Pushing the Number of Qubits Below the "Minimum": Realizing Compact Boolean Components for Quantum Logic

Alwin Zulehner Robert Wille
Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
alwin.zulehner@jku.at robert.wille@jku.at

***Abstract*—Research on quantum computers has gained attention since they are able to solve certain tasks significantly faster than classical machines (in some cases, exponential speed-ups are possible). Since quantum computations typically contain large Boolean components, design automation techniques are required to realize the respective Boolean functions in quantum logic. They usually introduce a significant amount of additional qubits – a highly limited resource. In this work, we propose an alternative method for the realization of Boolean components for quantum logic. In contrast to the current state-of-the-art, we dedicatedly address the main reasons causing the additionally required qubits (namely the number of the most frequently occurring output pattern as well as the number of primary outputs of the function to be realized) and propose to manipulate the function so that both issues are addressed. The resulting methods allow to push the number of required qubits below what is currently considered the minimum.**

I. Introduction

Quantum computers [10] have gained significant attention since they might be used in the future to solve certain tasks significantly faster than classical machines – including applications for which exponential speed-ups are possible. In the recent years, physical realizations of quantum computers have made significant progress [5]. Well-known companies like Google, Microsoft, Intel, or IBM already announced to realize quantum computers with 50 qubits within the near future – a huge step towards quantum computers that are able to solve practically relevant tasks. But even with all these accomplishments, qubits are still a rather limited resource and their number should be kept as small as possible. This is additionally motivated because quantum computations are subject to frequent errors caused by quantum decoherence. As a result, information stored in qubits is very unstable and requires constant error-correction [11] – mainly relying on redundant computations which increase the number of qubits further.

This poses a challenge to corresponding design automation techniques which are particularly needed to realize the Boolean components (often called *oracle*) that frequently occur in many quantum algorithms. Since quantum computations are inherently reversible, it has to be ensured that these components are realized in a reversible fashion, i.e. as a function realizing a unique mapping from the inputs to the outputs *and* vice versa (cf. [6, 17, 14, 21]). This usually requires a significant amount of additional qubits (also called *working* or *ancillae* qubits of the oracle) – as said, a highly limited resource.

In this work, we investigate how this overhead in qubits can be reduced. We think thereby beyond the current state-of-the-art [6, 17, 14, 21], which considers the given Boolean function to be realized as a fixed entity. More precisely, while previous work simply tried to minimize the number of additional qubits for a given Boolean function f (eventually determining a minimum number of qubits needed to realize f), we go one step further and dedicatedly address the reasons why these additional qubits become necessary in the first place. Indeed, we discuss how the number of times the most frequently output pattern of f occurs as well as the number of primary outputs of f significantly affects the number of needed qubits and propose coding techniques that manipulate f so that both issues are addressed. As a result, the number of additionally required qubits can be pushed below what is currently considered the minimum.

Experimental evaluations confirm that, for most commonly used benchmarks, the number of required qubits can significantly be reduced compared to the minimum obtained by the state-of-the-art. As a further positive side-effect, the manipulations applied to f also help to reduce the costs for realizing the Boolean components in quantum logic. In fact, improvements of 92.7% on average and of several orders of magnitudes in the best cases are observed.

This work has partially been supported by the European Union through the COST Action IC1405.

II. Number of Qubits for Boolean Components

A major obstacle in the design automation of Boolean components for quantum logic is the fact that the Boolean function to be realized has to be made reversible first, i.e. each possible input pattern must map to a unique output pattern [10]. Therefore, it is essential to understand how many non-unique output patterns exists in a function to be realized and how often they occur.

