Synthesizing Multiplier in Reversible Logic

Sebastian Offermann[1] Robert Wille[1] Gerhard W. Dueck[2] Rolf Drechsler[1]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
{offerman,rwille,drechsle}@informatik.uni-bremen.de
gdueck@unb.ca

Abstract—**In the past, reversible logic has become an intensely studied research topic. This is mainly motivated by its applications in the domain of low-power design and quantum computation. Since reversible logic is subject to certain restrictions (e.g. fanout and feedback are not allowed), traditional synthesis methods are not applicable and specific methods have been developed. In this paper, we focus on synthesis of multiplier circuits in reversible logic. Three methods are presented that address the drawbacks of previous approaches. In particular, the large number of circuit lines in the resulting realizations as well as the poor scalability. Finally, we compare the results to circuits obtained by general purpose synthesis approaches.**

I. Introduction

The number of elements integrated in digital circuits grows exponentially, leading to enormous challenges in *Computer Aided Design* (CAD). Due to this exponential growth physical boundaries will be reached in the near future. Furthermore, power consumption of circuits becomes a major issue. As a consequence, researchers expect that "traditional" technologies like CMOS will reach their limits in the near future [1]. To face this, alternative computation technologies are needed. This motivates research in the domain of reversible logic.

Reversible logic realizes bijections, i.e. one-to-one mappings of Boolean functions. The resulting reversibility allows promising applications e.g. in the domain of low-power design and quantum computation. In fact, it has been proven that zero power dissipation is only possible if computations are performed in an invertable manner [2], [3]. Reversible circuits are driven by their input signals only (and accordingly without additional power supplies) have already been physically implemented [4]. Besides that, the application in quantum computation is triggered by the fact that every quantum operation inherently is reversible [5]. Quantum computers allow to solve practically relevant problems (e.g. factorization) much faster than traditional circuits [6], [7].

As a result, reversible logic has become an intensely studied research area. In particular, synthesis aspects are of interest since reversible circuits are subject to certain restrictions, e.g. fanout and feedback are not allowed [5]. In the past, several general purpose synthesis approaches have been introduced (e.g. [8], [9], [10], [11], [12]). The realization of reversible circuits for arithmetic functions is of particular interest, since these functions occur naturally in circuit design.

In the following, we focus on synthesis of multiplier circuits in reversible logic. First reversible realizations of multiplier have already been introduced (see e.g. [13], [14], [15]). However, they often rely on a very special type of reversible gates (e.g. the TSG gate, the HNG gate, or the PFAG gate) and additionally have been proposed for very small bit-width only (in fact, 4×4 multiplier have been introduced). Besides that, multiplier can also be realized with the help of the (general purpose) synthesis approaches mentioned above. But since multiplication is an irreversible function, thereby often circuits with a significant number of additional circuit lines result. Furthermore, these approaches do not scale very well in particular for the multiplication function.

In this paper, we present three methods that (partially) address these drawbacks. More precisely, an adjusted multiplication specification is introduced that enables synthesis of a multiplier with a significant lower number of circuit lines. Even if this specification still is only applicable for very small bit-widths, it provides an interesting insight in how to exploit properties of a multiplier while synthesizing them as reversible circuit. Additionally, two constructive approaches for synthesis of multipliers with very large bit-width are proposed. While the first one is a hierarchical method based on partial products, the second one is motivated by the divide-and-conquer-method of Karatsuba's algorithm [16].

All proposed methods apply the well-established Toffoli gate library. The resulting circuits are evaluated with respect to number of circuit lines, number of gates, quantum cost, and transistor costs, respectively. Furthermore, we compare the results with circuits obtained by (general purpose) synthesis approaches. Overall, methods to generate reversible realizations for practical relevant multiplier with large bit-widths are presented and evaluated.

The remaining paper is structured as follows. Section II introduces the basics of reversible logic. Afterwards, the three methods to generate reversible multipliers are introduced in Section III. This also includes a brief discussion about the advantages and disadvantages of the respective approaches. Finally, experimental results are presented in Section IV and the paper is concluded in Section V, respectively.

II. Preliminaries

To keep the paper self-contained, this section introduces the basics of reversible functions and reversible circuits. For more details we refer to the respective publications.

Definition 1: A multiple-output function $f : \mathbb{B}^n \to \mathbb{B}^m$ is a *reversible function* iff (1) its number of inputs is equal to the number of outputs (i.e. $n = m$) and (2) it maps each input pattern to a unique output pattern.

