ROS: Resource-constrained Oracle Synthesis for Quantum Computers

Giulia Meuli
EPFL
Lausanne, Switzerland
giulia.meuli@epfl.ch

Mathias Soeken
Microsoft Quantum
Redmond, United States

Martin Roetteler
Microsoft Quantum
Redmond, United States

Giovanni De Micheli
EPFL
Lausanne, Switzerland

We present a completely automatic synthesis framework for oracle functions—a central part in many quantum algorithms. The proposed framework for resource-constrained oracle synthesis (*ROS*) is a LUT-based hierarchical method in which every step is specifically tailored to address hardware resource constraints. *ROS* embeds a LUT mapper designed to simplify the successive synthesis steps, costing each LUT according to the resources used by its corresponding quantum circuit. In addition, the framework exploits a SAT-based quantum garbage management technique. Those two characteristics give *ROS* the ability to beat the state-of-the-art hierarchical method both in number of qubits and in number of operations. The efficiency of the framework is demonstrated by synthesizing quantum oracles for Grover's algorithm.

1 Introduction

Practical quantum computers are nowadays a realistic prospect thanks to advances in fabrication technology and the effort of the research community to revolutionize computing [6, 17, 9]. Quantum systems enable computation over superposition of states and are based on physical phenomena that are fundamentally different from the ones exploited in classical computing systems. For this reason, they require the development of dedicated logic synthesis tools.

The peculiarities of quantum computation can be exploited to solve problems that cannot be solved with standard computers in a reasonable time, by running innovative quantum algorithms. The possible applications span among others factorization [20], quantum chemistry [3], and satisfiability solving [8].

Many quantum algorithms include combinational logic operations. The large amount of resources necessary to perform such computations can overcome the resources available, hence preventing some algorithms to be computed on a constrained quantum hardware. Consequently, there is a large interest in finding synthesis methods that minimize the impact of combinational logic on the cost of quantum algorithms.

Some automatic quantum circuit synthesis methods have been proposed [13, 7, 1, 19], which can be applied to relatively small logic designs. Hierarchical methods proved to be applicable to larger designs, as they are based on multi-level logic representations [18]. Among them, the LUT-based hierarchical reversible logic synthesis (*LHRS*) framework has been proposed in [21], and is currently part of the open source project *RevKit*.[1] It exploits classical logic synthesis methods to create quantum circuits of any given combinational logic component. Objective functions of the synthesis are: the number of qubits and the number of operations required to perform the target function.

[1] https://github.com/msoeken/revkit

Bob Coecke and Mathew Leifer (Eds.):
Quantum Physics and Logic 2019 (QPL)
EPTCS 318, 2020, pp. 119–130, doi:10.4204/EPTCS.318.8

LHRS uses LUT mapping to decompose the target function. The decomposition step is a crucial phase in this hierarchical method, as it is the starting point of the synthesis process. For this reason, it is of paramount importance to control its behavior. The k-LUT mapping technique that is used in *LHRS* originates from the open source logic synthesis tool *abc* [5] and has been originally designed for the synthesis of classical circuits.

In this work, we develop an alternative hierarchical framework: *Resource-constrained Oracle Synthesis* (ROS). It embeds a new quantum-aware LUT-mapper, that is specifically designed for the application into a hierarchical synthesis framework. Classical LUT-mappers, like the one used by *LHRS*, aim at minimizing area and delay, but none of them have an immediate direct counterpart in quantum circuit synthesis. Instead, our mapper is designed to minimize metrics that make each LUT easier to be synthesized into a quantum circuit.

We show that the hierarchical flow that integrates our new mapper achieves a consistent reduction in the number of gates of the final circuits. We also integrate in the flow a method for quantum garbage management that has been proposed in [11]. This method enables to efficiently uncompute intermediate results, giving control on the number of extra qubits (ancillae) of the circuit.

We show that our approach can effectively improve the state-of-the-art both in number of qubits and in number of operations. Rather than providing a method that generates additional Pareto optimal synthesis results, our approach systematically beats existing ones by improving the qubit count while not increasing the gate count—and vice versa—by improving the gate count while not increasing the qubit count. We apply the ROS flow to synthesize quantum oracles, which could be applied in algorithms such as Grover's search, and compare our results with the state-of-the-art hierarchical method (*LHRS*) showing improved results.

2 Preliminaries

2.1 Quantum circuits

Quantum computing processes qubits. A qubit can be in one of the "classical" logic states, 0 and 1, or in any superposition of these states. The state of a qubit q can be defined by the linear combination of the classical states using two complex coefficients, $q = a_0|0\rangle + a_1|1\rangle$, with $a_0, a_1 \in \mathbb{C}$ and $|a_0|^2 + |a_1|^2 = 1$.

