Towards HDL-based Synthesis of Reversible Circuits with No Additional Lines

Robert Wille*, Majid Haghparast†, Smaran Adarsh‡, and Tanmay M‡
*Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
†Yadegar-e-Imam Khomeini (RAH) Shahre Rey Branch, Islamic Azad University, Tehran, Iran
‡People's Education Society University, Bangalore, India
robert.wille@jku.at http://iic.jku.at/eda/research/quantum/

***Abstract*—Reversible circuits are needed in different emerging technologies, but their design is still mainly conducted on low abstraction levels thus far. *Hardware Description Languages* (HDLs) provide suitable description means to lift the design process to higher levels of abstractions. However, synthesis of HDL descriptions thus far still relies on non-reversible building blocks even if the corresponding statements are purely reversible. This leads to reversible circuits with additional circuit lines (i.e., circuit signals)—rendering HDL-based synthesis infeasible for many applications such as quantum computing. In this work, we present a synthesis method which realizes many of the HDL statements with no additional lines at all. To this end, we consider the respective (reversible) HDL statements as an entirety rather than breaking it down into (possibly non-reversible) building blocks. For the first time, this allows to realize many HDL descriptions with no additional circuit lines.**

I. Introduction

Reversible circuits realize an unconventional form of computing in which operations can be employed in both directions, i.e., from the inputs to the outputs and vice versa. The resulting properties are very useful for designing and realizing different emerging technologies. Most prominently, this finds application in the design of quantum circuits [1] which are inherently reversible and, caused by the current momentum in this area, many design approaches first realize reversible circuits which, afterwards, are mapped to corresponding quantum realizations [2], [3]. But also other areas such as adiabatic circuits [4], [5], [6], the design of encoders [7], on-chip interconnects [8], certain aspects of low power design [9], [10], or even verification [11] heavily utilize principles of reversible circuits.

Accordingly, a substantial amount of work has been spent on the efficient synthesis of reversible circuits. Originally, most synthesis approaches focused on realizing the desired circuit with a minimal number of circuit lines[1] (e.g., approaches based on truth-tables [12], permutations [13], positive-polarity Reed-Muller spectra [14], and Boolean satisfiability [15]). But since they relied on exponential function representations, their scalability remained limited. In contrast, hierarchical synthesis approaches have been proposed which e.g., employed a divide and conquer approach, i.e., they decompose the function to be realized into smaller functions for which corresponding building blocks are available. Afterwards, these building blocks are cascaded together with respect to the applied decomposition so that eventually the desired circuit results. Approaches relying on two level descriptions [16], [17] or decision diagrams [18], [19] fall in this category. Besides that, also hybrid approaches exists [20]. However, in all these approaches, the design is still mainly conducted on low abstraction levels.

[1]Note that, in the domain of reversible circuits, the respective circuit signals are usually called *circuit lines*.

Hardware Description Languages (HDLs) provide a suitable alternative to those approaches. Accordingly, several reversible HDLs have been introduced in the past (see e.g., [21], [22]). They offer description means which are similar to conventional HDLs (such as Verilog [23] or VHDL [24]), but, at the same time, respect restrictions and rules needed to describe reversible circuits. While this allows to lift the design of those circuits to higher levels of abstractions, the corresponding synthesis approaches still suffer from the main problem of generating too many additional circuit lines. This is, because the corresponding HDL statements are decomposed into sub-statements which may not be reversible anymore. Because of this, additional circuit lines are introduced to embed non-reversible functionality into a reversible one (this is described in more detail later in Section III-A). This leads to reversible circuits with additional circuit lines—rendering HDL-based synthesis infeasible for many applications such as quantum computing where the number of circuit lines is crucial.

Although recent approaches investigated e.g., on re-writing the given HDL description [25], undoing computations [26] [27], or realizing single operations in a more line-aware fashion [28], no solution exists yet, which is capable of realizing HDL descriptions with no additional circuit lines at all. This is particularly unsatisfactory because the originally given HDL is always purely reversible and, hence, at least in principle, allows for a fully reversible description of the circuit to be realized.

In this work, we provide a solution that addresses this problem. The proposed approach rests on a rather simple but not yet investigated idea: Rather than breaking down reversible statements into (possibly non-reversible) building blocks (which cause additional circuit lines), we aim to synthesize those statements as an entirety. In order to handle larger statements (for which purely reversible building blocks are not available and/or cannot efficiently be generated), a translation scheme is provided which maps many of those statements into smaller ones—without "loosing" reversibility and, hence, the need to introduce additional circuit lines. For the first time, this allows to realize many HDL descriptions with no additional circuit lines at all. By this, the proposed synthesis scheme provides the first feasible HDL-based synthesis method for emerging technologies relying on reversible circuits. Experimental evaluations confirm the benefits of the resulting approach. For HDL descriptions which also have been used in the past to evaluate HDL-based synthesis, reversible circuits are generated which do not require a single additional circuit line anymore.

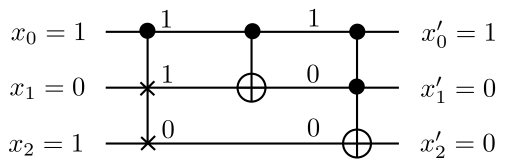


Fig. 1: Reversible circuit.