In other words, each reversible function is a bijection that permutes the set of input patterns. A function that is not reversible is termed *irreversible*. Quite often, (irreversible) multi-output Boolean functions should be represented by reversible circuits. This necessitates the irreversible function to be *embedded* into a reversible one which requires the addition of circuit lines leading to *constant inputs* (i.e. inputs that are

TABLE I
QUANTUM COST FOR TOFFOLI GATES

NO. OF CONTROL LINES	QUANTUM COST OF A TOFFOLI GATE
0	1
1	1
2	5
3	13
4	26, if at least 2 lines are unconnected 29, otherwise
5	38, if at least 3 lines are unconnected 52, if 1 or 2 lines are unconnected 61, otherwise
6	50, if at least 4 lines are unconnected 80, if 1, 2 or 3 lines are unconnected 125, otherwise

assigned to a fixed value) and *garbage outputs* (i.e. outputs that are don't care for all possible input conditions). The minimal number of circuit lines to be added is determined by the number of the occurrences of the most frequent output pattern [17].

To realize reversible functions some restrictions must be considered, e.g. fanouts and feedback are not allowed [5]. This is reflected in the definition of reversible circuits.

Definition 2: A *reversible circuit* G over inputs $X = \{x_i | 1 \leq i \leq n\}$ is a cascade of reversible gates g_i, i.e. $G = \top_{i=1}^{d} g_i$ where d is the number of gates[1].

In the literature, different reversible gates have been studied. The universal *multiple control Toffoli gate* [18] is the most commonly used gate.

Definition 3: Let $X := \{x_i | 1 \leq i \leq n\}$ be the set of domain variables. A multiple control Toffoli gate has the form $g(C,t)$, where $C = \{x_{j_i} | 1 \leq i \leq k\} \subset X$ is the set of *control lines* and $t = x_l$ with $t \notin C$ is the *target line*. It inverts the target line iff all control lines are assigned to 1, i.e. the gate maps $\top_{i=1}^{l-1} x_i, x_l, \top_{i=l+1}^{n} x_i$ to $\top_{i=1}^{l-1} x_i, x_l \oplus \bigwedge_{i=1}^{k} x_{j_i}, \top_{i=l+1}^{n} x_i$. If no control lines are given (C is empty), then the target line is inverted, i.e. the input vector of the gate is mapped to $\top_{i=1}^{l-1} x_i, x_l \oplus 1, \top_{i=l+1}^{n} x_i$. In the following, we refer to multiple control Toffoli gates for brevity as *Toffoli gates*. Furthermore, a Toffoli gate with no control line (with k control lines) is also called *NOT gate* (*C^kNOT gate*).

To determine the effort to realize a reversible circuit, the following cost models are applied depending on the target technology:

- *Line count* denotes the number of lines the circuit uses (in particular for the application in the domain of quantum computing this is an important cost criterion since the number of circuit lines correspond to the number of qubits – so far a very restricted resource).
- *Gate count* denotes the number of gates the circuit consists of (i.e. d).
- *Quantum cost* denotes the effort needed to transform a reversible circuit to a quantum circuit. Table I shows the quantum cost for a selection of Toffoli gate configurations as introduced in [19] and further optimized e.g. in [17]. As can be seen, gates of larger size are considerably more expensive than gates of smaller size. The sum of the quantum cost for all gates defines the quantum cost of the whole circuit.
- *Transistor cost* denotes the effort needed to realize a reversible circuit in CMOS according to [20]. The transistor cost of a Toffoli gate is $8 \cdot c$ where c is the number of control lines.

[1]For tuples, we are using the $\top$-symbol which is analogously defined to the sum-symbol $\sum$: $\top_{i=0}^{0} x_i = x_0; \top_{i=0}^{n+1} x_i = \left(\left(\top_{i=0}^{n} x_i\right), x_{n+1}\right)$.

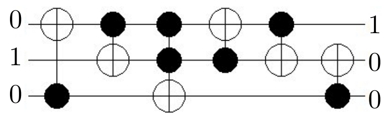


Fig. 1. Toffoli circuit

Example 1: Fig. 1 shows a Toffoli circuit with 3 circuit lines, 6 gates, quantum cost of 10, and transistor cost of 56, respectively. The control lines are thereby denoted by ●, while the target lines are denoted by $\oplus$. The annotated values illustrate the computation performed by this circuit.

