Lowering the T-depth of Quantum Circuits via Logic Network Optimization

THOMAS HÄNER and MATHIAS SOEKEN, Microsoft Quantum, Switzerland

6

The multiplicative depth of a logic network over the gate basis $\{\wedge, \oplus, \neg\}$ is the largest number of $\wedge$ gates on any path from a primary input to a primary output in the network. We describe a dynamic programming based logic synthesis algorithm to reduce the multiplicative depth of logic networks. It makes use of cut enumeration, tree balancing, and **exclusive sum-of-products (ESOP)** representations. Our algorithm has applications to cryptography and quantum computing, as a reduction in the multiplicative depth directly translates to a lower T-depth of the corresponding quantum circuit. Our experimental results show improvements in T-depth over state-of-the-art methods and over several hand-optimized quantum circuits, for instance, of AES, SHA, and floating-point arithmetic.

CCS Concepts: • **Security and privacy** → *Cryptography*; • **Theory of computation** → *Quantum computation theory*; • **Hardware** → *Logic synthesis;*

Additional Key Words and Phrases: Multiplicative depth, quantum compilation, exclusive sum-of-products, XOR-AND graphs

ACM Reference format:
Thomas Häner and Mathias Soeken. 2022. Lowering the T-depth of Quantum Circuits via Logic Network Optimization. *ACM Trans. Quantum Comput.* 3, 2, Article 6 (February 2022), 15 pages.
https://doi.org/10.1145/3501334

1 INTRODUCTION

Logic networks are the central data structure in algorithms for technology-independent optimization in electronic design automation applications [11, 43, 62]. Roughly speaking, the number of logic gates in a logic network corresponds to the size of its physical implementation, while the number of logic levels corresponds to its delay.

In recent years, the domain of applications for logic optimization has broadened to also target areas such as cryptography [9] and fault-tolerant quantum computing (see, e.g., [33, 34, 57, 60]). Logic networks are typically represented over a gate set consisting of 2-input AND gates, 2-input XOR gates, and inverters. This representation is known as an **XOR-AND graph (XAG)**, and for the implementation cost, only AND gates are considered.

The **multiplicative complexity (MC**, [54]) and the **multiplicative depth (MD,** [14]) of a Boolean function are two important theoretical metrics. The multiplicative complexity is the smallest number of AND gates necessary in any XAG that represents the function. Similarly, the

Authors' address: T. Häner and M. Soeken, Microsoft Switzerland, The Circle 02, 8058 Zürich, Switzerland.

2643-6817/2022/02-ART6 $15.00
https://doi.org/10.1145/3501334

multiplicative depth of a function is the smallest critical path (only considering AND gates) in any XAG that represents the function. We also refer to the length of the critical path (only considering AND gates) as AND-depth.

We refer to the number of AND gates and the AND-depth of an XAG by *MC/MD of an XAG*, respectively. We use just MC or MD if it is clear from the context whether we refer to the MC/MD of a Boolean function or to the MC/MD of a logic network. We note that the latter provides an upper bound to the former.

Both multiplicative complexity and multiplicative depth play important roles in cryptography and fault-tolerant quantum computing. A low multiplicative complexity corresponds to a higher vulnerability to some cryptographic attacks. In fault-tolerant quantum computing, the multiplicative complexity provides an upper bound on the number of expensive quantum operations (in a fault-tolerant setting) as well as on the number of qubits [36]. Furthermore, the multiplicative depth is an approximate measure for the execution time of a quantum algorithm [37].

Computing the multiplicative complexity of a Boolean function f is expensive. It has been shown that no algorithm exists to compute the multiplicative complexity that is polynomial in the size of the truth table for f [20] if one-way functions [30] exist. We are not aware of any theoretical results concerning the multiplicative depth.

Consequently, many heuristics have been proposed that aim to reduce the MC of an XAG (see, e.g., [9, 17, 49, 63, 64]) in order to arrive at tighter upper bounds on the MC of the function being implemented. Similarly, some heuristics have been proposed that aim to reduce the multiplicative depth [6, 14]. In this paper, we introduce a logic synthesis algorithm to reduce the MD of logic networks. Our algorithm is based on dynamic programming and makes use of cut enumeration [18], tree balancing [40], as well as ESOP [52] and ESPP [25] representations.

Contributions. We present a fully automatic logic synthesis algorithm that reduces the multiplicative depth of logic networks. We present benchmarks demonstrating that our algorithm is capable of reducing the MD by up to 3× for depth-optimized logic networks and up to 9× for MC-optimized logic networks. As a result, also the quantum circuits derived from our depth-optimized networks feature depths that are significantly smaller than state-of-the-art circuit designs. Crucially, these improvements in depth are possible with only a mild increase in space-time volume.

2 PRELIMINARIES

2.1 Logic Networks

In this work, we consider **XOR-AND graphs (XAGs)**, which are logic networks consisting of 2-input AND gates, and 2-input XOR gates. Such logic networks can represent all 0-preserving Boolean functions, i.e., functions f for which $f(0, \ldots, 0) = 0$. We are interested in logic networks that minimize the maximum number of AND gates on any path from an input to an output as a primary cost criteria, and the number of overall AND gates as a secondary cost criteria. Functions f, which are not 0-preserving, can be realized by finding an XAG for $\bar{f}$ and then inverting the output. Restricting to have inversions only at the outputs does not affect the AND gates in the circuit, as all inner inversions can be propagated to the outputs by only using XOR gates [54].

Formally, we model an XAG for a single-output Boolean function f over n variables $x_1, \ldots, x_n$ as a sequence of *steps*, or gates,

$$x_i = x_{j_{1i}} \circ_i x_{j_{2i}} \tag{1}$$

for $n < i \leq n + r$, and $\circ_i \in \{\oplus, \wedge\}$. The values $1 \leq j_{1i} < j_{2i} < i$ point to primary inputs or previous steps in the network. The function value is computed by the last step $f = x_{n+r}$. This model is readily extended to multi-output Boolean functions, by associating each output function

with some step in the network. The *logic level* of a primary input or gate i is defined as

$$\ell_i = \begin{cases} 0 & \text{if } i \leq n, \\ \max\{\ell_{j_{1i}}, \ell_{j_{2i}}\} & \text{if } i > n \text{ and } \circ_i = \oplus, \\ \max\{\ell_{j_{1i}}, \ell_{j_{2i}}\} + 1 & \text{if } i > n \text{ and } \circ_i = \wedge. \end{cases} \tag{2}$$

The depth of an XAG is $d = \max\{\ell_i \mid 1 \leq i \leq n + r\}$, the largest level among all gates. In other words, the logic level of a step is the earliest possible time in which a step must be computed, if we aim at parallelizing the evaluation of a logic network. Similarly, we define the *reverse logic level* ℓ_i^r as the latest possible time in which step i can be performed without increasing the depth of the logic network.