The remainder of this paper is structured as follows: The next section reviews reversible circuits and corresponding HDL descriptions. Based on that, the considered problem (how to synthesize those reversible HDL description without additional circuit lines?) as well as the general idea of the approach proposed in this work are discussed in Section III. Afterwards, the resulting synthesis scheme is described in Section IV. Finally, results obtained by our evaluations are discussed in Section V and the paper is concluded in Section VI.

II. Background

To keep the paper self-contained, this section briefly reviews the basics on reversible circuits as well as corresponding hardware description languages.

A. Reversible Circuits

Reversible circuits realize functions $f : \mathbb{B}^n \to \mathbb{B}^m$ over *variables* $X = \{x_1, \dots, x_n\}$ which have the same number of inputs and outputs (i.e., for which $n = m$) and which employ a unique mapping for all input/output patterns. That is, those circuits realize bijections. A reversible circuit $G = g_1 \dots g_d$ is composed as a cascade of reversible gates g_i. Each *reversible gate* in the circuit has the form $g_i(C,T)$, where $C = \{x_{i_1}, \dots, x_{i_k}\} \subset X$ are *control lines* and $T = \{x_{j_1}, \dots, x_{j_k}\} \subseteq X/C$ are *target lines*. The most important gates (which are also employed in this work) are the Toffoli gate [29] and the Fredkin gate [30]. The *Toffoli gate* has only one target line $T = \{x_j\}$ whose value is inverted, iff all positive (negative) control lines are assigned 1 (0). The *Fredkin gate* has two target lines $T = \{x_{j_1}, x_{j_2}\}$, whose values are swapped, iff all positive (negative) control values are assigned to 1 (0). In both cases, the value of all remaining lines pass through the gate unchanged.

Example 1. *Fig. 1 shows a reversible circuit realizing a reversible function over three inputs $X = \{x_0, x_1, x_2\}$ and with three gates (two Toffoli gates and one Fredkin gate). Each variable of the realized function is represented by a circuit line. The first gate realizes a Fredkin gate of the form $g_1(\{x_0\}, \{x_1, x_2\})$. The second and third gate realizes Toffoli gates of the form $g_2(\{x_0\}, \{x_1\})$ and $g_3(\{x_0, x_1\}, \{x_2\})$, respectively. In figures, control lines are usually denoted by $\bullet$, whereas the target line(s) are denoted by $\oplus$ and $\times$ in case of Toffoli gates and Fredkin gates, respectively. As can be seen in Fig. 1, this circuit maps the input pattern 101 to the output pattern 100 and vice versa (computations in both directions are possible).*

Since reversible circuits realize bijections which can be computed in both directions, the inverse of a reversible circuit G (denoted G^{-1} in the following and realizing the function f^{-1}) can be easily obtained by $G^{-1} = g_d^{-1} g_{d-1}^{-1} \cdots g_1^{-1}$, where g_i^{-1} is the inverse gate of g_i. Since Toffoli and Fredkin gates are self-inverse, $g_i = g_i^{-1}$ holds and, thus, G^{-1} can simply be obtained by reversing the order of the gates of G.

In order to measure the costs of a circuit, different metrics can be applied which often depend on the respectively defined application area. In this work, we consider metrics called *quantum costs* and *transistor costs* as defined in [2], [3] and [31], respectively.[2] They depend on the gates used in the circuit and, hence, are also called *gate costs* in the following. Besides that, and usually much more important, the number of circuit lines constitute a crucial cost metric. In principle, a reversible function over n variables can be realized using n circuit lines. However, sometimes additional circuit lines (usually employing a constant input) are added, e.g., to realize non-reversible functions (an aspect which becomes relevant and is discussed in more detail later in Section III-A).

B. Reversible Hardware Description Languages

Reversible Hardware Description Languages (reversible HDLs) allow for the description and realization of complex and large reversible circuits which would be hard to design manually. In this work, we consider the language *SyReC* which has been introduced in [21] and constitutes a suitable representative for this work.

SyReC utilizes the concept of *reversible assignments* which are of the form $v\oplus = e, \oplus \in (\hat{}, +, -)$, where the *left-hand side* (LHS) variable v must not appear in the *right-hand side* (RHS) expression e. Those assignments conduct a so-called *reversible update*, i.e., update the variable v using a reversible operation such as XOR (^=), increment $(+=)$, or decrement $(-=)$ to keep the entire statement reversible[3]. The RHS expression e may be non-reversible and can either be a signal identifier or a binary expression of the form of $v = e_{left} \odot e_{right}$, where $\odot$ is an arbitrary binary operation and e_{left} and e_{right} are again expressions or sub-expressions. A list of the most common binary operations which are directly applicable include arithmetic $(+, -, *, /, \%, *)$, bitwise $(\&, |, \hat{})$, logical $(\&\&, \|)$, and relational $(<, >, <=, >=, =, !=)$ operations. Although the RHS expression can be non-reversible, reversibility is always ensured because the input values remain unchanged and only reversible updates (for which inverses are available) are conducted—allowing to always employ a statement in both directions.

Example 2. *Consider the SyReC description as shown in Fig. 2. Here, the module signature first defines the signal names $(x_0, x_1, x_2, x_3, x_4, x_5)$, types $(in/inout/out)$, and bitwidths (64). The next lines define the statements to be executed. Each statement consists of an LHS signal and an RHS expression. The LHS signal is updated with the reversible operation and the RHS expression. For example, line 2 states that x_0 is XORed with the results of $((x_1 + x_2) - x_3)$ and, then, the results is stored in x_0(i.e., $x_{t+1} = x_t \oplus ((x_1 + x_2) - x_3)$)*

For a complete treatment of the reversible HDL considered here (including a complete grammar), we are referring to [26].