Finally, the following definition introduces the denotation of *controlled functions*.

Definition 4: Let $\circ : \mathbb{B}^n \to \mathbb{B}^n$ be a reversible function that is realized by the reversible circuit $G = \top_{i=1}^{d} g_i$. The *controlled function* $\overset{c}{\circ}$ is described by the circuit $\underset{c}{G}$ which is obtained by adding the circuit line c to the control set C of each gate g in the circuit G.

In this way, e.g. a controlled increaser $\overset{c}{+=}$ can be obtained from an increaser $+=$ by adding the control bit c to every gate of the circuit, that represents this increaser.

III. REALIZING MULTIPLICATION IN REVERSIBLE LOGIC

In this section, we introduce methods to efficiently realize multiplication in reversible logic. In a straight-forward way, a multiplier can be synthesized by specifying the underlying function in terms of a truth table, binary decision diagram, exclusive-sum-of-products, or similar descriptions, respectively, and pass this to an appropriate synthesis approach (e.g. [9], [11], [12]). Since multiplication is an irreversible function, thereby often circuits with a significant number of additional circuit lines result (even if in case of [9] circuits with minimal number of lines can be achieved). But for the particular case of multiplication, it is possible to reduce the number of circuit lines if an adjusted function specification is used. The corresponding approach is introduced in the first part of this section. However, even then multipliers can be realized only for very small bit-widths. Thus, two further approaches are introduced that enable synthesis of multipliers with scalable bit-widths. Finally, the newly proposed realizations are discussed.

A. Multiplier with Sub-minimal Circuit Lines

As mentioned above, usually additional circuit lines are needed to embed an irreversible function (like multiplication) into a reversible one. More precisely, lines must be added so that at least $\lceil \log_2(\mu) \rceil$ garbage outputs are available, where μ is the maximum number of times an output pattern is

TABLE II
RESULTS OF A 3-BIT MULTIPLICATION

product (result)	μ	$\lceil\log_2\mu\rceil$	factors
0 (zero)	15	4	$0 \cdot k, k \cdot 0$
6	4	2	$1 \cdot 6, 2 \cdot 3, 3 \cdot 2, 6 \cdot 1$
12	4	2	$2 \cdot 6, 3 \cdot 4, 4 \cdot 3, 6 \cdot 2$
4	3	2	$1 \cdot 4, 2 \cdot 2, 4 \cdot 1$
2	2	1	$1 \cdot 2, 2 \cdot 1$
$3, 5, 7$	similar to 2		
8	2	1	$2 \cdot 4, 4 \cdot 2$
$10, 14, 15, 18, 20, 21, 24, 28, 30, 35, 42$	similar to 8		
1	1	0	$1 \cdot 1$
9	1	0	$3 \cdot 3$
16,25,36,49	similar to 9		

repeated in the truth table (see [17])[2]. Thus, keeping μ as small as possible helps reducing the number of circuit lines. In the following, an adjusted specification of the multiplication function is presented that exploits this observation.

The most frequent output pattern in a multiplier is the binary encoding of zero. The product is zero, iff (at least) one factor is zero. Thus, an n-bit multiplication produces $\mu = 2 \cdot 2^n - 1 = 2^{n+1} - 1$ zeros. Hence, $\lceil \log_2(2^{n+1} - 1)\rceil = (n + 1)$ lines with garbage outputs are required. All other possible products (i.e. output patterns) are less frequent. The main idea of this approach is to reduce the occurrences of the zero-output by means of an additional *indicator output*. As a result, the value of μ is significantly decreased and the multiplier can be realized with less circuit lines.

More precisely, an indicator output is added, which becomes assigned to 1 iff the product of the multiplication is zero. In contrast, all primary outputs can be arbitrarily assigned in this case. In doing so, the result zero is obtained by measuring the indicator output, while all other results still can be obtained from the remaining primary outputs. Thus, the binary encoding of the zero is not longer applied to obtain the minimal number of garbage lines. Instead the next most frequent output pattern is used for that. Therewith, the number of garbage lines can be asymptotically reduced by one half.

Example 2: Consider a 3-bit multiplication. The possible products and their occurrences (i.e. μ) are depicted in Table II. The respective output patterns (ordered in terms of a truth table) are shown in Table III. Since zero is the most frequent product (in total occurring 15 times), at least $\lceil\log_2(15)\rceil = 4$ garbage lines are conventionally needed.