2.2 Cut Enumeration

Many algorithms for logic network optimization are based on applying local changes to small subnetworks instead of considering the whole logic network at once. An important family of single-rooted subnetworks are *cuts*. Formally, a *cut* C of a step i in a logic network is a set of steps, called *leaves*, such that (i) every path from step i to a primary input visits at least one leaf, and (ii) each leaf is contained in at least one path. Step i is called the *root* of the cut and each cut represents a subgraph that includes the root i and some internal steps, and has the leaves as primary inputs. A cut is k-feasible (referred to as k-cut), if $|C| \leq k$, i.e., it has at most k leaves.

Cut enumeration [18] is an algorithm that computes all or a subset of all k-cuts for each step in a network. It constructs a mapping $\mathsf{CUTS}(i)$ that maps each step to a set of cuts using the following recursive procedure:

$$\mathsf{CUTS}(i) = \begin{cases} \{\{i\}\} & \text{if } i \leq n, \\ \{\{i\}\} \cup \bigcup \{C_1 \cup C_2 \mid & \\ \quad C_1 \in \mathsf{CUTS}(j_{1i}), & \\ \quad C_2 \in \mathsf{CUTS}(j_{2i}) & \\ \quad \text{s.t. } |C_1 \cup C_2| \leq k\} & \text{otherwise.} \end{cases} \tag{3}$$

Cuts $\{\{i\}\}$ for root i are called *trivial* cuts. Note that these are essential, since otherwise the leaves of cuts can only be primary inputs. Cut enumeration can also compute the function $\mathsf{FUNC}(i, C)$ represented by a C for root i, by assigning $\mathsf{FUNC}(i, \{i\}) = x_i$ for all trivial cuts, and

$$\mathsf{FUNC}(i, C) = \mathsf{FUNC}(j_{1i}, C_1) \circ_i \mathsf{FUNC}(j_{2i}, C_2) \tag{4}$$

if C was constructed using C_1 and C_2 in (3). Support-normalized truth tables are typically used to represent the cut functions; e.g., truth tables for cut functions $x_1 \wedge x_3$ and $x_4 \wedge x_9$, are both represented by the 4-bitstring $(1000)_2$. To which variables the truth table refers can be determined from the cut's leaves.

2.3 Exclusive Sum-of-products

An ESOP for an n-variable Boolean function $f(x_1, \ldots, x_n)$ has the form

$$f(x_1, \ldots, x_n) = \bigoplus_{j=1}^{m} \left(x_1^{p_{1,j}} \wedge \cdots \wedge x_n^{p_{n,j}} \right) \tag{5}$$

for some m and *polarities* $p_{i,j}$, which take values from 0 to 2. Their meaning is that $x_i^0 = \bar{x}_i$, $x_i^1 = x_i$, and $x_i^2 = 1$. We call x_i^0 a negative literal, x_i^1 a positive literal, and x_i^2 an empty literal. If $m = 0$, we define $f(x_1, \ldots, x_n) = 0$. The constant-1 function can be represented by an ESOP where $m = 1$ and $p_{1,1} = \cdots = p_{n,1} = 2$.

Each term $(x_1^{p_{1,j}} \wedge \cdots \wedge x_n^{p_{n,j}})$ is called a *cube* of *degree* $d_j = |\{i \mid p_{i,j} \neq 2\}|$. It can be seen as an $(n-d_j)$-dimensional subcube of the n-dimensional hypercube, in which the 2^n vertices correspond to all bitstrings of length n. We require that no cube occurs more than once in an ESOP. The *degree* of the ESOP is $\max_{1 \leq j \leq m} d_j$.

An ESOP in which $p_{i,j} \neq 0$ for all $1 \leq i \leq n$, $1 \leq j \leq m$ is called the *algebraic normal form* of f. It is unique up to permutation of the cubes. The degree of the algebraic normal form is called the *algebraic degree* of f and is a lower bound for the degree of any ESOP for f. An ESOP can be brought into algebraic normal form by replacing each cube with 2^l cubes in which all $l = |\{i \mid p_{i,j} = 0\}|$ negative literals are replaced by all combinations of positive and empty literals.

Various exact and heuristic algorithms [10, 19, 24, 42, 46, 50, 52, 53, 61] exist that find ESOPs for Boolean functions, where the primary cost function is the number of cubes in the ESOP and the secondary cost function is the total number of non-empty literals. ESOP expressions are interesting for our approach because they have a small depth: the depth is bounded by its degree. However, the number of AND gates in a straightforward XAG representation of an ESOP is $\sum_{1 \leq j \leq m} d_j - m$, which correlates with the total number of non-empty literals. Consequently, ESOP optimization algorithms that target the number of literals as the *primary* cost function would be a better fit for our approach.

2.4 Quantum Computing

A quantum computer consists of quantum bits, so-called *qubits*, to which sequences of *quantum gates* are applied in order to solve a computational task. The quantum computer is controlled by a classical computer running a quantum program, which consists of both classical and quantum instructions: classical instructions are executed by the (classical) host computer, and quantum instructions get sent to the quantum co-processor for execution.

In each computational step, the classical computer decides on the sequence of quantum instructions to be executed on the co-processor. Such sequences can be depicted as *quantum circuits*. The circuit diagram is read from left to right, with each horizontal line representing a qubit, and quantum gates are represented as boxes/symbols on these lines. Figure 2 shows a quantum circuit that computes the majority-of-5 function and is derived from the logic network in Figure 1. The circuit consists of CNOT gates, AND gates, as well as uncomputing AND gates. CNOT gates act on two qubits and compute the XOR of both qubit values onto the lower (target) qubit, leaving the upper (control) qubit unchanged. The AND gate computes a 1 on a newly initialized target qubit, if and only if the two control qubits are 1. The uncomputing AND gate expects that the target qubit is 1 if and only if the two control qubits are are 1, and releases the target qubit in a clean state such that it can be used for subsequent computations.

In this paper, we target *fault-tolerant* quantum computing, which is necessary to run quantum algorithms with more than a few thousand operations, e.g., for chemistry simulations of practical interest [48, 66]. In this setting, the focus of circuit optimization shifts away from two-qubit gates (e.g., for NISQ devices [47]) toward gates that require distillation. In particular, when the surface code is used, the so-called T-gate incurs a large overhead [3, 45]. In fault-tolerant quantum computing, the cost of CNOTs are typically neglected, despite the actual cost being nonzero [13, 31]. The AND gate has a T-count of 4 and a T-depth of 1 if one additional helper qubit is used for its implementation [27] (otherwise, it can be implemented with a T-depth of 2 without the use of a helper qubit [21]). The uncomputing AND gate requires no T-gates, but does require an intermediate measurement [27]. While such measurements may be relatively costly in a NISQ setting, this

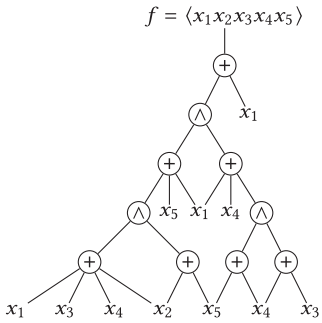

Fig. 1. XAG for the majority-of-5 function, with multiplicative complexity 3 and multiplicative depth 2; for 2-input XOR gates are merged into multi-input XOR gates.