[2]Note that we choose these cost metrics as they are also applied in the respective related work. However, the considerations conducted here do not rely on a particular cost metric and can be applied to any other metric as well.

[3]Note that further operations f can be used for the reversible update as long it has an inverse operator f^{-1} with $v = f^{-1}(f(v, e), e)$. Furthermore, note that XOR is self-inverse, while the inverse of the increment $(+=)$ is the decrement $(-=)$ and vice versa.

```
module main(inout $x_0(64)$, in $x_1(64)$, in $x_2(64)$, in $x_3(64)$, in $x_4(64)$, out $x_5(64)$)
$x_0$ ^= $((x_1 + x_2) - x_3)$
$x_0$ -= $(((x_1 - x_2) - x_3) + (x_1 - x_4))$
$x_0$ ^= $((x_1 + x_2)/(x_3 - x_4))$
$x_5$ ^= $(((x_0 * x_1) * x_1) + ((x_2 * x_1) + x_3))$
```

Fig. 2: A Simple HDL code

III. Motivation and General Idea

Synthesis of reversible circuits described in terms of a reversible HDL constitutes a non-trivial task which has intensely been considered in the past years. A major problem in all these endeavors was that, even though the respectively given HDL is purely reversible in principle, all synthesis approaches proposed thus far still required additional circuit lines. This section illustrates the problem and briefly discusses why this remained to be a challenge until today. Afterwards, we sketch an idea to eventually overcome this problem.

A. Considered Problem

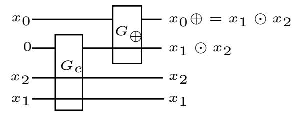
Reversible HDLs allow for the definition of complex reversible functionality. However, in order to eventually realize those descriptions in terms of reversible circuits, a hierarchical synthesis method has been applied thus far [26]. Here, realizations of the respective operations (both, the reversible assignment operations (^=), $(+ =)$, and $(- =)$ as well as all considered binary operations $(+, -, \&, |, \&\&, \|, <, >$, etc.) are first pre-computed or taken from literature such as [32], [33], [34]. Then, these *building blocks* are taken and used to realize each statement of the given HDL description towards the corresponding circuit realization. To this end, two steps are conducted as illustrated in Fig. 3a:

1) Realize the RHS expression e, which yields a sub-circuit G_e composed of existing building blocks realizing the respective binary operations.
2) Realize the reversible update, which yields a sub-circuit $G_\oplus$ composed of existing building blocks realizing the respective reversible assignment operation and, by this, the entire statement.

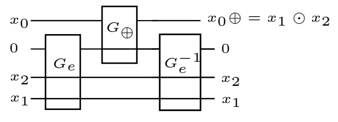
The application of this hierarchical flow in which the overall (reversible) statement is decomposed into sub-functions causes a significant drawback: Since the expression e (as well as possible sub-expressions e_{left} and e_{right} out of which e is composed of) may describe non-reversible functions (i.e., may include non-reversible binary operations), corresponding buildings block for those require additional circuit lines with constant inputs. The following example illustrates the issue.

Example 3. *Let's assume the statement x_0 ^= $x_1 + x_2$ shall be realized. Following the hierarchical flow sketched above, first a building block $G_e = G_{x_1+x_2}$ realizing the addition of x_1 and x_2 is required. But since addition is not reversible (e.g., the sum 3 may originate from two different input assignments $x_1 = 1/x_2 = 2$ or $x_1 = 2/x_2 = 1$), additional circuit lines are required to make this building block reversible. A possible realization is shown in the left-hand side of Fig. 3c (the sub-circuit labeled $G_{x_1+x_2}$), where additional lines with constant inputs are used to store the desired output while the originally given circuit lines keep the input values—making the computation reversible[4]. Using this building block in addition to a building block for the ^=-operation (the sub-circuit shown in the center of Fig. 3c and labeled $G_{x_0 \hat{=} x_1+x_2}$) eventually realizes the considered statement.*

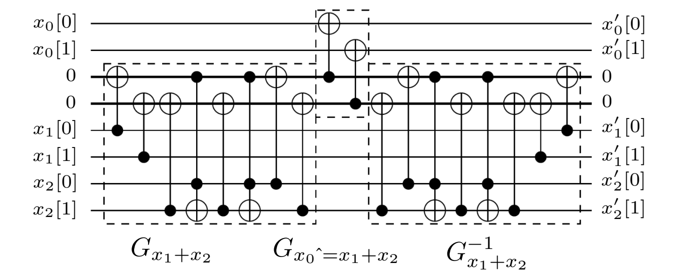
[4]Note that Fig. 3c shows a circuit where all signals are realized with bitwidth 2. Accordingly, $x_i[0]$ and $x_i[1]$ denote the least significant and most significant bit, respectively.



