One Additional Qubit is Enough:
Encoded Embeddings for Boolean Components in Quantum Circuits

Alwin Zulehner*, Philipp Niemann†, Rolf Drechsler†‡, and Robert Wille*
*Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
†Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
‡Department of Computer Science, University of Bremen, Bremen, Germany
alwin.zulehner@jku.at, philipp.niemann@dfki.de, drechsle@informatik.uni-bremen.de, robert.wille@jku.at

***Abstract*—Research on quantum computing has recently gained significant momentum since first physical devices became available. Many quantum algorithms make use of so-called *oracles* that implement Boolean functions and are queried with highly superposed input states in order to evaluate the implemented Boolean function for many different input patterns in parallel. To simplify or enable a realization of these oracles in quantum logic in the first place, the Boolean reversible functions to be realized usually need to be broken down into several non-reversible sub-functions. However, since quantum logic is inherently reversible, these sub-functions have to be realized in a reversible fashion by adding further qubits in order to make the output patterns distinguishable (a process that is also known as *embedding*). This usually results in a significant increase of the qubits required in total. In this work, we show how this overhead can be significantly reduced by utilizing coding. More precisely, we prove that one additional qubit is always enough to embed *any* non-reversible function into a reversible one by using a variable-length encoding of the output patterns. Moreover, we characterize those functions that do not require an additional qubit at all. The made observations show that coding often allows one to undercut the usually considered minimum of additional qubits in sub-functions of oracles by far.**

I. Introduction

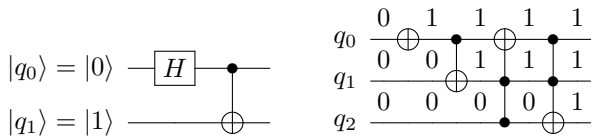
Quantum algorithms running on quantum computers allow for significant (exponential in the best case) speed-ups compared to their classical counterparts by exploiting quantum-mechanical phenomena like superposition, entanglement, and phase shifts [1]. Recently, devices that have been made publicly available—together with the commitment of companies like IBM, Google, Microsoft, and Rigetti—brought new momentum into a domain that has been considered as a "dream of the future" for a long time [2]–[4]. Even though these first devices are limited in qubit fidelity and their number of qubits (i.e., they are classified as NISQ devices [5]), they provide a first step towards building a fault-tolerant quantum computer that is capable of conducting hard and useful tasks in non-exponential time.

Many proposed quantum algorithms contain large Boolean parts (also called *oracles*) that are queried with a highly superposed input to gain quantum speed-up. Examples are the modular exponentiation in Shor's algorithm for integer factorization [6] or a Boolean description of the database that is queried in Grover's Algorithm [7]. In order to use these Boolean components on a quantum computer, they have to be described as quantum circuits (i.e., a sequence of quantum operations that are applied to the qubits)—an inherently reversible description means. Since it is very complex to determine a sequence of quantum operations (also denoted quantum gates) that realize the desired functionality (a process termed *synthesis* [8], [9]), the Boolean function to be realized is usually decomposed into several (not necessarily reversible) sub-functions [10]–[12]. Hence, even though the overall functionality of the oracle is inherently reversible, its sub-components may not be.

In order to realize non-reversible functions in quantum logic, further qubits (often called *ancillary*, *ancillae*, or *working* qubits) have to be added in order to make the output patterns distinguishable and, hence, obtain a reversible function (a process called *embedding* [13], [14]). Moreover, such additional qubits are often used to store intermediate results and have to be restored to their initial state (by de-computing intermediate results) before "leaving" the oracle. All this obviously increases the number of qubits needed to realize the oracle. In fact, even if the embedding process guarantees a minimum of ancillary/ancillae/working qubits, their number is frequently quite substantial—a severe drawback since qubits are a highly limited resource.

In order to overcome the issue outlined above, we propose to utilize *coded* embeddings where each occurring output pattern is encoded with another (smaller) unique pattern. This way, we utilize recently proposed embedding and synthesis schemes such as *one-pass synthesis* of reversible logic [15] as well as synthesis exploiting coding techniques [16], [17] for the realization of quantum oracles. Encoding outputs allows us to significantly reduce the number of qubits even below what is usually considered to be the minimum. Although this changes the intended functionality, using encoded values is still acceptable for the realization of oracles since subsequent sub-components just have to be slightly adjusted to handle the code, or need to be equipped with a small decoder beforehand (which often is easier to realize than the original functionality).