The Block sphere is a powerful representation of a qubit state. The two poles of the sphere represent the two classical states, while all the points of the sphere represent superposed states. On the equator of the block sphere there are all superposed states with $|a_0|^2 = |a_1|^2 = 1/2$ characterized by different angles with respect to the Z-axis.

A 2-qubit system can be defined as: $q = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$, with $a_{00}, a_{01}, a_{10}, a_{11} \in \mathbb{C}$ and $|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$. As a consequence, 4 complex coefficients are needed to represent a two-qubit state, while 8 complex coefficients are necessary to describe a 3-qubit system. In general, to represent the state of n qubits and to simulate the quantum system behavior on a classical computer, 2^n complex coefficients are required.

While modeling a combinational functionality for the use in a quantum computation, it is possible to consider all the inputs as Boolean values—even when embedded as part of a quantum algorithm where entangled states in superposition are being applied.

The state of a qubit can be modified by applying quantum operations. All possible operations are reversible and can be represented by unitary matrices. Both single-qubit operations and 2-qubit operations are available, the latter changing the state of a qubit according to the state of a second one.

There are different universal sets of quantum operations, targeting different technologies. In this work, we refer to the set that consists of the following operations: Controlled-NOT (CNOT), Hadamard (H) and rotations of an arbitrary angle θ over the Z-axis of the Block sphere ($R_z(\theta)$). All quantum operations can be represented by unitary matrices of dimension $2^n \times 2^n$, where n is the number of qubits affected by the operations. For the selected universal set, the representative matrices are:

$$\text{CNOT} = \begin{pmatrix} 1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0 \end{pmatrix},\ H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1&1\\1&-1 \end{pmatrix},\ R_z(\theta) = \begin{pmatrix} e^{\frac{-i\theta}{2}} & 0\\ 0 & e^{\frac{i\theta}{2}} \end{pmatrix}$$

A quantum oracle is defined as a "black box" operation performing a multi-output Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$. The effect of an oracle O performing the operation f over two registers, one of n qubits to store the inputs, $|x\rangle$, and one of m qubits to store the outputs, $|y\rangle$, can be described as follows:

$$O(|x\rangle \otimes |y\rangle) \mapsto |x\rangle \otimes |y \oplus f(x)\rangle$$

The cost of a quantum circuit depends on the number of qubits required for the computation, and the number of operations that are performed. Automatic tools can be used to take into account technology constraints by synthesizing low cost quantum circuits.

2.2 Rademacher-Walsh spectrum

We call a function $f : \mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{0,1\}$, a Boolean function over n variables. A Boolean function can be represented by its truth table in the $\{1,-1\}$ encoding, which is a bitstring $b_{2^n-1}b_{2^n-2}\ldots b_0$ of size 2^n where

$$b_x = (-1)^{f(x_1,\ldots,x_n)} \qquad \text{when } x = (x_1x_2\ldots x_n)_2$$

The Hadamard transform matrix over n variables is defined as:

$$H_n = \begin{pmatrix} H_{n-1} & H_{n-1}\\ H_{n-1} & -H_{n-1} \end{pmatrix}, \quad H_0 = 1$$

Each row of the Hadamard transform matrix is equal to the truth table of the parity function between a subset of the n variables. For example the last row of an n-variable Hadamard matrix will be the truth table of the parity function $p = x_1 \oplus x_2 \oplus \cdots \oplus x_n$.

The Rademacher-Walsh spectrum S of the function f expressed as a truth table in the $\{1,-1\}$ encoding F is defined as:

$$S = H_nF$$

Each coefficient of the spectrum represents the correlation with a parity function of a subset of the inputs.

Example 1 *Given the 3-input majority Boolean function $f(x_1,x_2,x_3) = \langle x_1x_2x_3\rangle$, its truth table is:*

$$F = \begin{pmatrix} 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \end{pmatrix}$$

The Rademacher-Walsh spectrum of f is:

