G = G\ t(\hat{c}(v), \{buf\})$
4      $G = G\ (\{buf\} \mapsto \hat{g}(v))$
5      $G = G\ t(\hat{c}(v), \{buf\})$
6  else
7      if $v \in V_l$ then
8          $G = G\ \hat{g}(v)$
9      else
10         foreach $u \in s(v)$ do
11             $G = G$ optCascade( $u$, $G$, $buf$ )
12 return $G$
```

Fig. 6. Algorithm for exact cascade optimization

additional control lines (namely a, b) are applied to the negation. This amounts to a cost of $stdCost$ =365.

- *If the optimized $\{a, b\}$ controlled-cascade is applied, only two additional control lines (namely buf, c) are applied to the realization of the additions and only one additional control line (namely buf) is applied to the negation. But then, two additional Toffoli gates $t(\{a, b\}, \{buf\})$ are required amounting to a total cost of 219.*
- *However, a cheaper realization results, if optimal controlled cascades are applied at Node v_4 and Node v_6. Then, one additional control line (namely buf) is applied to the additions and two additional control lines (namely a, b) are applied to the negation. Additionally, four Toffoli gates $t(\{a, b, c\}, \{buf\})$ are required. Nevertheless, this leads to total cost of 149.*

Accordingly, the decide-*function returns* false *and no optimized cascade is applied at Node v_2.*

V-C. Synthesis

Combining the data-structures and strategies introduced above, an algorithm as shown in Fig. 6 results. This algorithm synthesizes a reversible circuit G from a given control-connection tree with improved handling of the control structures. It gets the node corresponding to the HDL program that should be realized (usually the root node r) and an additional line that can be used for the optimization of the controlled cascades. As result, the algorithm returns the synthesized circuit.

The algorithm traverses the control connection tree beginning with a given node. For each node, it checks whether the currently represented cascade should be optimized or not. Therefore, the *decide*-function introduced above is applied (Line 2). If this function returns *true*, the respective, optimized $\hat{C}$-controlled cascade is generated (Line 3 to Line 5). Otherwise, the algorithm is recursively called for all successors $u \in s(v)$ (Line 11).

VI. Experiments

The proposed approach has been implemented in C++ and evaluated using HDL programs specified in the SyReC language [20]. The programs define various components of a simple processor including an arithmetic logic unit realizing addition, subtraction, multiplication, and xor (denoted by *alu*), a logic unit realizing Boolean operations (denoted by *lu*), different control units (denoted by *contr_unit1*, *contr_unit2*, and *contr_unit3*), and different versions of a program counter (denoted by *pc1* and *pc2*). The logic units are thereby evaluated using different bit-widths. Beyond that, also an arbiter with 8 clients (denoted by *arb8*) has been considered. All experiments have been carried out on an AMD DualCore Athlon 3GHz machine with 32 GB of main memory.

Table II summarizes the obtained results. Besides the proposed approach, also circuits realizing control logic using duplication and circuits realizing control logic using additional controls (see Section III) are considered for comparison. The respective columns give the name of the circuit (denoted by *Benchmark*), the applied bit-width (denoted by *bit-width*), the number of primary inputs (denoted by *PI*), the number of additional lines with constant inputs (denoted by *CI*), the quantum cost (denoted by *QC*), the transistor cost (denoted by *TC*), and the needed synthesis time (in CPU seconds and denoted by *Time*).

The results confirm that the proposed approach provides a trade-off between the existing methods. In comparison to the duplication-based method, the number of additional lines is decreased by nearly 50% on average. Thus, with respect to the number of lines, nearly as good results as with the method using additional controls are achieved. Moreover, while the method based on additional controls leads to circuits with very large quantum cost and transistor cost, this increase is moderate using the proposed approach. In cases of the control units (that inherently include a significant portion of control logic) even some substantial reductions can be achieved. Overall, using the proposed approach, both, the number of circuit lines as well as the respective gate costs, remain moderate, while the previously introduced approaches suffer from either one of these criteria.

VII. Conclusions

In this paper, a new method to realize conditional statements in reversible logic is presented. Therefore, the observation that existing realizations include a significant amount of redundancies is exploited. In doing so, the drawbacks of the existing realizations are explicitly addressed. In fact, the proposed approach enables to realize control logic with less additional circuit lines, while – at the same time – the increase of the quantum cost and transistor cost remains moderate or is even reduced.

VIII. Acknowledgment

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

IX. References

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

[2] R. Landauer, “Irreversibility and heat generation in the computing process,” *IBM J. Res. Dev.*, vol. 5, pp. 183, 1961.

