Scaling-up ESOP Synthesis for Quantum Compilation

Bruno Schmitt, Mathias Soeken, Giovanni De Micheli
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
{bruno.schmitt, mathias.soeken, giovanni.demicheli}@epfl.ch

Alan Mishchenko
University of California, Berkeley
Berkeley, USA
alanmi@berkeley.edu

***Abstract*—Today's rapid advances in quantum computing hardware call for scalable synthesis methods to map combinational logic represented as multi-level Boolean networks (e.g., an and-inverter graph, AIG) to quantum circuits. Such synthesis process must yield reversible logic function, since quantum circuits are reversible. Thus, logic representations using exclusive sum-ofproducts (ESOP) are advantageous because of their natural relation to Toffoli gates, one of the primitives in reversible logic. This motivates developing effective methods to collapse AIG logic networks into ESOPs. In this work, we present two state-of-the-art methods to collapse an AIG into an ESOP expression, describe their shortcomings and introduce a new approach based on the divide-and-conquer paradigm. We demonstrate the effectiveness of our method in collapsing IEEE-compliant half precision floating point networks. Results show that our method can collapse designs—which were previously not solvable within a week—in less than 5 minutes. We also describe a technique capable of taking advantage of this new method to generate quantum circuits with up to 50% fewer T gates compared to state-of-the-art methods.**

***Index Terms*—ESOP, Collapsing, Reversible Logic, Quantum Computing**

I. Introduction

Quantum circuits are an abstraction for the physical interaction with a quantum computer. They consist of low-level operations, called quantum gates, that model the state update of one or multiple qubits [1]. Admissible gates for today's quantum computers interact with one or two qubits. Quantum compilation is the task of translating a quantum algorithm into a quantum circuit. Quantum algorithms consist of quantum and classical operations. Depending on the type of operations, different compilation techniques are applied to find a quantum circuit. For classical operations, which are the target of this paper, the compilation typically takes two steps: (i) the combinational operation is represented in terms of a reversible logic network, and (ii) the reversible gates in the network are translated into quantum gates. Toffoli gates are usually used as gates in the reversible logic networks, and efficient methods exist for their translation into quantum gates [2], [3].

Due to a natural affinity between exclusive sum-of-products (ESOP) expressions and Toffoli gates, ESOP-based reversible logic synthesis [4] has been proven to be highly effective. Given an ESOP expression for a Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$, one can easily derive a reversible network that acts on $n + m$ qubits. The number of qubits does not increase when compiling the reversible network into a quantum circuit.[1]

We assume that the combinational operations in the quantum algorithms are expressed in terms of multi-level Boolean gate level networks. In order to utilize ESOP-based reversible logic synthesis, it is necessary to convert such networks into two-level ESOP expressions. The conversion of a Boolean network into a two-level representation is called collapsing. There are two state-of-the-art techniques to collapse a multi-level Boolean network represented as an and-inverter graph (AIG) into a two-level ESOP expression: the AIG extract method and the BDD extract method [5]. To illustrate the difference, consider the simple AIG shown in Fig. 1a.

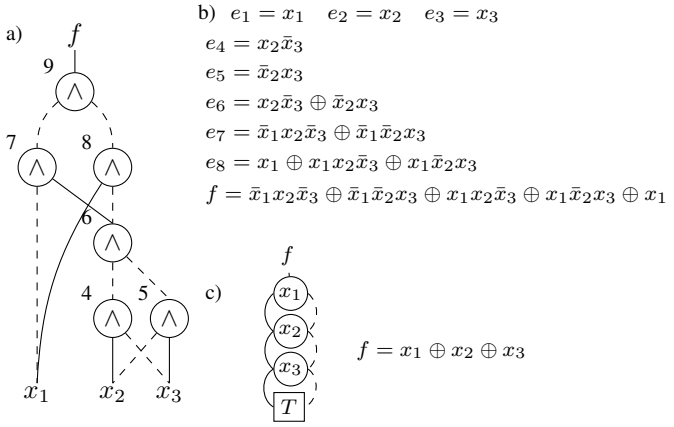

Fig. 1. In this figure, for the interest of space, we omit the $\wedge$ operator when forming cubes, e.g., $x_2 \wedge \bar{x}_3 = x_2\bar{x}_3$. a) An AIG which implements $f = x_1 \oplus x_2 \oplus x_3$. b) The result of AIG extract. c) The result of BDD extract.