$$S = H_3F = \begin{pmatrix} 0 & 4 & 4 & 0 & 4 & 0 & 0 & -4 \end{pmatrix}$$

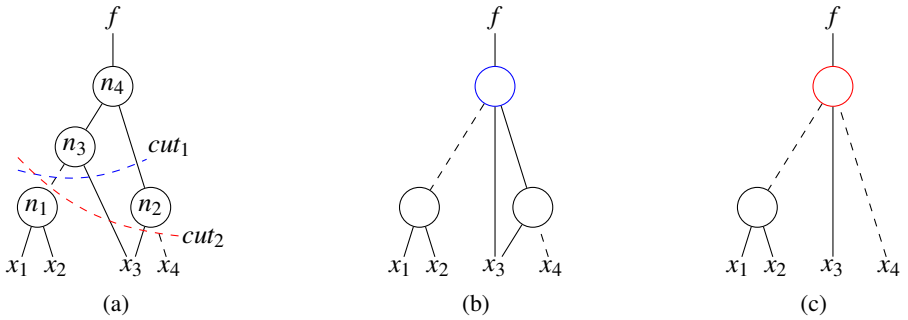


Figure 1: (a) an AIG graph performing the function $f = (\overline{x}_1 + \overline{x}_2)x_3\overline{x}_4$ with two possible 3-feasible cuts; (b) 3-LUT network generated by cut_1; (c) 3-LUT network generated by cut_2.

We later make use of the fact that one can derive a quantum gate implementation for a Boolean function from the function's spectral coefficients [1, 19].

2.3 k-LUT mapping

Lookup table (LUT) mapping is a decomposition method that has originally been used to map a logic design into components of FPGAs (Field Programmable Gate Array) capable of computing any Boolean function up to a given number of inputs, i.e., lookup tables. Later, LUT mappers found a successful application in logic synthesis and circuits optimization [14], as they allow to decompose large functionality into smaller functions. Several efficient state-of-the-art mappers are available and they are traditionally designed to minimize delay and area of the resulting circuit.

The input of the LUT mapping is a multi-level logic network representing a Boolean function. A multi-level logic network is represented by a graph, in which each node performs a Boolean operation and edges define data dependencies. The inputs of the function are represented by the primary inputs of the network. According to the characteristics of the network, we distinguish different graph representations. In And-Inverter Graphs (AIG), nodes perform the 2-input AND Boolean function, while edges can be complemented to perform inversion. A different network representation, called Xor-And-inverter Graphs (XAG) implements the 2-input XOR in addition to the 2-input AND operation. For example, a logic network representing the function $f = (\overline{x}_1 + \overline{x}_2)x_3\overline{x}_4$ is shown in Fig. 1(a). All nodes of this network perform the 2-input AND operation between the node's inputs, dashed edges represent Boolean inversion, and x_1, x_2, x_3 and x_4 are the primary inputs.

The k-LUT mapper decomposes the multi-level network using k-feasible cuts. A cut for a node n is a set of leaves $l_1, \ldots, l_n$ such that each path from n to a primary input includes one of the leaves. A k-feasible cut is a cut that has at most k leaves. Leaves are nodes or primary inputs of the network. In Fig. 1(a) two 3-feasible cuts are shown for the node n_4. The fist cut has leaves n_1, x_3 and n_2, while the second cut has leaves n_1, x_3 and x_4. Fig. 1(b) and (c) show the 3-LUT networks generated by the first and the second cut, respectively. In the first graph, the node highlighted in blue is a LUT with 3 inputs that performs the combined operations of nodes n_4 and n_3, while in the second graph, the LUT highlighted in red performs the combined operations of three nodes. Comparing the two networks, generated by two different cuts, it is clear how the choices made during the mapping process affect the number of nodes of the resulting LUT network and the complexity of the function performed by each LUT.

2.4 k-LUT based hierarchical reversible synthesis

Automatic quantum circuit synthesis methods transform a high level Boolean function representation into a quantum circuit, exploiting reversible circuits as intermediate representations. These circuits are composed by single target gates, that are generalizations of the multiple-controlled Toffoli gates. A single-target gate $T_f(\{x_1,\ldots,x_n\},x_k)$ is characterized by: a set of controls $x_1,\ldots,x_n$, a control function $f:\mathbb{B}^n\to\mathbb{B}$ and a target x_k. The value of x_k is complemented if the function $f(x_1,\ldots,x_n)$ evaluates to one. Efficient methods are known for the decomposition of these reversible gates into quantum operations [2, 10, 12].

Hierarchical methods for the synthesis of quantum circuits have shown the ability to synthesize large functions and enable to explore the trade-off between number of operations and number of qubits. Hierarchical means that the method starts from a multi-level representation of the function, i.e., a graph. Inputs to the function are stored on a set of existing qubits. Additional qubits are used to store intermedite results computed by each node of the graph. Finally, the output results are available on some of the additional qubits. Among the hierarchical reversible logic synthesis methods, *LHRS* [21] exploits a state-of-the-art area-oriented LUT mapper, called *mf* that is part of the logic synthesis framework *abc* [5] to generate a LUT network that is used as starting representation for the synthesis flow.