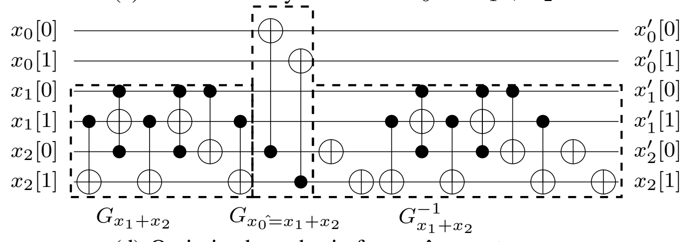
(a) Hierarchical synthesis.



(b) Optimized synthesis.



(c) Hierarchical synthesis for x_0 ^= $x_1 + x_2$.



(d) Optimized synthesis for x_0 ^= $x_1 + x_2$.

Fig. 3: Synthesis of a statement of the form $x_0 \oplus = x_1 \odot x_2$.

In the worst case, each non-reversible binary operation that occurs in the expression e causes a further set of additional circuit lines. In order to reduce this amount of additional lines, several improvements have been proposed. The most effective one [35] is to undo all operations conducted in order to determine the result of the expression e, after the reversible update has been executed. More precisely, after the two steps from above have been conducted, a third step as illustrated in Fig. 3b is employed:

3) Undo the realization of expression e, i.e., add a sub-circuit G_e^{-1}.

Example 4. *Additionally conducting the third step for the statement x_0 ^= $x_1 + x_2$ considered in the previous example eventually yields the complete circuit shown in Fig. 3c (including the sub-circuit shown in the right-hand side of Fig. 3c and labelled $G^{-1}_{x_1+x_2}$)[5].*

Although adding G_e^{-1} substantially increases the gate costs of the circuit (by almost doubling it), it sets the additional circuit lines back to their constant values. By this, those constant values can be used again for the realization of the next statement—preventing the need for additional circuit lines for each single statement. Motivated by that, several methods have been proposed (see e.g., [25], [26], [27], [28]) to keep the length of the statement small and/or to optimize expressions so that the number of additionally required circuit lines are kept as small as possible. However, despite these efforts, no solution is known yet, which is capable of realizing these reversible statements with no additional circuit lines at all.

[5]Note that $G^{-1}_{x_1+x_2}$ can be easily determined by just inversing the gates from $G_{x_1+x_2}$.

B. General Idea

The current state-of-the-art in HDL-based synthesis of reversible circuits clearly is unsatisfactory. Although a fully reversible description of the circuit to be realized is available in terms of an HDL that allows to define purely reversible statements, existing synthesis schemes still rely on non-reversible building blocks introducing additional circuit lines. An obvious way to avoid that is to always consider a reversible statement in its entirety, i.e., without breaking it down into possibly non-reversible building blocks. That this works in principle is illustrated by the following example:

Example 5. *Consider again the realization of the statement x_0 ^= $x_1 + x_2$. However, rather than following a hierarchical approach which combines the building block for the $+$-operation and the ^=-operation (yielding additional circuit lines), the statement can also be synthesized in its entirety at once—yielding, e.g., the circuit as shown in Fig. 3d.*

However, a big problem with this approach is that considering a statement in its entirety requires the availability of building blocks for *all* possible statements. That is, instead of utilizing and composing building blocks only for single operations such as ^=, $+=$, and $-=$ as well as $+, -, \&, |, \&\&, \|, <, >$, etc., such an approach would additionally require building blocks for all possible combinations of them—an infinite amount which obviously leads to an infeasible amount of building blocks to pre-compute. Moreover, even an "on-the-fly"-synthesis often is not feasible—in particular when statements are getting complex and, hence, often cannot be efficiently synthesized anymore (e.g., already a statement such as x_0 ^= $(x_1 + x_2)$ ^ $(x_3 - x_4) + x_5$ is highly non-trivial to synthesize in its entirety for a decent bitwidth). Because of these problems, researchers and engineers still opt back to the hierarchical synthesis scheme reviewed above—even if they yield additional circuit lines.

In this work, we propose an alternative that eventually addresses this problem for many of the cases. Our approach rests on a rather simple but not yet investigated idea: Many of the combinations between operations can be easily translated into a sequence of simpler statements. As a very simple example, a statement such as $x_0 += x_1 + x_2 + x_3$ (requiring additional circuit lines for each of the binary operations when following the established hierarchical synthesis flow) can be translated into an equivalent sequence of $x_0 += x_1$, $x_0 += x_2$, and $x_0 += x_3$ (for which building blocks with no additionally circuit lines are available and/or can be generated easily). Following this premise, an HDL-based synthesis for reversible circuits with no additional circuit lines is possible if

1) arbitrary statements (combining various operations) can be translated into a sequence of *simple statements* (i.e., statements with at most one non-reversible operation only), and
2) building blocks for those simple statements are available and/or can be determined easily.

IV. Resulting HDL-based Synthesis Scheme

In this section, we provide details of the proposed HDL-based synthesis scheme. More precisely, we first discuss methods to translate arbitrary statements into a universal subset of simple statements. Then, we discuss why determining building blocks realizing those simple statements in its entirety and, hence, without additional circuit lines is a feasible task.

A. Translating Arbitrary Statements to Simple Statements

HDLs such as SyReC reviewed in Section II-B allow for arbitrary definitions of statements. This is defined through a corresponding grammar available in [26]. Having that, statements can take any form of $v\oplus = e$ (see also Section II-B). The RHS-expression e (with $e = e_{left} \odot e_{right}$) can be represented in a tree-like fashion, i.e., an *Abstract Syntax Tree* (AST), where e_{left} and e_{right} can themselves be composed of sub-expressions (i.e., represent either operators or operands).[6] The root node of the AST represents the operator $\odot$ of the expression e.