The method *AIG extract* collapses all nodes of the AIG into ESOPs in a topological order; Fig. 1b shows the result of its application. Without going into details, one can already see that size explosion is the main drawback of directly collapsing AIG nodes into ESOP expressions. This problem can be mitigated by means of some ESOP optimization techniques, but these are most often not sufficient.

The alternative method *BDD extract* is presented in Fig. 1c, which shows the BDD computed from the AIG, and an ESOP

[1] In the special case where $n > 2$, $m = 1$, and the size of f's ON-set is odd, all current mapping algorithms for Toffoli gates will require one additional qubit.

extracted from the BDD. For this case and for the majority of our benchmarks, BDD extract performs much better than AIG extract in both runtime and quality of results. Unfortunately, BDD extract does not come without its own shortcoming: the lack of scalability of the BDD construction process. For some circuits, BDD construction suffers from the BDD memory explosion problem—the BDD size is exponential in the number of input variables.

Both state-of-the-art techniques lack the necessary scalability to cope with the increasing complexity of the logic functions used on quantum computers. Therefore, in this paper, we present the results of research directed towards the development of a new ESOP extraction tool:

- We revisit both the AIG and BDD extract methods and propose ways to improve them (Section III).
- We introduce a divide-and-conquer collapsing method, called *DC extract* (Section IV), that overcomes scalability limitations of the state-of-the-art approaches.
- We apply the new collapsing method for an ESOP-based reversible logic synthesis technique, allowing us to find quantum circuits with up to 50% reduced quantum gate cost (Section V).

The experimental results in Section VI confirm the effectiveness of the proposed method. Specifically, we show that we are able to collapse AIGs that previous methods were unable to. This opens up the possibility of a qubit-optimal quantum network realization of these circuits, which is crucial due to the severe resource limits imposed by today's and near-term quantum hardware.

II. Preliminaries

A *Boolean variable* x is a variable that takes one of the two values from the domain $\mathbb{B} = \{false, true\}$, or $\{0, 1\}$. A *positive literal* is the Boolean variable x and a *negative literal* is its complement $\bar{x}$. The Boolean AND of k literals is a *cube*, or product, i.e., $c = l_1 \wedge \cdots \wedge l_k$ (we may omit the symbol $\wedge$ in forming cubes, e.g., $l_1 \wedge \cdots \wedge l_k = l_1 \cdots l_k$). If a variable is not represented by a positive literal or a negative literal in a cube, then its value is said to be a *don't care literal*. A *minterm* is a cube, in which every variable is represented by either a negative or positive literal. A cube with k don't care literal values covers 2^k minterms.

The *distance* of two cubes is the number of variables for which the corresponding literals have different sets of value, i.e given three cubes: $c_1 = x_1x_2$, $c_2 = \bar{x}_1$ and $c_3 = \bar{x}_1x_2$, the distance of c_1 and c_2 is two, while the distance between c_1 and c_3 is one.

A *multiple-output Boolean function* $f : \mathbb{B}^n \to \mathbb{B}^m$ maps n Boolean input variables to m Boolean output values. We can represent f as a m-tuple of n-variable Boolean functions $(f_1, \ldots, f_m)$. The *support* of f is the subset of variables that influence the output value of the function f. Unless stated otherwise, we assume that a Boolean function is completely specified. The *cofactors* are derived from the function by substituting constant values for the input variables. For example, Boole's expansion of a function f, often called Shannon's expansion, where $f_{\bar{x}_i} = f(x_i = 0)$ and $f_{x_i} = f(x_i = 1)$ are the negative and positive cofactors of the function f with respect to variable x_i, respectively.