The flow of *LHRS* is shown in Fig. 2(a). The first step of the flow, i.e., the k-LUT mapping, has a large impact on the final result, as it defines: (i) the number of required qubits, and (ii) the complexity of each sub-network. Fig. 3(a) show an input network that is transformed by the mapper into the 2-LUT network in Fig. 3(b). In the successive step, the k-LUT network is transformed into a reversible circuit, a network made of single-target gates (STG network). In this reversible representation, each line corresponds to a single qubit. This step is performed by exploiting the one-to-one correspondence between nodes of the k-LUT network and reversible single-target gates. It transforms the network in Fig. 3(b) into one of the STG networks in Fig. 3(c) and (d). The Gray synthesis method [1, 19] (see Fig. 2(a)) decomposes each single-target gate with control function f into a quantum circuit that consists of the following quantum operations: CNOT, H, $R_z(\theta)$. The method is characterized by a direct dependence between nonzero coefficients in the Hadamard-Walsh spectrum of the function f and number of CNOT and R_θ gates to be synthesized. This method is performed after the k-LUT mapping, this means that by modifying the mapping we get some control on the characteristics of the control functions in the LUT network, that are input to the Gray synthesis.

In *LHRS*, k-LUT mapping is performed considering metrics as delay and area, that have no immediate correlation in this application; so when k-LUT mapping is used in the context of quantum circuit synthesis, the classical metrics must be changed to context-related ones. In this work we address this criticality by integrating in ROS a new quantum-aware k-LUT mapper that aims at minimizing the number of gates required to synthesize the quantum circuit of each LUT using the Gray synthesis method.

2.5 Quantum memory management

Quantum circuits are required to be garbage free, that is, any intermediate result needs to be accessible from the outputs. Otherwise, as many states can be entangled together, measurement of intermediate results may compromise the computation. Recently, a method for quantum memory management has been proposed that is based on solving instances of the reversible pebbling game [11]. This technique

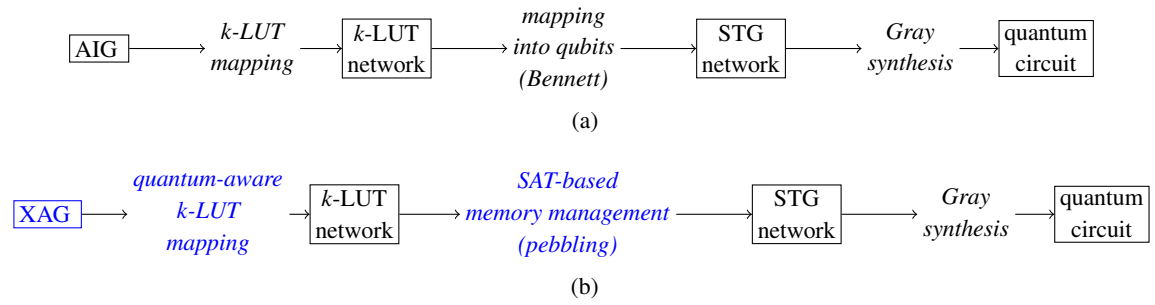


Figure 2: (a) state-of-the-art hierarchical synthesis framework; (b) proposed hierarchical synthesis framework.

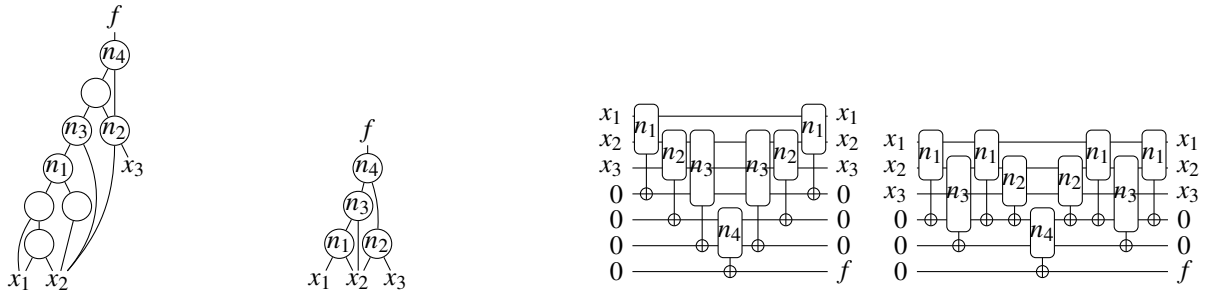


Figure 3: (a) a three-input multi-level logic network performing the Booelan function f; (b) an example of a 2-LUT network for f; (c) the reversible circuit for the 2-LUT network obtained using the Bennett clean-up strategy; (d) the reversible circuit for the 2-LUT network obtained using the quantum garbage management clean-up strategy [11].

can grant control on the number of qubits that are used in the clean-up process, exploiting a state-of-the-art SAT solver [16]. In [11], the authors show how the problem of uncomputing intermediate results corresponds to the reversible pebbling game.

