# Constructing Quantum Implementations with the Minimal *T*-depth or Minimal Width and Their Applications

Zhenyu Huang[1,2]*, Fuxin Zhang[1,2], and Dongdai Lin[1,2]

[1] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, China
`{huangzhenyu, zhangfuxin, ddlin}@iie.ac.cn`

**Abstract.** With the rapid development of quantum computers, optimizing the quantum implementations of symmetric-key ciphers, which constitute the primary components of the quantum oracles used in quantum attacks based on Grover and Simon's algorithms, has become an active topic in the cryptography community. In this field, a challenge is to construct quantum circuits that require the least amount of quantum resources. In this work, we aim to address the problem of constructing quantum circuits with the minimal $T$-depth or width (number of qubits) for nonlinear components, thereby enabling implementations of symmetric-key ciphers with the minimal $T$-depth or width. Specifically, we propose several general methods for obtaining quantum implementation of generic vectorial Boolean functions and multiplicative inversions in $\mathbb{F}_{2^n}$, achieving the minimal $T$-depth and low costs across other metrics. As an application, we present a highly compact $T$-depth-3 Clifford+$T$ circuit for the AES S-box. Compared to the $T$-depth-3 circuits presented in previous works (ASIACRYPT 2022, IEEE TC 2024), our circuit has significant reductions in $T$-count, full depth and Clifford gate count. Compared to the state-of-the-art $T$-depth-4 circuits, our circuit not only achieves the minimal $T$-depth but also exhibits reduced full depth and closely comparable width. This leads to lower costs for the DW-cost and T-DW-cost. Additionally, we propose two methods for constructing minimal-width implementations of vectorial Boolean functions. As applications, for the first time, we present a 9-qubit Clifford+$T$ circuit for the AES S-box, a 16-qubit Clifford+$T$ circuit for a pair of AES S-boxes, and a 5-qubit Clifford+$T$ circuit for the $\chi$ function of SHA3. These circuits can be used to derive quantum circuits that implement AES or SHA3 without ancilla qubits.

**Keywords:** Quantum Circuit, $T$-depth, Width, AES, SHA3

* corresponding author

## 1 Introduction

In recent years, the potential arrival of a large-scale quantum computer has led to heightened examination of the post-quantum security of cryptographic primitives. It is well known that if large-scale quantum computers are ever built Shor's algorithm [48] will completely break many widely employed public-key schemes such as RSA, DSA, and ECC. For symmetric-key schemes, Grover's algorithm [24], which provides a quadratic speedup for unstructured database search, can be used to perform quantum key search attacks to block ciphers and quantum pre-image attacks to hash functions. Additionally, various quantum attacks on symmetric-key schemes have been developed using Simon's period-finding algorithm, under different attack scenarios. The first scenario involves attackers having access to keyed quantum oracles [11,32], while the second scenario allows only classical queries and offline quantum computations [10, 17, 26]. Furthermore, quantum attacks stemming from dedicated cryptanalytic techniques are extensively investigated [12–14, 25, 27, 33, 41].

To precisely estimate the complexities of these attacks, the quantum circuits implementing them should be constructed using a basic quantum gate set, such as the Clifford+$T$ gate set, which is a commonly used universal fault-tolerant gate set. For the quantum attacks to symmetric-key cipher based on Grover's and Simon's algorithm, primary components for the attack circuits are the quantum circuits that implement these ciphers. Optimizing these quantum circuits can reduce the attack complexities, and provide the designers with precise insights into the costs for evaluating security margins. Moreover, in the National Institute of Standards and Technology (NIST)'s call for proposals for post-quantum cryptography standards, the gate complexities of the quantum attacks to AES and SHA3 are used as different baselines to categorize the different security levels for the post-quantum public-key schemes, with a suggested attack scenario that the depth of the attack circuit is limited by a `MAXDEPTH` [44, 45]. For these reasons, synthesizing and optimizing quantum circuits for various ciphers, especially for AES, is receiving significant attention in our community.

**Related Work.** In [23], Grassl et al. presented the first quantum circuit for AES, where reducing the width (the number of qubits) of the circuit is their primary optimization goal. Then slight improvement on the width and Toffoli-count of the quantum circuit for AES was provided in [2]. In [34], quantum circuits for the AES S-box derived from efficient classical circuits were presented, resulting in a substantial improvement to the AES circuit. At ASIACRYPT 2020, Zou et al. presented further optimized quantum circuits for the AES S-box and an improved zig-zag structure. Based on these, they presented a quantum circuit for AES-128 with width 512 [56]. As depth and $T$-depth are important cost metrics which determine the runtime of the quantum circuits, at EUROCRYPT 2020, Jaques et al. investigated the optimization of depth and $T$-depth, and presented a shallow quantum circuit for AES [31]. At ASIACRYPT 2022 [28], Huang and Sun proposed some general techniques for designing in-place quantum circuits and low $T$-depth quantum circuits, by which they improved the quantum circuits presented in [31] and [56]. At ASIACRYPT 2023 [39], Liu et al. proposed two methods to optimize the $T$-depth-4 circuit for the AES S-box proposed in [28], and together with a modified pipeline structure, they further improved the quantum circuits for AES. Recently, the individual works in [47, 54] both achieved similar results by reducing the depth of the CNOT circuit for MixColumn of AES and the $T$-depth of the AES encryption oracle. Additionally, there are other works focused on optimizing the quantum circuits for AES [30,36,37], and some works exploring the quantum implementation for other schemes [4,5,31].

Although many previous works investigated reducing the quantum resources of the quantum circuits for AES and other schemes, constructing quantum circuits that achieve the theoretical minimum for a certain metric, particularly width or $T$-depth, has not been thoroughly discussed, and efficient general approaches to this problem are still lacking. For the implementation of AES, the only known result regarding the minimal-width circuit was presented in [23], where it was announced that a 9-qubit in-place quantum circuit for the AES S-box with no more than 9695 $T$ gates and 12631 Clifford gates can be found, but the implementation of the circuit was not presented in detail. For the minimal-$T$-depth implementations, in [28], an out-of-place quantum circuit for the AES S-box with $T$-depth 3 was achieved using heuristic techniques, and it was later improved through manual optimizations in [54].

**Our Contributions.** We present a general algorithm for converting an AND-depth-$s$ classical circuit to a highly compact out-of-place quantum circuit with $T$-depth $s$. Our algorithm ensures that the number of ancilla qubits required for applying each $T$-layer is theoretically minimal. Based on this algorithm, we can resolve the problem of constructing out-of-place quantum circuits with minimal $T$-depth by constructing classical circuits with minimal AND-depth.

We introduce a general approach, termed the Top-down approach, which enables the construction of minimal AND-depth classical circuits for vectorial Boolean functions based solely on their algebraic normal forms. As an application, we construct a classical circuit implementing the AES S-box with AND-depth 3 and AND-count 76. Compared to the AND-depth-3 classical circuit given in ASIACRYPT 2022, this circuit reduces the AND-count from 78 to 76.

We further investigate implementing multiplicative inversions in $\mathbb{F}_{2^n}$, by using a new concept called the parallel addition chain. We present some generic results for implementing multiplicative inversions in $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^{2m}}$ with minimal AND-depth. As an application, we construct a classical circuit for the AES S-box with AND-depth 3 and AND-count 42. Furthermore, by using our conversion algorithm, we achieve a highly compact $T$-depth-3 circuit for the AES S-box. Compared to the $T$-depth-3 circuit presented in [54], this circuit reduces the $T$-count by 36%, width by 25%, and other gate-counts by 28%.

We propose two methods to construct the *in-place* quantum circuits for vectorial Boolean functions with width achieving theoretical minimum. The first one combines the method for finding MCT decompositions of permutations proposed in [35] with the Clifford+$T$ decompositions of MCT gates introduced in [1]. As an application, we present, for the first time, a 9-qubit Clifford+$T$ circuit for the AES S-box. Compared to the $T$-count and Clifford gate-count reported in [23], our circuit achieves a reduction of 59% in the $T$-count and 41% in the Clifford gate-count. The second method is a SAT-based method, which can efficiently obtain the minimal-width circuits for 4-bit S-boxes and some 5-bit S-boxes. As an application, we present, for the first time, a 5-qubit Clifford+$T$ circuit for the $\chi$ function of SHA3. This circuit uses 49 $T$ gates, decomposed from 7 Toffoli gates, and this Toffoli count closely approaches the theoretical minimum of 5 (without width restrictions).

## 2 Preliminaries

For an $n$-qubit quantum system, its state, typically written as $|u\rangle$, can be described by a unit vector in $\mathbb{C}^{2^n}$. A quantum algorithm operates on the state of an $n$-qubit system by applying a series of unitary transformations and measurements, where a unitary transformation is a linear map $U$ over $\mathbb{C}^{2^n}$ with $UU^\dagger = U^\dagger U = I$. A quantum circuit is a widely used model for quantum computation. The building blocks of a quantum circuit are quantum gates, which are basic unitary transformations operating on a small number of qubits.

In this work, we are mainly concerned with the quantum circuits that can implement vectorial Boolean functions. More specifically, for a given vectorial Boolean function $\mathcal{F}$, our goal is to construct a quantum circuit capable of producing $|\mathcal{F}(x)\rangle$ as a component of its output when the input is in the computational basis $|x\rangle$. This circuit often requires some ancilla qubits as intermediate storage space, which should be returned to their initial states (usually $|0\rangle$) at the end of the circuit. There are the following two types of quantum circuits that can implement $\mathcal{F}$. Here $|0\rangle$ denotes possible ancilla qubits initialized as $|0\rangle$.

1) A circuit that maps $|x\rangle|y\rangle|0\rangle$ to $|x\rangle|y \oplus \mathcal{F}(x)\rangle|0\rangle$. It is called an *out-of-place* circuit for $\mathcal{F}$.
2) A circuit that maps $|x\rangle|0\rangle$ to $|\mathcal{F}(x)\rangle|0\rangle$, if $\mathcal{F}$ is invertible. It is called an *in-place* circuit for $\mathcal{F}$.

Sometimes (for example, for implementing the AES S-box in `SubBytes`), we only need an out-of-place implementation that works when $|y\rangle = |0\rangle$, which means it maps $|x\rangle|0\rangle|0\rangle$ to $|x\rangle|\mathcal{F}(x)\rangle|0\rangle$. Here, using the terminology introduced in [28], we refer to this circuit as a $\mathfrak{C}^0$-circuit for $\mathcal{F}$. Meanwhile, an out-of-place circuit that works for any $|y\rangle$ is called a $\mathfrak{C}^*$-circuit for $\mathcal{F}$.