Table II. Experimental results

			Realizing control logic using											
			duplication				additional controls				the proposed approach			
Benchmark	Bit-width	PI	CI	QC	TC	Time	CI	QC	TC	Time	CI	QC	TC	Time
alu	8	26	65	1918	5792	0.049s	41	11073	13128	0.052s	42	4705	8784	0.048s
alu	16	50	121	7082	19808	0.053s	73	40697	45160	0.061s	74	16829	29168	0.055s
alu	32	98	233	27202	72416	0.058s	137	155625	166056	0.088s	138	63541	105264	0.075s
lu	8	26	64	253	1328	0.047s	40	1965	2976	0.045s	41	744	2040	0.043s
lu	16	50	120	461	2544	0.047s	72	3773	5600	0.046s	73	1320	3640	0.045s
lu	32	98	232	877	4976	0.025s	136	7389	10848	0.050s	137	2472	6840	0.047s
contr_unit1	32	215	320	4439	15472	0.130s	124	14481	22760	0.084s	125	2858	9984	0.085s
contr_unit2	32	200	74	6839	13248	0.008s	38	13568	17312	0.012s	39	2432	7072	0.016s
contr_unit3	32	527	4414	572882	797920	1.006s	318	1744399	1607064	0.959s	319	209170	388848	1.096s
pc1	–	8	17	53	312	0.018s	5	153	328	0.005s	6	97	304	0.009s
pc2	–	8	10	26	168	0.007s	0	34	104	0.007s	1	32	120	0.012s
arb8	–	16	44	80	640	0.302s	8	746	800	0.271s	9	391	576	0.262s

[3] C. H. Bennett, “Logical reversibility of computation,” *IBM J. Res. Dev*, vol. 17, no. 6, pp. 525–532, 1973.

[4] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, “Synthesis of reversible logic circuits,” *IEEE Trans. on CAD*, vol. 22, no. 6, pp. 710–722, 2003.

[5] D. Maslov, G. W. Dueck, and D. M. Miller, “Toffoli network synthesis with templates,” *IEEE Trans. on CAD*, vol. 24, no. 6, pp. 807–817, 2005.

[6] P. Gupta, A. Agrawal, and N. K. Jha, “An algorithm for synthesis of reversible logic circuits,” *IEEE Trans. on CAD*, vol. 25, no. 11, pp. 2317–2330, 2006.

[7] R. Wille, H. M. Le, G. W. Dueck, and D. Große, “Quantified synthesis of reversible logic,” in *Design, Automation and Test in Europe*, 2008, pp. 1015–1020.

[8] R. Wille and R. Drechsler, “BDD-based synthesis of reversible logic for large functions,” in *Design Automation Conf.*, 2009, pp. 270–275.

[9] D. Y. Feinstein, M. A. Thornton, and D. M. Miller, “Partially redundant logic detection using symbolic equivalence checking in reversible and irreversible logic circuits,” in *Design, Automation and Test in Europe*, 2008, pp. 1378–1381.

[10] R. Wille, M. Soeken, and R. Drechsler, “Reducing the number of lines in reversible circuits,” in *Design Automation Conf.*, 2010, pp. 647–652.

[11] G. F. Viamontes, I. L. Markov, and J. P. Hayes, “Checking equivalence of quantum circuits and states,” in *Int’l Conf. on CAD*, 2007, pp. 69–74.

[12] S.-A. Wang, C.-Y. Lu, I-M. Tsai, and S.-Y. Kuo, “An XQDD-based verification method for quantum circuits,” *IEICE Transactions*, vol. 91-A, no. 2, pp. 584–594, 2008.

[13] R. Wille, D. Große, D. M. Miller, and R. Drechsler, “Equivalence checking of reversible circuits,” in *Int’l Symp. on Multi-Valued Logic*, 2009, pp. 324–330.

[14] K. N. Patel, J. P. Hayes, and I. L. Markov, “Fault testing for reversible circuits,” *IEEE Trans. on CAD*, vol. 23, no. 8, pp. 1220–1230, 2004.

[15] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes, “A family of logical fault models for reversible circuits,” in *Asian Test Symp.*, 2005, pp. 422–427.

[16] R. Wille, H. Zhang, and R. Drechsler, “ATPG for reversible circuits using simulation, Boolean satisfiability, and pseudo Boolean optimization,” in *IEEE Annual Symposium on VLSI*, 2011.

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

[18] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, “Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance,” *Nature*, vol. 414, pp. 883–887, 2001.

[19] B. Desoete and A. De Vos, “A reversible carry-look-ahead adder using control gates,” *INTEGRATION, the VLSI Jour.*, vol. 33, no. 1-2, pp. 89–104, 2002.

[20] R. Wille, S. Offermann, and R. Drechsler, “SyReC: A programming language for synthesis of reversible circuits,” in *Forum on Specification and Design Languages*, 2010, pp. 184–189.

[21] Y. Takahashi and N. Kunihiro, “A linear-size quantum circuit for addition with no ancillary qubits,” *Quantum Information and Computation*, vol. 5, pp. 440–448, 2005.

[22] M.S. Islam, M.M. Rahman, Z. Begum, and M.Z. Hafiz, “Low cost quantum realization of reversible multiplier circuit,” *Information Technology Journal*, vol. 8, no. 2, pp. 208–213, 2009.

[23] S. Offermann, R. Wille, G. W. Dueck, and R. Drechsler, “Synthesizing Multiplier in Reversible Logic,” in *Int’l Symp. on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 335–340.

[24] D. M. Miller, R. Wille, and R. Drechsler, “Reducing reversible circuit cost by adding lines,” in *Int’l Symp. on Multi-Valued Logic*, 2010, pp. 217–222.

[25] T. Toffoli, “Reversible computing,” in *Automata, Languages and Programming*, W. de Bakker and J. van Leeuwen, Eds., p. 632. Springer, 1980, Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[26] E. F. Fredkin and T. Toffoli, “Conservative logic,” *International Journal of Theoretical Physics*, vol. 21, no. 3/4, pp. 219–253, 1982.

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

[28] M. K. Thomson and R. Glück, “Optimized reversible binary-coded decimal adders,” *J. of Systems Architecture*, vol. 54, pp. 697–706, 2008.