Consider Fig. 3(c), here the k-LUT network is mapped into qubits, and each node is transformed in its corresponding reversible gate. The intermediate result is stored on a qubit that was initialized to $|0\rangle$. After the result is computed, all the intermediate values n_1, n_2, and n_3 must be uncomputed. This is done by performing the same operation twice. The order in which nodes are computed and uncomputed is a clean-up strategy, in Fig. 3(c) the strategy used is called Bennett strategy [4]. The SAT-based method described in [11] is capable of finding clean-up strategies that reduce the number of required qubits by computing and uncomputing the same reversible operation more than once. An example is shown in Fig. 3(d).

3 Resource-constrained Oracle synthesis

Even if quantum computing is promising to beat its classical counterpart in many applications, quantum device technology is still developing and is fairly new with respect to standard CMOS technology. With our tool we aim at providing the designer with the capability to tune the synthesis with respect to the

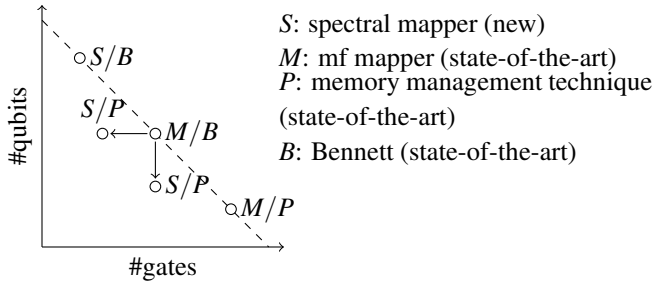


Figure 4: qualitative description of ROS's capability

available hardware.

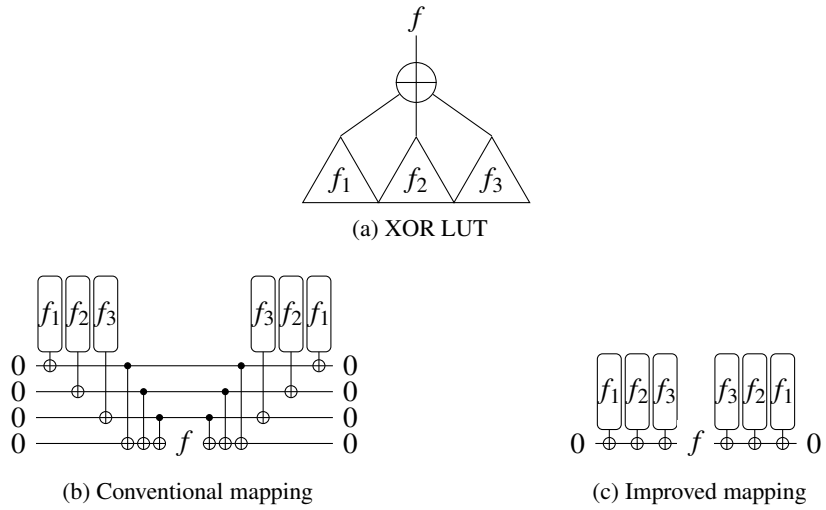
We propose ROS, a hierarchical synthesis framework built to leverage the quantum circuit cost, both in terms of number of qubits *and* number of gates. ROS's synthesis flow is shown in Fig. 2(b). It introduces two main contributions with respect to the state-of-the-art flow (see Fig. 2(a)).

- First, it embeds a new k-LUT mapper that is used to decompose the initial functionality into LUTs, in such a way to minimize the cost of each LUT. Such cost is defined according to the complexity of the LUT function to be synthesized by the Gray algorithm. If we analyze the result of this mapper against the state-of-the-art *mf* mapper, we get in general more LUTs, each one corresponding to fewer gates.
- Second, it exploits the quantum garbage management technique presented in [11] to control the number of qubits.

We claim that by using those two techniques together with the Gray synthesis method, we can beat the state-of-the-art results both in number of qubits and number of gates. Fig. 4 shows a qualitative description of the performance advantages we expect to obtain using ROS. In the plot, the state-of-the-art result is the one marked as M/B (corresponding to the *LHRS* synthesis framework). If we only apply the memory management technique in [11], but not the new mapper, we will obtain a circuit with fewer qubits and more gates, that correspond in the figure to M/P. If instead we embed into *LHRS* our quantum aware k-LUT mapper, but no quantum memory management, we can obtain a circuit with few gates but more qubits: S/B. Only by combining both techniques (S/P), we can beat the state-of-the-art tool in both qubits and gates. In fact, we can tune the approach to only improve qubit count by not increase gate count, or vice versa. This qualitative description is supported by the results in Section 5.