For synthesizing a quantum circuit, a commonly used universal fault-tolerant gate set is the Clifford+$T$ gate set, which contains the Clifford gates and the $T$ gate. The Clifford gates used to implement vectorial Boolean functions usually include the Hadamard gate $H$, the Phase gate $S$, the CNOT gate, and the Pauli-X gate, which is referred to as the NOT gate in this paper.

A typical way to obtain a Clifford+$T$ implementation for a vectorial Boolean function is first obtaining its *NCT implementation*, which is a quantum circuit composited by gates in the gate set {NOT, CNOT, Toffoli}, then decomposing each Toffoli gate into a Clifford+$T$ sub-circuit.[3] An NCT circuit can be also

[3] Different Clifford+$T$ circuits for the Toffoli gate are presented in Appendix A

seen as a classical reversible circuit, since the CNOT and the Toffoli gate are reversible versions of the classical XOR and AND gate.

In some cases, the functionality of a classical AND gate can be implemented by a *quantum AND gate* (QAND), which maps $|a\rangle|b\rangle|0\rangle$ to $|a\rangle|b\rangle|ab\rangle$. Compared to the Toffoli gate, which maps $|a\rangle|b\rangle|c\rangle$ to $|a\rangle|b\rangle|c \oplus ab\rangle$, the quantum AND gate only works when the target qubit is $|0\rangle$, hence can be seen as a $\mathfrak{C}^0$-circuit for bitwise multiplication. As shown in [31], QAND can be implemented by a Clifford+$T$ circuit with $T$-depth 1 and width 4, while QAND$^\dagger$ can be implemented without $T$ gates by using a measurement. In comparison, the $T$-depth-1 Clifford+$T$ implementation for the Toffoli gate has width 7. Readers can refer to Appendix A for the specific implementations of QAND and QAND$^\dagger$. By using this implementation for QAND$^\dagger$, as in [28, 39, 54], the Clifford+$T$ circuits presented in this work may contain measurements.

**Quantum Circuit Complexity.** The complexity of a quantum circuit can be measured by various metrics, such as width, gate-count, and depth. The *width* of a quantum circuit is defined as the number of qubits used in the circuit, including the ancilla qubits. It can intuitively indicate whether the quantum circuit can be performed on a certain quantum computer. The *gate-count* of a quantum circuit is the total number of gates used in the circuit. It quantifies the computational complexity of a quantum circuit, and provides insights into the potential execution time of a quantum circuit. For a Clifford+$T$ circuit, since the high cost of the fault-tolerant implementation of a $T$ gate, *$T$-count*, the total number of $T$ and $T^\dagger$ gates, is a much more significant metric.

In this work, we assume parallel quantum gates are allowed when they are acting on different qubits. In this case, the runtime of a quantum circuit is primarily determined by its *depth*, also referred to as *full depth* or *overall depth*. For a Clifford+$T$ circuit, its runtime is dominated by its *$T$-depth*. There are two equivalent definitions for the depth ($T$-depth) of a circuit:

1) the maximum number of gates ($T$ and $T^\dagger$ gates) on its all critical paths;
2) the minimum number of layers for parallel applications of gates ($T$ and $T^\dagger$ gates) in the circuit.

The first definition can be used to fast calculate the depth ($T$-depth) of a given circuit. In contrast, the second definition is more commonly seen and more appropriate for quantum computation, since actually we want to know how to parallelize the gates in each layer, which indicates how the qubits are manipulated at each stage. Note that, there are different ways of parallelizing gates in each layer, leading to different numbers of layers. In most cases, when the number of $T$-layers equals the $T$-depth, the number of all layers will not equal the full depth.

Besides these basic metrics, some other composite metrics were also considered in previous works [28, 31, 39]. Denote the width, depth, and gate-count of a circuit as $\mathcal{W}, \mathcal{D}, \mathcal{G}$ respectively. The DW-cost, defined as $\mathcal{D} \cdot \mathcal{W}$, is a metric which can indicate the total cost for quantum error correction [31]. $\mathcal{G} \cdot \mathcal{D}$ and $\mathcal{W} \cdot \mathcal{D}^2$ are metrics that indicate the gate-count and DW-cost for parallel Grover search under a depth limit, which is a suggested scenario for estimating the costs of quantum attacks in NIST's call for proposals for post-quantum cryptography standards. For these metrics, we can also choose to count only $T$ gates towards gate-count and depth.

For implementing a vectorial Boolean function, a Clifford+$T$ implementation is always obtained from an NCT implementation by decomposing Toffoli gates. For the NCT implementations, there are theoretical lower bounds for their width and the Toffoli-depth. These bounds can be seen as the lower bounds for the Clifford+$T$ implementations generated from NCT implementations. In this paper, we aim to construct Clifford+$T$ implementations that reach these lower bounds while simultaneously optimizing other metrics. Note that, linear vectorial Boolean functions can be implemented in-place by CNOT gates, and these implementations obviously have the minimal width and $T$-depth 0. Since such CNOT implementations have been sufficiently discussed in previous works [28,47,52,54], in this work, we focus on the above implementation problem for nonlinear vectorial Boolean functions.

### 2.1 Tools for Resource Estimation

The resource estimators in Q# [40] and ProjectQ [50] were widely used in previous works for quantum resource estimation [28,31,39,54]. Unfortunately, both of them have some errors in the value of width when they automatically combine small sub-circuits into an entire one. One more disadvantage for ProjectQ is that it cannot obtain the $T$-depth of a given Clifford+$T$ circuit.

To achieve correct values of all metrics, we developed a quantum resource estimator based on Python. Given a Clifford+$T$ circuit expressed by a sequence of specific gates and measurements in the QASM format, the estimator first optimizes the circuit by identifying simple cancellations (cancel two consecutive gates that are the inverse of each other, and replace two consecutive $T$ gates by an $S$ gate), then outputs the values of various metrics along with the layer structure of the optimized circuit. In the output layer structure, the number of $T$ layers is equal to the $T$-depth, and by this structure, one can know how to parallelize these gates. We also developed an NCT version of the estimator. Unless stated otherwise, the costs of the circuits presented in the following sections are obtained using this resource estimator. Moreover, the specific layer structures of all these quantum circuits are available at `https://github.com/hzy-cas/Minimal_T-depth_Width`.

## 3 Implementing Nonlinear Functions with the Minimal $T$-depth

Given a vectorial Boolean function $\mathcal{F}$, we consider the problem of implementing $\mathcal{F}$ with the minimal $T$-depth. In the following of this paper, for simplicity, when we say a Boolean function, we mean a vectorial Boolean function.

Since the minimal $T$-depth of out-of-place implementations is always not bigger than the minimal $T$-depth of in-place implementations (for some simple functions, these two $T$-depths may be equal), we focus on the *out-of-place implementations* for $\mathcal{F}$. For constructing an out-of-place implementation, a commonly used approach is first constructing a circuit $\mathsf{C}_0$, which maps $|x\rangle|y\rangle|0\rangle$ to $|x\rangle|y \oplus \mathcal{F}\rangle|g(x)\rangle$ with some garbage output $|g(x)\rangle$, then performing uncomputation to erase the garbage output. Here uncomputation can be implemented by the reverse circuit of $\mathsf{C}_0$ with deleting some gates involving the second register. In what follows, we refer to $\mathsf{C}_0$ as a *forward circuit* for $\mathcal{F}$, and the circuit for uncomputation as a *backward circuit* for $\mathcal{F}$.

By combining the forward and backward circuits, one can obtain an out-of-place circuit for $\mathcal{F}$ (without garbage outputs). Therefore, to achieve an out-of-place implementation with the minimal $T$-depth, a critical step is constructing a forward circuit with the minimal $T$-depth. Our strategy is first finding a forward NCT circuit with the minimal Toffoli-depth, then decomposing each Toffoli gate into a $T$-depth-1 Clifford+$T$ sub-circuit. In some forward NCT circuit, the target qubits of Toffoli gates are all initialized as $|0\rangle$, hence these Toffoli gates can be replaced by QAND gates. In this case, by using QAND$^\dagger$ gates instead of Toffoli gates in the backward circuit, we can obtain a backward circuit without $T$ gates, and the entire out-of-place circuit will have the same $T$-depth as the forward part.

According to Theorem 1 of [28], if we have a classical circuit that implements $\mathcal{F}$ with AND-depth $s$, then with sufficient ancilla qubits, we can construct a Toffoli-depth-$s$ forward circuit for $\mathcal{F}$, deriving a $T$-depth-$s$ forward circuit for $\mathcal{F}$. For the AND-depth of the classical circuit implementing a Boolean function, there is a well-known result.

**Lemma 1.** *Let* $\mathcal{F} = (f_1(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n))$ *be a Boolean function with* $\max_i(\deg(f_i)) = D$. *Let* $d$ *be the AND-depth for a classical circuit that implements* $\mathcal{F}$. *Then* $d \geq \lceil \log_2(D) \rceil$ *and there is a classical circuit that implements* $\mathcal{F}$ *with AND-depth* $\lceil \log_2(D) \rceil$.

*Proof.* With one AND layer, we can at most double the highest degree of all achieved polynomials, thus we have the bound $\lceil \log_2(D) \rceil$ for the number of AND layers. For constructing a classical implementation with AND-depth $\lceil \log_2(D) \rceil$, we can construct monomials with degrees in the range $[2^k + 1, 2^{k+1}]$ in the $k$-th AND layer, based on the monomials with degrees in the range $[2^{k-1}+1, 2^k]$. After $\lceil \log_2(D) \rceil$ AND layers, all monomials with degree $\leq D$ have been generated. Then we can construct all $f_i$'s from these monomials by using some XOR and NOT gates. □

Based on a classical circuit that implements $\mathcal{F}$ with AND-depth $\lceil \log_2(D) \rceil$, we can derive a forward NCT circuit for $\mathcal{F}$ with Toffoli-depth $\lceil \log_2(D) \rceil$, resulting in a Clifford+$T$ implementation with $T$-depth $\lceil \log_2(D) \rceil$. In this paper, a *minimal $T$-depth Clifford+$T$ circuit* for $\mathcal{F}$ means a Clifford+$T$ implementation having this $T$-depth, and our goal is constructing a minimal $T$-depth implementation with low width and low gate-count.

According to the above discussions, the process for constructing a minimal $T$-depth implementation can be divided into two steps.

1) Construct a classical minimal AND-depth implementation.
2) Convert it to a Clifford+$T$ circuit with the $T$-depth being equal to this minimal AND-depth.