Any Boolean function can be represented as a two-level ESOP, which is a Boolean XOR of cubes (i.e., $f = c_1 \oplus c_2 \oplus \cdots \oplus c_j$). Using the fundamental property of the XOR operation, the two following propositions can be proven:

Proposition 1. *Two identical cubes, i.e., distance-0 cubes, can be added to any ESOP without changing the function represented by it.*

Proposition 2. *The XOR of two distance-1 cubes can be represented by a single cube.*

We also recall here the three basic expansions: $f = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$, $f = f_{\bar{x}_i} \oplus \bar{x}_i \frac{\partial f}{\partial x}$ and $f = f_{x_i} \oplus \bar{x}_i \frac{\partial f}{\partial x}$ where $\frac{\partial f}{\partial x} = f_{\bar{x}_i} \oplus f_{x_i}$. These are called Shannon expansion, the positive Davio expansion, and the negative Davio expansion, respectively.

A *Boolean network* is a directed acyclic graph where nodes represent logic gates or primary inputs/primary outputs, and edges represent wires that form the interconnections among the gates. An AIG [6] is a homogeneous Boolean network in which the logic gates are two-input ANDs and the edges can be either regular or complemented.

A BDD [7] is a graph-based representation of a function that is based on the Shannon decomposition. Applying this decomposition recursively allows dividing the function into many smaller sub-functions. BDDs make use of the fact that for many functions of practical interest, smaller sub-functions occur repeatedly and need to be represented only once. BDDs are ordered in the sense that the Shannon decomposition is applied with respect to some given variable ordering which also has an effect on the BDD's number of nodes. Improving the variable ordering for BDDs is NP-complete [8] and many heuristics have been presented that aim at finding a good ordering.

III. State-of-the-art Collapsing

In this section, we describe both the AIG and BDD extract methods [5], [9]. We present some of their implementation details and their respective shortcomings. We also discuss some initial ideas to improve on the state-of-the-art techniques that did not lead to sufficient improvements. These failed attempts further affirm the inability of the two approaches to cope with the complexity of large benchmarks.

We shall limit the following discussion for the cases where we are dealing with AIGs, but these technique can be easily extended to deal with more generic Boolean networks.

A. AIG extract

This method computes an ESOP expression for each node in an AIG in a topological order. First, each input variables x_i is assigned the trivial ESOP expression x_i. The ESOP expression of each subsequent node is computed by conjoining the ESOP expressions of its previous nodes:

$$e_i = \begin{cases} x_i & \text{if } 1 \leq i \leq n \\ e_{j(i)} \wedge e_{k(i)} & \text{if } n+1 \leq i \leq n+r \end{cases} \quad (1)$$

where r is the number of nodes, $j(i)$ and $k(i)$ identify the two inputs of node i, $1 \leq j(i) < i$ and $1 \leq k(i) < i$. Extra care must be taken when a node depends on the complementation of its prior one.

As illustrated by Fig. 1, the number of product terms in the resulting ESOP expression of each node is the product of the number of terms of its inputs. Hence, the direct implementation of this technique is highly limited in both scalability and quality of results. Therefore, in the following we discuss improvements over this naïve implementation.

Improving performance by using ESOP minimization: Over the years, several ESOP minimization strategies have been created, however none of them could be fully applied after collapsing each node without paying a large runtime penalty. To solve this problem, our implementation uses a greedy, low effort minimization strategy while constructing the ESOP expression of each node. The strategy is based on propositions 1 and 2 and is greedy in the sense that whenever a new cube is to be added to the resulting expression, it first tries to find another cube which is either distance-0 or distance-1, if this search is successful, then it transforms both cubes accordingly. With this improvement the size of the resulting ESOP of a node is only the product of the number of terms of its children in the worst case. Empirical observations indicate that most of the time the size is about the same as the size of its fanins' largest ESOP.

B. BDD extract

The method BDD extract first expresses the AIG in terms of a BDD by translating each node into a BDD in topological order. From the BDD a special case of an ESOP expression, a Pseudo-Kronecker expression, is extracted using the algorithm presented in [10]. A detailed discussion of the algorithm is given in [9].

The algorithm traverses the BDD twice. During the first pass, the best expansion is found for each node and saved in a hash table. Three possible canonical expansions (Shannon, positive Davio, or negative Davio) are considered and their costs are evaluated by the number of cubes they add to the solution. During the second pass, in each node the best expansion is chosen and, depending on the expansion, one literal is added to an array of the variable values representing the current cube. When the traversing procedure reaches the bottom of the diagram, it uses all the variable values collected to generate a cube and add it to the resulting ESOP expression.