4 Quantum-aware k-LUT mapping

A main contribution of this work is to develop a k-LUT mapper designed to reduce the resources needed to synthesize the quantum circuit of each LUT. As explained in Section 2.4, LUT mapping is used to decompose a logic design that is too large to be synthesized into a quantum circuit by the existing methods. In this section, we describe how to perform this decomposition to facilitate the successive synthesis steps.



(a) XOR LUT

(b) Conventional mapping

(c) Improved mapping

Figure 5: Mapping LUTs that represent the XOR function

4.1 Cut enumeration and costing

An Xor-And-inverter Graph, or XAG, is input to the k-LUT mapping process. This is a graph where each node performs the XOR or the AND operation, and where edges can be complemented to perform inversion. The choice of this particular data structure is not accidental, but reflects the fact that it is relatively cheap to perform the XOR operation in fault tolerant quantum circuits. Only one CNOT gate is needed to perform the XOR between two qubits, and in general $m-1$ CNOT gates are needed to perform the CNOT of an m-input XOR gate. If the result should be stored on a free ancilla line, m CNOT gates are needed in total.

Once the input is defined, we first perform cut enumeration. This step consists of enumerating all possible cuts for each node of the input graph, traversing the graph from the bottom to the top. Only the best cuts are stored for each node, a technique called "priority cuts" to reduce the memory requirement of cut enumeration [15]. During cut enumeration each node is assigned to a set of p cuts that are ordered following a user defined cost criteria. In our case, as we aim at integrating the mapper into a quantum synthesis framework, we use as cost function the number of nonzero spectral coefficients of the function performed by the selected cut. As pointed out in the preliminaries, the Gray synthesis method generates smaller quantum circuits when the input function presents a spectrum with many zeros. At the end of the cut enumeration step, each node of the XAG is assigned with an ordered set of p cuts, with the order criteria defined by the spectrum of the cut's function.

4.2 XOR-block matching

We chose XAG graphs as logic representation, due to the inexpensive implementation of the XOR operation in fault tolerant quantum circuits. Following this idea we modify the mapping algorithm to identify and select cuts that performs the parity function. They can be synthesized as multiple-input XOR gates.

After we perform cut enumeration we refine the cuts, looking for multi-input XOR blocks. In fact, if all leaves of the cuts have a fan-out size of 1, i.e., they only fan-in into the XOR gate, we can apply an

	M/B		S/B		S/P_match_q		S/P_match_g		M/P	
	gates	qubits	gates	qubits	gates	qubits	gates	qubits	gates	qubits
addassoc4	1376	25	1029	34	1141	25	1371	22	1904	19
addassoc5	2987	36	1586	49	1798	36	1804	31	5365	24
addassoc6	2394	43	1445	58	1513	42	1729	35	8268	26
addassoc7	3243	51	1941	70	2201	50	2361	44	4383	36
addassoc8	3221	62	2018	79	2312	57	2430	49	4787	40
addassoc9	3603	70	2385	89	2453	67	2773	56	5569	42
addassoc10	4528	80	2835	97	3575	70	3549	58	6142	50
multassoc4	6682	34	2751	60	3057	34	3193	33	10834	19
multassoc5	10519	54	4811	104	5321	55	5321	55	16687	31
multassoc6	17653	93	7395	172	8565	96	8565	96	22933	53
multassoc7	25395	138	11099	240	15425	135	14607	128	37717	74
multassoc8	32443	181	13781	323	20713	179	22997	166	51757	94
multassoc9	37599	212	17881	394	34305	203	32489	200	66267	110
multassoc10	47795	289	22843	525	41825	281	41081	262	101627	143
multdistr4	4812	29	2368	54	3262	29	3694	25	5034	19
multdistr5	9011	54	4569	94	5441	54	5441	54	22717	25
multdistr6	13327	78	6092	143	7138	80	7138	80	15169	46
multdistr7	18268	110	8771	200	13849	109	13849	109	21746	63
multdistr8	26151	149	11888	276	17896	149	17520	143	39449	81
multdistr9	30427	184	14477	332	22917	182	22445	181	43819	99
multdistr10	37571	226	17808	414	29714	214	31570	219	55583	122
S/P vs M/B average results					-32.31%	-1.86%	-29.77%	-8.38%		

Table 1: Comparison between *ROS* and *LHRS*

alternative mapping strategy that leads to reduction of qubits and gates.