The proof of Lemma 1 induces an approach for the first step, but the AND-count of the obtained classical circuit is too large for most instances. For the second step, if a straightforward conversion is adopted, the resulting quantum circuit will have a large width. In the following sections, we first consider how to improve the converting step.

## 3.1 Constructing a Compact $T$-depth-$s$ Quantum Circuit from an AND-depth-$s$ Classical Circuit

In [28], Huang and Sun introduced a way to convert an AND-depth-$s$ classical circuit that implements a Boolean function $\mathcal{F}$ into a $T$-depth-$s$ forward circuit for $\mathcal{F}$. Its main steps can be outlined as follows. First calculate the AND-depth of all gates and regroup gates by their AND depths. Then, for converting the AND gates with the same AND-depth to parallelized Toffoli (or QAND) gates, copy their common inputs to new allocated qubits. Finally, decompose Toffoli (or QAND) gates into $T$-depth-1 Clifford+$T$ sub-circuits.

For this conversation, we can observe that the $T$-count and $T$-depth of the obtained circuit are determined by the AND-count and AND-depth of the original classical circuit. To reduce the width, gate-count, and depth of the circuit, we should focus on reconstructing the CNOT sub-circuits that generate the inputs of the parallelized Toffoli (or QAND) gates. Here we propose a general algorithm for this problem. For simplicity, we assume the given classical circuit does not contain NOT gates. If it does, we only need to add some NOT gates after the resulting CNOT circuits to modify the constant terms of the algebraic expressions computed by these CNOT circuits.

For a classical circuit, denote its inputs as Boolean variables $x_1, x_2, \ldots, x_n$, and the output of each AND gate as a new $x$ variable. By this manner, the inputs of all AND gates and the outputs of the circuit can be written as linear functions w.r.t. these $x$ variables. Then preparing the inputs of the AND gates, which have the same AND-depth, based on previous outputs, can be seen as generating a sequence of linear functions $\{T_1, T_2, \ldots, T_m\}$ from a given sequence of linear functions $\{L_1, L_2, \ldots, L_t\}$. For this process, we have the following lemma (the proof can be found in the Appendix B).

**Lemma 2.** *Let $\{L_1, L_2 \ldots, L_t\}$ and $\{T_1, T_2 \ldots, T_m\}$ be two sequences of linear functions with respect to Boolean variables $x_1, x_2, \ldots, x_n$. Suppose the rank of $L_1, L_2, \ldots, L_t$ is $n$, and the rank of $T_1, T_2, \ldots, T_m$ is $k$. If $|L_1, L_2, \ldots, L_t\rangle$ is the input of a $t$-qubit register, then to output the state $|T_1, T_2, \ldots, T_m\rangle$ using a CNOT circuit, $m - k - (t - n)$ additional qubits are necessary and sufficient. Additionally, if $m - k - (t - n) < 0$, it means no additional qubits are required. Instead, $t - n - m + k$ qubits can be returned to $|0\rangle$.*

Based on this lemma, we can calculate the minimal width of the CNOT sub-circuit between two Toffoli layers. Once the minimal width of a CNOT sub-circuit is determined, we can identify its input state $|S_1\rangle$, output state $|S_2\rangle$, and a Boolean matrix $M$ that maps $|S_1\rangle$ to $|S_2\rangle$. Furthermore, we can apply the state-of-the-art CNOT circuit optimization algorithms to find a compact CNOT sub-circuit that implements $M$. Finally, by combining all these CNOT sub-circuits with different Toffoli layers, we obtain the desired final circuit. We propose an algorithm called `ClassicalToQuantum`, which automatically implements these processes, hence can convert an AND-depth-$s$ classical circuit to a forward Toffoli-depth-$s$ NCT circuit with the optimal width and low CNOT-count. The specific steps with detailed explanations of this algorithm are given in the Appendix C. Here we present some new compact quantum circuits for the AES S-box [21] to demonstrate the effectiveness of this algorithm.

We take Boyar and Peralta's AND-depth-4 classical circuit [15] for the AES S-box as the input of `ClassicalToQuantum`. In [28], the first $T$-depth-4 quantum circuit for the AES S-box was designed based on this classical circuit. With manual optimizations, some $T$-depth-4 implementations with lower width and CNOT-count are given in [39, 54]. Using this input, `ClassicalToQuantum` can output a forward NCT $\mathfrak{C}^0$-circuit for the AES S-box with Toffoli-depth-4 and the optimal width 66. It can be easily extended to an entire $\mathfrak{C}^0$-circuit (including uncomputation). We can utilize the method proposed in [28] to convert it to a $\mathfrak{C}^*$-circuit by adding some CNOT gates. Table 1 compares the costs of these two circuits with the NCT circuits presented in [39]. Note that, these out-of-place circuits have Toffoli-depth 8, since they include the Toffoli-depth-4 backward circuits. We can illustrate that our circuits have lower costs for all critical metrics.

**Table 1.** Comparison of different out-of-place NCT circuits (including uncomputation) originated from Boyar and Peralta's AND-depth-4 classical circuit for the AES S-box.

| Type | #NOT | #CNOT | #Toffoli | Toffoli-depth | Full Depth | Width | Source |
|---|---|---|---|---|---|---|---|
| $\mathfrak{C}^*$ | 4 | 312 | 68 | 8 | 78 | 90 | [39] |
| $\mathfrak{C}^*$ | 4 | 368 | 68 | 8 | 105 | 76 | [39] |
| $\mathfrak{C}^0/\mathfrak{C}^*$ | 4 | **227/240** | **60** | 8 | **60** | **66** | This work |

By using QAND/QAND$^\dagger$ gates instead of Toffoli gates in the NCT $\mathfrak{C}^0$-circuit, we can obtain a Clifford+$T$ $\mathfrak{C}^0$-circuit for the AES S-box. Note that, to obtain a Clifford+$T$ $\mathfrak{C}^*$-circuit, we should slightly modify the last CNOT sub-circuit that generates the circuit outputs, as the target qubits of the QAND gate cannot be the same qubits used for the outputs (thus requiring additional ancilla qubits). The comparison of our Clifford+$T$ circuits with the Clifford+$T$ circuits presented in [39, 54] is shown in Table 2.

*Remark 1.* Some values presented in Table 2 and Table 3 (in Section 4.2) are slightly different from those presented in [39, 54]. The reason is that we utilize a more compact Clifford+$T$ implementation for QAND$^\dagger$, correct an error regarding the width of the $\mathfrak{C}^*$ Clifford+$T$ circuits (where the output qubits cannot serve as target qubits for QAND gates), account for the worst-case scenario for QAND$^\dagger$ (where measurement results consistently yield 1), and fully consider the impact of measurement feedback on the depth of QAND$^\dagger$ (since conditioned operations on the first two qubits cannot be applied before the measurement on the third qubit).

**Table 2.** Comparison of different out-of-place Clifford+$T$ circuits (including uncomputation) for the AES S-box, where #M means the number of measurements.

| Type | #CNOT | #1qClifford | #$T$ | #M | $T$-depth | Full Depth | Width | Source |
|---|---|---|---|---|---|---|---|---|
| $\mathfrak{C}^0/\mathfrak{C}^*$ | 618/608 | 220 | 136 | 34 | 4 | 109 | 99/107 | [39] |
| $\mathfrak{C}^0/\mathfrak{C}^*$ | 527/554 | 206/238 | 136 | 26/34 | 4 | 95/90 | 84/92 | [54] |
| $\mathfrak{C}^0/\mathfrak{C}^*$ | **525/539** | 206/238 | 136 | 26/34 | 4 | **93**/92 | 84/92 | This work |

The above instances demonstrate that, given a classical implementation with AND-depth $s$, our algorithm can construct a highly compact Clifford+$T$ implementation with $T$-depth $s$. Therefore, using this algorithm, to achieve a minimal $T$-depth implementation with low costs for other metrics, we can focus on constructing a minimal AND-depth classical circuit with low AND-count.

## 3.2 Constructing Classical Circuits with the Minimal AND-depth

Let $\mathcal{F} = (f_1, f_2, \ldots, f_n)$ be a Boolean function with $\max_i\{deg(f_i)\} = D$. As mentioned before, the proof of Lemma 1 induces a straightforward approach for constructing a minimal AND-depth classical circuit for $\mathcal{F}$. That is first constructing all monomials with degree $\leq D$ by $\lceil \log_2(D) \rceil$ AND-layers, then generating all $f_i$'s by XOR and NOT gates. For some Boolean functions with simple ANFs, this straightforward approach can yield an implementation with low AND-count. The following example shows how to construct a classical circuit for the $\chi$ function of SHA3 [7] with AND-depth 1 and AND-count 5 by this approach.

*Example 1.* (**A $T$-depth-1 circuit for the $\chi$ function of SHA3**) The $\chi$ function of SHA3 is $(f_1, f_2, f_3, f_4, f_5) = (x_1 + (x_2+1)x_3, x_2 + (x_3+1)x_4, x_3 + (x_4+1)x_5, x_4 + (x_5+1)x_1, x_5 + (x_1+1)x_2)$. Apparently, with 5 multiplications $x_2 \cdot x_3, x_3 \cdot x_4, x_4 \cdot x_5, x_5 \cdot x_1, x_1 \cdot x_2$, which can be applied by 5 AND gates in one AND-layer, we can generate all the quadratic monomials occurring in these $f_i$'s. Then, using 10 XOR gates, we can construct $f_1, f_2, f_3, f_4, f_5$ from $x_1, x_2, x_3, x_4, x_5, x_2x_3, x_3x_4, x_4x_5, x_5x_1, x_5x_2$. This classical circuit can be easily converted to an out-of-place Clifford+$T$ circuit that implements the $\chi$ function with $T$-depth 1, $T$-count 20, and width 20.

For complex problems, the circuit obtained through this straightforward approach will have a large AND-count. In [8], Bilgin et al. presented a SAT-based method to minimize the AND-depth and AND-count of the classical implementations. They showed that most Boolean functions with 4 variables can be implemented with the minimal AND gate count and AND depth by their method. However, for large Boolean functions, this method could not ensure the output circuit has the minimal AND-depth. In the following, we propose a new general approach to construct classical implementations having the minimal AND-depth and low AND-count.

First, we introduce some new notations and terminologies. Let $\mathbb{M}_k$, where $k \geq 1$, denote the set of Boolean polynomials with degree in the range $[2^{k-1}+1, 2^k]$, and $\mathbb{M}_0$ denote the set of all affine polynomials. In other words, $\mathbb{M}_k$ consists of polynomials whose classical implementations have the minimal AND-depth $k$. Moreover, we use $\mathbb{M}_{\leq k}$ to denote $\cup_{i=0}^{k}\mathbb{M}_i$. Let $f \in \mathbb{M}_d$. Suppose $\mathcal{M} = \{M_1, M_2, \ldots, M_s\}$ are all monomials contained in $f$. The $i$-th m-layer of $f$, denoted as $ML_i(f)$, is the sum of all monomials in $\mathcal{M} \cap \mathbb{M}_i$. Obviously, we have $f = ML_d(f) + ML_{d-1}(f) + \cdots + ML_0(f)$.