Moreover, in this work we show for the first time that utilizing all that potential indeed allows for the realization of Boolean non-reversible sub-components with at most one additional qubit only. In addition to that, we exactly identify the cases where even this additional qubit is not necessary. By this, we can provably show that, using coding, one additional qubit is enough, i.e., that the proposed scheme often allows one to undercut the usually considered minimum of additional qubits in oracles by far. This is additionally confirmed by experimental evaluations. Note that while we only cover the two-valued case here—since this is the de facto standard in quantum computation and a large set of benchmarks is available for evaluation—we expect that our (theoretical) results can be extended to the multiple-valued case with a radix $r > 2$ in a straightforward fashion (e.g., using the generalization as proposed in [18]).



(a) Quantum Circuit (b) Reversible Circuit

Fig. 1: Circuit Diagrams

The remainder of this work is structured as follows. In Section II, we briefly introduce the basics of quantum circuits as well as how non-reversible functions can be realized by them. Section III provides a technique for encoding the function to be realized. Here, we also formally prove that using a variable-length code indeed allows for realizations with at most one additional qubit. Section IV compares the number of required qubits in coded embeddings to those embeddings (without encoding) that have been considered to be the minimum thus far. Section V concludes the paper.

II. Background

In this section, we briefly recap the basics of quantum circuits, as well as how to realize Boolean components occurring in them.

A. Quantum Circuits

Quantum computations are conducted by applying operations to qubits—entities that cannot only be in one of its two basis states (denoted $|0\rangle$ and $|1\rangle$), but also in an (almost) arbitrary superposition of both. Typical operations acting on a single qubits are negating the state of a qubit (NOT operation, denoted by X or $\oplus$), setting a qubit into superposition (Hadamard operation, denoted by H), or conducting a phase shift by i (denoted by S). Moreover, these operations may be controlled by other qubits. Then, the operation is only conducted if all controlling qubits are in basis state $|1\rangle$. All these computations may be represented by means of circuit diagrams, where each qubit is represented by a horizontal line and quantum gates (i.e., operations that are applied to the qubits) on these lines determine (from left to right) in which order the respective operations are applied to the qubits.

Example 1. *The quantum circuit shown in Fig. 1a is composed of two qubits and two gates. First, a Hadamard operation is applied to qubit q_0, setting q_0 into a superposition. Afterwards, a controlled NOT (CNOT) operation is conducted, where q_0 serves as control qubit and q_1 is the target qubit. Here, the value of q_1 is inverted if q_0 is in the basis state $|1\rangle$.*

Reversible circuits are a subset of quantum circuits that can be modeled in the classical domain. Hence, these circuits are used when designing Boolean components for quantum circuits and are usually composed of multiple-controlled Toffoli gates. These gates are composed of a (possibly empty) set of control qubits and a so-called target qubit. The value of the target qubit is inverted if, and only if, all control qubits are in basis state $|1\rangle$. Hence, the CNOT gate discussed above is a multiple-controlled Toffoli gate with a single control.

TABLE I: Truth table of the half adder function

(a) Before embedding

x_1	x_2	y_1	y_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) After embedding

a	x_1	x_2	g	y_1	y_0
0	**0**	**0**	0	**0**	**0**
0	**0**	**1**	1	**0**	**1**
0	**1**	**0**	0	**0**	**1**
0	**1**	**1**	1	**1**	**0**
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

Example 2. *Consider the reversible circuit shown in Fig. 1b that is composed of three qubits and four gates. The target qubit of a Toffoli gate is again denoted by $\oplus$, whereas control qubits are denoted by $\bullet$. Additionally, we have labeled the intermediate values of the qubits throughout the circuit when applying $|q_0 q_1 q_2\rangle = |000\rangle$ as input. Since a reversible circuit can be modeled in the classical domain (no quantum effects are exploited), we use 0 and 1 to indicate the basis states (rather than $|0\rangle$ and $|1\rangle$). The first gate has not control qubits and, thus, inverts the value of q_0 from 0 to 1. The second gate is controlled by q_0. Since the value of q_0 is 1, the value of q_1 is inverted from 0 to 1. The second gate does not affect the state of the qubits, since the control qubit q_2 is set to 0. Eventually, the last gate inverts the state of q_2 from 0 to 1.*

B. Boolean Components in Quantum Circuits

Typically, quantum circuits contain large Boolean components (also called *oracles*) which can be realized by *reversible circuits*. Decompositions of the gates occurring in reversible circuits into elementary quantum operations (e.g., into the well-known *Clifford+T* library [19]) can be determined using approaches such as [20].