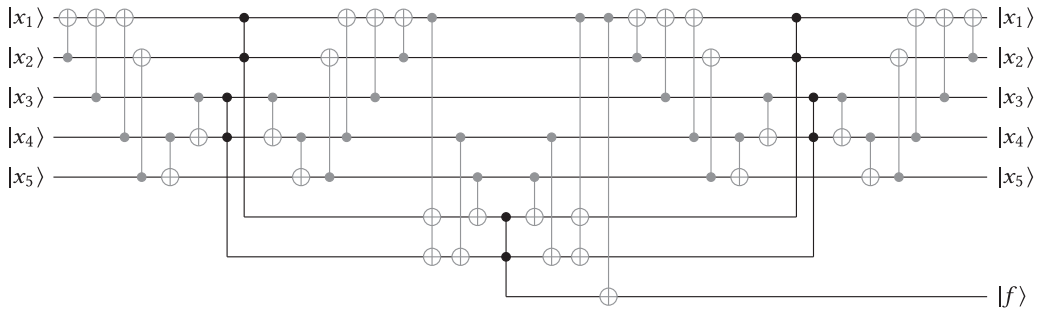

Fig. 2. Quantum circuit for the majority-of-5 function, derived from the logic network in Figure 1. Each of the computing AND gates, can be realized in T-depth 1 when using one helper qubit. The circuit therefore has T-count 12, T-depth 2, and 10 qubits, incl. qubits for function inputs and outputs.

is not the case in fault-tolerant quantum computing, where measurements are anyway constantly used to extract error syndromes.

While previous work for translating classical functionality into quantum circuits [36] focused on reducing the number of costly T-gates, we aim to shorten the time to solution by reducing the T-depth instead. Amy et al. considered T-depth optimization in a resynthesis approach that takes a quantum circuit as input and applies peephole optimization to rewrite parts of it [2]. Further, alternative methods to decompose large multiple-controlled Toffoli gates into quantum circuits aiming to reduce the circuit depth have been proposed [7, 44, 56, 59]. These methods provide alternative approaches to T-depth reduction.

In contrast, our proposed algorithm targets the more specific task of translating classical functionality, and not T-depth optimization of universal quantum circuits. Our algorithm is thus able to exploit optimization opportunities at a higher level of abstraction—the XAG of the classical function—and we show that these optimizations result in a quantum circuit with a lower T-depth.

3 MULTIPLICATIVE DEPTH REDUCTION

In this section, we introduce various methods that reduce the multiplicative depth of logic networks. Then, we present a procedure to map these networks to quantum circuits while maintaining depth improvements.

3.1 Cut-based Balancing

Algorithm 1 describes a generic balancing algorithm based on dynamic programming and cut enumeration inspired by [40].[1] It takes as input a logic network for an n-variable Boolean function with r steps and returns a new depth-optimized logic network. Traversing all steps i in topological order, it computes depth-optimized candidates for each cut C of i, and stores the best candidate in a mapping $\mathsf{BEST}(i)$. The output of the depth-optimized network is $\mathsf{BEST}(n+r)$ after all steps have been visited. For each cut C of step i, the algorithm tries to resynthesize the cut function $\mathsf{FUNC}(i, C)$ with the target to reduce the level of step i. For this purpose, it assumes the best candidates for the cut's leaves.

The algorithm invokes a *balance* subroutine that resynthesizes the cut function. It is therefore generic and can be customized by applying various resynthesis procedures. One possible resynthesis procedure is presented in [40]. It computes a **sum-of-products (SOP)** representation for the cut function and then translates each term in the SOP into a weight-balanced tree of AND gates, as well as all terms into a weight-balanced tree of OR gates. Our work adapts this method by using an ESOP representation instead. Instead of OR gates from the SOP representation, which add to the multiplicative complexity of the network, the outer operations in an ESOP representation are XORs that do not contribute to the logic network's multiplicative complexity/depth.

ALGORITHM 1: Generic cut-based balancing

for $i = 1, \ldots, n$ **do**
  $\mathsf{BEST}(i) \leftarrow i$
end for
for $i = n+1, \ldots, n+r$ **do**
  $j_{\text{best}} \leftarrow \Lambda$
  $\ell_{j_{\text{best}}} \leftarrow \infty$
  for $C \in \mathsf{CUTS}(i)$ s.t. $|C| > 1$ **do**
    $\{l_1, \ldots, l_k\} \leftarrow C$
    $j \leftarrow \mathit{balance}(\mathsf{FUNC}(i, C), \mathsf{BEST}(l_1), \ldots, \mathsf{BEST}(l_k))$
    if $\ell_j < \ell_{j_{\text{best}}}$ **then**
      $j_{\text{best}} \leftarrow j$
    end if
  end for
  $\mathsf{BEST}(i) \leftarrow j_{\text{best}}$
end for
return $\mathsf{BEST}(n+r)$

3.2 ESOP Balancing

In this section we discuss a rebalancing algorithm based on ESOP forms, which can be used in Algorithm 1. ESOP forms offer a potentially low-depth implementation as an XAG. For the sake of a simpler description of the algorithm, we assume that the ESOP form is given in algebraic normal form, however, in the implementation we consider ESOP forms that also contain negative literals, since they allow for a more compact representation.

Given a k-cut $C = \{l_1, \ldots, l_k\}$ of root i with cut function

$$\mathsf{FUNC}(i, C) = f(\hat{x}_1, \ldots, \hat{x}_k),$$

[1] In the algorithm we use 'Λ' to refer to a null value.

where $\hat{x}_i = x_{\mathsf{BEST}(l_i)}$ with corresponding level $\hat{\ell}_i$. If we are given an ESOP for f with m cubes, then each cube is translated into a tree of 2-input AND gates that is balanced with respect to the leaf levels. Then all outputs of these AND-trees are combined by a tree of 2-input XOR gates, which does not add to the multiplicative depth.

ALGORITHM 2: Tree balancing computation of product term j

Require: Product term $p_{1,j}, \ldots, p_{k,j}$, variables $\hat{x}_1, \ldots, \hat{x}_k$ at levels $\hat{\ell}_1, \ldots, \hat{\ell}_k$
  let Q be a priority queue of steps, ordered by the steps' level in ascending order
  for $i = 1, \ldots, k$ s.t. $p_{i,j} = 1$ **do**
    $push(Q, \hat{x}_i)$
  end for
  while $|Q| > 1$ **do**
    $u \leftarrow pop(Q)$
    $v \leftarrow pop(Q)$
    $push(Q, u \wedge v)$
  end while
  return $pop(Q)$

The algorithm to balance a non-constant ESOP cube with respect to the leaf levels is described in Algorithm 2. First, all non-empty literals are inserted into a priority queue Q according to their levels in ascending order. Then, as long as the queue has more than one element, the two top-most elements are popped from the queue and merged with an AND gate. The resulting step is then pushed back into the queue, taking into account the level of the step for the ordering.