In contrast, encoding the zero output by a separate indicator output, 6 and 12 become the most frequent products – each with only 4 occurrences. Hence, only $\lceil\log_2(4)\rceil = 2$ additional garbage outputs are needed to realize this function. As a result, this encoding only requires 3 additional outputs (the indicator output and the two garbage outputs) in comparison to the 4 additional outputs required by the conventional method. Column *sub-minimal* in Table III shows a possible embedding exploiting this encoding.

Having this adjusted specification, any truth table-based synthesis approach (e.g. [9]) can be applied to realize the multiplier. However, since this approach still relies on a

TABLE III
ENCODING OF A 3-BIT-MULTIPLICATION

		result encoded		
a	b	decimal	conventional	sub-minimal
000	000	0	000000 - - - -	000000 1- -
000	001	0	000000 - - - -	000001 1- -
000	010	0	000000 - - - -	000010 1- -
⋮	⋮	⋮	⋮	⋮
001	000	0	000000 - - - -	001000 1- -
001	001	1	000001 - - - -	000001 0- -
001	010	2	000010 - - - -	000010 0- -
⋮	⋮	⋮	⋮	⋮
011	111	21	010101 - - - -	010101 0- -
100	000	0	000000 - - - -	100000 1- -
⋮	⋮	⋮	⋮	⋮
111	111	49	110001 - - - -	110001 0- -

truth table description, it is only applicable for multiplier of very small bit-widths. In the following two scalable synthesis methods are proposed.

B. Hierarchical Method

The realization of multiplier by a hierarchical method using controlled increasers is described in this section.

A common way to multiply two factors $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$ and $b = \sum_{i=0}^{n-1} b_i \cdot 2^i$ is to compute the partial products and add them together, i.e. $a \cdot b = \sum_{i=0}^{n-1} \left(a_i \cdot \sum_{j=0}^{n-1} b_j \cdot 2^j \right) \cdot 2^i$. That is, the respective bit of b_j multiplied by the respective power of 2 is added to the product, iff the respective bit a_i is assigned to 1. This can easily be realized by controlled functions (or more precisely controlled increasers as sketched at the end of Section II).

Thus, using the hierarchical method an n-bit multiplier is realized by n single n-bit controlled increasers. The respective algorithm for an n-bit multiplication with the factors $a = \top_{i=0}^{n-1} a_i$ and $b = \top_{i=0}^{n-1} b_i$ as well as the product $c = \top_{i=0}^{2 \cdot n-1} c_i$ is depicted in Fig. 2.

Here, the ith controlled increaser is controlled by a_i. It conditionally adds the value of b to $\top_{j=i}^{n-1+i} c_i$, i.e. to the n bits of the product c beginning from the i-th bit. The lower bits do not have to be considered, since the j-th bit of the product is only modified until the j-th controlled increaser. Beyond that, the value remains unchanged. Therefore, the controlled increasers can sequentially be realized by an implicit bit-shift after every increase and without concern for the lower bits. Also, the i-th increaser writes the carry-over in the $n + i$-th position of the product, which has not been used so far and, thus, holds the value 0.

Example 3: Consider a 3-bit hierarchical multiplication with the factors $a = a_2a_1a_0$ and $b = b_2b_1b_0$ as well as the product $c = c_5c_4c_3c_2c_1c_0$. In total, three controlled increasers

```
for $i = 0$ to $n - 1$
{
        $\top_{j=i}^{n+i} c_i \overset{a_i}{+=} \top_{i=0}^{n-1} b_i$
}
```

Fig. 2. Hierarchical method

[2]Note that this states not only for truth tables but for any other function description as well.

are needed to realize this multiplication. The first controlled increaser is controlled by a_0. It conditionally adds the second factor b to the three least significant bits of the product c, i.e. to $c_2c_1c_0$. It also writes the carry-over into c_3. The second controlled increaser is controlled by a_1. It conditionally adds b to $c_3c_2c_1$. Again the carry-over is written into the next product-bit c_4. Finally, the third controlled increaser is controlled by a_2. It conditionally adds the second factor b to $c_4c_3c_2$. The carry-over is written into the most significant bit c_5 of the product.