For $f \in \mathbb{M}_d$, if there are $f_1 \in \mathbb{M}_s$ and $f_2 \in \mathbb{M}_t$ such that $f = f_1 \cdot f_2$ and $s, t < d$, we say $f$ is *depth-decreasing factorable*, and $f_1, f_2$ are *depth-decreasing factors* of $f$. Apparently, all monomials with degrees higher than one are depth-decreasing factorable. A *max-depth cover* of $f$ is a set of depth-decreasing factorable Boolean polynomials $\mathcal{C} = \{C_1, C_2, \ldots, C_k\} \subseteq \mathbb{M}_d$ such that

$$f = C_1 + C_2 + \cdots + C_k + R, \tag{1}$$

for some $R \in \mathbb{M}_{\leq d-1}$. From (1), we know that $\sum_i C_i = ML_d(f)$. Since $C_i$ is depth-decreasing factorable, we have $C_i = D_{i,1} \cdot D_{i,2}$ for some $D_{i,1}, D_{i,2} \in \mathbb{M}_{\leq d-1}$. Here $D_{1,1}, D_{1,2}, \ldots, D_{k,1}, D_{k,2}$ are called the factors of the max-depth cover, and $k$ is called the size of the max-depth cover. Unless otherwise stated, 'cover' refers to a max-depth cover in the following.

Let $\mathcal{F} = (f_1, f_2, \ldots, f_m)$ be a Boolean function with all $f_i \in \mathbb{M}_d$. For any $f_i$, suppose we can find a max-depth cover $\{C_1^i, C_2^i, \ldots, C_{k_i}^i\}$ of $f_i$, and $D_{1,1}^i, D_{1,2}^i, \ldots, D_{k_i,1}^i, D_{k_i,2}^i$ are the factors of this cover. Then by applying $k_1 + k_2 + \cdots + k_m$ AND gates in one AND layer, we can generate all these covers from their factors. Obviously, if we can construct an AND-depth-$(d-1)$ circuit that can generate these factors and the remainders $R^1, R^2, \ldots, R^m$, then we can derive an AND-depth-$d$ circuit that can generate $\mathcal{F}$. Then the original problem is induced to the problem of constructing an AND-depth-$(d-1)$ circuit for the Boolean function

$$\mathcal{F}_1 = (R^1, D_{1,1}^1, D_{1,2}^1, \ldots, D_{k_1,1}^1, D_{k_1,2}^1, \ldots, R^m, D_{1,1}^m, D_{1,2}^m, \ldots, D_{k_m,1}^m, D_{k_m,2}^m).$$

This provides a recursive approach, which is called the *Top-down approach*, to construct an AND-depth-$d$ circuit for $\mathcal{F}$. In each iteration of the approach we only need to find the max-depth covers of the input polynomials. Algorithm 1 presents the specific steps of the Top-down approach.

From Algorithm 1, we know that the AND-count of the final circuit is $\sum_{s=1}^{d} |\mathsf{CA}_s|$, and in each iteration (Step 2-11), $\mathsf{CA_s}$ contains $|\cup_i \mathcal{C}_i|$ AND gates.

**Algorithm 1:** `Top-down approach`

```
input  : A Boolean function $\mathcal{F} = (f_1, f_2, \ldots, f_m)$ with at least one $f_i \in \mathbb{M}_d$
output: A classical circuit that implements $\mathcal{F}$ with AND-depth $d$
1  $\mathcal{H} \leftarrow \{f_1, f_2, \ldots, f_m\}$;
2  for $s$ from $d$ to 1 do
3      $\mathsf{CA}_s \leftarrow \emptyset$, $\mathsf{CX}_s \leftarrow \emptyset$, $\mathcal{G} \leftarrow \mathcal{H} \cap \mathbb{M}_s$, $\mathcal{H} = \mathcal{H} \setminus \mathcal{G}$;
4      for each $g_i$ in $\mathcal{G}$ do
5          Find a max-depth cover $\mathcal{C}_i = \{C_1, C_2, \ldots, C_k\}$ of $g_i$;
6          Let $R$ be the remainder and $\{D_1^1, D_1^2, \ldots, D_k^1, D_k^2\}$ be the factors of $\mathcal{C}_i$;
7          Add the AND gates that generate $C_1, C_2, \ldots, C_k$ into $\mathsf{CA}_s$;
8          Add the XOR gates that generate $C_1 + C_2 + \ldots + C_k$ into $\mathsf{CX}_s$;
9          $\mathcal{H} \leftarrow \mathcal{H} \cup \{D_1^1, D_1^2, \ldots, D_k^1, D_k^2, R\}$;
10     end
11 end
12 Construct $\mathsf{CX}_0$, which is the set of XOR and NOT gates that generate the
    polynomials in $\mathcal{H}$ ;    /* Here $\mathcal{H}$ only contains affine polynomials */
13 return {$\mathsf{CX}_0, \mathsf{CA}_1, \mathsf{CX}_1, \ldots, \mathsf{CA}_d, \mathsf{CX}_d$}
```

Hence, the total AND-count of the obtained circuit is primarily determined by the sizes of the obtained covers.

A trivial max-depth cover of a Boolean polynomial $f$ is the set of the monomials that are in $ML_d(f)$. Its size provides an upper bound for the sizes of covers of $f$. It is easy to see that the straightforward approach introduced earlier is equivalent to consistently using trivial max-depth covers in the Top-down approach. Actually, covers with smaller sizes exist for many Boolean polynomials. For example, if two monomials in $ML_d(f)$ have a common factor in $\mathbb{M}_{d-1}$, then $f$ has a nontrivial cover.

*Example 2.* Let $f = x_1x_2x_3x_4x_5x_6 + x_1x_2x_3x_4x_5 + x_1x_2x_3x_4x_6 + x_1x_3x_4x_5x_6 + x_1x_2x_4x_5x_6 + x_1x_2x_3x_5x_6 + x_2x_3x_4x_5x_6 + r$, where $r$ is a Boolean polynomial with degree less than 5. It is easy to check that

$$\mathcal{C} = \{(x_1x_2x_3 + x_2x_3 + x_1x_2 + x_1x_3) \cdot (x_4x_5x_6 + x_4x_5 + x_4x_6 + x_5x_6)\}$$

is a max-depth cover of $f$. Since it contains only one polynomial, its size is 1.

**Methods for finding covers.** We propose two methods to find covers with small size. The first one is a greedy method. From a cover $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ of $f$, we want to use $C_1 + C_2 + \cdots + C_k$ to generate $G = ML_d(f)$. Our strategy is using $C_1$ to generate a sufficiently large part of $G$. Let $\mathcal{M}$ be the set of monomials involved in $G$. We begin with two sets $S_1 = \{p_1\}$ and $S_2 = \{q_1\}$, where $p_1$ and $q_1$ are the depth-decreasing factors of a monomial in $\mathcal{M} \cap \mathbb{M}_d$. Then $S_1$ is updated as follows. If there is a $p$, which is the depth-decreasing factor of a monomial in $\mathcal{M}$, satisfying that $(\sum_{p_i \in S_1} p_i + p)(\sum_{q_j \in S_2} q_j)$ contains more monomials in $\mathcal{M}$ than $(\sum_{p_i \in S_1} p_i)(\sum_{q_j \in S_2} q_j)$, we add $p$ into $S_1$. Similarly, we can update $S_2$. If $S_1$ and $S_2$ cannot be updated, set $C_1 = D_1^1 \cdot D_2^1$ with $D_1^1 = \sum_{p_i \in S_1} p_i$ and

$D_2^1 = \sum_{q_j \in S_2} q_j$. Then let $G = G - C_1$, and find $C_2$ similarly. For example, we can easily construct the cover in Example 2 by this method. Sometimes, we can update $S_1$ (or $S_2$) by adding $p$ (or $q$) such that we can generate new monomials in $ML_{\leq d-1}(f)$.

The second method is a SAT-based method. According to Equation (1), we can encode the problem of *determining whether there is a max-depth cover of* $f \in \mathbb{M}_d$ *with size* $k$, into the following relation.

$$f + \sum_{i=1}^{k} \mathsf{D}_i^1 \cdot \mathsf{D}_i^2 + \mathsf{R} = 0. \tag{2}$$

Here $\mathsf{D}_i^1, \mathsf{D}_i^2, \mathsf{R}$ are Boolean polynomials that contain all monomials in $\mathbb{M}_{\leq d-1}$, and have different Boolean variables as their coefficients. Then, from the constraint that the coefficients of monomials ($\in \mathbb{M}_d$) contained in $f + \sum_{i=1}^{k} \mathsf{D}_i^1 \cdot \mathsf{D}_i^2$ should all equal zero, we can generate a set of Boolean polynomial equations for the coefficient variables of $\mathsf{D}_i^1, \mathsf{D}_i^2$ ($1 \leq i \leq k$). We denote these equations as $\mathrm{EQN}_c(f, d, k)$.

By setting $k$ from 1 to $K - 1$, where $K$ is the size of the trivial max-depth cover of $f$, we attempt to solve $\mathrm{EQN}_c(f, d, k)$ using an off-the-shelf SAT-solver. If the solver returns `SAT` for some $k$, then we can obtain a max-depth cover of $f$ with size $k$.

**Improving techniques.** The following three techniques can be used to improve Algorithm 1: (1) Perform Gaussian elimination on $\mathcal{G}$ to eliminate $ML_s(g_i)$ in some $g_i$; (2) Change $\mathcal{G}$ to different invertible linear combinations of the original polynomials to find covers with smaller sizes; (3) Simultaneously find a cover of $f$ along with covers for its factors and remainder. Detailed descriptions of these techniques can be found in Appendix D.

**Application in implementing the AES S-box.** We show how to construct a minimal AND-depth circuit for the AES S-box by the Top-down approach. Let $(f_1, f_2, \ldots, f_8)$ be the Boolean function corresponding to the AES S-box. It is well known that $\deg(f_i) = 7$ and $f_i \in \mathbb{M}_3$ for $1 \leq i \leq 8$, hence our target circuit has AND-depth 3.