Since Boolean components occurring in quantum circuits commonly describe very complex functionality, they are usually split into several non-reversible parts (e.g., the modular exponentiation in Shor's algorithms can be build up from several adders)—either manually [10]–[12] or by automated synthesis tools (using methods as reviewed, e.g., in [8], [9]). But since quantum computations are inherently reversible, it has to be ensured that these sub-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.

Example 3. *Consider the truth table of a half adder shown in Table Ia and assume that this functionality shall be realized as a sub-function of an oracle. Since the output pattern 01 occurs twice, the function is not reversible—the input cannot be determined uniquely having the output only.*

To ensure a unique input-output mapping, the non-reversible function to be realized is *embedded* into a reversible one that typically has a much larger number of variables.[1] This embedding process can either be conducted explicitly [13], [14] (required when using synthesis approaches such as [21], [22]) or implicitly (using synthesis schemes following one-pass synthesis as employed in [15], [16]). However, conducting

[1]Note that each variable of the function is realized by means of a qubit in the quantum circuit.

the embedding often yields circuits where the number of additional variables and, hence, qubits is significant. Since qubits are a limited resource (especially in NISQ devices [5]) their number shall be kept as small as possible. But even following the state of the art reviewed above, still a rather substantial number of qubits results. In fact, the minimal number of qubits required for embedding thus far is defined as follows:

Definition 1. *Consider a Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ with output patterns $p_1, p_2, \ldots, p_k \in \mathbb{B}^m$ ordered by the number of corresponding input patterns (in the following denoted as $\mu(p_i) = |\{x \in \mathbb{B}^n \mid f(x) = p_i\}|$). Since the embedding process has to make all output patterns distinguishable, at least $\lceil \log_2 \mu(p_1) \rceil$ additional so-called garbage outputs are required (where p_1 is the most frequently occurring output pattern). Moreover, since the number of inputs and outputs has to be equal to realize a reversible function as quantum circuit, a total of $\min(n, m + \lceil \log_2 \mu(p_1) \rceil)$ qubits are required to embed a function $f\colon \mathbb{B}^n \to \mathbb{B}^m$. If this implies to add further inputs, the desired output is obtained when setting all ancillary inputs to a specific value (usually 0).*

Example 3 (continued)**.** *Since the most frequently occurring output pattern $p_1 = 01$ occurs twice, $\lceil \log_2 2 \rceil = 1$ garbage output is required to make this output pattern distinguishable. To align the number of inputs with the number of outputs, one ancillary input is required. Table Ib shows one possible embedding of the half adder function. The desired function can be obtained at the primary outputs by setting the ancillary input a to 0 (highlighted in bold). All garbage variables as well as the primary outputs when $a \neq 0$ can be chosen arbitrarily as long as a reversible function results.*

The ancillary qubits of all sub-functions of an oracle have to be de-computed to their initial state to allow for a correct execution within the oracle and enable a later reuse.

III. One Ancillary Qubit is Enough

The authors of [16], [17] have shown that it is possible to undercut the theoretical lower bound on the number of required qubits (discussed in Section II-B) by using coding techniques, i.e., by using a 1-to-1 mapping of the output patterns to others. In this section, we first review the main idea of this approach and then formally prove that, using a variable-length encoding, at most one ancillary qubit is enough to realize any desired non-reversible function—and, by this, any sub-component of an oracle. Afterwards, in Section IV, it is experimentally confirmed that this indeed allows one to significantly reduce the number of overall required qubits (even below the minimum considered thus far) in many cases.