C. Karatsuba Method

This section describes the realization of multipliers in reversible logic based on the divide-and-conquer-method of Karatsuba's algorithm [16]. The idea behind the Karatsuba algorithm is to realize the multiplication by multiplying two factors with smaller bit-width and additionally perform some less expensive operations. Consider an n-bit multiplication with $n = 2 \cdot k$. Both factors (e.g. $a = \sum_{i=0}^{2\cdot k-1} a_i \cdot 2^i$) are partitioned in an upper half ($\overline{a} := \sum_{i=k}^{2\cdot k-1} a_i \cdot 2^{i-k}$) and a lower half ($\underline{a} := \sum_{i=0}^{k-1} a_i \cdot 2^i$) such that $a = \overline{a} \cdot 2^k + \underline{a}$. With this representation the following equations are deducible:

$$\begin{aligned}
a \cdot b &= \left(\overline{a} \cdot 2^k + \underline{a}\right) \cdot \left(\overline{b} \cdot 2^k + \underline{b}\right) \\
&= \overline{a} \cdot \overline{b} \cdot 2^{2\cdot k} + (\underline{a} \cdot \overline{b} + \overline{a} \cdot \underline{b}) \cdot 2^k + \underline{a} \cdot \underline{b} \\
&= \overline{a} \cdot \overline{b} \cdot 2^{2\cdot k} + (\underline{a} \cdot \overline{b} + \overline{a} \cdot \underline{b} + \overline{a} \cdot \overline{b} + \underline{a} \cdot \underline{b} - \overline{a} \cdot \overline{b} - \underline{a} \cdot \underline{b}) \cdot 2^k + \underline{a} \cdot \underline{b} \\
&= \overline{a} \cdot \overline{b} \cdot 2^{2\cdot k} + (\underline{a} \cdot (\underline{b} + \overline{b}) + \overline{a} \cdot (\underline{b} + \overline{b}) - \overline{a} \cdot \overline{b} - \underline{a} \cdot \underline{b}) \cdot 2^k + \underline{a} \cdot \underline{b} \\
&= \overline{a} \cdot \overline{b} \cdot 2^{2\cdot k} + ((\underline{a} + \overline{a}) \cdot (\underline{b} + \overline{b}) - \overline{a} \cdot \overline{b} - \underline{a} \cdot \underline{b}) \cdot 2^k + \underline{a} \cdot \underline{b}
\end{aligned}$$

These equations show that a $(2 \cdot k)$-bit multiplication can be realized by three k-bit multiplications, some additions, subtractions, and bit-shifts, respectively. In reversible logic, additionally a number of circuit lines are required (as already shown in [21]). However, the latter can be reduced by choosing appropriate targets of the intermediate results as well as good orderings of the respective operations. Fig. 3 shows an optimized approach to generate reversible multiplier based on this observation (which requires less additional circuit lines than the method of [21]). The general idea is illustrated by the following example. Since the Karatsuba method is not applicable for very small bit-widths, the variable *turningPoint* denotes the bit-width below which the hierarchical multiplication is used. Beyond the *turningPoint* the Karatsuba method is used for multiplication.

Example 4: Consider an 8-bit Karatsuba multiplication and turning point $t = 6$. Since 8 is greater than t and even, the two conditionals (lines 1 and 4) do not hold and $k = 4$ is computed (line 7). Then, new variables d, e, f are initialized as shown in line 8. This leads to $4 \cdot 4 + 4 = 20$ garbage lines. Then, the two smaller multiplications $\underline{c} = c_7c_6c_5c_4c_3c_2c_1c_0 = a_3a_2a_1a_0 \cdot b_3b_2b_1b_0 = \underline{a} \cdot \underline{b}$ (line 9) and $\overline{c} = \overline{a} \cdot \overline{b}$ (line 10) are performed, respectively. Since these are 4-bit multiplications and the turning point in this example is $t = 6$, these multiplications are realized using the hierarchical approach described in the previous section. Afterwards, the result of these multiplication is directly assigned to the bits of the product. Furthermore, this result will be used later to modify the product of the sums that are computed next. These sums are $d = d_4d_3d_2d_1d_0 = a_7a_6a_5a_4 + a_3a_2a_1a_0 = \overline{a} + \underline{a}$ (line 11)

```
1  if ($n < turningPoint$)
2        $c = MULT_H\,(a, b)$
3
4  if ($n\%2 = 1$)
5        init $a_n, b_n, c_{2\cdot n}, c_{2\cdot n+1}$ with 0
6
7  $k := \lfloor \frac{n}{2} \rfloor$
8  init $d, e$ ($k+1$ bits), $f$ ($2 \cdot k+2$ bits) with 0
9  $\overline{c} = MULT_K(\overline{a}, \overline{b})$
10 $\underline{c} = MULT_K\,(\underline{a}, \underline{b})$
11 $d = \overline{a} + \underline{a}$
12 $e = \overline{b} + \underline{b}$
13 $h = MULT_K(d, e)$
14 $h -= \overline{c}$
15 $h -= \underline{c}$
16 $\top_{i=k}^{3\cdot k+3} c_i += h$
```