Implementing BDD extract is quite straightforward by means of a BDD package such as CUDD [11]. The main drawback of this method is the lack of scalability of the BDD construction, which bounds its performance. The BDD construction process can suffer from a poor selection of variables order, which can be mitigated by allowing BDD dynamic variable reordering. For some circuits, however, regardless of reordering being use, the BDD construction suffers from the BDD memory explosion problem—the BDD size is exponential in the number of input variables—and therefore the naïve implementation is impractical.

IV. Divide-and-Conquer extraction

The main technical contribution of this paper is the introduction of a divide-and-conquer collapsing method. This new approach is based on a simple and natural idea: divide the given problem into small-enough subproblems; solve these subproblems independently, then combine these solutions to arrive at a final solution.

A. The set of single-cube cofactors

In the following we discuss a useful relation for the coming discussion.

Lemma 1. *Any Boolean function f can be written as*

$$f = f \wedge p_1 \oplus f \wedge p_2 \oplus \cdots \oplus f \wedge p_i \oplus \cdots \oplus f \wedge p_N \qquad (2)$$

for N Boolean functions p_N over the same support as f, if

$$p_1 \oplus p_2 \oplus \cdots \oplus p_i \oplus \cdots \oplus p_N = 1.$$

Proof: By factoring f in the right-hand-side of (2) becomes $f(p_1 \oplus \cdots \oplus p_N)$. ∎

The careful selection of a set $P = \{p_1, p_2, \ldots, p_i, \ldots, p_N\}$ is a key component of our method because it directly affects its effectiveness. Indeed, we need to choose P in such a way that makes easier to collapse *all* individual $f \wedge p_i$. This might sound counter-intuitive at first, but if we limit our choices for p_i so that P is a set of single-cube cofactors of f, then the variables present in p_i can be ignored when collapsing $f \wedge p_i$.

In practice, our implementation represents f as an AIG and p_i as cube; $f \wedge p_i$ is represented as a tuple (f_i, p_i), where f_i is the original AIG f with the variables present in p_i being assigned constant values in accordance to their polarity in the cube. These constant values simplify the AIG through constant propagation.

After dividing the problem, we use BDD extract to collapse these f_i AIGs into ESOP e_i. Finally, we can combine our results in two different ways: one will leave the cofactors multiplying the intermediate ESOPs, while the other will expand the multiplications. The following example illustrates our method.

Example 1. *Let f be a 3-variable Boolean function. Using Lemma 1, we can represent the function as*

$$f = f \wedge p_1 \oplus f \wedge p_2 \oplus f \wedge p_3 \oplus f \wedge p_4$$

with $p_1 = x_1 \wedge x_2$, $p_2 = \bar{x}_1 \wedge x_2$, $p_3 = x_1 \wedge \bar{x}_2$ and $p_4 = \bar{x}_1 \wedge \bar{x}_2$. With f given as an AIG, we can divide our problem by creating the four tuples (f_1, p_1), (f_2, p_2), (f_3, p_3), (f_4, p_4). We solve it by collapsing the simplified f_i separately, where $i = \{1, 2, 3, 4\}$, thus generating: (e_1, p_1), (e_2, p_2), (e_3, p_3), (e_4, p_4). Finally, we can report the combined solution either as:

$$f = e_1 \wedge p_1 \oplus e_2 \wedge p_2 \oplus e_3 \wedge p_3 \oplus e_4 \wedge p_4$$

or as an expanded ESOP expression, with the multiplications $e_i \wedge p_i$ carried out.

B. Selection heuristics

The efficiency of our method is intimately related to the selection of variables to cofactor. Therefore, in this section, we define different heuristics to select the variables.

Before, it is helpful to present a concept which aids the visualization of how both heuristics behave. Hence, in the context of this work, we define a cofactoring diagram as a directed acyclic graph with a single root node and 2^n leaves, where n is he number of cofactored variables. Each leaf is associated with a cube c which contains the literals of the cofactored variables. Each intermediate node is associated with a variable and has two out-going edges, representing the positive and negative cofactors. Examples of such diagrams can be seen in Fig. 2. We tested the following two heuristics for picking the variables:

Fixed variable selection: This heuristic chooses variables to cofactor using only one criteria: the combined size, in number of nodes, of the resulting AIGs. The idea is to minimize this number as much as possible. The first variable is chosen by temporarily cofactoring all variables, one at a time, and choosing one with minimal combined size. The two resulting AIGs (equivalent to the positive and negative cofactors) are used as starting point for the selection of the second variable, which, in turn, temporarily cofactor all remaining variables, one at a time, from the two AIGs and apply the same criteria do select a variable. This process continues until the desired number of variables is cofactored. Fig. 2a illustrates the implicit cofactor diagram generated by this process.