Having this description, term re-writing techniques can be employed to translate a given statement to a corresponding sequence of simple statements. This works particularly well in cases where the statement or the (sub-)expression(s) are composed of operations that are associative. But also certain combinations of operations work well, which are not per se associative but can be adjusted accordingly (e.g., statements/expressions involving several combinations of $+$ and $-$ operations can be translated into simple statements by translating a subtraction into an addition with inverted inputs). In contrast, a few combinations of operations such as $-$ and ^ are harder to translate. But since those combinations rarely occur in existing HDL descriptions (as also confirmed by evaluations summarized in Section V), it is acceptable to ignore those combinations for now.

As a result, many of the arbitrary statements can be translated into a subset of simple statements. This is described for many of the frequently occurring cases in the following. In all following descriptions, we assume thereby that the RHS-expression is provided in terms of an AST in which each internal node corresponds to an operator and each leaf node corresponds to an operand.

Then, the simplest case is where the expression e is solely composed of ^, $+$, and $-$ (i.e., operations which are also available as reversible assignment operations). If additionally the signals belonging to e_{left} of the expression are not repeated more than once on every level of the AST, apply the following steps:

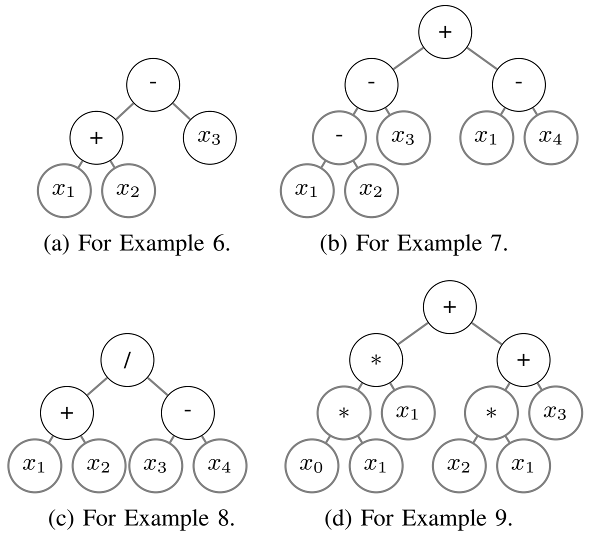
1) Recursively traverse the AST in a post order fashion and add statements $e_{left}\odot = e_{right}$ for each operation node.
2) Once the overall expression e is realized, add the statement $v\oplus = e$ which describes the assignment of the root expression e to the LHS.
3) Add statements which inverse the statements from Step 1, i.e., add $e_{left}\odot^{-1} = e_{right}$ for each depth.

Example 6. *Consider again the HDL description shown in Fig. 2. The statement in Line 2 (whose RHS-expression is represented by an AST as shown in Fig. 4a) can be translated to the following sequence of statements:*

$$x_1 += x_2$$
$$x_1 -= x_3$$
$$x_0 \mathbin{\hat{}}= x_1$$
$$x_1 += x_3$$
$$x_1 -= x_2$$

Otherwise, if the expression e is solely composed of ^, $+$, and $-$ (i.e., operations which are also available as reversible assignment operations), but the signals belonging

[6]Note that, according to the grammar proposed in [26], e_{right} is never larger than e_{left}.



(a) For Example 6. (b) For Example 7.

(c) For Example 8. (d) For Example 9.

Fig. 4: ASTs of the expressions from the Examples 6 to 9.

to e_{left} of the expression are repeated more than once on every level, slightly more elaborated translations are needed. Still, as long as the respective operations of the statement/(sub-)expression(s) are associative (or can easily be made associative, e.g., by inverting them), proper translations are possible. More precisely, if the assignment operator is of $\oplus \in \{\hat{}\}(\oplus \in \{-\})$ and a binary operator $\odot \in \{-\}(\odot \in \{\hat{}\})$ operates on the leaf nodes only, then the following steps can be applied:

1) Traverse the AST in a post order fashion and, for the deepest operation node, add the statement $v\oplus = e_{left} \odot e_{right}$.
2) For the rest of the AST:
   - If the currently considered operator node has one leaf node, add the statement $v\odot = e_{left/right}$ (if $\oplus \in \{\hat{}, +\}$) or $v\odot^{-1} = e_{left/right}$ (otherwise)
   - If the currently considered operator node has two leaf nodes, add the statement $v\odot_{top} = e_{left} \odot e_{right}$ (if $\oplus \in \{\hat{}, +\}$) or $v\odot_{top}^{-1} = e_{left} \odot e_{right}$ (otherwise) where $\odot_{top}$ refers to the next top level operation of the currently considered node.