A. Utilizing Coding

As shown in [16], [17], the number of additionally required output patterns can be significantly reduced by exploiting coding techniques. The general idea for using coding is motivated by the fact that usually not all output patterns occur equally many times and, thus, do not require the same number of garbage outputs. Hence, a variable-length encoding can be utilized, where frequently occurring output patterns are represented by a short code word (together with a large number of garbage outputs) and rarely occurring output patterns are represented by a longer code word (together with a smaller number of garbage outputs).

TABLE II: Encoding a non-reversible function

(a) Orig. function

x_3	x_2	x_1	x'_3	x'_2	x'_1
0	0	0	1	1	0
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	1	1	0

(b) Encoding

i	p_i	$\mu(p_i)$	$code(p_i)$
1	110	4	0 - -
2	000	2	1 0 -
3	100	1	1 1 0
4	111	1	1 1 1

(c) Encoded function

x_3	x_2	x_1	x'_3	x'_2	x'_1
0	0	0	0	-	-
0	0	1	1	0	-
0	1	0	0	-	-
0	1	1	1	1	0
1	0	0	1	0	-
1	0	1	1	1	1
1	1	0	0	-	-
1	1	1	0	-	-

Example 4. *Consider the Boolean function with $n = 3$ inputs and $m = 3$ outputs shown in Table IIa. Using an embedding scheme as discussed in Section II-B yields a reversible function with five variables (thus, requiring five qubits). However, using the code as shown in Table IIb allows one to reduce the number of required qubits to three. For example, the most frequently occurring output pattern $p_1 = 110$ (which requires $\lceil \log_2 4 \rceil = 2$ garbage outputs) is encoded as $code(p_1) = 0$, while the output pattern $p_3 = 100$ is encoded by $code(p_3) = 110$. The number of variables/qubits required for each output pattern is then determined by the sum of the code length and the number of required garbage outputs—resulting in the encoded function shown in Table IIc (dashes indicate garbage variables).*

To generate a code as shown above, a *Pseudo-Huffman encoding* is employed. To this end, one starts with terminal nodes—one for each output pattern with $\mu(p_i) > 0$ (no code has to be assigned to output patterns that do not occur)—and attaches a weight representing the number of required garbage outputs (i.e., $\lceil \log_2 \mu(p_i) \rceil$). The *Pseudo-Huffman tree* is then generated by repeatedly combining the two nodes a and b with the smallest attached weights $w(a)$ and $w(b)$ to a new node c with attached weight $w(c) = \max(w(a), w(b)) + 1$ until a single node results. The weight of such a node $w(c)$ then gives the number of outputs required to represent all combined output patterns uniquely, i.e., one additional variable is required (aside from $\max(w(a), w(b))$) to distinguish between a and b. Hence, the weight of the root node determines the number of overall required outputs in the encoded function. Building the *Pseudo-Huffman tree* inherently gives such a variable-length encoding of the output patterns by, e.g., assigning 0 (1) to the left (right) successor of each node. Concatenation of the values attached to the path from the root node to a terminal representing an output pattern p_i determines $code(p_i)$.

Example 5. *Figure 2 shows the Pseudo-Huffman tree for the function shown in Table IIa. Since there exist four output patterns with $\mu(p_i) > 0$, we start with four terminal nodes (labeled v_1, v_2, v_3, and v_4, respectively) and attach the number of required garbage outputs as weights (drawn as numbers inside the nodes). First, we combine the nodes v_3 and v_4 to a new node v_5 with weight $w(v_5) = \max(0, 0) + 1 = 1$. Next, we combine the nodes v_2 and v_5 to a new node v_6 with weight $w(v_6) = \max(1, 1) + 1 = 2$. Eventually, the*

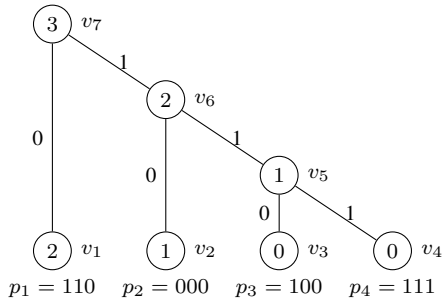
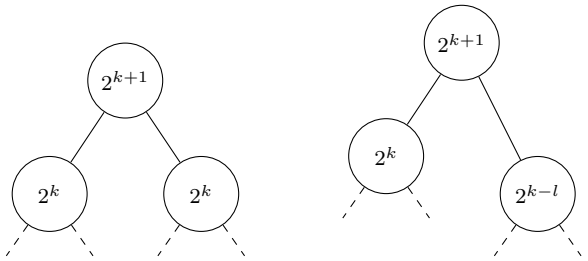