First, for any $f_i$, with the SAT-based method, we can find that it has a cover with size 1. Based on this result, we can combine the second and the third improving techniques as follows. Change the input polynomials as all possible invertible linear combinations of $f_1, f_2, \ldots, f_8$. Suppose $g_1, g_2, \ldots, g_8$ are the new input polynomials. For each $g_i$, attempt to simultaneously find its cover and covers of the corresponding factors and remainders by the SAT-based method. Here we set the size of the cover of $g_i$ to 1, and the sizes of the covers of the two factors and one remainder as $1, 1, 3$ respectively. We can successfully find a set of new input polynomials $\{\bar{g}_1, \bar{g}_2, \ldots, \bar{g}_8\}$ such that the SAT-solver returns `SAT`. Therefore, we can generate the size-1 covers of $\bar{g}_1, \bar{g}_2, \ldots, \bar{g}_8$ by applying 8 AND gates in the third AND-layer, and generate the covers of the factors and remainder for each $\bar{g}_i$ by applying $1 + 1 + 3 = 5$ AND gates in the second

AND-layer, resulting in a total AND-count $8 \times 5 = 40$ for the second AND-layer. Finally, we considered the iteration with input polynomials in $\mathbb{M}_1$. After performing Gaussian elimination, the rank of the first m-layer (i.e. the quadratic parts) of these input polynomials is 28, which implies that at least 28 AND gates are required to generate these polynomials. Therefore, we can use the trivial covers, which contains 28 quadratic monomials, for this iteration. In summary, we construct a AND-depth-3 circuit [4] with AND-count being equal to $8+40+28 = 76$. Compared to the AND-depth-3 circuit proposed in [28], which has 78 AND gates, our circuit reduces the AND-count by 2.

Note that the AND-depth-3 circuit proposed in [28] was achieved by applying some heuristic modification on Boyar and Peratal's AND-depth-4 circuit, which was constructed by utilizing the algebraic property of the multiplicative inversion in $\mathbb{F}_{2^8}$. In comparison, the Top-down approach is a general approach, which can be applied with only knowing the ANF of the Boolean function. This leads to the subsequent section. Considering the multiplicative inversion in $\mathbb{F}_{2^n}$, which is commonly used in cryptography, we aim to develop a specific method that can construct a minimal AND-depth implementation with lower AND-count.

## 4 Implementing the Multiplicative Inversion in $\mathbb{F}_{2^n}$

For an element $\alpha \in \mathbb{F}_{2^n}$, we denote its multiplicative inverse, as $\alpha^{-1}$ (where $\alpha^{-1} = 0$ if $\alpha = 0$). Suppose $\{\beta_1, \beta_2, \ldots, \beta_n\}$ is a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. $\alpha$ can be expressed as $x_1\beta_1 + x_2\beta_2 + \cdots + x_n\beta_n$, where each $x_i \in \mathbb{F}_2$. Then $\alpha^{-1}$ can be written as $f_1(x_1, x_2, \ldots, x_n)\beta_1 + f_2(x_1, x_2, \ldots, x_n)\beta_2 + \cdots + f_n(x_1, x_2, \ldots, x_n)\beta_n$ for some Boolean polynomials $f_1, f_2, \ldots, f_n$. The problem of implementing the multiplicative inversion in $\mathbb{F}_{2^n}$, or simply the inversion in $\mathbb{F}_{2^n}$, is to implement the Boolean function $\mathcal{F}_{inv}^{(n)} = (f_1, f_2, \ldots, f_n)$. A property of $\mathcal{F}_{inv}^{(n)}$ is that at least one of its coordinate functions has algebraic degree $n-1$. A proof of this property can be found in Appendix E. Then, according to Lemma 1, we have the following theorem.

**Theorem 1.** *The minimal AND-depth for implementing the inversion in $\mathbb{F}_{2^n}$ is $\lceil \log_2(n-1) \rceil$.*

As our method for implementing inversion will involve implementing multiplications in various $\mathbb{F}_{2^k}$, we first introduce the following lemma about multiplication implementations, whose proof can be found in Appendix F.1.

**Lemma 3.** *If $k = 2^r s$, for some positive number $r$ and odd number $s$, then the multiplication of two elements in $\mathbb{F}_{2^k}$ can be implemented by one AND layer and $3^r s^2$ AND gates.*

In the following paragraphs of this section, we always assume that the multiplication in $\mathbb{F}_{2^k}$ is implemented by one AND layer. Moreover, for $k = 2^r s$, we denote this AND-count $3^r s^2$ as $\omega(k)$.

[4] This circuit is available at `https://github.com/hzy-cas/Minimal_T-depth_Width`.

### 4.1 Parallel Addition Chain

Let $\alpha \in \mathbb{F}_{2^n}$. Note that, if $\alpha \neq 0$, $\alpha^{2^n-2} \cdot \alpha = \alpha^{2^n-1} = 1$, and if $\alpha = 0$, $\alpha^{2^n-2} = 0$, which means $\alpha^{-1} = \alpha^{2^n-2}$. Therefore computing $\alpha^{-1}$ is equivalent to computing the exponentiation $\alpha^{2^n-2}$.

There is a connection between the procedure of computing $\alpha^{2^n-2}$ and the addition chain of $n-1$. For a number $k$, its *addition chain* is a sequence $1 = a_0 < a_1 < a_2 < \cdots < a_r = k$, with the property that for all $s = 1, 2, \ldots, r$, $a_s = a_i + a_j$ for some $i \leq j < s$. The number $r$ is called the length of the addition chain. We use an example to illustrate such connection.

Let $\alpha \in \mathbb{F}_{2^6}$, then $\alpha^{-1} = \alpha^{2^6-2} = \alpha^{62}$. We can use two kinds of operations, squaring and multiplication, to compute $\alpha^{62}$. For example, we sequentially compute $\alpha^2$, $\alpha^3 = \alpha \cdot \alpha^2$, $\alpha^6 = (\alpha^3)^2$, $\alpha^7 = \alpha^6 \cdot \alpha$, $\alpha^{28} = ((\alpha^7)^2)^2$, $\alpha^{31} = \alpha^{28} \cdot \alpha^3$, $\alpha^{62} = (\alpha^{31})^2$. If we express these exponents by their binary expressions, which are denoted by binary strings with 2 as their subscript, then the change of these exponents can be illustrated by the following steps.

$$1_2(1) \xrightarrow{\wedge 2} 10_2(2) \xrightarrow{\times} 11_2(3) \xrightarrow{\wedge 2} 110_2(6) \xrightarrow{\times} 111_2(7) \xrightarrow{\wedge 4} 11100_2(28) \xrightarrow{\times} 11111_2(31) \xrightarrow{\wedge 2} 111110_2(62)$$

Note that, with a polynomial basis $\{1, \beta, \beta^2, \ldots, \beta^5\}$ of $\mathbb{F}_{2^6}$, the squaring operation for $\alpha = a_1\beta^5 + a_2\beta^4 + \cdots + a_4\beta + a_5$ corresponds to compute some linear combinations of these $a_i$'s, while with a normal basis $\{\beta, \beta^2, \ldots, \beta^{2^5}\}$ of $\mathbb{F}_{2^6}$, the squaring operation for $\alpha = a_1\beta + a_2\beta^2 + \cdots + a_4\beta^{2^4} + a_5\beta^{2^5}$ corresponds to reordering these $a_i$'s. In both cases, implementing the squaring operations does not need AND gates. Since we focus on reducing AND-count, the steps corresponding to squaring can be ignored in the above chain. Moreover, we can observe that, in the above steps, squaring corresponds to appending a zero at the end of a binary number, and this does not change the Hamming weights of this binary number. For a binary number $k$, its *Hamming weight*, denoted as $\mathrm{HW}(k)$, is the number of ones that occur in $k$. Then the change for the Hamming weights of these binary numbers can be illustrated by an addition chain for 5: $1 \to 2 \to 3 \to 5$. This chain has length 3, which implies that 3 multiplications in $\mathbb{F}_{2^6}$ are applied in the corresponding process for computing $\alpha^{-1}$.

Let $\alpha \in \mathbb{F}_{2^n}$. Suppose we have an addition chain for $n-1$ with length $r$. We can construct the following procedure for computing $\alpha^{2^n-2}$. If $a = b + c$ in the chain, then we can obtain $\alpha^{2^a-1}$ (the binary expression of $2^a - 1$ is $a$ continuous 1's) from $\alpha^{2^b-1}$ and $\alpha^{2^c-1}$ by first squaring $\alpha^{2^b-1}$ for $c$ times, resulting in $\alpha^{2^{b+c}-2^c}$, then computing $\alpha^{2^{b+c}-2^c} \cdot \alpha^{2^c-1} = \alpha^{2^{b+c}-1}$. In this way, we can obtain $\alpha^{2^{n-1}-1}$ with $r$ multiplications in $\mathbb{F}_{2^n}$. Finally, we square $\alpha^{2^{n-1}-1}$ and achieve $\alpha^{2^n-2}$. Obviously, from this procedure, we can construct a circuit that computes $\alpha^{-1}$ with AND-depth $r$ and AND-count $\omega(n)r$.

However, based on the above approach, sometimes, even from a shortest addition chain for $n-1$, we cannot obtain a minimal AND-depth circuit for inversion in $\mathbb{F}_{2^n}$. For example, the shortest addition chain for 7 has length 4, but inversion in $\mathbb{F}_{2^8}$ can be implemented with AND-depth 3, according to Theorem 1.

In order to better characterize the AND-depth, we introduce a new concept called the parallel addition chain.

A *parallel addition chain* for $n$ with length $d$ and width $w$, is defined by a matrix $A = (a_{i,j})_{w\times(d+1)}$ whose entries are non-negative integers, with the following properties:

1) $a_{1,1} = 1$, $a_{s,d+1} = n$ for some $1 \le s \le k$, and $a_{i,d+1} = 0$ for any $i \ne s$;
2) For any row, suppose its nonzero entries are $a_{i,j_1}, a_{i,j_2}, \dots, a_{i,j_t}$ with $j_1 < j_2 < \cdots < j_t$, then $a_{i,j_1} < a_{i,j_2} < \cdots < a_{i,j_t}$.
3) Suppose $a_{i,s}$ is the first nonzero entry in a row where $i > 1$. Then $a_{i,s} = a_{k,s}$ for some $k \ne i$. (This means the subsequent operations share the same input)
4) Suppose $a_{i,j}$ is a nonzero entry in a row, but not the first nonzero element. Then $a_{i,j} = a_{i,p} + a_{s,q}$ where $s \ne i$, $p < j$, and $q < j$.