Fig. 3. Karatsuba method

and $e = \overline{b} + \underline{b}$ (line 12). They can be performed by copying the first summand to the (still uninitialized) target and then increasing it by the second summand. The results of these two sums are multiplied to get the third sub-product $h = d \cdot e$ (line 13). Again, since the turning point is greater than the size of the factors, this multiplication is performed by the hierarchical approach. After that, this third sub-product must be modified by subtracting the two earlier computed products from lines 9 and 10 as can be seen in lines 14 and 15 . Finally, the result is obtained by performing the addition of this value to the product using an implicit bitshift of k bits (line 16).

D. Discussion

As reviewed in Section II, there are different cost metrics to judge the quality of reversible circuits, namely the number of circuit lines, the number of gates, the quantum cost, and the transistor cost, respectively. In this section, the costs of the circuits for an n-bit multiplier obtained by the proposed methods are briefly discussed. Experimental results are afterwards given in Section IV.

The sub-minimal approach significantly reduces the minimal number of garbage lines using approximately $\frac{n}{2}$ garbage lines (instead of $n + 1$ garbage lines that the conventional approach minimally requires). The number of gates, the quantum cost, and the transistor costs depend on the applied synthesis method.

For the hierarchical approach, an n-bit adder without garbage lines is used, which needs $(5 \cdot n - 5)$ CNOT and $(2 \cdot n - 1)$ C^2NOT gates [22]. Accordingly, a controlled adder consists of $(5 \cdot n - 5)$ C^2NOT gates and $(2 \cdot n - 1)$ C^3NOT gates. The hierarchical method realizes a multiplier with n controlled adders. The first controlled addition can therefore be replaced with a controlled duplication (since the product initially is assigned to 0 and, thus, nothing has to be added). Consequently, this implementation of the hierarchical method leads to circuits with $(n - 1)$ controlled adders and one controlled duplication (which is done with n CNOT gates). In total, circuits result including $(5 \cdot n^2 - 9 \cdot n + 5)$ C^2NOT gates as well as $(2 \cdot n^2 - 3 \cdot n + 1)$ C^3NOT gates. Therefore, these circuits have quantum costs of $51 \cdot n^2 - 84 \cdot n + 38$ and transistor costs of $128 \cdot n^2 - 216 \cdot n + 104$, respectively. Furthermore, this approach uses separate lines for primary inputs (n lines for each factor) and primary outputs ($2 \cdot n$ lines for the product), i.e. in total $4 \cdot n$ lines.

The Karatsuba multiplication needs $(4 \cdot \lceil \frac{n}{2} \rceil + 5)$ garbage outputs per recursion step. Thus, if t is chosen as turning point and T_t represents the number of circuit lines needed for a t-bit hierarchical multiplier, circuits with approximately $3^{log_2(n)-log_2(t)} \cdot T_t + \sum_{i=0}^{log_2(n)-log_2(t)-1} 3^i \cdot (4 \cdot \lceil \frac{n}{2} \rceil + 5) \approx \frac{n}{t}^{log_2(3)} \cdot (T_t + 4 \cdot t) + 5 \cdot \frac{n}{t} - 4 \cdot n - 5$ garbage outputs are required (additionally to the $2 \cdot n$ lines for the primary inputs). The number of gates, the quantum cost, and the transistor costs are in $O(n^{log_2(3)})$.

In summary, the Karatsuba approach asymptotically has less quantum cost and transistor costs than the hierarchical approach. But, the hierarchical method requires less circuit lines. The sub-minimal approach leads to circuits with the lowest line count, but is limited by the truth table-based specification.

IV. Experiments

The proposed synthesis methods for multiplier have been implemented in C++. In this section, we provide experimental results generated by these methods and compare them to realizations obtained by general purpose approaches (namely the transformation-based [9], the ESOP-based [11], and the BDD-based [12] approach, respectively). For the sub-minimal specification, we used the transformation-based approach [9] to synthesize the circuits. The turning point for the Karatsuba approach was set to $t = 8$. The timeout (denoted by *TO*) was set to 1000 CPU seconds.