Definition 1. *Let $f : \mathbb{B}^n \to \mathbb{B}^m$ be a Boolean function. Then, $p_i \in \mathbb{B}^m$, $1 \leq i \leq 2^m$ is one possible output pattern for f. The function $\mu : \mathbb{B}^m \to \mathbb{N}_0$ gives how often an output pattern $p_i \in \mathbb{B}^m$ occurs in the function f. Furthermore, the indices i are sorted with respect to the number of occurrences, i.e. p_1 represents the most frequently occurring output pattern, while p_{2^m} represents the least frequently occurring output pattern.*

If $\mu(p_i) = 1$ for all possible patterns $p_i \in \mathbb{B}^m$ of a given function f, then f is already reversible (each output pattern p_i occurs only once and, hence, can uniquely be mapped to an input pattern). In this case, nothing else needs to be done. However, for many functions to be realized, this is often not the case and output patterns that occur more than once have to be made distinguishable. To this end, the respective output pattern p_i is extended by so-called *garbage outputs* which allow to make all occurrences of p_i distinguishable [6, 17, 14, 21]. Obviously, $\lceil \log_2 \mu(p_i) \rceil$ garbage outputs are sufficient for this purpose as this allows for $\mu(p_i)$ unique ways to extend p_i. Since (w.l.o.g. following the definition from above) p_1 is the most frequently occurring pattern, a total of $\lceil \log_2 \mu(p_1) \rceil$ garbage outputs are required to make an arbitrary function f reversible.

Example 1. *Consider the Boolean function f shown in Table Ia. The most frequently occurring output pattern is $p_1 = 100$. Since this patterns occurs $\mu(p_1) = 5$ times, at least $\lceil \log_2 \mu(p_1) \rceil = \lceil \log_2 5 \rceil = 3$ garbage outputs are required in order to make the occurrences of the output pattern p_1 distinguishable. Using these additional outputs, p_1 can be extended*

TABLE I: Truth table of a Boolean function f

(a) Original

x_3	x_2	x_1	y_3	y_2	y_1
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	0	1	0

(b) Reversible

a_3	a_2	a_1	x_3	x_2	x_1	y_3	y_2	y_1	g_3	g_2	g_1
0	0	0	**0**	**0**	**0**	**1**	**0**	**0**	0	0	0
0	0	0	**0**	**0**	**1**	**1**	**0**	**0**	0	0	1
0	0	0	**0**	**1**	**0**	**1**	**1**	**0**	0	0	0
0	0	0	**0**	**1**	**1**	**1**	**0**	**0**	0	1	0
0	0	0	**1**	**0**	**0**	**0**	**1**	**0**	0	0	0
0	0	0	**1**	**0**	**1**	**1**	**0**	**0**	0	1	1
0	0	0	**1**	**1**	**0**	**1**	**0**	**0**	1	0	0
0	0	0	**1**	**1**	**1**	**0**	**1**	**0**	0	1	0
			⋮				⋮				

as e.g. shown in Table Ib. The originally intended function f can still be extracted by applying input patterns where the first three qubits are set to 0 *(highlighted bold in Table Ib), while the overall functionality ensures reversibility.*

Overall, adding garbage outputs obviously increases the number of required qubits in order to realize f in quantum logic (i.e. in a reversible fashion). In fact, given a function $f : \mathbb{B}^n \to \mathbb{B}^m$ with p_1 being the most frequently occurring output pattern, the correspondingly resulting reversible function requires at least $n' = \max(n, m + \lceil \log_2 \mu(p_1) \rceil)$ qubits. In other words, following the state of the art, a function realizing f in a reversible fashion with a minimum number of required qubits would be of the form $f_{sota} : \mathbb{B}^{n'} \to \mathbb{B}^{n'}$.

The scheme of determining a reversible function out of an arbitrarily given function f as reviewed above is called *embedding* and addressed e.g. in [6, 14, 21]. Afterwards, the resulting reversible function serves as input e.g. for synthesis approaches such as [12, 9, 7, 13]. Besides that, also alternative solutions have been proposed which take a Boolean function and implicitly determine the reversibility during the synthesis (see e.g. [4, 19, 15]). They, however, yield realizations with a large number of required qubits (usually magnitudes above the minimum as observed e.g. in [17]).

III. Pushing the Number of Required Qubits Below the Minimum

Since the emergence of initial design solutions for quantum logic, a main objective was to keep the number required qubits (and, hence, the overhead caused by ensuring reversibility) as small as possible. This led to several solutions which guarantee a minimum of required qubits as already discussed above (and proposed e.g. in [6, 14, 21]). However, all these endeavors did not address the main reasons which lead to a large number of required qubits, namely:

- The number of times the most frequent output pattern occurs (i.e. $\mu(p_1)$) – basically defining the number of garbage outputs.
- The number m of primary outputs.