Example 7. *Consider again the HDL description shown in Fig. 2. The statement in Line 3 (whose RHS-expression is represented by an AST as shown in Fig. 4b) can be translated to the following sequence of statements:*

$$x_0 - = (x_1 - x_2)$$
$$x_0 + = x_3$$
$$x_0 - = (x_1 - x_4)$$

Furthermore, also translations to single statements are possible if the expression e is not solely composed of $\hat{}$, $+$, and $-$ (i.e., if the expression is composed of operations which are not available as assignment operations). For example, this is possible if the assignment operator is of $\oplus \in \{\hat{}\}$ $(\oplus \in \{-\})$ and a binary operator $\odot \in \{-\}$ $(\odot \in \{\hat{}\})$ operates only on leaf nodes. Then, if additionally the signals belonging to e_{left} of the expression are not repeated more than once on every level of the AST *and* no two consecutive nodes of the AST have both operators, $\odot \notin \{\hat{}, +, -\}$, the following steps can be applied:

1) Traverse the AST in post order fashion and, for the deepest operation node with $\odot \notin \{\hat{}, +, -\}$, add the statement $v\oplus = e_{left} \odot e_{right}$, where e_{left} and e_{right} are either signals or results of the respective sub-expression.
2) For the rest of the AST:
   - If the currently considered operator node has two leaf nodes, add the statement $e_{left}\odot = e_{right}$ (if $\odot \in \{\hat{}, +, -\}$)
   - If the currently considered operator node has one leaf node and if $\odot \in \{\hat{}, +, -\}$, add the statement $v\odot = e_{left/right}$ (if $\oplus \in \{\hat{}, +\}$) or $v\odot^{-1} = e_{left/right}$) (otherwise)
   - If the currently considered operator node $\odot \notin \{\hat{}, +, -\}$, add the statement $v\odot_{top} = e_{left} \odot e_{right}$ (if $\oplus \in \{\hat{}, +, \}$) or $v\odot_{top}^{-1} = e_{left} \odot e_{right}$ (otherwise) where $\odot_{top}$ refers to the next top level operation of the currently considered node which, according to the case assumption stated above, has to be $\odot_{top} \in \{\hat{}, +, -\}$).
3) Add corresponding inverse statements for all statements which have been added in Step 2 with $\odot \in \{\hat{}, +, -\}$, i.e add $e_{left} \odot^{-1} e_{right}$ for all these cases.

Example 8. *Consider again the HDL description shown in Fig. 2. The statement in Line 4 (whose RHS-expression is represented by an AST as shown in Fig. 4c) can be translated to the following sequence of statements:*

$$x_1 + = x_2$$
$$x_3 - = x_4$$
$$x_0 \,\hat{}= (x_1 / x_3)$$
$$x_3 + = x_4$$
$$x_1 - = x_2$$

The cases discussed so far cover a huge amount of frequently occurring statements. However, for the statements which do not fall into any of the cases mentioned above, no translation down to simple statements that can be easily realized with no additional circuit lines has been found. Nevertheless, applying the rules from above, still yield reductions with respect to the number of lines as illustrated by the following example.

Example 9. *Consider again the HDL description shown in Fig. 2. The statement in Line 5 (whose RHS-expression is represented by an AST as shown in Fig. 4d) can be translated to the following sequence of statements:*

$$x_5 \,\hat{}= ((x_0 * x_1) * x_1)$$
$$x_5 + = (x_2 * x_1)$$
$$x_5 + = x_3$$

Here, all the statements except $x_5 \,\hat{}= ((x_0 * x_1) * x_1)$ can be easily realized with no additional circuit lines. Compared to the number of additional lines required when realizing the original statement from Line 6, this allows for a substantial reduction.

Overall, the general idea and the rules proposed above indeed allow for a translation of arbitrary statements into simple statements in most of the cases (this is also confirmed by experimental evaluations summarized later in Section V). By this, the main basis for the proposed synthesis scheme is laid out. Based on that, only a restricted set of building blocks for those simple statements is needed anymore.

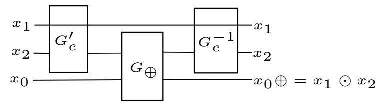
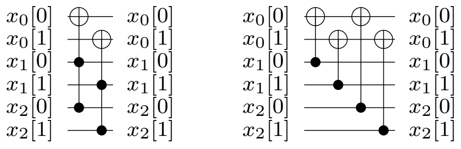


Fig. 5: Structure of a building block for $x_0 \oplus = x_1 \odot x_2$.



(a) x_0 ^= $x_1 \& x_2$ (b) x_0 ^= x_1 ^ x_2

Fig. 6: Building blocks with Boolean operations.

B. Determining Building Blocks for Simple Statements

Having an HDL description translated into simple statements, the determination of the correspondingly needed building blocks gets much easier. Indeed rather than to realize arbitrary combinations of expressions, now only building blocks for a restricted amount of combinations is needed. If we restrict ourselves to the statements with at most one non-reversible operator in the right-hand side expression, we end up with a total of 51 different combinations (3 possible assignment operations times 17 possible binary operations). By providing purely reversible building blocks for these combinations, the realization of many statements with no additionally required circuit lines is possible.

In fact, most of these building blocks have a structure which is similar than the one sketched in Fig. 5.

Here, the sub-circuit G'_e now realizes the respective binary operations in expression e, with the results stored in one of the input lines (rather than an additional circuit line). Then, the sub-circuit $G_\oplus$ realizes the respective reversible assignment operation (as also done in the original HDL-based synthesis). Finally, the sub-circuit G_e^{-1} applies the inverse again. Since the sub-circuit G'_e is now guaranteed to be composed of at most one binary operation $\oplus$ only, it can be realized easily. Fig. 3d shows a corresponding example for the statement x_0 ^= $x_1 + x_2$, where the sub-circuits for addition and the inverse addition (i.e., subtraction) are designed using the methods proposed in [34].