Free variable selection: This heuristic selects variables using the same criteria as the previous one. The difference is the recursive way it chooses them. After selecting the first variable, the two resulting AIGs are used as two different starting points for the selection of the second one. In practice this means that the variable selected as second cofactor in one branch might be different from the variable selected in the other. Fig. 2b illustrates this.

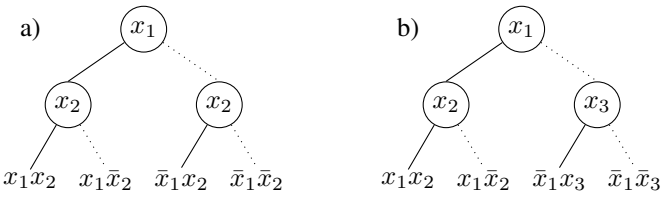

Fig. 2. Implicit cofactor diagrams generated by: a) Fixed variable selection heuristic, b) Free variable selection heuristic.

V. Application: Reversible Circuits

ESOP-based logic synthesis is heavily used in reversible logic synthesis, due to natural correspondence of product terms in an ESOP expressions and Toffoli gates. A reversible Toffoli gate acts on a number of variables (that represent qubits). Let X be the set of variables in the quantum circuit. Then a (multiple-controlled mixed-polarity) Toffoli gate has a (possibly empty) set of control lines, which are literals over X and one target line which is a variable in X that is not a control line. The Toffoli gate will invert the variable assigned to the target line, if and only if the polarity of all inputs to the control lines match their polarity. Fig. 3 illustrates one Toffoli gate that acts on 5 qubits.

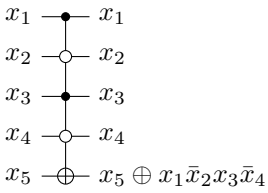

Fig. 3. Example Toffoli gate

By concatenating Toffoli gates that act on the same target line, one can build up an ESOP expression on the target line. Vice versa, given an ESOP expression, one can easily extract a sequence of Toffoli gates with control lines according to the literals in the product terms. When considering multiple-output functions, the resulting quantum circuit requires $n + m$ qubits where m is the number of outputs.

Example 2. *Assume* $f = \bar{x}_1\bar{x}_2x_3x_4x_5 \oplus \bar{x}_1x_2x_3\bar{x}_4 \oplus \bar{x}_1x_2x_4x_5 \oplus x_1\bar{x}_2x_4x_5 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4$. *Then a quantum circuit composed of Toffoli gates to realize* f *is depicted in Fig. 4.*

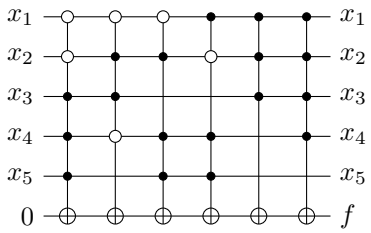

Fig. 4. Example of a synthesized reversible circuit.

Toffoli gates are an intermediate representation. They are mapped into quantum gate library using an additional step (see, e.g., [3]). An universal fault-tolerant quantum gate library is the *Clifford+T gate* set [12]. The cost model for quantum operations is non-trivial, but a good heuristic will try to minimize both the number of Toffoli gates and the number of controls in a Toffoli gates, which corresponds, respectively, to the number of product terms in the ESOP expressions and the number of literals in the product terms. The cost of a Toffoli gate is typically measured in the number of T gates it requires in its mapping to a sequence of quantum operations, since the T gate is considered the most expensive gate in the Clifford+T quantum gate library [13]. Table I gives the costs of implementing multiple-controlled Toffoli gates, according to [3]. The circuit in Fig. 4 has a cost of 143 T gates.