3.3 ESPP Optimization

An **exclusive sum-of-pseudoproducts (ESPP** [25]) for an n-variable Boolean function $f(x_1, \ldots, x_n)$ has the form

$$f(x_1, \ldots, x_n) = \bigoplus_{j=1}^{m} \left(L_0^{p_{0,j}} \wedge \cdots \wedge L_{2^n-1}^{p_{2^n-1,j}} \right) \tag{6}$$

where $L_i = b_1 x_1 \oplus \cdots \oplus b_n x_n$ when $i = (b_n \ldots b_1)_2$ is the linear function (or parity function) that contains variables according to the positions of 1s in the binary expansion of i. The polarity variables $p_{i,j}$ play the same role as defined for ESOP forms, i.e., the parity function L_i in term j is negated if $p_{i,j} = 0$, used as is if $p_{i,j} = 1$, and omitted if $p_{i,j} = 2$. The terms in (6) are called *pseudoproducts* [32]. Note that each ESOP is also an ESPP, but an ESPP is only an ESOP if $(p_{i,j} \neq 2) \rightarrow (\nu(i) = 1)$ (where $\nu(i)$ is the sideways sum of i, i.e., the number of 1s in its binary expansion).

Ishikawa et al. [25] presented an exhaustive search algorithm to find small ESPPs, and some theoretical investigations on the form have been conducted [55]. However, to the best of our knowledge, no efficient heuristic optimization algorithm for ESPPs has been presented.

As a simple heuristic, we implemented a greedy algorithm to minimize the number of terms in an ESPP. The algorithm iteratively merges cubes to increase the use of linear functions as cube literals, thereby minimizing the number of AND operations. The algorithm starts with an initial ESPP form that corresponds to an ESOP form, extracted from a cut function. It then checks whether there exists two distinct terms with indices j_1 and j_2 such that there exist two indices $0 \leq i_1, i_2 < 2^n$ such that $p_{i_1,j_1} = p_{i_2,j_2} = 2$ but $p_{i_1,j_2} \neq 2$ and $p_{i_2,j_1} \neq 2$, and for all other indices $i \notin \{i_1, i_2\}$, it holds that $p_{i,j_1} = p_{i,j_2}$. Then, the two terms can be combined into a single term j, with $p_{i,j} = p_{i,j_1}$ for all

Table 1. Experimental Results for Applying ESOP-balancing

Benchmark	State-of-the-art [6, 14]			Min. MC baseline					Min. depth baseline				
	MC	MD	Run-time	MC (before)		MD (before)		Run-time	MC (before)		MD (before)		Run-time
Arithmetic functions [1]													
adder	16378	**9**	125.00	481	(128)	34	(128)	0.15	2761	(1742)	12	(14)	10.13
bar	4193	10	0.70	1303	(832)	**4**	(7)	0.33	3516	(3334)	8	(11)	2.42
div	190855	532	3731.00	158795	(5288)	973	(2243)	26.18	120327	(120327)	**523**	(620)	541.33
hyp	135433	15230	172000.00	120765	(56635)	4428	(8784)	166.07	780220	(417567)	**1287**	(1558)	324.31
log2	31573	129	94.00	34133	(10906)	**104**	(201)	778.78	83177	(33951)	114	(171)	130.33
max	7666	26	14.50	3839	(890)	93	(252)	1.81	8368	(4027)	**25**	(28)	4.17
multiplier	23059	57	30.73	15138	(7653)	65	(149)	13.50	39628	(28331)	**56**	(86)	77.72
sin	5507	74	4.50	6822	(2603)	62	(105)	9.45	14067	(6424)	**61**	(89)	58.87
sqrt	321555	2084	107814.00	71587	(5381)	951	(2167)	45.75	185061	(65762)	**769**	(936)	290.40
square	11306	26	12.50	6348	(4672)	59	(155)	8.41	10777	(14570)	**20**	(36)	38.75
Random control [1]													
arbiter	5183	**10**	43.00	3128	(1174)	13	(50)	2.10	7276	(6205)	11	(12)	1.35
cavlc	667	9	0.00	447	(394)	**7**	(11)	1.15	564	(576)	8	(10)	0.45
ctrl	109	5	0.00	54	(45)	**4**	(5)	0.10	77	(80)	4	(8)	0.06
dec	304	3	0.00	328	(328)	3	(3)	0.08	292	(292)	**3**	(3)	0.02
i2c	1213	7	0.10	816	(557)	**7**	(11)	0.87	1122	(1007)	7	(8)	0.37
int2float	216	7	0.00	104	(85)	**6**	(11)	0.87	184	(190)	7	(8)	0.13
mem_ctrl	54816	40	85.00	9983	(4695)	**14**	(39)	17.56	78044	(37519)	35	(41)	20.37
priority	876	102	0.50	442	(323)	11	(95)	1.08	522	(479)	**10**	(13)	0.28
router	198	11	0.00	116	(93)	**8**	(13)	0.10	227	(196)	10	(12)	0.19
voter	4288	30	112.42	7335	(4257)	26	(40)	31.95	3255	(6716)	**17**	(48)	6.14
Cryptographic functions [5]													
AES-128				8400	(6400)	**50**	(60)	5.49	33953	(85547)	80	(299)	65.52
AES-192				9408	(7168)	**60**	(72)	5.98	39533	(96979)	99	(359)	55.39
AES-256				11592	(8832)	**70**	(84)	7.49	53775	(120627)	123	(417)	90.26
Keccak-f				38400	(38400)	**24**	(24)	—	38630	(567395)	28	(266)	129.00
SHA-256				22573	(22573)	1607	(1607)	—	450447	(296951)	**1519**	(1936)	247.16
SHA-512				57947	(57947)	3303	(3303)	—	1988586	(831166)	**2383**	(2894)	1489.64
IEEE floating-point operations [5]													
FP-add				16721	(5384)	96	(235)	9.93	27541	(15879)	**64**	(83)	15.42
FP-div				3829444	(82265)	1646	(3619)	2994.35	732932	(200112)	**885**	(1157)	400.12
FP-eq				315	(315)	9	(9)	—	220	(336)	**9**	(10)	0.02
FP-f2i				3290	(1467)	24	(94)	3.01	3405	(2881)	**21**	(29)	3.41
FP-mul				23886	(19614)	92	(129)	14.78	62254	(47213)	**87**	(140)	54.51
FP-sqrt				4946577	(91499)	3763	(6507)	2981.18	893849	(264130)	**1877**	(2374)	506.28

$i \notin \{i_1, i_2\}$ and

$$p_{i_1 \oplus i_2, j} = [p_{i_1, j_2} = p_{i_2, j_1}] \tag{7}$$

if $p_{i_1 \oplus i_2, j_1} \in \{2, [p_{i_1, j_2} = p_{i_2, j_1}]\}$. If $p_{i_1 \oplus i_2, j_1} = 1 - [p_{i_1, j_2} = p_{i_2, j_1}]$, the two terms cancel and can be removed without adding another term to the ESPP. We iterate this procedure until no more such two terms can be found. In our implementation, empty parity functions are not explicitly stored, and therefore this procedure can be implemented efficiently.