Fig. 2: Pseudo-Huffman tree for the function from Table IIa



(a) Joining nodes with equal weights

(b) Joining nodes with different weights

Fig. 3: Joining nodes in the construction of the PH-tree

nodes v_1 and v_6 are combined to a node v_7 with weight $w(v_7) = \max(2,2)+1 = 3$—the single root node of the tree. The code for the individual output patterns is then determined by the path from the root node to the respective terminal. For example, output pattern $p_2 = 000$ is encoded by $code(p_2) = 10$ since the path traverses the right edge of node v_7 and the left edge of node v_6. Overall, the code shown in Table IIb results.

B. Proving an Upper Bound of $n+1$ Qubits

In this section, we prove that encoding the output patterns of an n-input function as shown above results in a coded function requiring at most $n+1$ variables. Moreover, we show precisely in which cases this additional qubit is required and in which not. To this end, we first formally define the Pseudo-Huffman tree utilized to determine the encoding.

Definition 2 (Pseudo-Huffman Tree)**.** *Let $G = (V, E)$ be a connected, arborescence (a directed rooted tree) composed of a set of nodes $V = \{v_1, v_2, \ldots, v_{|V|}\}$ and a set of edges $E \subset V \times V$, and let $w\colon V \to \mathbb{N}_0$ be a labeling of the graph nodes in terms of non-negative weights. Moreover, let $T = \{t \in V \mid \forall v \in V\colon (t,v) \notin E\} \subseteq V$ denote the set of all terminal nodes. Then, $PH = (G, w)$ is called a* Pseudo-Huffman tree*, if, and only if,*

1) *each internal node $v \in V \setminus T$ has exactly two children $a, b \in V$ and $w(v) = \max(w(a), w(b)) + 1$ and*
2) *for any two different internal nodes $v_1, v_2 \in V \setminus T$ with children a_1, b_1 and a_2, b_2, respectively, it holds that $w(a_1) \leq w(b_1)$ implies*

$$\big(w(a_2), w(b_2) \leq w(a_1)\big) \vee \big(w(a_2), w(b_2) \geq w(b_1)\big).$$

In other words, the tree can be formed from the terminal nodes by successively combining nodes with the lowest available weights as described in Section III-A.

The following theorem yields a condition on the terminal nodes of a Pseudo-Huffman tree that is sufficient to restrict the weight of the tree's root node.

Theorem 1. *Let $PH = ((V, E), w)$ be a Pseudo-Huffman tree. If there exists an assignment s_v for each terminal node $v \in T = \{t \in V \mid \forall v \in V\colon (t,v) \notin E\}$ such that $2^{w(v)} \geq s_v > 2^{w(v)-1}$ (where $w(v)$ denotes the weight of node v) and $\sum_{v \in T} s_v = 2^n$, then the weight $w(v_r)$ of the root node v_r of the tree is either n or $n+1$.*

Proof. Replace all weights using the rule $w \mapsto 2^w$. Then the rule for computing the weight of a new node changes from $\max(w(a), w(b)) + 1$ to $2 \cdot \max(w(a), w(b))$. Accordingly, all weights in the tree will be a power of 2.

We perform the proof by arguing about the weights of the nodes when constructing a Pseudo-Huffman tree. To this end, consider the set of all nodes V_r^i of the tree-under-construction that are the root nodes of the already connected components after step i of the algorithm. Let $w_{total}^i = \sum_{v \in V_r^i} w(v)$ denote the sum of the weights over all these nodes.