Besides that, building blocks realizing Boolean operations such as AND and XOR can be realized in an even simpler fashion. Here, Fig. 6a and Fig. 6b provide corresponding examples for x_0 ^= $x_1 \& x_2$ and x_0^= x_1 ^ x_2, respectively.

V. Experimental Evaluation

The concepts introduced above have been implemented in C++ and resulted in a synthesis tool which is capable to realize arbitrary HDL descriptions with no additional circuit lines in almost all cases. In order to confirm the benefits, intensive evaluations have been conducted using HDL descriptions provided in [36] which also have been used in the past to evaluate HDL-based synthesis of reversible circuits. Using those benchmarks, we realized corresponding reversible circuits using the resulting synthesis tool and compared them to circuits generated

- using the original synthesis tool presented in [21] and
- using the state-of-the-art solution available thus far (additionally employing line-aware synthesis as proposed in [26]).

All evaluations have been conducted on an 64-bit Intel machine with 2.66 GHz and 8 GB of main memory. In the following, the main results are presented and discussed.

Table I summarizes the obtained results. The first column provides the name of the respectively considered benchmark as well as the considered bitwidth (for each benchmark, we realized circuits for bitwidth of 16 and 32). Afterwards, the number of additionally required circuit lines and the number of gates as well as the respectively resulting gate costs (based on the metrics reviewed in Section II-A) are provided for each resulting circuit.

The numbers clearly show that the proposed synthesis scheme realizes the HDL descriptions with no additional circuit lines. Considering that all previous work only managed to reduce the number of circuit lines (compare the results from the original synthesis tool to the results from the line-aware synthesis tool) but never where able to completely get rid of them, this confirms the main benefit of the synthesis scheme proposed in this work. Moreover, this improvement can be obtained while, at the same time, hardly further increasing the resulting costs. In fact, in almost all cases the costs are much lower than that of line-aware synthesis scheme. This is particularly remarkable since previous studies frequently showed a trade-off between the number of circuit lines and the quantum costs.

VI. Conclusions

In this paper, we proposed to optimize HDL based synthesis of reversible circuits by considering reversible HDL statements as an entity, instead of breaking it down into non-reversible blocks. Since, there are infinite number of possible statements (leading to an infeasible amount of building blocks), we propose to translate arbitrary statements into a universal subset of simple statements (i.e., statements with at most one non-reversible operation) for which building blocks can be determined easily. Following this approach, the number of additional lines can be reduced to zero, whenever arbitrary description can be fully translated to simple statements. But also in all other cases, the descriptions can be translated and, hence, a reduction becomes possible. As the number of circuit lines is usually considered to be a very limited resource (e.g., in the domain of quantum computation), these achievements are promising.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct System Lab funded by the State of Upper Austria.

References

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

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

[3] D. M. Miller, R. Wille, and Z. Sasanian, "Elementary quantum gate realizations for multiple-control toffoli gates," in *Int'l Symp. on Multi-Valued Logic*, 2011, pp. 288–293.

[4] M. P. Frank, "Throwing computing into reverse," *IEEE Spectrum September 2017*, 2017.

TABLE I: Experimental Results

		Original Synthesis [21]				Line-aware Synthesis [26]				Proposed Synthesis			
Benchmark	Bitwidth	Add. Lines	Gates	QC	TC	Add. Lines	Gates	QC	TC	a.l.	gates	QC	TC
new_alu	16	32	306	1874	5088	17	516	3524	9168	0	514	3266	8640
new_alu	32	64	626	3858	10464	33	1060	7268	18896	0	1058	6754	17856
for	16	128	1218	7490	20352	17	2052	14084	36624	0	2050	13058	34560
for	32	256	2498	15426	41856	33	4228	29060	75536	0	4226	27010	71424
parity	16	30	65	313	1000	16	127	615	1976	0	65	313	1000
parity	32	62	129	633	2024	32	255	1255	4024	0	129	633	2024
call	16	32	514	3266	8640	17	692	4884	12464	0	482	3106	8128
call	32	64	1058	6754	17856	33	1428	10100	25776	0	994	6434	16832
parity_check	16	32	69	333	1064	17	135	655	2104	0	69	333	1064
parity_check	32	64	133	653	2088	33	263	1295	4152	0	133	653	2088
gray_binary	16	30	94	462	1472	2	156	764	2448	0	64	312	992
gray_binary	32	62	190	942	3008	2	316	1564	5008	0	128	632	2016
deutsch1	16	16	48	48	384	16	80	80	640	0	32	32	256
deutsch1	32	32	96	96	768	32	160	160	1280	0	64	64	512
deutsch2	16	16	64	64	384	16	96	96	640	0	48	48	256
deutsch2	32	32	128	128	768	32	192	192	1280	0	96	96	512
Single_long_statement	16	80	504	864	4240	80	864	1584	8352	0	800	1520	7072
Single_long_statement	32	160	1032	1776	8720	160	1776	3264	17184	0	1152	2144	9664
Multiple_statement	16	48	560	1040	4928	16	768	1488	7584	0	768	1488	6816
Multiple_statement	32	96	1152	2144	10176	32	1584	3072	15648	0	1584	3072	14112
inputs_repeated	16	96	816	1536	7456	48	1296	2496	12768	0	1144	1984	9808
inputs_repeated	32	192	1680	3168	15392	96	2672	5152	26336	0	2360	4096	20304
operators_repeated	16	32	272	512	2656	16	424	784	4112	0	240	480	2400
operators_repeated	32	64	560	1056	5472	32	872	1616	8464	0	496	992	4960

Add. Lines: Number of additionally required circuit lines   Gates: Number of gates   QC/TC: Quantum costs and transistor costs (cf. Section II-A)

[5] A. Rauchenecker, T. Ostermann, and R. Wille, "Exploiting reversible logic design for implementing adiabatic circuits," in *Int'l Conf. Mixed Design of Integrated Circuits and Systems*, 2017, pp. 264–270.