The results are presented in Table IV. Besides the respective bit-width and the resulting number of primary inputs (*PI*), the line count (*LC*), the gate count (*GC*), the quantum cost (*QC*), and the transistor costs (*TC*) are listed for each method, respectively. Additionally, the time (in CPU seconds) required to generate the results is listed in column *Time*.

First of all, the results confirm that using the adjusted specification from Section III-A, multiplier with a lower number of circuit lines can be generated. But due to the truth table-based description, the method works for very small bit-widths only. A similar behavior can be observed, if general purpose synthesis approaches based on ESOP and BDDs are applied. Indeed, somewhat larger multiplier can be synthesized, but practical relevant bit-widths (e.g. a 32-bit or 64-bit multiplier) cannot be generated. This can be explained by the fact that in particular for the multiplication no efficient representation as ESOP or as BDD, respectively, exists.

In contrast, the hierarchical method and the Karatsuba approach enable the synthesis of reversible multipliers for nearly arbitrary sizes. As discussed in the last section, the Karatsuba method requires more circuit lines, but leads to smaller realizations with respect to quantum cost and transistor cost (in particular for large bit-width). For a better overview, the results of both approaches are additionally illustrated in Fig. 4.

V. Conclusions

In this paper, we introduced three methods for multiplier synthesis that particularly address the drawbacks of previous approaches (e.g. the large number of circuit lines in the resulting realizations as well as the poor scalability). We showed that multiplier with a lower number of circuit lines can be obtained by using an adjusted specification of the underlying function. Besides that, two constructive approaches for synthesis of multipliers with very large bit-width are proposed. Experiments confirmed that using these methods, multipliers with large bit-widths can efficiently be synthesized, while previous approaches as well as general purpose synthesis methods do not scale very well on multiplication.

Acknowledgment

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

References

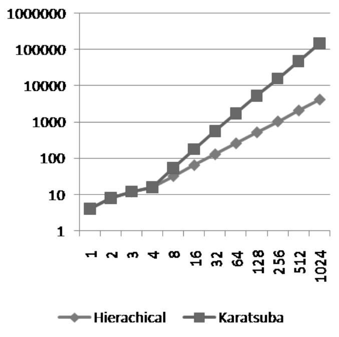
[1] V. V. Zhirnov, R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff, "Limits to binary logic switch scaling – a gedanken model," *Proc. of the IEEE*, vol. 91, no. 11, pp. 1934–1939, 2003.
[2] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Dev.*, vol. 5, p. 183, 1961.
[3] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev*, vol. 17, no. 6, pp. 525–532, 1973.
[4] B. Desoete and A. D. Vos, "A reversible carry-look-ahead adder using control gates," *INTEGRATION, the VLSI Jour.*, vol. 33, no. 1-2, pp. 89–104, 2002.
[5] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.
[6] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Foundations of Computer Science*, pp. 124–134, 1994.
[7] 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, p. 883, 2001.
[8] 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.
[9] 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.
[10] 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.
[11] K. Fazel, M. A. Thornton, and J. E. Rice, "ESOP-based Toffoli gate cascade generation," in *PACRIM*, 2007, pp. 206–209.
[12] R. Wille and R. Drechsler, "Bdd-based synthesis of reversible logic for large functions," in *DAC*, 2009, pp. 270–275.
[13] H. Thapliyal and M. B. Srinivas, "Novel reversible multiplier architecture using reversible TSG gate," in *International Conference on Computer Systems and Applications*, 2006, pp. 100–103.
[14] M. Haghparast, S. Jassbi, K. Navi, and O. Hashemipour, "Design of a novel reversible multiplier circuit using HNG gate in nanotechnology," *World Applied Sciences Journal*, vol. 3, no. 6, pp. 974–978, 2008.
[15] M. Islam, M. Rahman, Z. Begum, and M. Hafiz, "Low cost quantum realization of reversible multiplier circuit," *Information Technology Journal*, vol. 8, no. 2, pp. 208–213, 2009.
[16] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Doklady Akad. Nauk SSSR*, vol. 145, 1963.
[17] D. Maslov and G. W. Dueck, "Reversible cascades with minimal garbage," *IEEE Trans. on CAD*, vol. 23, no. 11, pp. 1497–1509, 2004.
[18] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, W. de Bakker and J. van Leeuwen, Eds. Springer, 1980, p. 632, technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.
[19] 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.
[20] M. K. Thomson and R. Glück, "Optimized reversible binary-coded decimal adders," *J. of Systems Architecture*, vol. 54, pp. 697–706, 2008.
[21] L. A. B. Kowada, R. Portugal, and C. M. H. Figueiredo, "Reversible Karatsuba's algorithm," *The Journal of Universal Computer Science*, vol. 12, no. 5, pp. 499–511, 2006.
[22] Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," in *Asian Conference on Quantum Information Science*, 2009.