At each step i of the algorithm, two nodes $a, b \in V_r^i$ with minimal weight are chosen and joined to a new node c such that $V_r^{i+1} = \{c\} \cup V_r^i \setminus \{a, b\}$. There are two cases:

1) both nodes a and b have the same weight 2^k. Then, they are replaced by a node with weight 2^{k+1} such that $w_{total}^{i+1} = w_{total}^i$ (see Fig. 3a), i.e., the sum of the weights over the root nodes remains constant.
2) one node—assume without loss of generality a—has weight $w(a) = 2^k$ and the other node (b) has weight $w(b) = 2^{k-l}$ for some $k \geq l > 0$. Then, they are replaced by a node c with weight $w(c) = 2^{k+1}$ (see Fig. 3b).
   Since we always take the nodes with minimal weight, there might not be any other node $d \in V_r^i$ with $w(d) < 2^k$ as this node would have a higher priority to be joined with b. Thus, all nodes in V_r^i aside from b have a weight that—by construction—is a power of 2 that is greater than or equal to 2^k. Consequently, after joining a and b, w_{total}^i is increased to a number w_{total}^{i+1} that is divisible by 2^k. More precisely, it is increased by

$$\begin{aligned} w(c) - w(a) - w(b) &= 2^{k+1} - 2^k - 2^{k-l} \\ &= 2^k - 2^{k-l} \\ &< 2^k, \end{aligned}$$

   such that w_{total}^{i+1} is the *smallest* number that is greater than w_{total}^i and divisible by 2^k.
   Clearly, this case happens at most once for each $k > 0$, since afterwards there is no more node in V_r^{i+1} with a weight less than 2^k and all nodes that will be added to V_r^j (for $j > i$) have higher weights.

By the assumption of Theorem 1, we initially have $2^n = \sum_{v \in T} s_v \leq w^0_{total}$ and $w^0_{total} < 2^{n+1}$. Thus, we will at some point denoted $final$ reach the case that all nodes in V_r^{final} have a weight greater than or equal to 2^n such that w^{final}_{total} is divisible by 2^n. Since 2^{n+1} is divisible by all potencies 2^k for $k = 0, \ldots, n$, w^{final}_{total} will never exceed 2^{n+1}, as we are always increasing w^i_{total} to the smallest larger number divisible by 2^k for a $k \in \{1, \ldots, n\}$. Consequently, we have at least one and at most two nodes in V_r^{final} with a weight of 2^n. Thus, the root node of the resulting tree either is the single node with weight 2^n or the single node with weight 2^{n+1} constructed from the two nodes with weight 2^n. Hence, the root node of the original Pseudo-Huffman tree has weight n or $n+1$ as desired. □

Now let us interpret this result in the setting of coded Boolean functions. Consider a Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ to be encoded. We can construct a Pseudo-Huffman tree with $|T| = |\{p_i \in \mathbb{B}^m \mid \mu(p_i) > 0\}|$ terminal nodes (which is always possible), where each terminal node $v \in T$ uniquely corresponds to one output pattern p_i and has assigned $s_v = \mu(p_i)$ (thus, having a weight $w(v) = \lceil \log_2 \mu(p_i) \rceil$). As this assignment clearly satisfies the conditions of Theorem 1, the height of this tree is either n or $n+1$. Hence, there exists a coding (which is inherently given by the constructed tree) that requires at most one additional qubit when realizing f in quantum logic.

Moreover, we can precisely determine in which cases this additional qubit is required. In fact, the additional qubit is required whenever there exists an output pattern p_i where $\mu(p_i) > 0$ is not a power of two.

Corollary 1. *The root node of a Pseudo-Huffman tree satisfying the same assumptions as in Theorem 1 has weight n if, and only if, $\sum_{v \in T} 2^{w(v)} = 2^n$.*

Proof. Given that $\sum_{v \in T} 2^{w(v)} = 2^n$ we may apply Theorem 1 by using the assignment $s_v = 2^{w(v)}$ for all $v \in T$. Following the argumentation in the proof of Theorem 1, the root node of the Pseudo-Huffman tree has weight n if w^i_{total} does not exceed 2^n at any time, i.e., if the second case (which increases w^i_{total}) does not occur at all. This is clearly the case if $w^0_{total} = 2^n$ in the beginning. Conversely, if $\sum_{v \in T} 2^{w(v)} \neq 2^n$, we have $w^0_{total} > 2^n$ in the beginning, such that $w^{final}_{total} = 2^{n+1}$ in the end. □

IV. Comparison to Embeddings without Coding

In this section, we compare the idea of coded embeddings to previous approaches and discuss their effect on the design of quantum oracles.