If the ESOP expression is generated from the proposed DC extract algorithm, we can cluster product terms that have the same cofactor and use an additional helper qubit, called ancilla, to store the cofactor values. Fig. 5 illustrates this procedure. The top-most qubit is the ancilla that is being initialized to 0

TABLE I
THE COSTS OF MULTIPLE-CONTROLLED TOFFOLI GATES IN NUMBER OF T GATES.

# Controls	0	1	2	3	4	5
# T gates	0	0	7	16	24	31

and then computes all cofactors x_1x_2, $\bar{x}_1x_2$, $\bar{x}_1\bar{x}_2$, and $x_1\bar{x}_2$. By computing the cofactors in a Gray code order [14], the transition from one cofactor to another can be done using a single CNOT gate, a gate with a single control line. In general, if the cofactors have k literals, we can transition from one cofactor to another using a single $(k-1)$-controlled Toffoli gate, since the distance of two cofactors in Gray code order is 1. In order to exploit this property, DC extract must use the fixed variable selection heuristic. The circuit in Fig. 5 implements the same function; while, at the expense of using one more qubit, it has a cost of 109 T-gates.

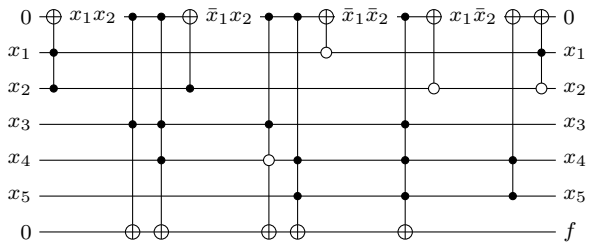

Fig. 5. A new realization of the circuit in Fig. 4 with a lower cost in number of T gates.

VI. EXPERIMENTAL RESULTS

A. Experimental setup

We implemented the algorithm described in Section IV in C++ using both CUDD and ABC [16] as external static libraries. ABC is an open-source tool, designed for logic synthesis, technology mapping, and formal verification for logic circuits, we use it to manipulate AIGs through its AIG package GIA and to check the results for equivalence. The CUDD package is used to manipulate BDDs.

To evaluate our method, we use a set of Verilog netlists of several IEEE-compliant arithmetic floating point designs in half (16-bit) precision. For synthesis all Verilog files were translated into AIGs and optimized for size using ABC's *resyn2* script. The collapsing of the these function is used as an example to illustrate the strength of this method in finding a good ESOP representation which is a good starting point for a quantum logic synthesis flow.

All experimental results were run on an Intel(R) Xeon(R) CPU E5-2690 v4 at 2.60GHz. The reported runtimes and peak memory usage were obtained using the GNU 1.7 version of the command *time*.

B. Collapsing floating-point benchmarks

The results of collapsing for each benchmark are reported in Table II. The first column identifies each benchmark by its name. The second column give the number of inputs and outputs (i/o). The remaining columns reports the results of using different techniques for collapsing in terms of ESOP number of cubes (# terms), time and peak memory usage (mem).

Both state-of-the-art methods failed to collapse the whole set of benchmarks. In this experiment the timeout was set to one week. Note that AIG extract has the worst runtime and quality of result for all benchmarks. BDD extract, run with dynamic variable reordering, improve these results and is able to collapse an additional benchmark. DC extract was configured to cofactor cubes with 2, 4, and 8 variables using the free variable selection heuristic. The improvement is remarkable; when cubes of 8 variables are cofactored, not only all benchmarks were successfully collapsed, but also *fp_and*, *fp_div* and *fp_sub* were processed within four minutes and using much less memory—when compared with the other runs of DC extract. Note *fp_mult* runs out of memory (OOM) for in the other runs of DC extract.

Finally, in order to evaluate the impact of cofactoring without taking into account the variable selection heuristics, we implemented a random variable selection procedure which behaves in a similar way as the free variable selection heuristic, but instead of trying to minimize the size of the resulting AIGs at each recursion step, it randomly select one of the remaining variables. We used this heuristic to collapse each benchmark a hundred times. Not surprisingly, the results shown in Table III indicate the need to carefully selecting the variables.