3.4 Mapping to Quantum Circuit

Given a logic network over the gate basis $\{\wedge, \oplus, \neg\}$, it is straightforward to generate a quantum circuit that computes the same function: Each $\wedge$ node in the network can be mapped to a Toffoli

Table 2. Estimates for T-depth Optimized Quantum Circuits Obtained from Depth-optimized XAGs

Benchmark	T-count	T-depth	Qubits	Instance
Cryptographic functions				
AES-128	25600	60	7324	Min. MC baseline (ASAP)
AES-128	33600	50	9384	Min. MC opt (ASAP)
AES-192	28672	72	8156	Min. MC baseline (ASAP)
AES-192	37632	60	10456	Min. MC opt (ASAP)
AES-256	35328	84	9884	Min. MC baseline (ASAP)
AES-256	46368	70	12704	Min. MC opt (ASAP)
Keccak-f	153600	24	46400	Min. MC baseline (ASAP)
SHA-256	90292	1607	23684	Min. MC baseline (ASAP)
SHA-256	1801788	1519	458974	Min. MC opt (ASAP)
SHA-512	231788	3303	60448	Min. MC baseline (ASAP)
SHA-512	7954344	2383	2008595	Min. MC opt (ASAP)
IEEE floating-point operations				
FP-add	21384	235	5969	Min. MC baseline (ALAP)
FP-add	100832	64	28154	Min. MC opt (ASAP)
FP-div	290848	3604	81066	Min. MC baseline (ALAP)
FP-div	3054524	885	792188	Min. MC opt (ASAP)
FP-eq	880	9	655	Min. MC opt (ALAP)
FP-eq	1260	9	976	Min. MC baseline (ASAP)
FP-f2i	5832	94	1821	Min. MC baseline (ALAP)
FP-f2i	13620	21	4846	Min. MC opt (ASAP)
FP-mul	76368	118	26890	Min. MC baseline (ALAP)
FP-mul	249052	87	69347	Min. MC opt (ASAP)
FP-sqrt	315924	6498	84017	Min. MC baseline (ALAP)
FP-sqrt	3575396	1877	901087	Min. MC opt (ASAP)

We report quantum circuits that achieve the smallest number of qubits (first row) and the lowest T-depth (second row) over all T-depth optimized circuits.

that writes the output into an extra qubit starting in $|0\rangle$; each $\oplus$ and $\neg$ node can be computed inplace using a (controlled) NOT gate [36].

While the resulting quantum circuit computes the same function, a significant amount of parallelism is lost due to input-dependencies. As a remedy, we copy the inputs of those gates that can be executed in parallel, thus removing these dependencies [37].

4 EXPERIMENTAL RESULTS

We use various arithmetic and random-control functions from [1] as well as cryptographic functions and IEEE floating-point operations [5] as benchmarks for our algorithm. Our algorithm has been implemented in C++ on top of the EPFL logic synthesis libraries [58]. All experiments were run on a Microsoft Azure virtual machine, on a general purpose Standard D8s v3 size configuration, running on an Intel Xeon Platinum 8171M 2.40GHz CPU with 32 GiB memory and Ubuntu 18.04.

We choose two different baselines as starting points, heavily optimized XAGs for low MC (Min. MC baseline) and heavily optimized AIGs (AND-inverter graph) for low (general) logic network depth (Min. depth baseline). The Min. MC baseline is obtained using the MC optimization

Table 3. Estimates from Related Work

Benchmark	T-count	T-depth	Qubits	Comment
AES-128 [28]	—	960	21854	1
AES-128 [22]	1060864	50688	984*	
AES-192 [22]	1204224	44352	1112*	
AES-256 [22]	1505280	59904	1336*	
AES-128 [29]	118580	7520	864*	2
AES-192 [29]	137060	6560	896*	2
AES-256 [29]	166320	8640	1232*	2
AES-128 [26]	54400	120	1785*	
AES-192 [26]	60928	120	2105*	
AES-256 [26]	75072	126	2425*	
Keccak-f [4]	24640	33	3200*	
SHA-256 [28]	—	30336	938*	1
SHA-256 [4]	228992	70400	2402*	
FP-add [23]	26348	7224	268*	3
FP-mul [23]	122752	52116	315*	3

A *indicates that this value is better compared to the best value reported in Table 2.
[1]Authors report no Toffoli-count or T-count; T-depth is derived from reported Toffoli-depth by multiplication with 3 [3]; authors report six different candidates, from which we picked the one with the best T-depth.
[2]T-count and T-depth are derived from reported Toffoli-count and Toffoli-depth in the paper.
[3]The floating-point designs in the paper are not IEEE-compliant and do not account for special cases or denormalized numbers.

algorithm from the work by Testa et al. [64].[2] The Min. MC baseline is obtained by calling the ABC [12] optimization scripts `resyn2rs` (depth-preserving size optimization [38, 41]), followed by `if -K 6 -y` (AIG depth optimization [67]), followed by another round of `resyn2rs`, each run until depth is no longer improved.

4.1 Multiplicative-depth Optimization

As a first step, we apply ESOP-balancing with a cut size of 6 and exorcism [42] to obtain ESOPs for the cut functions to the chosen benchmarks for both baselines. We call the algorithm repeatedly until no further reduction in the multiplicative depth can be obtained. We report the results in Table 1. For the EPFL benchmarks we list the currently best-known results for multiplicative depth obtained from the state-of-the-art multiplicative depth optimization approach in [6, 14]. That approach has not been applied to the cryptographic and floating-point operations. For each baseline we list MC and MD after optimization together with the initial values in parentheses, as well as the runtime in seconds. The result with the lowest MD is highlighted in bold; in case of a tie we compare MC as a second cost metric.

Our algorithm successfully improves the best-known results in 18 out of 20 cases. For the arithmetic functions, the largest MD reductions were obtained when applying our approach to the Min. depth baseline, whereas for the random control functions, the Min. MC baseline turns out to be the better starting point. Note that in some cases (e.g., *hyp* and *priority*) we obtain a 10× improvement over the state of the art. For the cryptographic and floating-point functions, we can

[2]The cryptographic and floating-point operations were not further optimized, as they are already optimized for MC.