A. Evaluation

We compare the idea of coded embeddings to approaches that do not consider coding when realizing a Boolean function $f \colon \mathbb{B}^n \to \mathbb{B}^m$ in quantum logic. More precisely, we compare to exact methods utilizing $\max(n, m + \lceil \log_2 \mu(p_1) \rceil)$ qubits [13], [14] as well as to heuristic ones that always utilize a Bennett embedding with $n+m$ qubits [23], [24] (e.g., generated when using an ESoP based synthesis approach [25]).

TABLE III: Number of required qubits

Benchmark			Embedding		
name	n	m	Bennett [23]	Min. [13], [14]	Encoded
f51m_159	14	8	22	19	15
tial_214	14	8	22	19	15
cu_141	14	11	25	25	15
misex3_180	14	14	28	28	15
misex3c_181	14	14	28	28	15
table3_209	14	14	28	28	15
s1488_split	14	25	39	38	15
s1494_split	14	25	39	38	15
b12	15	9	24	22	16
in0_162	15	11	26	25	16
parity_188	16	1	17	16	16
ryy6_198	16	1	17	17	17
t481_208	16	1	17	17	17
cmb_134	16	4	20	20	17
pcler8_190	16	5	21	21	17
cm163a_133	16	13	29	25	17
pdc_191	16	40	56	55	17
spla_202	16	46	62	61	17
table5	17	15	32	32	18
s298_split	17	20	37	29	18
s208.1_split	18	9	27	19	19
cm151a_129	19	9	28	27	20
cm150a_128	21	1	22	22	22
mux_185	21	1	22	22	22
duke2	22	29	51	50	23
cordic_138	23	2	25	25	24
cps_140	24	109	133	132	25
vg2	25	8	33	32	26
misex2	25	18	43	42	26
frg1_160	28	3	31	30	29
apex2_101	39	3	42	42	40
seq_201	41	35	76	75	42
apex1	45	45	90	89	46
apex3	54	50	104	103	55
e64_149	65	65	130	129	65

To this end, we have implemented the proposed idea in C++ and utilized the QMDD package [26] as well as the BDD package CUDD [27] to gain a compact representation of the considered functions—allowing us to determine the number of required qubits in negligible runtime. As benchmarks we use the functions from RevLib [28], as well as from the ISCAS [29] and IWLS [30] benchmark suites.[2]

Table III summarizes the obtained results. The first three columns of the benchmark as well as the number of inputs n and the number of outputs m. In the next three columns we list the number of required qubits when using Bennett embedding (i.e., $m+n$), when using a minimal encoding without considering coding (i.e., $\max(n, m + \lceil \log_2 \mu(p_1) \rceil)$), and when using coded embeddings as described in this work (i.e., n or $n+1$), respectively.

As can be seen in Table III, the number or required qubits can significantly be reduced when considering coded embeddings—especially in cases where $m > n$. Consider for example benchmarks *cps_140* and *e64_149*, where the number of required qubits can be reduced by 107 and 65, respectively, using coding techniques. Overall, a possible reduction of 36.4% can be observed on average.

[2]Note that we only consider non-reversible functions from these benchmarks suits since reversible ones do not require embedding.

B. Discussion

Concerning the design of quantum oracles, coded embeddings as proposed above can be exploited in two different ways:

- On the one hand, one can apply the coding technique *locally* on each and every sub-component and use decoders (after each sub-component) to translate the encoded results to the original ones which are then used as inputs of the subsequent components. This essentially reduces the complexity of synthesis for the individual sub-components (since a smaller number of qubits needs to be considered). While this offers a significant improvement of synthesis run-time (as also observed in [16]), the total number of additional qubits does not change (due to the decoders).
- On the other hand, one can apply the coding technique *globally* such that the encoded outputs of one sub-component are directly used as input for subsequent components and a single decoder at the end translates the final results to the desired ones. This approach significantly reduces the number of extra qubits required during the computation of the oracle's sub-components such that the total number of extra qubits is likely to stay close to the theoretical minimum given by the oracle's overall functionality (which is zero). On the downside, a redesign of the sub-components might be required in order to work with encoded values.