In this work, we propose alternative solutions that actively takes $\mu(p_1)$ as well as m into consideration and aims for reducing these values by using coding techniques. Accordingly, this allows for reducing the number of required qubits – which in many cases can be pushed below a value that is considered the minimum thus far. The general ideas of the proposed solution for reducing the number of required qubits are (1) to treat frequently occurring output patterns in a special fashion and/or (2) to dedicatedly encode output patterns – both in order to reduce the values of $\mu(p_1)$ and m, respectively.[1]

[1]Note that both ideas eventually yield a different function than the original one (i.e. f_{sota}). However, the proposed ideas can nevertheless be exploited to realize more compact Boolean components, because those components (the oracles) typically employ a hierarchical structure. Here, several sub-components are required that compute intermediate results stored in the working or ancillae qubits. If those sub-components respectively consider that the intermediate results are partially represented by special outputs and/or codings, the functionality of the top component (eventually used by the remainder of the quantum circuit) still can be realized as desired.

A. Adding Special Outputs

Functions f where the number of occurrences of p_1 is very dominant compared to the other output patterns (i.e. for functions with $\mu(p_1) \gg \mu(p_2)$), often require a substantial number of garbage outputs that serve no purpose except making p_1 distinguishable. In contrast, we propose to treat p_1 by a *special output* s_{p_1}. That is, we extend the initially given function by a new output s_{p_1} which always assumes the value 1 iff an input assignment is supposed to generate p_1. At the same time, the original output values can be set arbitrarily in these cases allowing for a reduction of $\mu(p_1)$ and, hence, a reduction of the number of required garbage outputs.

The general concept of this can, in principle, also be generalized for the next frequently occurring patterns p_2, p_3, etc. To this end, we can simply add up to $k \in \{1, \ldots, m\}$ special outputs $s_1, \ldots, s_k$ to an originally given function f. These indicate whether an input assignment is supposed to generate either one of the $2^k - 1$ most frequently occurring output patterns (in case $[s_1, \ldots, s_k]_2 = i > 0$, the output pattern p_i is supposed to be generated) or one of the other output patterns (in case $[s_1, \ldots, s_k]_2 = 0$ the output pattern is supposed to be extracted from the actual primary outputs). An example demonstrates the idea:

Example 2. *Consider a Boolean function with 6 inputs and 5 outputs, where the output patterns are distributed as shown in Table IIa. Furthermore, assume that we add $k = 2$ special outputs to the function and, by this, handle the most frequently occurring output patterns p_1, p_2, and p_3 by them. This reduces the remaining values of $\mu(p'_i)$ as shown in Table IIb. If the function is supposed e.g. to evaluate to output pattern $p_2 = 11001$, the special outputs s_1 and s_2 are set to 1 and 0, respectively. The values of the primary outputs do not matter in this case and, hence, can be set arbitrarily. This degree of freedom is then used to make the occurrences of $s_1 s_2 = 01$, $s_1 s_2 = 10$, and $s_1 s_2 = 11$ (which represent p_1, p_2, and p_3, respectively) distinguishable. Consequently, the primary outputs serve as garbage outputs in cases where $s_1 s_2 \neq 00$. Since $\lceil \log_2 \mu(p_1) \rceil = 5$, the five primary outputs are sufficient to distinguish all occurrences.*

In contrast, if the function of interest evaluates e.g. to output pattern $p_4 = 01011$, the special outputs s_1 and s_2 are both set to 0. Then, the desired output value can be obtained from the primary outputs, i.e. they are assigned 01011. *Since p_4 is now the most frequent pattern that can be obtained at the primary outputs, only $\lceil \log_2 \mu(p_4) \rceil = 2$ actual garbage outputs are required (compared to the $\lceil \log_2 \mu(p_1) \rceil = 5$ garbage outputs in case the original function from Table IIa is considered).*

While this approach often yields reductions in the number of required qubits, its effect strongly depends on the respective occurrences $\mu(p_i)$ of output patterns p_i of f and how they are distributed. In simple words, if most frequently occurring output patterns are very dominant (i.e. occur quite often compared to the remaining output patterns) large reductions can be achieved. If their occurrences, however, differ only slightly, no improvements or even a larger number of required qubits results.