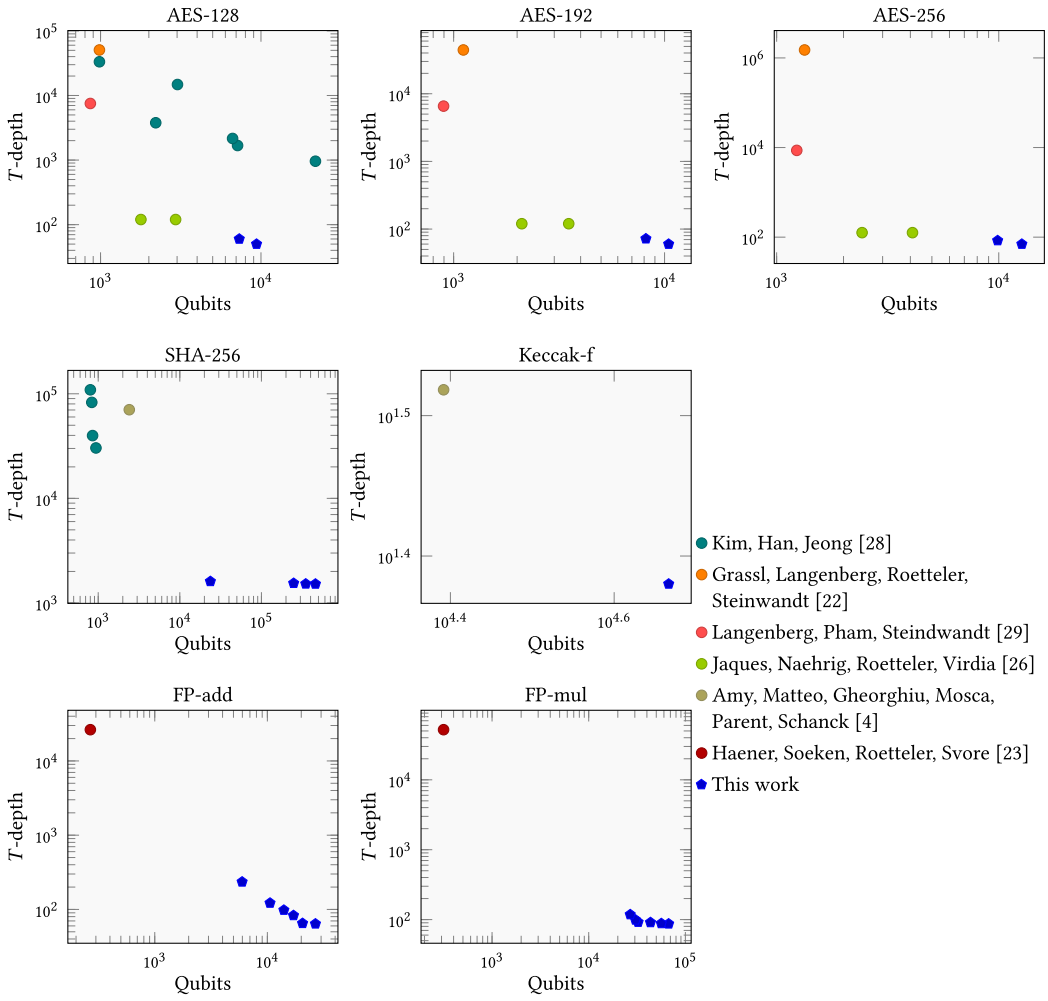

Fig. 3. These plots contain various resource estimates that can be found in the literature, together with all Pareto-optimal results for our approach that we obtained during the experimental evaluation. These include also results from intermediate optimization steps.

improve the MD compared to both baselines for all benchmarks except for *Keccak-f*. Because we use heavily-optimized networks as the baseline, we do not expect large gains for cryptographic functions, especially since MC and MD are important quantities in cryptography. In contrast, we find depth-reductions of up to 3× for floating-point operations (e.g., *FP-add* and *FP-f2i*) with only moderate increases in MC.

4.2 T-depth Optimization

In a second step, we map our depth-optimized XAGs to quantum circuits using two straightforward heuristics for upper-bounding the number of qubits: the ***as soon as possible* (ASAP)** heuristic computes all AND gates in parallel that have the same logic level and the ***as late as possible* (ALAP)** heuristic computes all AND gates in parallel that have the same reverse logic level. We present the resulting T-counts, T-depths, and qubit estimates in Table 2. For each cryptographic

function and floating-point operation, we report the two quantum circuits with the fewest number of qubits (first row) and the lowest T-depth (second row). The corresponding XAG and heuristic (ASAP or ALAP) is given in the last column. Compared to state-of-the-art manually-crafted quantum circuit designs, we achieve significant reductions in depth without drastic increases in qubit requirements. A comparison of our automatically-generated designs to a variety of state-of-the-art circuits for several cryptographic and floating-point functions is given in Figure 3, and we report the best known implementations with respect to T-depth explicitly in Table 3.

We note that, in addition to reduced circuit depths compared to the state of the art, our approach has the clear advantage that it is completely automatic. This stands in stark contrast to the circuits found in the literature, since those are manual designs that were not created by the push of a button.

5 CONCLUSIONS

In this work, we presented a dynamic programming algorithm to minimize the multiplicative depth of XAGs that makes use of cut enumeration, tree balancing, as well as ESOP and ESPP representations. Our results constitute significant improvements over state-of-the-art MD optimization algorithms [6, 14]. We used our algorithm to find quantum circuit implementations of various cryptographic and floating-point operations that improve the T-depth over state-of-the-art circuit designs.

Our ESOP-balancing algorithm was inspired by an existing algorithm for SOP-based balancing of Boolean logic networks. Since XOR gates corresponding to the outer XOR operator of the ESOP forms do not contribute to the depth, the resulting algorithm works well for reducing the multiplicative depth of XAGs in practice. In future work, we plan to investigate how this change in the cost function and underlying logic representation may benefit from alternative depth optimization algorithms such as MUX-based optimization [8, 39], generalized select transform algorithms [35, 51], or BDD-based techniques [15, 16].

We presented a post-optimization algorithm for ESOPs based on ESPPs, a generalization of ESOPs. XP^2 forms are a generalization of ESPPs and a minimization algorithm for such forms has been presented in [65]. We expect that such forms can help to further reduce the number of AND gates in the rebalancing step of our algorithm without increasing the multiplicative depth.

In classical optimization flows for logic synthesis, it is customary to interleave depth-optimization algorithms with size-optimization algorithms to obtain good trade-off points. We plan to adapt heuristic MC optimization algorithms in order to make them depth-preserving, i.e., allowing the minimization of AND gates only if the multiplicative depth does not increase. This allows for a reduction of the T-count and qubit count in the corresponding quantum circuits without an increase in T-depth.

REFERENCES

[1] Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Int'l Workshop on Logic and Synthesis*.