*Example 3.* The following matrix is a parallel addition chain for 7. It corresponds to a procedure for computing 7 with 3 addition layers.

$$A = \begin{bmatrix} 1 & 2 & 4 & 7 \\ 1 & 0 & 3 & 0 \end{bmatrix}$$

$$\begin{tikzcd}
\mathcal{C}_1 : 1 \arrow[r, "+1"] & 2 \arrow[r, "+2"] & 4 \arrow[r, "+3", "\mathcal{C}_2"'] & 7 \\
\mathcal{C}_2 : 1 \arrow[r] & 1 \arrow[r, "+2", "\mathcal{C}_1"'] & 3 &
\end{tikzcd}$$

Here, 3 ($\in \mathcal{C}_2$) is obtained by adding 1 ($\in \mathcal{C}_2$) and 2 ($\in \mathcal{C}_1$). 7 ($\in \mathcal{C}_1$) is obtained by adding 4 ($\in \mathcal{C}_1$) and 3 ($\in \mathcal{C}_2$). Based on this parallel addition chain, we can construct the following procedure for computing $\alpha^{-1} \in \mathbb{F}_{2^8}$.

$$\begin{tikzcd}
\alpha \arrow[r, "\wedge 2"] & \alpha^{10_2} \arrow[r, "\times\alpha^{1}"] & \alpha^{11_2} \arrow[r, "\wedge 4"] & \alpha^{1100_2} \arrow[r, "\times\alpha^{11_2}"] & \alpha^{1111_2} \arrow[r, "\wedge 16"] & \alpha^{11110000_2} \arrow[r, "\times\alpha^{1110_2}"] & \alpha^{11111110_2} \\
 & \alpha^{10_2} \arrow[r] & \alpha^{10_2} \arrow[r] & \alpha^{10_2} \arrow[r, "\times\alpha^{1100_2}"] & \alpha^{1110_2} & &
\end{tikzcd}$$

This procedure corresponds to an AND-depth-3 circuit for the inversion in $\mathbb{F}_{2^8}$.

For a parallel addition chain $A$ for $n$ with length $d$ and width $w$, the nonzero entries in the same column can be generated in parallel, hence we have a depth-$d$ procedure for computing $n$. Moreover, if the number of nonzero entries in $A$ is $m$, then the addition count for the whole procedure is $m - w$. This procedure corresponds to a classical circuit that implements the inversion in $\mathbb{F}_{2^{n+1}}$ with AND-depth $d$ and AND-count $(m-w)\omega(n)$. Obviously, the minimal length of the parallel addition chains for $n$ is $\lceil \log_2(n) \rceil$. Following the Itoh-Tsujii algorithm [29] for inversion, we can construct a parallel addition chain for $n$ with the minimal length.

**Lemma 4.** *For any $n$, there is a parallel addition chain for $n$ with the minimal depth $\lceil \log_2(n) \rceil$ and involving $\mathrm{HW}(n) + \lfloor \log_2(n) \rfloor - 1$ additions.*

*Proof.* Suppose $\mathrm{HW}(n) = 1$, then $n = 2^k$ for some $k$, and $\lfloor \log_2(n) \rfloor = \lceil \log_2(n) \rceil = k$. In this case, $[1, 2, 2^2, \cdots, 2^{k-1}, 2^k]$ is a parallel addition chain for $n$. This chain has length $k = \lceil \log_2(n) \rceil$ and addition-count $k = \mathrm{HW}(n) + \lfloor \log_2(n) \rfloor - 1$

Now suppose $\mathrm{HW}(n) = m > 1$. We can write $n$ as $2^{k_m} + 2^{k_{m-1}} + \cdots + 2^{k_1}$, where $\lfloor \log_2(n) \rfloor \geq k_m > k_{m-1} > \cdots > k_1 \geq 0$. Then we can construct the following parallel addition chain for $n$.

$$\begin{bmatrix} 1 & 2 \cdots 2^{k_1} & 2^{k_1+1} \cdots 2^{k_2} & 2^{k_2+1} & \cdots & 2^{k_3} \cdots & 2^{k_m} & 0 \\ 0 & 0 \cdots 2^{k_1} & 0 \quad \cdots \quad 0 & \sum_{i=1}^{2} 2^{k_i} & \cdots & 0 \quad \cdots & 0 & \sum_{i=1}^{m} 2^{k_i} \end{bmatrix}$$

The length of this chain is $k_m + 1 = \lfloor \log_2(n) \rfloor + 1 = \lceil \log_2(n) \rceil$, and the addition count is $k_m + \mathrm{HW}(n) - 1 = \mathrm{HW}(n) + \lfloor \log_2(n) \rfloor - 1$. □

From this lemma, we can induce the following result about implementing the inversion in $\mathbb{F}_{2^n}$ with the minimal AND-depth.

**Theorem 2.** *There is a classical circuit implementing the inversion in $\mathbb{F}_{2^n}$ with AND-depth $\lceil \log_2(n-1) \rceil$ and AND-count $\omega(n)(\mathrm{HW}(n-1) + \lfloor \log_2(n-1) \rfloor - 1)$.*

In the following, we show that the AND-count of the circuit derived from a parallel addition chain can be reduced for some instances. In a parallel addition chain, we may compute the addition of a number and itself. This corresponds to the multiplication of $\alpha^a$ and $\alpha^b$ with $\mathrm{HW}(a) = \mathrm{HW}(b)$. Sometimes, we can significantly reduce the AND-counts for implementing $\alpha^a \cdot \alpha^b$ and the multiplications of $\alpha^{a+b}$ and other elements.

**Lemma 5.** *Suppose $n = 2s$ for some integer $s$. Let $a$, $b$ be two integers with $b = a2^s$. Then the multiplication of $\alpha^a$ and $\alpha^b$ in $\mathbb{F}_{2^n}$ can be implemented by one AND layer, and $\omega(s)$ AND gates. Moreover, the multiplication result $\alpha^{a+b}$ is in $\mathbb{F}_{2^s}$ hence the the multiplication of $\alpha^{a+b}$ and any element in $\mathbb{F}_{2^n}$ can be implemented by one AND layer and $2\omega(s)$ AND gates.*

*Proof.* Since $n = 2s$, $\alpha^a$ can be expressed as $a_1\beta_1 + a_2\beta_2$, where $a_1, a_2 \in \mathbb{F}_{2^s}$, and $\{\beta_1, \beta_2\}$ is a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_{2^s}$. Since $a_i^{2^s} = a_i$, we have $\alpha^b = (\alpha^a)^{2^s} = a_1\beta_1^{2^s} + a_2\beta_2^{2^s}$. Then $\alpha^a \cdot \alpha^b = a_1^2\beta_1^{2^s+1} + a_2^2\beta_2^{2^s+1} + a_1a_2(\beta_1^{2^s}\beta_2 + \beta_2^{2^s}\beta_1)$. As we mentioned before, implementing $a_1^2$ and $a_2^2$ does not cost AND gates, and implementing $a_1a_2$ requires one AND layer and $\omega(s)$ AND gates. This proves the first conclusion. Moreover, we have $\alpha^{a+b} = (\alpha^a)^{2^s+1}$. Let $T = \alpha^a$, then $(T^{2^s+1})^{2^s} = T^{2^{2s}+2^s} = T^{2^n}T^{2^s} = T \cdot T^{2^s} = T^{2^s+1}$, which implies $\alpha^{a+b}$ is an element in $\mathbb{F}_{2^s}$. For any element $r \in \mathbb{F}_{2^n}$, it can be expressed as $r_1\beta_1 + r_2\beta_2$. Then $\alpha^{a+b} \cdot r = \alpha^{a+b}r_1\beta_1 + \alpha^{a+b}r_2\beta_2$, which means only two parallelized multiplications in $\mathbb{F}_{2^s}$ are required to compute $\alpha^{a+b} \cdot r$. This proves the lemma. □

Compared to the implementation from Lemma 3, which requires $3s^2$ AND gates for $n = 2s$, this implementation reduces the AND-count for the multiplication of two powers by 2/3, and the AND-count for the subsequent multiplication by 1/3.

*Example 4.* (**Implementing the RAIN-128 S-box**) We consider the S-box of the MPC-friendly block cipher RAIN-128 proposed in CCS 2022 [22], which is the inversion in $\mathbb{F}_{2^{128}}$. According to Theorem 2, to obtain an AND-depth 7 implementation, we can construct the following parallel addition chain for 127:

$$A = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 32 & 64 & 127 \\ 1 & 0 & 3 & 7 & 15 & 31 & 63 & 0 \end{bmatrix},$$

which induces an AND-depth-7 classical circuit using $3^7 \cdot 12 = 26244$ AND gates. Since $128 = 2 \cdot 64$, to apply Lemma 5, we can set the power production corresponding to $64 = 32 + 32$ as $\alpha^{(2^{32}-1)2^{64}} \cdot \alpha^{(2^{32}-1)} = \beta$, then this step can be implemented by $3^6 = 729$ AND gates. Moreover, $\beta$ is in $\mathbb{F}_{2^{64}}$, hence the power production $(\beta)^{2^{32}} \cdot (\alpha^{(2^{32-1})2^{64}} \cdot \alpha^{(2^{31}-1)2}) = \alpha^{2^{128}-2}$, which corresponds to $127 = 64+63$, can be implemented by $2 \cdot 3^6 = 1458$ AND gates. In this way, we can reduce the AND-count to 24057. Furthermore, using `ClassicalToQuantum`, we can obtain an NCT forward implementation with the minimal Toffoli-depth 7 and Toffoli-count 24057.

*Remark 2.* Implementing the inversion in $\mathbb{F}_{2^n}$ is also important for applying Shor's algorithm to attack binary elliptic curve schemes [5]. For example, to attack the binary elliptic curves recommended by NIST's Federal Information Processing Standards 186-4 [43], implementations for $n = 163, 233, 283, 409, 571$ are required. Based on Theorem 2, we can construct the minimal-$T$-depth implementations for these values of $n$.

### 4.2 Implement the Multiplicative Inversion in $\mathbb{F}_{2^{2m}}$

In this section, we further investigate optimizing the implementation of inversion in $\mathbb{F}_{2^{2m}}$, when $m$ satisfies the following property: let $m - 1 = \sum_{i=1}^{s} 2^{k_i}$ with $0 \leq k_1 < k_2 < \cdots < k_s$ and $s \geq 2$, and assume $k_s > \lceil \log_2(\sum_{i=1}^{s-1} 2^{k_i}) \rceil$.