Example 3. *Consider again the Boolean function with 6 inputs and 5 outputs, where the output patterns are distributed as shown in Table IIa. Since the most frequent output pattern occurs 21 times (i.e. $\mu(p_1) = 21$), 5 garbage outputs are required (resulting in a reversible function which requires $\max(6, 5 + 5) = 10$ qubits). Adding $k = 1$ special output to the function reduces the number of required garbage outputs*

TABLE II: Distribution of the output patterns

(a) Original

i	p_i	$\mu(p_i)$
1	00010	21
2	11001	14
3	01110	6
4	01011	4
⋮	⋮	⋮
8	00000	4
9	01111	3
10	01001	0
⋮	⋮	⋮

(b) With special outputs

i	$s_1 s_2$	p'_i	$\mu(p'_i)$
1	0 1	–	–
2	1 0	–	–
3	1 1	–	–
4	0 0	01011	4
⋮	⋮	⋮	⋮
8	0 0	00000	4
9	0 0	01111	3
10	0 0	01001	0
⋮	⋮	⋮	

(c) With coding

i	p'_i	$\mu(p'_i)$
1	0000	21
2	0001	14
3	0010	6
4	0011	4
5	0100	4
6	0101	4
7	0110	4
8	0111	4
9	1000	3
10	1001	0
⋮	⋮	⋮

to $\lceil \log_2 \mu(p_{2^k}) \rceil = 4$*, but again results in a function with* $\max(6, 5+4)+1 = 10$ *qubits. Setting the number of special outputs to* $k = 2$ *reduces the number of required qubits to* $\max(6, 5 + \lceil \log_2 \mu(p_{2^k}) \rceil) + 2 = 9$*. Adding one more special output (*$k = 3$*), the number of required qubits again increases to* $\max(6, 5+2)+3 = 10$*. Hence, adding more special outputs does not always result in a function that requires fewer qubits.*

Overall, adding k special outputs to the original function reduces the overall number of outputs if, and only if, $k + \lceil \log_2 \mu(p_{2^k}) \rceil < \lceil \log_2 \mu(p_1) \rceil$. Since the overall number of required qubits also depends on the number of primary inputs, the optimal number of special outputs k_{opt} for a Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ is determined by

$$k_{opt} = \underset{0 \leq k < m}{\arg\min} \left(\max \left(n, m + \lceil \log_2 \mu(p_{2^k}) \rceil \right) + k \right).$$

This optimal k_{opt} heavily depends on the considered function. However, it can easily be determined by inspection of the function and, hence, prior to the design.

B. Coding Output Patterns

Besides reducing the number of garbage outputs (as accomplished using special outputs discussed above), the number of required qubits can further be optimized by reducing the number of primary outputs m. This is particularly beneficial if the number of primary outputs is larger than the number of primary inputs, i.e. iff $m > n$. Then, at least one of the outputs pattern $p_i \in \mathbb{B}^m$ exists which never occurs as an output pattern (i.e. $\mu(p_i) = 0$)[2].

A precise approach accomplishing that reads as follows: Consider again a Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ and let h be the number of output patterns that occur at least once, i.e. $h = |\{p_i | \mu(p_i) > 0\}|$. Then, $\lceil \log_2 h \rceil$ coded outputs $c_1, \ldots, c_{\lceil \log_2 h \rceil}$ are required to encode all output patterns, where $\left[c_1, \ldots, c_{\lceil \log_2 h \rceil}\right] = i$ represents output pattern p_{i+1}. The original outputs are then not required anymore – resulting in a new function $f' : \mathbb{B}^n \to \mathbb{B}^{\lceil \log_2 h \rceil}$. An example demonstrates the idea:

Example 4. *Consider again a Boolean function with 6 inputs and 5 outputs, where the output patterns are distributed as shown in Table IIa. There exist* $h = 9$ *output patterns that occur at least once. Consequently, four coded outputs are required to represent all relevant output patterns – resulting in the distribution of coded output patters* p'_i *shown in Table IIc. For example, the coded outputs* $p'_2 = c_1c_2c_3c_4 = 0001$ *and* $p'_8 = c_1c_2c_3c_4 = 0111$ *represent output patterns* $p_2 = 11001$ *and* $p_8 = 00000$*, respectively. Applying a coding for this function reduces the number of primary outputs from 5 to 4 and the number of required qubits from 10 to 9.*

[2]Our experimental evaluations summarized later in Section IV confirm that many functions indeed map the inputs patterns to a rather small subset of output patterns.