V. Conclusions

In this work, we have proven that one additional qubit is enough to determine a coded embedding of any non-reversible function for quantum circuits. By this, one can significantly reduce the overall number of qubits required for realizing Boolean oracles, since their functionality is usually split into several non-reversible parts. Our experimental evaluation shows that the number of required qubits can indeed be reduced by 36.4% on average, when comparing to embeddings that do not utilize encoding and that have been considered as the minimum thus far. Possible applications on the design of oracles for quantum circuits are discussed.

Acknowledgements

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

References

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

[2] Rachel Courtland. Google aims for quantum computing supremacy. *IEEE Spectrum*, 54(6):9–10, 2017.

[3] Lee Gomes. Quantum computing: Both here and not here. *IEEE Spectrum April 2018*.

[4] Jeremy Hsu. CES 2018: Intel's 49-qubit chip shoots for quantum supremacy. *IEEE Spectrum Tech Talk*, 2018.

[5] John Preskill. Quantum computing in the NISQ era and beyond. *arXiv preprint arXiv:1801.00862*, 2018.

[6] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Foundations of Computer Science*, pages 124–134, 1994.

[7] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Theory of computing*, pages 212–219, 1996.

[8] Rolf Drechsler and Robert Wille. From truth tables to programming languages: Progress in the design of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 78–85, 2011.

[9] Mehdi Saeedi and Igor L. Markov. Synthesis and optimization of reversible circuits - a survey. *ACM Comput. Surv.*, 45(2):21, 2013.

[10] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n+2 qubits with Toffoli based modular multiplication. *Quantum Information & Computation*, 17(7&8):673–684, 2017.

[11] Stéphane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Information & Computation*, 3(2):175–185, 2003.

[12] Thomas Haener, Mathias Soeken, Martin Roetteler, and Krysta M Svore. Quantum circuits for floating-point arithmetic. In *International Conference on Reversible Computation*, pages 162–174. Springer, 2018.

[13] Mathias Soeken, Robert Wille, Oliver Keszocze, D. Michael Miller, and Rolf Drechsler. Embedding of large Boolean functions for reversible logic. *J. Emerg. Technol. Comput. Syst.*, 12(4):41:1–41:26, December 2015.

[14] Alwin Zulehner and Robert Wille. Make it reversible: Efficient embedding of non-reversible functions. In *Design, Automation and Test in Europe*, pages 458–463, 2017.

[15] Alwin Zulehner and Robert Wille. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 37(5):996–1008, 2018.

[16] Alwin Zulehner and Robert Wille. Exploiting coding techniques for logic synthesis of reversible circuits. In *Asia and South Pacific Design Automation Conf.*, pages 670–675, 2018.

[17] Alwin Zulehner and Robert Wille. Pushing the number of qubits below the "minimum": Realizing compact boolean components for quantum logic. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1179–1182. IEEE, 2018.

[18] Alwin Zulehner, P. Mercy Nesa Rani, Kamalika Datta, Indranil Sengupta, and Robert Wille. Generalizing the concept of scalable reversible circuit synthesis for multiple-valued logic. In *Int'l Symp. on Multi-Valued Logic*, pages 115–120, 2018.

[19] P Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. A new universal and fault-tolerant quantum basis. *Information Processing Letters*, 75(3):101–107, 2000.

[20] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin 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.

[21] Mathias Soeken, Robert Wille, Christoph Hilken, Nils Przigoda, and Rolf Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *Asia and South Pacific Design Automation Conf.*, pages 85–92, 2012.

[22] Alwin Zulehner and Robert Wille. Improving synthesis of reversible circuits: Exploiting redundancies in paths and nodes of QMDDs. In *Int'l Conf. of Reversible Computation*, pages 232–247, 2017.

[23] C.H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, Nov 1973.

[24] 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.

[25] K. Fazel, M.A. Thornton, and J.E. 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.

[26] Philipp Niemann, Robert Wille, D. Michael Miller, Mitchell A. Thornton, and Rolf Drechsler. QMDDs: Efficient quantum function representation and manipulation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 35(1):86–99, 2016.

[27] Fabio Somenzi. CUDD: CU decision diagram package release 3.0. 0. 2015.

[28] 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.

[29] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Int'l Symp. Circuits and Systems (ISCAS 85)*, pages 677–692. IEEE Press, Piscataway, N.J., 1985.

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