Here we use the tower field architecture of $\mathbb{F}_{2^{2m}}$ [15, 16]. For $\alpha \in \mathbb{F}_{2^{2m}}$, we have $\alpha^{-1} = (\alpha \cdot \alpha^{2^m})^{-1} \alpha^{2^m}$. Let $b = \alpha \cdot \alpha^{2^m} = \alpha^{2^m+1}$ , then $\alpha^{-1} = b^{-1} \cdot \alpha^{2^m}$. Note that $b^{2^m-1} = \alpha^{(2^m+1)(2^m-1)} = \alpha^{2^{2m}-1} = 1$, which implies $b \in \mathbb{F}_{2^m}$ . With a normal basis $\{\beta, \beta^{2^m}\}$ of $\mathbb{F}_{2^{2m}}$ over $\mathbb{F}_{2^m}$, $\alpha$ can be written as $a_0\beta^{2^m} + a_1\beta$ , where $a_0, a_1 \in \mathbb{F}_{2^m}$. Then we have

$$\alpha^{2^m} = a_1\beta^{2^m} + a_0\beta, \quad b = \alpha \cdot \alpha^{2^m} = a_0a_1\beta^{2^{m+1}} + (a_0^2 + a_1^2)\beta^{2^m+1} + a_0a_1\beta^2$$

and

$$\alpha^{-1} = b^{-1}a_1\beta^{2^m} + b^{-1}a_0\beta.$$

This means we can compute $\alpha^{-1}$ by the following three steps:

(1) Compute $b$. The only nonlinear operation for computing $b$ is computing $a_0a_1$, hence it can be implemented by one AND layer and $\omega(m)$ AND gates.
(2) Compute $b^{-1}$. Since $b \in \mathbb{F}_{2^m}$, this step can be implemented by $\lceil \log_2(m-1) \rceil$ AND layers.
(3) Compute $b^{-1}a_1$ and $b^{-1}a_0$ simultaneously, then obtain $\alpha^{-1}$ by applying some linear operations. This step can be implemented by one AND layer.

The circuit obtained by these three steps will have AND-depth $\lceil \log_2(m-1)\rceil + 2$, which is higher than the minimal AND-depth $\lceil \log_2(2m-1)\rceil$. In the following, we show that we can reduce the AND-depth by modifying the above process.

According to Section 4.1, the problem of computing $b^{-1} \in \mathbb{F}_{2^m}$ with the minimal AND-depth can be converted to finding a parallel addition chain for $m-1$. Suppose in a parallel addition chain $A_{w\times(d+1)}$ for $m-1$ with the minimal length $d$, $m-1$ is obtained by computing $m_1+m_2$. Without loss of generality, assume $m_1$ is in the $d$-th column. If $m_2$ is in a different column, which means the two rows containing $m_1$ and $m_2$ are as follows,

$$\begin{bmatrix} \cdots & * & \cdots & m_1 & m-1 \\ \cdots & m_2 & \cdots & 0 & 0 \end{bmatrix},$$

then $m_2$ is obtained before $m_1$. Note that, in this chain the step of obtaining a number $k$ corresponds to a step of computing $b^{s(k)}$ by a multiplication in $\mathbb{F}_{2^m}$, where $s(k)$ is a binary number with $\mathrm{HW}(s(k)) = k$. Since we have already obtained $b^{s(m_2)}$ before computing $b^{s(m_1)}$, we can compute $b^{s(m_1)}$, $b^{s(m_2)}a_1$, $b^{s(m_2)}a_0$ in parallel. Then we compute $b^{s(m_1)} \cdot b^{s(m_2)}a_1 = b^{-1}a_1$ and $b^{s(m_1)} \cdot b^{s(m_2)a_0} = b^{-1}a_0$ in parallel.

By this strategy, the above Step (2) and (3) can be implemented by $\lceil \log_2(m-1)\rceil$ AND layers, and the whole procedure can be implemented by $\lceil \log_2(m-1)\rceil + 1$ AND layers. It is easy to prove that $\lceil \log_2(2m-1)\rceil = \lceil \log_2(2m-2)\rceil = \lceil \log_2(m-1)\rceil + 1$, which means this implementation has the minimal AND-depth. We refer to the above strategy as the *merging strategy*, since it merges two steps. Note that, compared to the original strategy, which first computes $b^{s(m_1)} \cdot b^{s(m_2)} = b^{s(m-1)}$, then simultaneously computes $b^{s(m-1)} \cdot a_1$ and $b^{s(m-1)} \cdot a_2$, the merging strategy needs one more multiplication in $\mathbb{F}_{2^m}$.

*Example 5.* We show how to compute $\alpha^{-1} \in \mathbb{F}_{2^{24}}$ by the merging strategy. In a parallel addition chain, an addition corresponds to a multiplication of two powers. Here we use the symbol "$\boxplus$" to denote an extension version of addition, which corresponds to the multiplication of a power and a given element. Suppose we have already achieved $b \in \mathbb{F}_{2^{12}}$ by Step (1), which requires one multiplication in $\mathbb{F}_{2^{12}}$. Based on $A_1$, which is a parallel addition chain for 11 with length 4, we can construct an extended parallel chain $A_2$.

$$A_1 = \begin{bmatrix} 1 & 2 & 4 & 8 & 11 \\ 0 & 2 & 3 & 0 & 0 \end{bmatrix} \;\Rightarrow\; A_2 = \begin{bmatrix} 1 & 2 & 4 & 8 & 11 \boxplus a_1 \\ 0 & 2 & 3 & 3 \boxplus a_0 & 11 \boxplus a_0 \\ 0 & 0 & 3 & 3 \boxplus a_1 & 0 \end{bmatrix}$$

$A_2$ corresponds to the following procedure for computing $b^{-1}a_1 = b^{4094}a_1$ and $b^{-1}a_0 = b^{4094}a_0$.

$$\begin{tikzcd}
b \arrow[r, "\wedge 2"] & b^2 \arrow[r, "\times b^1"] & b^3 \arrow[r, "\wedge 4"] & b^{12} \arrow[r, "\times b^3"] & b^{15} \arrow[r, "\wedge 16"] & b^{240} \arrow[r, "\times b^{15}"] & b^{255} \arrow[r, "\wedge 16"] & b^{4080} \arrow[r, "\times b^{14} a_1"] & b^{4094}a_1 \\
 & & b^3 \arrow[r, "\wedge 2"] & b^6 \arrow[r, "\times b^1"] & b^7 \arrow[r, "\wedge 2"] & b^{14} \arrow[r, "\times a_0"] & b^{14}a_0 \arrow[r] & b^{14}a_0 \arrow[r, "\times b^{4080}"] & b^{4094}a_0 \\
 & & & & & b^{14} \arrow[r, "\times a_1"] & b^{14}a_1 & &
\end{tikzcd}$$

Then, the whole procedure for computing $\alpha^{-1}$ has 5 multiplication layers, and involves 9 multiplications in $\mathbb{F}_{2^{12}}$.

**Theorem 3.** *Let $m-1=\sum_{i=1}^{s} 2^{k_i}$ with $0 \le k_1 < k_2 < \cdots < k_s$ and $s \ge 2$. If $k_s > \lceil \log_2(\sum_{i=1}^{s-1} 2^{k_i}) \rceil$, then $\alpha^{-1} \in \mathbb{F}_{2^{2m}}$ can be implemented by a classical circuit with the minimal AND-depth $\lceil \log_2(2m-1) \rceil$ and AND-count $\omega(m)(\mathrm{HW}(m-1)+\lfloor \log_2(m-1) \rfloor + 3)$.*

*Proof.* Let $b \in \mathbb{F}_{2^m}$ be the element obtained from Step (1). Now we consider the implementation of $b^{-1}$. If $s=2$, we have the following parallel addition chain $A$ for $m-1$ with the minimum length.

$$A = \begin{bmatrix} 1 & 2 \cdots & 2^{k_1} & 2^{k_1+1} \cdots & \boldsymbol{2^{k_2}} & 0 \\ \\ 0 & 0 \cdots & \boldsymbol{2^{k_1}} & 0 & \cdots\ 0 & m-1 \end{bmatrix}.$$

If $s > 2$, according to the proof of Lemma 4 and the property that $k_s > \lceil \log_2(\sum_{i=1}^{s-1})k_i \rceil$, we have the following parallel addition chain $A$ for $m-1$ with the minimum length.

$$A = \begin{bmatrix} 1 & 2 \cdots & 2^{k_1} & 2^{k_1+1} \cdots & 2^{k_2} & 2^{k_2+1} & \cdots & 2^{k_{s-1}} & 2^{k_{s-1}+1} & \cdots & \boldsymbol{2^{k_s}} & 0 \\ \\ 0 & 0 \cdots & 2^{k_1} & 0 & \cdots\ 0 & \sum_{i=1}^{2} 2^{k_i} & \cdots & 0 & \boldsymbol{\sum_{i=1}^{s-1} 2^{k_i}} & \cdots & 0 & m-1 \end{bmatrix}.$$

Note that, in both cases, $m-1$ is obtained by adding $2^{k_s}$ and $\sum_{i=1}^{s-1} 2^{k_i}$, and $2^{k_s}$ and $\sum_{i=1}^{s-1} 2^{k_i}$ are not in the same column. Therefore we can apply the merging strategy, and implement $\alpha^{-1}$ by $\lceil \log_2(2m-1) \rceil$ AND layers. Now we give the AND-count. Since $A$ involves $\mathrm{HW}(m-1)+\lfloor \log_2(m-1) \rfloor - 1$ additions, which corresponds to $\mathrm{HW}(m-1)+\lfloor \log_2(m-1) \rfloor - 1$ multiplications in $\mathbb{F}_{2^m}$. Then the whole procedure will involve $1+(\mathrm{HW}(m-1)+\lfloor \log_2(m-1) \rfloor - 1)+3 = \mathrm{HW}(m-1)+\lfloor \log_2(m-1) \rfloor + 3$ multiplications in $\mathbb{F}_{2^m}$, which cost $\omega(m)(\mathrm{HW}(m-1)+\lfloor \log_2(m-1) \rfloor + 3)$ AND gates. □

**Application in implementing the AES S-box.** The nonlinear part of the AES S-box is the inversion in $\mathbb{F}_{2^8}$. Note that, $4-1=2^1+2^0$, hence we can apply the merging strategy. A minimal AND-depth implementation of the AES S-box can be constructed based on the following extended addition chain.