A coding for the output patterns does not affect the number of the required garbage outputs, because the coding is unique. Hence, the number of occurrences of the patterns remain the same. In case that $\lceil \log_2 h \rceil < m$ and $m + \lceil \log_2 \mu_{p_1} \rceil > n$, the coding scheme proposed here indeed will allow for the realization of a given function f with a smaller number of required qubits.

C. Combination of Both Approaches

The approaches for reducing the number of required qubits described in this section are independent to each other. While the approach discussed in Section III-A aims for a reduction of the garbage outputs, the coding scheme proposed in Section III-B aims for a reduction of the primary outputs. Therefore, both schemes can be applied in a combined fashion. First, we apply a coding for the outputs, and then we add special outputs to reduce the number of garbage outputs. An example illustrates the idea.

Example 5. *Consider again the Boolean function with 6 inputs discussed in Example 4. Applying a coding for the relevant output patterns (those that occur at least once) reduces the number of primary outputs from 5 to 4 – resulting in the distribution of the output patters shown in Table IIc.*

The overall number of required qubits is $\max(6, 4 + \lceil \log_2 p_1 \rceil) = 9$*. To reduce the number of garbage outputs, we again add* $k = 2$ *special outputs (cf. Example 3) – resulting in a function with* $\max(6, 4 + \lceil \log_2 p_4 \rceil) + 2 = 8$ *qubits. Hence, in this example, the reductions of the individual approaches sum up.*

IV. Experimental Results

We experimentally evaluated the approaches proposed above for the realization of Boolean components in quantum logic with less than the minimum number of required qubits. To this end, we implemented the schemes from Sections III-A and III-B in C++ and evaluated them on benchmarks taken from commonly used benchmarks suits such as RevLib [16], ISCAS [1], IWLS [8], and ITC [3].

A. Reduction Below the Minimum

In a first series of experiments, we evaluated how much the number of required qubits can be pushed below the minimum with the approaches proposed in this work. To this end, we analyze the functions f_{sota} obtained using the state-of-the-art solutions from [6, 17, 14, 21] and the functions f_{prop} obtained by the solution proposed in this work (i.e. the combined approach discussed in Section III-C). The accordingly obtained numbers are listed in Table III. The first three columns list the name of the benchmark, the number of primary inputs (n), and the number of primary outputs (m). In the columns labeled q we list the number of required qubits for f_{sota} (i.e. the number that is considered the minimum thus far) and for f_{prop}. For f_{prop}, we additionally list the optimal number k_{opt} of special outputs. The last column of Table III provides the obtained improvements regarding the number of required qubits. Note that the runtime for determining the number of required qubits is not listed in this table, since it was negligible for all benchmarks (less than 10 seconds).

The experimental evaluation clearly shows the benefit of the approach proposed in this work. In 38 out of 46 cases, the function of interest can be realized with fewer qubits than previously considered to be the minimum. For 21 benchmarks, the number of required qubits can be reduced by more than 10 compared to the state-of-the-art. In the case of benchmark *cps*, even more than 100 qubits can be saved – a reduction of