[2] Matthew Amy, Dmitri Maslov, and Michele Mosca. 2014. Polynomial-time T-depth optimization of clifford+t circuits via matroid partitioning. *IEEE Trans. on CAD of Integrated Circuits and Systems* 33, 10 (2014), 1476–1489. https://doi.org/10.1109/TCAD.2014.2341953

[3] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. 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 (2013), 818–830. https://doi.org/10.1109/TCAD.2013.2244643

[4] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. 2016. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *Int'l Conf. on Selected Areas in Cryptography*. 317–337. https://doi.org/10.1007/978-3-319-69453-5_18

[5] David Archer, Victor Arribas Abril, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. 'Bristol Fashion' MPC circuits. Retrieved on Dec. 2021 from https://homes.esat.kuleuven.be/~nsmart/MPC/.
[6] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. 2020. Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of Boolean circuits. In *The Cryptographers' Track at the RSA Conference*. 345–363. https://doi.org/10.1007/978-3-030-40186-3_15
[7] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Physical Review A* 52, 5 (1995), 3457.
[8] C. Leonard Berman, David J. Hathaway, Andrea S. LaPaugh, and Louise H. Trevillyan. 1990. Efficient techniques for timing correction. In *Int'l Symp. on Circuits and Systems*. 415–419. https://doi.org/10.1109/ISCAS.1990.112064
[9] Joan Boyar, Philip Matthews, and René Peralta. 2013. Logic minimization techniques with applications to cryptology. *Journal of Cryptology* 26, 2 (2013), 280–312. https://doi.org/10.1007/s00145-012-9124-7
[10] Daniel Brand and Tsutomu Sasao. 1993. Minimization of AND-EXOR expressions using rewrite rules. *IEEE Trans. on Computers* 42, 5 (1993), 568–576. https://doi.org/10.1109/12.223676
[11] Robert K. Brayton, Gary D. Hachtel, and Alberto L. Sangiovanni-Vincentelli. 1990. Multilevel logic synthesis. *Proc. IEEE* 78, 2 (1990), 264–300.
[12] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification*. 24–40. https://doi.org/10.1007/978-3-642-14295-6_5
[13] Benjamin J. Brown, Katharina Laubscher, Markus S. Kesselring, and James R. Wootton. 2017. Poking holes and cutting corners to achieve Clifford gates with the surface code. *Physical Review X* 7, 2 (2017), 021029. https://doi.org/10.1103/PhysRevX.7.021029
[14] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey. 2017. A multi-start heuristic for multiplicative depth minimization of Boolean circuits. In *Int'l Workshop on Combinatiorial Algorithms*. 275–286. https://doi.org/10.1007/978-3-319-78825-8_23
[15] Lei Cheng, Deming Chen, and Martin D. F. Wong. 2007. DDBDD: Delay-driven BDD synthesis for FPGAs. In *Design Automation Conference*. 910–915. https://doi.org/10.1145/1278480.1278705
[16] Mihir R. Choudhury and Kartik Mohanram. 2010. Bi-decomposition of large Boolean functions using blocking edge graphs. In *Int'l Conf. on Computer-Aided Design*. 586–591. https://doi.org/10.1109/ICCAD.2010.5654210
[17] Stelvio Cimato, Valentina Ciriani, Ernesto Damiani, and Maryam Ehsanpour. 2019. An OBDD-based technique for the efficient synthesis of garbled circuits. In *Int'l Workshop on Security and Trust Management (Lecture Notes in Computer Science)*, Sjouke Mauw and Mauro Conti (Eds.), Vol. 11738. Springer, 158–167. https://doi.org/10.1007/978-3-030-31511-5_10
[18] Jason Cong, Chang Wu, and Yuzheng Ding. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Int'l Symp. on Field Programmable Gate Arrays*. 29–35. https://doi.org/10.1145/296399.296425
[19] Rolf Drechsler. 1999. Preudo-Kronecker expressions for symmetric functions. *IEEE Trans. on Computers* 48, 9 (1999), 987–990. https://doi.org/10.1109/12.795226
[20] Magnus Gausdal Find. 2014. On the complexity of computing two nonlinearity measures. In *Int'l Computer Science Symposium in Russia*. 167–175. https://doi.org/10.1007/978-3-319-06686-8_13
[21] Craig Gidney. 2018. Halving the cost of quantum addition. *Quantum* 2 (2018), 74. https://doi.org/10.22331/q-2018-06-18-74 arXiv:1709.06648v3.
[22] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. 2016. Applying Grover's algorithm to AES: Quantum resource estimates. In *Int'l Workshop on Post-Quantum Cryptography*. 29–43. https://doi.org/10.1007/978-3-319-29360-8_3
[23] Thomas Häner, Mathias Soeken, Martin Roetteler, and Krysta M. Svore. 2018. Quantum circuits for floating-point arithmetic. In *Int'l Conf. on Reversible Computation*. 162–174. https://doi.org/10.1007/978-3-319-99498-7_11
[24] Martin Helliwell and Marek A. Perkowski. 1988. A fast algorithm to minimize multi-output mixed-polarity generalized reed-muller forms. In *Design Automation Conference*. 427–432. http://portal.acm.org/citation.cfm?id=285730.285799.
[25] Ryoji Ishikawa, Takashi Hirayama, Goro Koda, and Kensuke Shimizu. 2004. New three-level Boolean expression based on EXOR gates. *IEICE Trans. on Information & Systems* 87-D, 5 (2004), 1214–1222. http://search.ieice.org/bin/summary.php?id=e87-d_5_1214.
[26] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. 2019. Implementing Grover oracles for quantum key search on AES and LowMC. *arXiv preprint arXiv:1910.01700* (2019).
[27] Cody Jones. 2013. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A* 87, 2 (2013), 022328.
[28] Panjin Kim, Daewan Han, and Kyung Chul Jeong. 2018. Time-space complexity of quantum search algorithms in symmetric cryptanalysis: Applying to AES and SHA-2. *Quantum Information Processing* 17, 12 (2018), 339. https://doi.org/10.1007/s11128-018-2107-3