TABLE IV
EXPERIMENTAL RESULTS

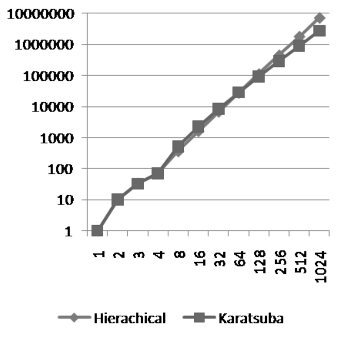
		Sub-minimal (+ transformation-based [9])					Minimal (+ transformation-based [9])				
Bit-width	PI	LC	GC	QC	TC	Time	LC	GC	QC	TC	Time
1	2	3	3	11	32	<0.01	4	2	6	24	0.01
2	4	6	158	1126	2184	0.01	7	344	3294	5032	0.02
3	6	9	2323	24904	32712	0.36	–	–	–	–	TO
4	8	–	–	–	–	TO	–	–	–	–	TO

		ESOP-based [11]					BDD-based [12]				
Bit-width	PI	LC	GC	QC	TC	Time	LC	GC	QC	TC	Time
1	2	4	1	5	16	0.01	8	5	9	32	0.01
2	4	8	6	72	128	0.01	17	20	44	160	0.01
3	6	12	36	591	720	0.02	29	49	117	448	0.01
4	8	16	169	3693	3744	0.06	47	103	279	1064	0.01
8	16	–	–	–	–	TO	609	2798	8842	33840	0.06
16	32	–	–	–	–	TO	531001	2806841	9225345	33932664	957,26
32	64	–	–	–	–	TO	–	–	–	–	TO

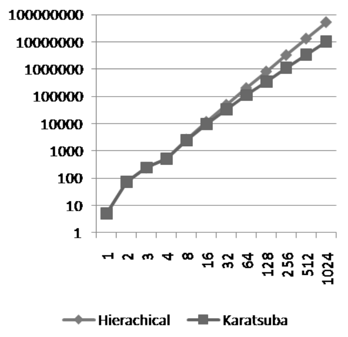
		Hierachical					Karatsuba				
Bit-width	PI	LC	GC	QC	TC	Time	LC	GC	QC	TC	Time
1	2	4	1	5	16	<0.01	4	1	5	16	<0.01
2	4	8	10	74	184	<0.01	8	10	74	184	<0.01
3	6	12	33	245	608	<0.01	12	33	245	608	<0.01
4	8	16	70	518	1288	<0.01	16	70	518	1288	<0.01
8	16	32	358	2630	6568	0.01	54	517	2437	7032	<0.01
16	32	64	1606	11750	29416	0.01	176	2304	9696	29352	<0.01
32	64	128	6790	49574	124264	0.01	554	8492	34000	105296	0.01
64	128	256	27910	203558	510568	0.01	1712	28710	111966	351096	0.04
128	256	512	113158	824870	2069608	0.27	5234	92672	355972	1124432	0.12
256	512	1024	455686	3320870	8333416	0.61	15896	291174	1108206	3516312	0.38
512	1024	2048	1828870	13326374	33443944	1.34	48074	899912	3405340	10835696	0.98
1024	2048	4096	7327750	53391398	133996648	5.94	144992	2752590	10377606	33081336	1.43



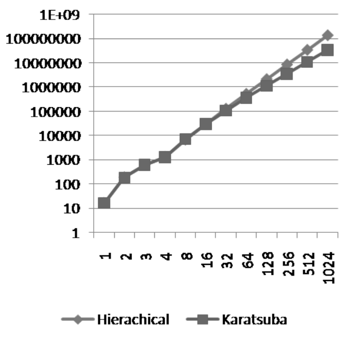
(a) Line count (LC)



(b) Gate count (GC)



(c) Quantum cost (QC)



(d) Transistor cost (TC)

Fig. 4. Comparison between Hierachical method and Karatsuba method