TABLE III: Synthesis results (using QMDD-based synthesis)

			f_{sota} as input			f_{prop} (Sec. III-C) as input				
Name	n	m	q	t	T-depth	k_{opt}	q	t	T-depth	Δ_q
decod	5	16	20	0.3	50 172	1	6	0.0	441	14
bw	5	28	32	1.6	62 376	1	6	0.0	852	26
b11	8	31	36	2.3	1 070 055	0	10	0.0	5 022	26
ex5p	8	63	68	22.7	3 913 413	0	12	0.7	162 111	56
clip	9	5	11	–	–	0	11	–	–	0
dk27	9	9	15	0.1	134 538	0	12	0.0	59 766	3
apex4	9	19	26	4.4	1 160 232	2	14	2.6	297 222	12
sao2	10	4	14	0.5	402 333	2	12	0.0	11 772	2
alu2	10	6	14	–	–	0	14	–	–	0
example2	10	6	14	–	–	0	14	–	–	0
x2	10	7	16	0.1	290 679	2	12	0.0	6 381	4
alu3	10	8	14	0.3	605 802	0	13	0.8	589 215	1
ex1010	10	10	18	2.0	1 952 757	1	15	1.3	821 118	3
dk17	10	11	19	0.9	1 142 160	0	13	0.4	353 694	6
apla	10	12	22	0.9	1 720 944	2	13	0.1	107 679	9
cm85a	11	3	13	–	–	0	13	–	–	0
add6	12	7	13	–	–	0	13	–	–	0
alu4	14	8	19	32.7	20 530 881	0	17	45.7	15 406 296	2
f51m	14	8	19	–	–	0	19	–	–	0
tial	14	8	19	44.2	21 111 420	1	17	45.3	13 364 280	2
cu	14	11	25	4.2	7 087 479	2	17	0.1	110 949	8
misex3	14	14	28	86.0	64 760 979	2	16	6.6	3 488 655	12
misex3c	14	14	28	91.7	64 535 337	2	16	5.5	3 627 549	12
table3	14	14	28	79.6	66 211 986	3	17	1.5	371 895	11
s1488	14	25	38	58.1	78 345 783	2	19	35.5	13 747 935	19
s1494	14	25	38	46.5	62 481 213	2	19	40.4	14 174 664	19
in0	15	11	25	155.2	94 018 080	1	19	41.2	13 458 195	6
cmb	16	4	20	0.6	2 026 779	1	17	0.0	7 011	3
pcler8	16	5	21	12.6	7 833 705	1	17	4.2	4 333 239	4
pdc	16	40	55	406.6	121 308 453	3	22	869.3	7 472 673	33
spla	16	46	61	524.0	337 363 242	3	22	60.4	12 289 548	39
table5	17	15	32	>1h	–	1	20	13.3	4 238 280	12
s208.1	18	9	19	–	–	0	19	–	–	0
cm151a	19	9	27	>1h	–	1	21	2827.6	444 535 209	6
duke2	22	29	50	>1h	–	3	26	>1h	–	24
cordic	23	2	25	>1h	–	1	24	1366.1	33 781 044	1
cps	24	109	132	>1h	–	2	28	>1h	–	104
vg2	25	8	32	>1h	–	1	29	61.1	2 641 671	3
misex2	25	18	42	>1h	–	3	28	>1h	–	14
frg1	28	3	30	–	–	0	30	–	–	0
apex2	39	3	42	>1h	–	1	41	>1h	–	1
seq	41	35	75	>1h	–	3	44	>1h	–	31
apex1	45	45	89	>1h	–	1	51	>1h	–	38
apex3	54	50	103	>1h	–	1	57	>1h	–	46
e64	65	65	129	>1h	–	3	68	>1h	–	61
ex4p	128	28	146	>1h	–	0	130	>1h	–	16

almost 79%. Overall, substantial improvements are obtained compared to the number of required qubits which are currently considered the minimum.

B. Further Benefits

Having a function with significantly fewer qubits obviously also effects/improves the costs of the resulting quantum circuits (i.e. a sequence of quantum operations that are applied to the qubits). This has been evaluated in a second series of experiments, in which we realize the function f_{sota} and f_{prop} in quantum logic. As synthesis engine we utilized QMDD-based synthesis which was originally proposed in [13] and recently improved in [20]. Table III again lists the respectively obtained results, i.e. the runtime t required to synthesize the circuit, as well as the complexity of the circuit (by means of *T-depth* [2]). The number of required qubits is inherently given by the first series of experiments (and, thus, listed in the columns labeled q). Note that we list only synthesis results where the number of qubits indeed can be pushed below the minimum.