C. ESOP-based reversible logic synthesis

We implemented the synthesis method described in Section V as a standalone tool capable of using the cofactored results of DC extract to synthesize reversible circuits. Table IV shows the the results in number of qubits (q), T gates (T). In the case of our technique, we also report the number of cofactored variables (v). The first column name the benchmarks. The next column presents the results obtained by directly mapping product terms into multiple-controlled Toffoli gates when the benchmark is collapsed without cofactoring, followed by the results of optimizing the direct mapping by means of [15], an approach which cofactors Toffoli gates that share the same controls and stores the cofactor on an ancilla qubit. The third column corresponds to results obtained when using the synthesis method of Section V. As these experimental results show, our method of deriving the cofactors more explicitly, when compared to [15], leads to circuits with fewer quantum gates. Finally, in the last column are the results of [5], which is a complete synthesis flow for quantum—hence our methods should not be viewed as substitute for it, but as a complement. Nevertheless, for the unitary operators benchmarks, we are able to achieve an improvement of up to 50% when compared to [5]. However, the results for the binary operators, namely *fp_add*, *fp_sub*, *fp_div*, and *fp_cmp*, have very large T gate count. The problem lies with the large size of their ESOP expressions, which, to our knowledge, cannot be handled by any state-of-the-art ESOP optimization tool. Consequently, we did not use the direct method nor [15] to synthesize them; however, we expect an even larger T gate count if these techniques were used.

TABLE II
RUNTIME OF AIG, BDD AND DC EXTRACT FOR THE HALF PRECISION (16-BIT) ARITHMETIC FLOATING POINT DESIGNS.

		AIG extract			BDD extract			DC extract (2 cofactored vars)			DC extract (4 cofactored vars)			DC extract (8 cofactored vars)		
name	i / o	# terms	time (s)	mem (Mb)	# terms	time (s)	mem (Mb)	# terms	time (s)	mem (Mb)	# terms	time (s)	mem (Mb)	# terms	time (s)	mem (Mb)
fp_add	32 / 16		TO			TO		16762634	27289.42	80190.32	17352120	10423.72	17077.70	14600025	134.81	1789.19
fp_cmp	32 / 04		TO		184363	0.80	16.86	184347	0.58	16.48	184353	0.54	17.22	184687	0.71	16.57
fp_div	32 / 16		TO			TO		29504220	4246.47	6397.96	30650976	1594.69	4618.52	30110188	242.20	2478.21
fp_exp	16 / 16	291726	14697.99	825.28	11470	282.24	156.88	11214	90.81	74.28	11087	21.59	47.43	11266	5.96	21.72
fp_invsqrt	16 / 16	15907	74.05	35.12	2213	2.00	18.70	2238	1.50	15.54	2925	1.38	13.55	5188	0.93	18.80
fp_ln	16 / 16	49240	6.40	10.21	16724	1.87	74.55	16301	1.14	23.48	16600	1.09	14.26	24684	0.65	14.59
fp_log2	16 / 16	30476	3.04	7.61	10816	0.60	17.84	10822	0.41	14.79	11502	0.28	13.04	18833	0.45	14.57
fp_mult	32 / 32		TO			TO				OOM			OOM	94869392	62811.34	120460.29
fp_recip	16 / 16	17478	3.89	8.11	3483	0.39	13.20	3576	0.23	12.79	3619	0.21	12.82	7346	0.79	18.47
fp_sincos	17 / 16	35992	10.03	11.08	5901	0.79	18.97	6285	0.57	18.95	6127	0.28	15.55	8949	0.39	13.51
fp_sqrt	16 / 16	19150	8.91	7.26	1963	0.16	12.01	2058	0.14	13.07	2218	0.10	12.47	3968	0.25	13.61
fp_square	16 / 16	7256	0.23	2.66	2683	0.02	12.01	2731	0.02	12.44	2853	0.06	12.37	6834	0.23	13.25
fp_sub	32 / 16		TO			TO		16801296	30392.64	115557.24	16988606	7211.76	16467.87	14502087	134.65	1784.04

TABLE III
RUNTIME OF DC EXTRACT FOR THE HALF PRECISION ARITHMETIC FLOATING POINT DESIGNS WHEN CHOOSING COFACTORS RANDOMLY.