[29] Brandon Langenberg, Hai Pham, and Rainer Steinwandt. 2020. Reducing the cost of implementing the advanced encryption standard as a quantum circuit. *IEEE Trans. on Quantum Engineering* 1 (2020), 1–12.
[30] Leonid A. Levin. 2003. The tale of one-way functions. *Problems of Information Transmission* 39, 1 (2003), 92–103. https://doi.org/10.1023/A%3A1023634616182
[31] Daniel Litinski and Felix von Oppen. 2018. Lattice surgery with a twist: Simplifying Clifford gates of surface codes. *Quantum* 2 (2018), 62. https://doi.org/10.22331/q-2018-05-04-62
[32] Fabrizio Luccio and Linda Pagli. 1999. On a new Boolean function with applications. *IEEE Trans. on Computers* 48, 3 (1999), 296–310. https://doi.org/10.1109/12.754996
[33] Igor L. Markov and Mehdi Saeedi. 2012. Constant-optimized quantum circuits for modular multiplication and exponentiation. *Quantum Information and Computation* 12, 5&6 (2012), 361–394.
[34] Igor L. Markov and Mehdi Saeedi. 2013. Faster quantum number factoring via circuit synthesis. *Physical Review A* 87, 1 (2013), 012310. https://doi.org/10.1103/PhysRevA.87.012310
[35] Patrick C. McGeer, Robert K. Brayton, Alberto L. Sangiovanni-Vincentelli, and Sartaj Sahni. 1991. Performance enhancement through the generalized bypass transform. In *Int'l Conf. on Computer-Aided Design*. 184–187. https://doi.org/10.1109/ICCAD.1991.185226
[36] Giulia Meuli, Mathias Soeken, Earl Campbell, Martin Roetteler, and Giovanni De Micheli. 2019. The role of multiplicative complexity in compiling low T-count oracle circuits. In *Int'l Conf. on Computer-Aided Design*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942093
[37] Giulia Meuli, Mathias Soeken, Martin Roetteler, and Giovanni De Micheli. 2020. Enumerating optimal quantum circuits using spectral classification. In *Int'l Symp. on Circuits and Systems*.
[38] Alan Mishchenko and Robert K. Brayton. 2006. Scalable logic synthesis using a simple circuit structure. In *Int'l Workshop on Logic and Synthesis*. 15–22.
[39] Alan Mishchenko, Robert K. Brayton, and Stephen Jang. 2010. Global delay optimization using structural choices. In *Int'l Symp. on Field Programmable Gate Arrays*. 181–184. https://doi.org/10.1145/1723112.1723144
[40] Alan Mishchenko, Robert K. Brayton, Stephen Jang, and Victor N. Kravets. 2011. Delay optimization using SOP balancing. In *Int'l Conf. on Computer-Aided Design*. 375–382. https://doi.org/10.1109/ICCAD.2011.6105357
[41] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. 2006. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *Design Automation Conference*. 532–535. https://doi.org/10.1145/1146909.1147048
[42] Alan Mishchenko and Marek A. Perkowski. 2001. Fast heuristic minimization of exclusive-sum-of-products. In *Reed-Muller Workshop*.
[43] Saburo Muroga. 1993. Logic synthesizers, the transduction method and its extension, SYLON. In *Logic Synthesis and Optimization*, Tsutomu Sasao (Ed.). Springer, 59–86.
[44] Philipp Niemann, Anshu Gupta, and Rolf Drechsler. 2019. T-depth optimization for fault-tolerant quantum circuits. In *Int'l Symp. on Multiple-Valued Logic*. 108–113. https://doi.org/10.1109/ISMVL.2019.00027
[45] Joe O'Gorman and Earl T. Campbell. 2017. Quantum computation with realistic magic-state factories. *Physical Review A* 95, 3 (2017), 032338. https://doi.org/10.1103/PhysRevA.95.032338
[46] George K. Papakonstantinou. 2014. A parallel algorithm for minimizing ESOP expressions. *Journal of Circuits, Systems, and Computers* 23, 1 (2014). https://doi.org/10.1142/S0218126614500157
[47] John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79. *arXiv preprint arXiv:1801.00862v3*.
[48] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. 2017. Elucidating reaction mechanism on quantum computers. *Proceedings of the National Academy of Sciences* 114, 29 (2017), 7555–7560.
[49] M. Sadegh Riazi, Mojan Javaheripi, Siam U. Hussain, and Farinaz Koushanfar. 2019. MPCircuits: Optimized circuit generation for secure multi-party computation. In *Int'l Symp. on Hardware-Oriented Security and Trust*. 198–207. https://doi.org/10.1109/HST.2019.8740831
[50] Heinz Riener, Rüdiger Ehlers, Bruno Schmitt, and Giovanni De Micheli. 2020. Exact synthesis of ESOP forms. In *Advanced Boolean Techniques*, Rolf Drechsler and Mathias Soeken (Eds.). Springer. *arXiv preprint arXiv:1807.11103* (2020).
[51] Alexander Saldanha, Heather Harkness, Patrick C. McGeer, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. 1994. Performance optimization using exact sensitization. In *Design Automation Conference*. 425–429. https://doi.org/10.1145/196244.196448
[52] Tsutomu Sasao. 1993. AND-EXOR expressions and their optimization. In *Logic Synthesis and Optimization*, Tsutomu Sasao (Ed.). Kluwer Academic.
[53] Tsutomu Sasao and Philipp Besslich. 1990. On the complexity of mod-2 sum PLA's. *IEEE Trans. on Computers* 39, 2 (1990). https://doi.org/10.1109/12.45212
[54] Claus-Peter Schnorr. 1988. The multiplicative complexity of Boolean functions. In *Int'l Conf. on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. 45–58. https://doi.org/10.1007/3-540-51083-4_47

[55] Svetlana Nikolaevna Selezneva. 2014. On the length of Boolean functions in the class of Exclusive-OR sums of pseudoproducts. *Moscow University Computational Mathematics and Cybernetics* 38, 2 (2014), 64–68. https://doi.org/10.3103/S0278641914020083

[56] Peter Selinger. 2013. Quantum circuits of T-depth one. *Physical Review A* 87 (2013), 042302.

[57] Vivek V. Shende, Aditya K. Prasad, Igor L. Markov, and John P. Hayes. 2003. Synthesis of reversible logic circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems* 22, 6 (2003), 710–722. https://doi.org/10.1109/TCAD.2003.811448

[58] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, and Giovanni De Micheli. 2018. The EPFL logic synthesis libraries. *arXiv preprint arXiv:1805.05121v2* (2018).

[59] Mathias Soeken and Martin Roetteler. 2020. Quantum circuits for functionally controlled NOT gates. *arXiv preprint arXiv:2005.12310* (2020).

[60] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. 2019. LUT-based hierarchical reversible logic synthesis. *IEEE Trans. on CAD of Integrated Circuits and Systems* 38, 9 (2019), 1675–1688. https://doi.org/10.1109/TCAD.2018.2859251

[61] Stergios Stergiou, Konstantinos Daskalakis, and George K. Papakonstantinou. 2004. A fast and efficient heuristic ESOP minimization algorithm. In *ACM Great Lakes Symposium on VLSI*. 78–81. https://doi.org/10.1145/988952.988971

[62] Eleonora Testa, Mathias Soeken, Luca Gaetano Amarù, and Giovanni De Micheli. 2019. Logic synthesis for established and emerging computing. *Proc. IEEE* 107, 1 (2019), 165–184. https://doi.org/10.1109/JPROC.2018.2869760

[63] Eleonora Testa, Mathias Soeken, Luca G. Amarù, and Giovanni De Micheli. 2019. Reducing the multiplicative complexity in logic networks for cryptography and security applications. In *Design Automation Conference*. 74. https://doi.org/10.1145/3316781.3317893

[64] Eleonora Testa, Mathias Soeken, Heinz Riener, Luca Gaetano Amarù, and Giovanni De Micheli. 2020. A logic synthesis toolbox for reducing the multiplicative complexity in logic networks. In *Design, Automation and Test in Europe*.

[65] Ajay K. Verma, Philip Brisk, and Paolo Ienne. 2008. XP^2: A new compact representation for manipulating arithmetic circuits. In *Int'l Workshop on Logic and Synthesis*.

[66] Vera von Burg, Guang Hao Low, Thomas Häner, Damian S. Steiger, Markus Reiher, Martin Roetteler, and Matthias Troyer. 2020. Quantum computing enhanced computational catalysis. *arXiv preprint arXiv:2007.14460* (2020).

[67] Wenlong Yang, Lingli Wang, and Alan Mishchenko. 2012. Lazy man's logic synthesis. In *Int'l Conf. on Computer-Aided Design*. 597–604. https://doi.org/10.1145/2429384.2429513

Received May 2021; revised September 2021; accepted November 2021