The results clearly show that the approaches proposed in this work do not only reduce the number of required qubits, but also yield a significantly smaller circuit complexity (in terms of T-depth). In fact, an average improvement of 92.7% is observed; in some cases (e.g. *decod*, *cmb*, etc.), cost reductions of several orders of magnitudes can be reported. Functions obtained by the proposed approaches also performs better in terms of runtime for most benchmarks. In four cases, the optimized function f_{prop} even allowed to complete the synthesis within the time limit whereas the synthesis of the state-of-the-art function f_{sota} failed.

These results are particularly remarkable since, thus far, a reduction of the number of required qubits almost always yielded an increase in the resulting circuit costs (as e.g. observed in [18]). Hence, using the approaches proposed in this work do not only allow for realizing Boolean functions in quantum logic with less than the minimum number of required qubits, but also enables designers to synthesize the respective circuits with significantly fewer cost.

V. Conclusions

In this work, we proposed a novel approach for realizing Boolean components in quantum logic using a smaller number of qubits than what is currently considered to be the minimum. To this end, we explicitly considered the main reasons that lead to a large number of qubits thus far: the number of times the most frequent output pattern occurs and the number of primary outputs. Following the proposed approaches yield significant reductions and, hence, allow for a much more efficient realization of the corresponding quantum computations. Considering that reducing the number of required qubits for quantum computation has been considered for a long time, pushing this number below what is currently considered the minimum as proposed in this work offers a new direction in the design of Boolean components for quantum logic. A first complete synthesis approach towards this direction has recently been proposed in [22].

References

[1] ISCAS'89 benchmark information. *www.cbl.ncsu.edu/ CBL_Docs/ iscas89.html*, 1997.

[2] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[3] F. Corno, M. Reorda, and G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. *Design Test of Computers, IEEE*, 17(3):44–53, Jul 2000.

[4] K. Fazel, M. Thornton, and J. Rice. ESOP-based Toffoli gate cascade generation. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 206 –209, 2007.

[5] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, page 201618020, 2017.

[6] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Trans. on CAD*, 23(11):1497–1509, 2004.

[7] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. on Design Automation of Electronic Systems*, 12(4), 2007.

[8] K. McElvain. IWLS'93 benchmark set: Version 4.0. In *Int'l Workshop on Logic Synth.*, 1993.

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

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

[11] M. Reed, L. DiCarlo, S. Nigg, L. Sun, L. Frunzio, S. Girvin, and R. Schoelkopf. Realization of three-qubit quantum error correction with superconducting circuits. *Nature*, 482(7385):382–385, 2012.

[12] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *Int'l Conf. on CAD*, pages 353–360, 2002.

[13] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *ASP Design Automation Conf.*, pages 85–92, 2012.

[14] M. Soeken, R. Wille, O. Keszocze, D. M. Miller, and R. Drechsler. Embedding of large Boolean functions for reversible logic. *J. Emerg. Technol. Comput. Syst.*, 12(4):41:1–41:26, Dec. 2015.

[15] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Automation Conf.*, pages 270–275, 2009.

[16] 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*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[17] R. Wille, O. Keszöcze, and R. Drechsler. Determining the minimal number of lines for large reversible circuits. In *Design, Automation and Test in Europe*, 2011.

[18] R. Wille, M. Soeken, D. M. Miller, and R. Drechsler. Trading off circuit lines and gate costs in the synthesis of reversible logic. *Integration*, 47(2):284–294, 2014.

[19] Z. Zilic, K. Radecka, and A. Kazamiphur. Reversible circuit technology mapping from non-reversible specifications. In *Design, Automation and Test in Europe*, pages 558–563, 2007.

[20] A. Zulehner and R. Wille. Improving synthesis of reversible circuits: Exploiting redundancies in paths and nodes of QMDDs. In *Conf. on Reversible Computation*, 2017.

[21] A. Zulehner and R. Wille. Make it reversible: Efficient embedding of non-reversible functions. In *Design, Automation and Test in Europe*, 2017.

[22] A. Zulehner and R. Wille. Exploiting coding techniques for logic synthesis of reversible circuits. In *ASP Design Automation Conf.*, 2018.