Fig. 5 illustrates the improved mapping strategy for an XOR cut with three inputs. In Fig. 5(a) this LUT is drawn as an XOR symbol. The conventional mapping, shown in Fig. 5(b), maps each child in to a clean ancilla, and then uses another clean ancilla to map the result of the XOR cut. The result of that cut, f, can then be used by its parents in subsequent gates. We illustrate this fact by simply annotating the circuit line where it represents the value f. However, since in this case the child cuts f_1, f_2, and f_3 are composed via the XOR operator, one can directly map them in to a single qubit without the need of requiring an additional ancilla for each child LUT, see Fig. 5(c).

Note that the size of the XOR gates does not need to be bounded by the LUT size k. In order to build XOR blocks in XAGs, we first detect 2-input XOR gates. Afterwards, sub-trees of XOR gates are grouped together. Finally, we adjust cut enumeration such that XOR cuts are assigned with cost 0, in order to force the LUT mapping to prefer XOR blocks.

5 Applications and Results

We have implemented our algorithms into the hierarchical quantum synthesis framework caterpillar[2] in C++. In this section we illustrate the efficiency of our proposed approach by synthesizing oracles, which can be used in algorithms such as Grover's search algorithm [8], which is capable of computing a satisfying assignment for a quantum oracle optimally with a quadratic speedup.

For our benchmarks we suppose that we want to perform equivalence checking between two designs. Equivalence checking is a well-known problem in logic synthesis that has been addressed by many logic synthesis tools, as for example *abc* [5] or *Formality*®. We need to synthesize an oracle quantum circuit of the function f, where f is satisfied when the two graphs performs a different operation. The algorithm would either prove that the two circuit are equivalent, or would provide the input set for which the two functions evaluate differently.

Our benchmark consists of XAG graphs. Each graph represents an equivalence checking miter of two circuits that perform the same function but using a different network structure. The miter of two networks is a network built by joining their input sets and by computing the 2-input XOR between their outputs. Further, one or more injected faults (a node performing a different computation) are injected in one of the two circuits. We consider three type of benchmarks: *addassoc*, where the algorithm should verify the validity of the associative property of addition; *multassoc*, where the two designs should be equivalent thanks to the associativity of the multiplication, and *multdistr*, to prove the distributivity of the multiplication. Each benchmark is considered with bitwidths from $w = 4$ to 10 bits. Consequently, each benchmark has $3w$ inputs and 1 output.

Our experimental results are reported in Table 1. The first two columns show the results of the state-of-the-art (M/B) synthesis flow, that uses a classic k-LUT mapper and the Bennett strategy to deal with garbage results (*LHRS*).

As expected, data shows that by only changing the k-LUT mapper (S/B) we always reduce the number of gates, paying in an increased number of qubits. On the other hand, by only applying the quantum garbage management technique (M/P), the number of qubits is always reduced, and the number of gates increased.

In the S/P_match_q experiment we have used ROS, setting the number of qubits to match M/B. In most of the cases we obtain an improvement in both qubits and gates, with the exception of *multassoc5*, *multdistr6* and *multassoc6*. For the latter cases, the SAT solver that is used in the quantum garbage management technique had reached our limit of 50000 conflicts. For this reason, we needed to slightly increase the number of qubits, still obtaining in all cases a reduction in gates with respect to M/B. *ROS* in this setting reduces the number of gates of 32.31% and the number of qubits of 1.86% on average.

In the S/P_match_g experiment, we start from the results in S/P_match_q and try to beat them, by decrementing the number of qubits, as long as the number of gates does not exceed the one in M/B. Also here ROS manages to obtain better results than the state-of-the-art flow both in gates and qubits. Gates are reduced of 29.77% while qubits are reduced of 8.38% on average, with respect to M/B.

Most of the synthesis runs completed within a few seconds, none required more than one minute in the worst-case.

[2] https://github.com/gmeuli/caterpillar

6 Conclusion

In this work we introduce ROS: a hierarchical quantum synthesis flow based on k-LUT networks. ROS exploits a k-LUT mapper that has been specifically designed for this application. This mapper is capable of generating LUTs that are easy to be synthesized by the Gray synthesis method, and leads to a quantum circuit with fewer gates if compared with existing mappers, i.e. *mf* from *abc*. In addition, ROS exploits a SAT-based quantum memory management technique to gain control over the number of qubits of the generated circuits. In our experiments we apply ROS for the synthesis of quantum oracles, which may be used in algorithms as the Grover's algorithm. Experimental results prove the ability of ROS to break the border of the pareto-point synthesis results, beating the existing framework in both qubits and number of gates.

Acknowledgments

This research was supported by the Swiss National Science Foundation (200021-169084 MAJesty).