	DC extract (8 random cofactored vars)			
name	min	max	avg	stdev
fp_add	98.88	4515.3	976.63	1355.33
fp_cmp	0.55	0.94	0.71	0.16
fp_div	202.08	425.5	265.17	80.97
fp_exp	5.58	7.4	6.27	0.50
fp_invsqrt	1.18	1.79	1.51	0.25
fp_ln	0.65	1.11	1.00	0.18
fp_log2	0.55	1.02	0.91	0.19
fp_mult	—	—	—	—
fp_recip	0.78	1.3	1.17	0.18
fp_sincos	0.42	0.95	0.77	0.17
fp_sqrt	0.32	0.81	0.70	0.20
fp_square	0.3	0.66	0.50	0.15
fp_sub	150.61	2586.4	822.07	715.83

TABLE IV
SYNTHESIS RESULTS IN NUMBER OF QUBITS (q) AND T GATES (T).

	Direct map		[15]		Our method			[5]	
name	q	T	q	T	v	q	T	q	T
fp_add					10	49	$1.44\,10^9$	156	33521
fp_cmp					8	37	18296672	40	30426
fp_div					12	49	$2.84\,10^9$	144	589721
fp_exp	32	784887	33	752558	9	33	344603	32	1193083
fp_invsqrt	32	136023	33	129692	7	33	101204	32	169282
fp_ln	32	969757	33	931516	7	33	585468	32	1623461
fp_log2	32	623948	33	566315	7	33	400281	32	850331
fp_mult								267	141657
fp_recip	32	170245	33	150448	4	33	136137	32	198167
fp_sincos	33	376733	34	326403	6	34	191136	34	452129
fp_sqrt	32	122793	33	106216	2	33	58184	32	133489
fp_square	32	129797	33	126803	5	33	108935	32	165545
fp_sub					9	49	$1.47\,10^9$	156	33626

The multiplication benchmark (*fp_mult*) was only synthesized using [5].

VII. CONCLUSION AND FUTURE WORK

We presented a new collapsing method that outperforms existing state-of-the-art methods. Improved scalability and the effectiveness in collapsing all half precision (16 bits) floating point arithmetic benchmarks are the main advantages. Using these results as starting point, design flows for the synthesis of reversible logic in quantum computers, such as [5], would be able to map these circuits into qubit-optimum quantum circuits of 48 qubits or less—an important step towards the goal of quantum advantage [17].

REFERENCES

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

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, no. 5, p. 3457, 1995.

[3] D. Maslov, "Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization," *Physical Review A*, vol. 93, p. 022311, 2016.

[4] A. Mishchenko and M. A. Perkowski, "Logic syntheis of reversible wave cascades," in *Int'l Workshop on Logic and Synthesis*, 2002.

[5] M. Soeken, M. Roetteler, N. Wiebe, and G. D. Micheli, "Logic Synthesis for Quantum Computing," *arXiv preprint arXiv:1706.02721v1*, 2017.

[6] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.

[7] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[8] B. Bollig and I. Wegener, "Improving the variable ordering of obdds is np-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, Sep. 1996.

[9] R. Drechsler, "Preudo-Kronecker expressions for symmetric functions," *IEEE Trans. on Computers*, vol. 48, no. 9, pp. 987–990, 1999.

[10] A. Mishchenko and M. A. Perkowski, "Fast heuristic minimization of exclusive-sum-of-products," in *Reed-Muller Workshop*, 2001.

[11] F. Somenzi, "CUDD: Colorado university decision diagram package," 1996.

[12] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. E. Wright, and C. Monroe, "Experimental comparison of two quantum computing architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

[13] 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*, vol. 32, no. 6, pp. 818–830, 2013.

[14] F. Gray, "Pulse code communication," March 1953, uS Patent 2,632,058.

[15] D. M. Miller, R. Wille, and R. Drechsler, "Reducing reversible circuit cost by adding lines," *Multiple-Valued Logic and Soft Computing*, vol. 19, no. 1–3, pp. 185–201, 2012.

[16] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Computer Aided Verification*, 2010, pp. 24–40.

[17] A. W. Harrow and A. Montanaro, "Quantum computational supremacy," *Nature*, vol. 549, no. 7671, pp. 203–209, 2017.