[6] A. Zulehner, M. P. Frank, and R. Wille, "Design automation for adiabatic circuits," in *ASP Design Automation Conf.*, 2019, pp. 669–674.

[7] A. Zulehner and R. Wille, "Taking one-to-one mappings for granted: Advanced logic design of encoder circuits," in *Design, Automation and Test in Europe*, 2017, pp. 818–823.

[8] R. Wille, O. Keszocze, S. Hillmich, M. Walter, and A. Garcia-Ortiz, "Synthesis of approximate coders for on-chip interconnects using reversible logic," in *Design, Automation and Test in Europe*, 2016, pp. 1140–1143.

[9] A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, "Experimental verification of Landauer's principle linking information and thermodynamics," *Nature*, vol. 483, no. 7388, p. 187, 2012.

[10] B. Desoete and A. De Vos, "A reversible carry-look-ahead adder using control gates," *Integration, the VLSI Journal*, vol. 33, no. 1-2, pp. 89–104, 2002.

[11] L. Amarú, P.-E. Gaillardon, R. Wille, and G. De Micheli, "Exploiting inherent characteristics of reversible circuits for faster combinational equivalence checking," in *Design, Automation and Test in Europe*, 2016, pp. 175–180.

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

[13] 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.

[14] D. Maslov and G. W. Dueck, "Reversible cascades with minimal garbage," *IEEE Trans. on CAD*, vol. 23, no. 11, pp. 1497–1509, 2004.

[15] 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.

[16] K. Fazel, M. A. Thornton, and J. Rice, "ESOP-based toffoli gate cascade generation," in *Pacific Rim Conference on Communications, Computers and Signal Processing*, 2007, pp. 206–209.

[17] Y. Sanaee and G. W. Dueck, "ESOP-based toffoli network generation with transformations," in *Int'l Symp. on Multi-Valued Logic*, 2010, pp. 276–281.

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

[19] C.-C. Lin and N. K. Jha, "RMDDS: Reed-Muller decision diagram synthesis of reversible logic circuits," vol. 10, no. 2, p. 14, 2014.

[20] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. on CAD*, vol. 37, no. 5, pp. 996–1008, 2017.

[21] 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. 1–6.

[22] M. K. Thomsen, "A functional language for describing reversible logic," in *Forum on Specification and Design Languages*, 2012, pp. 135–142.

[23] *IEEE Std 1800-2005: IEEE Standard for System Verilog - Unified Hardware Design, Specification, and Verification Language*. [Online]. Available: https://books.google.co.in/books?id=0WL4wQEACAAJ

[24] *IEEE Standard VHDL Language Reference Manual Amendment 1: Procedural Language Application Interface*.

[25] Z. Alwardi, R. Wille, and R. Drechsler, "Re-writing HDL descriptions for line-aware synthesis of reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2016, pp. 31–36.

[26] R. Wille, E. Schönborn, M. Soeken, and R. Drechsler, "SyReC: a hardware description language for the specification and synthesis of reversible circuits," *INTEGRATION, the VLSI Jour.*, vol. 53, pp. 39–53, 2016.

[27] R. Wille, M. Soeken, E. Schönborn, and R. Drechsler, "Circuit line minimization in the HDL-based synthesis of reversible logic," in *IEEE Annual Symposium on VLSI*, 2012, pp. 213–218.

[28] Z. Al-Wardi, R. Wille, and R. Drechsler, "Towards line-aware realizations of expressions for HDL-based synthesis of reversible circuits," in *Conference on Reversible Computation*, 2015, pp. 233–247.

[29] T. Toffoli, "Reversible computing," in *International Colloquium on Automata, Languages, and Programming*. Springer, 1980, pp. 632–644.

[30] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of theoretical physics*, vol. 21, no. 3-4, pp. 219–253, 1982.

[31] M. K. Thomsen and R. Glück, "Optimized reversible binary-coded decimal adders," *Journal of Systems Architecture*, vol. 54, no. 7, pp. 697–706, 2008.

[32] Y. Takahashi and N. Kunihiro, "A linear-size quantum circuit for addition with no ancillary qubits," *Quantum Information & Computation*, vol. 5, no. 6, pp. 440–448, 2005.

[33] S. Offermann, R. Wille, G. W. Dueck, and R. Drechsler, "Synthesizing multiplier in reversible logic," in *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 335–340.

[34] H. Thapliyal and N. Ranganathan, "Design of efficient reversible logic-based binary and bcd adder circuits," vol. 9, no. 3, p. 17, 2013.

[35] R. Wille, M. Soeken, E. Schönborn, and R. Drechsler, "Circuit line minimization in the HDL-based synthesis of reversible logic," in *IEEE Annual Symposium on VLSI*, 2012, pp. 213–218.

[36] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: an online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.