References

[1] Matthew Amy, Parsiad Azimzadeh & Michele Mosca (2018): *On the controlled-NOT complexity of controlled-NOT–phase circuits*. Quantum Science and Technology 4(1), p. 015002, doi:10.1088/2058-9565/aad8ca.

[2] Matthew Amy, Dmitri Maslov, Michele Mosca & Martin Roetteler (2013): *A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(6), pp. 818–830, doi:10.1109/TCAD.2013.2244643.

[3] Ryan Babbush, Dominic W Berry, Ian D Kivlichan, Annie Y Wei, Peter J Love & Alán Aspuru-Guzik (2016): *Exponentially more precise quantum simulation of fermions in second quantization*. New Journal of Physics 18(3), p. 033032, doi:10.1088/1367-2630/18/3/033032.

[4] Charles H Bennett (1989): *Time/space trade-offs for reversible computation*. SIAM Journal on Computing 18(4), pp. 766–776, doi:10.1137/0218053.

[5] Robert Brayton & Alan Mishchenko (2010): *ABC: An academic industrial-strength verification tool*. In: International Conference on Computer Aided Verification, Springer, pp. 24–40, doi:10.1007/978-3-642-14295-6_5.

[6] S Debnath et al. (2016): *Demonstration of a small programmable quantum computer with atomic qubits*. Nature, doi:10.1038/nature18648.

[7] Daniel Große, Robert Wille, Gerhard W Dueck & Rolf Drechsler (2009): *Exact multiple-control Toffoli network synthesis with SAT techniques*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28(5), pp. 703–715, doi:10.1109/TCAD.2009.2017215.

[8] Lov K Grover (1996): *A fast quantum mechanical algorithm for database search*. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, ACM, pp. 212–219, doi:10.1145/237814.237866.

[9] Esteban A. Martinez et al. (2016): *Real-time dynamics of lattice gauge theories with a few-qubit quantum computer*. Nature, doi:10.1038/nature18318.

[10] Dmitri Maslov (2016): *Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization*. Physical Review A 93(2), p. 022311, doi:10.1103/PhysRevA.93.022311.

[11] Giulia Meuli, Mathias Soeken, Martin Roetteler, Nikolaj Bjorner & Giovanni De Micheli (2019): *Reversible pebbling game for quantum memory management*. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019*, doi:10.23919/DATE.2019.8715092.

[12] Giulia Meuli, Mathias Soeken, Martin Roetteler, Nathan Wiebe & Giovanni De Micheli (2018): *A best-fit mapping algorithm to facilitate ESOP-decomposition in Clifford+T quantum network synthesis*. In: *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, IEEE Press, pp. 664–669, doi:10.1109/ASPDAC.2018.8297398.

[13] D Michael Miller, Dmitri Maslov & Gerhard W Dueck (2003): *A transformation based algorithm for reversible logic synthesis*. In: *Design Automation Conference, 2003. Proceedings*, IEEE, pp. 318–323, doi:10.1145/775832.775915.

[14] Alan Mishchenko, Satrajit Chatterjee & Robert Brayton (2006): *DAG-aware AIG rewriting a fresh look at combinational logic synthesis*. In: *Proceedings of the 43rd annual Design Automation Conference*, ACM, pp. 532–535, doi:10.1145/1146909.1147048.

[15] Alan Mishchenko, Satrajit Chatterjee & Robert K Brayton (2007): *Improvements to technology mapping for LUT-based FPGAs*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26(2), pp. 240–253, doi:10.1109/TCAD.2006.887925.

[16] Leonardo de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In C. R. Ramakrishnan & Jakob Rehof, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, doi:10.1007/978-3-540-78800-3_24.

[17] Peter J. J. O'Malley et al. (2016): *Scalable Quantum Simulation of Molecular Energies*. *PRX*, doi:10.1103/PhysRevX.6.031007.

[18] Mariusz Rawski (2015): *Application of functional decomposition in synthesis of reversible circuits*. In: *International Conference on Reversible Computation*, Springer, pp. 285–290, doi:10.1007/978-3-319-20860-2_20.

[19] Norbert Schuch & Jens Siewert (2003): *Programmable networks for quantum algorithms*. *Physical review letters* 91(2), p. 027902, doi:10.1103/PhysRevLett.91.027902.

[20] P. W. Shor (1994): *Algorithms for quantum computation: discrete logarithms and factoring*. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, doi:10.1109/SFCS.1994.365700.

[21] Mathias Soeken, Martin Roetteler, Nathan Wiebe & Giovanni De Micheli (2018): *LUT-based Hierarchical Reversible Logic Synthesis*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, doi:10.1109/TCAD.2018.2859251.