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 0 \end{bmatrix} \quad \Rightarrow \quad A_2 = \begin{bmatrix} 1 & 2 & 3 \boxplus a_1 \\ 1 & 1 \boxplus a_0 & 3 \boxplus a_0 \\ 1 & 1 \boxplus a_1 & 0 \end{bmatrix}$$

Let $b \in \mathbb{F}_{2^4}$ be the element obtain by Step (1). Then, in $A_2$, the addition, which obtains 2, corresponds to a multiplication of $b^{a_1}$ and $b^{a_2}$, where $a_1, a_2$ are two integer satisfying $\mathrm{HW}(a_1) = \mathrm{HW}(a_2) = 1$. Note that, $b$ and $b^4$ satisfy the assumption of Lemma 5. There we can design the computation procedure corresponding to $A_2$ as follows.

$$\begin{array}{lllll} b \xrightarrow{\wedge 4} b^{100_2} \xrightarrow{\times b^1} b^{101_2} & \xrightarrow{\wedge 2} & b^{1010_2} & \xrightarrow{\times b^{100_2} a_1} & b^{1110_2} a_1 = b^{-1} a_1 \\ \quad\ \ b^{100_2} \xrightarrow{\times a_0} b^{100_2} a_0 & \rightarrow & b^{100_2} a_0 & \xrightarrow{\times b^{1010_2}} & b^{1110_2} a_0 = b^{-1} a_0 \\ \quad\ \ b^{100_2} \xrightarrow{\times a_1} b^{100_2} a_1 & \rightarrow & b^{100_2} a_1 & & \end{array} \tag{3}$$

In this procedure, $b^{100_2} \cdot b$ costs 3 AND gates. Moreover, we have $b^{1010_2} \in \mathbb{F}_{2^2}$, hence $b^{1010_2} \cdot b^{100_2} a_1$ and $b^{100_2} a_0 \cdot b^{1010_2}$ cost 6 AND gates respectively. Based on this procedure, we can construct a classical circuit the implements the AES S-box with 3 AND layers and a total of $9+3+2\cdot 9+2\cdot 6 = 42$ AND gates. In Appendix G, we present this classical circuit. Compared to the AND-depth-3 circuit proposed in [28], which has 78 AND gates, this implementation reduces the AND-count by about 46%. Furthermore, by applying `ClassicalToQuantum`, we achieve a highly compact $T$-depth-3 quantum circuit for the AES S-box. Table 3 compares this circuit with the $T$-depth-3 circuit proposed in [28,54]. Moreover, in Appendix H, we present the costs of the Grover Oracle and Encryption Oracle for AES when using our new $T$-depth-3 S-box.

**Table 3.** Comparison of different out-of-place Clifford+$T$ circuits (including uncomputation) for the AES S-box with $T$-depth 3.

| Type | #CNOT | #1qClifford | $\#T$ | #M | $T$-depth | Full Depth | Width | Source |
|---|---|---|---|---|---|---|---|---|
| $\mathfrak{C}^0/\mathfrak{C}^*$ | 1396/1398 | 494 | 312 | 78 | 3 | 119 | 218/226 | [28] |
| $\mathfrak{C}^*$ | 1110 | 448 | 264 | 66 | 3 | 92 | 129 | [54] |
| $\mathfrak{C}^0/\mathfrak{C}^*$ | **827/856** | **266/298** | **168** | **34/42** | 3 | **85/87** | **89/97** | This work |

## 5 Implementing Nonlinear Functions with the Minimal Width

In this section, we consider implementing Boolean functions with the minimal number of qubits. For a Boolean function $\mathcal{F}$ with $n$ variables, if it is invertible, it can be seen as a permutation $\mathcal{P}$ on $\mathbb{F}_2^n$. If the corresponding $\mathcal{P}$ is an even (odd) permutation, we say $\mathcal{F}$ is even (odd). A well-known result for the minimal number of qubits required for implementing an invertible Boolean function $\mathcal{F}$ in-place by NCT gates is as follows.

**Lemma 6.** *[46] When $n \geq 4$, if $\mathcal{F}$ is even, the minimal number of qubits required to implement $\mathcal{F}$ by NCT gates is $n$. If $\mathcal{F}$ is odd, the minimal number of qubits required to implement $\mathcal{F}$ in-place by NCT gates is $n+1$.*

Suppose we already have an NCT circuit with this minimal width, then by decomposing the Toffoli gates into Clifford+$T$ sub-circuits without using ancilla qubits, we can obtain a Clifford+$T$ circuit for $\mathcal{F}$ with the same width. In this paper, we only consider the strategy of constructing Clifford+$T$ implementations originated from NCT implementations, hence in the following, when we say a minimal-width Clifford+$T$ implementation, we mean an $n$-qubit ($(n+1)$-qubit) Clifford+$T$ circuit for an even (odd[5]) $\mathcal{F}$.

[5] If we use some different gate sets, for example, the Clifford+$Z_N$ gate set, which includes the $N^{th}$ roots of the Pauli-$Z$ gate, we can construct an $n$-qubit implementation for an odd $\mathcal{F}$ [9]

Note that, this minimal-width implementation is an in-place implementation for $\mathcal{F}$, and exists only when $\mathcal{F}$ is invertible. If $\mathcal{F}$ is non-invertible, it can only be implemented by an out-of-place circuit. In this case, we can equivalently consider the problem of finding a minimal-width implementation for the invertible Boolean function: $(x, y) \to (x, y \oplus \mathcal{F}(x))$. Therefore, in the following, we suppose $\mathcal{F}$ is invertible, and consider the problem of *finding a minimal-width in-place Clifford+T circuit for* $\mathcal{F}$.

In [46], Lemma 6 was proven constructively, and one can derive a minimal-width implementation from the proof. However, the gate-count for the resulting circuit would be quite large. For this reason, we aim to explore methods that can construct minimal-width implementations with low gate-count.

## 5.1 Method Based on MCT Implementations

A *multiple controlled Toffoli (MCT) gate* with $k$ control qubits, denoted as $C^kX$, is a gate that maps $|x_1, x_2, \cdots, x_k\rangle|x_{k+1}\rangle$ to $|x_1, x_2, \cdots, x_k\rangle|x_{k+1} \oplus x_1x_2\cdots x_k\rangle$, for any $x_1, x_2 \ldots, x_{k+1} \in \mathbb{F}_2$. Clearly, the $C^0X$, $C^1X$, and $C^2X$ gates are the NOT, CNOT, and Toffoli gates respectively. We introduce a method for constructing the minimal-width implementation based on the MCT implementations of Boolean functions. To implement an $n$-bit invertible Boolean function $\mathcal{F}$, this method involves the following three steps.

1) Implement $\mathcal{F}$ by MCT gates without using any ancilla qubit.
2) Decompose each MCT gate into NCT gates with at most one ancilla qubit.
3) Decompose each Toffoli gate into a 3-qubit Clifford+$T$ sub-circuit, then assemble all sub-circuits.

Here the first step is constructing an $n$-qubit MCT implementation for $\mathcal{F}$. For this problem, a trivial solution is decomposing the corresponding permutation into exchanges, then based on the Gray code, further decomposing each exchange into the composition of exchanges that swap two elements having one bit difference [42], which can be implemented by a $C^{n-1}X$ gate and some NOT gates. However, the number of $C^{n-1}X$ gates of the obtained circuit is too large, resulting in a minimal-width implementation with a substantial gate-count. In [35], Lee et al. proposed a tensor decomposition-based method to obtain compact MCT implementations of invertible Boolean functions. The idea of their algorithm can be described as follows. Let $U_{\mathcal{F}}$ be the $2^n \times 2^n$ permutation matrix corresponding to $\mathcal{F}$. Try to find some $M_1, M_2, \ldots, M_k$ such that $M_kM_{k-1}\cdots M_1U_{\mathcal{F}}$ can be decomposed as $U_{n-1} \otimes I_2$, where $U_{n-1}$ is a permutation matrix of order $2^{n-1}$, $I_2$ is the identity matrix of order 2, and $M_i$ is the permutation matrix corresponding to an MCT gate on some of the input $n$ qubits. Then by recursively decomposing $U_{n-1}$, one can obtain an $n$-qubit MCT implementation of $\mathcal{F}$. Their experiments show that their algorithm can efficiently obtain a compact MCT circuit for $\mathcal{F}$ with $n \leq 8$.

After obtaining an $n$-qubit MCT implementation, we need to decompose each MCT gate into NCT gates. Here we introduce the decomposition presented by in [6]. We will involve two kinds of ancilla qubits, the clean ones, which are initialized to $|0\rangle$, and the dirty ones, which are initialized to some state $|x\rangle$.

**Lemma 7.** *[6] Suppose $p$ and $q$ are two integers such that $p+q=m+1$. Then with one dirty ancilla qubit, one can implement a $C^mX$ gate by two $C^pX$ gates and two $C^qX$ gates, while with one clean ancilla qubit, one can implement a $C^mX$ gate by two $C^pX$ gate and one $C^qX$ gate.*

**Lemma 8.** *[6] A $C^mX$ gate with $m \geq 3$ can be implemented by $4(m-2)$ Toffoli gates with $m-2$* dirty *ancilla qubits.*

In Appendix I, we illustrate how to decompose $C^7X$ and $C^5X$ according to Lemma 7 and 8, respectively. By combing the circuits obtained from Lemma 7 and 8, we can easily obtain an NCT implementation for the $C^mX$ gate with $m \geq 3$ by using one ancilla qubit. Note that, a $C^{n-1}X$ gate on $n$ qubits corresponds to an odd permutation, since it only exchanges two elements in $\mathbb{F}_2^n$. Therefore, the NCT implementation for $C^{n-1}X$ obtained in this way has the minimal width.

Now consider the $n$-qubit MCT implementation obtained. When $\mathcal{F}$ is odd, we can decompose each $C^mX$ $(m \leq n-1)$ with one clean ancilla qubit, and obtain an $(n+1)$-qubit NCT implementation. When $\mathcal{F}$ is even, the tensor decomposition-based method ensures that the resulting MCT circuit does not contain the $C^{n-1}X$ gate for the majority of cases. Therefore, we can decompose each $C^mX$ $(m \leq n-2)$ with one dirty ancilla qubit, and this will induce an $n$-qubit NCT implementation.

After obtaining the minimal-width NCT implementation, our next step is decomposing Toffoli gates and assembling small Clifford+$T$ sub-circuits. In [1], Abdessaied et al. presented a method to reduce the $T$-depth of the Clifford+$T$ implementation for an MCT gate based on the the symmetric and cascade structure of the Toffoli sequence implementing this MCT gate. They proved that a $C^mX$ gate with $m \geq 5$ can be implemented with one clean ancilla qubit and $T$-depth $6(m-2)+2$. Here we adopt their method, and utilize our quantum resource estimator to further improve the circuit. We found that some $C^mX$ can be implemented by lower $T$-depth than the bound given in [1], due to the cancellation of gates and the unpredictable parallelization of $T$ gates. For example, we found that $C^7X$ can be implemented by $T$-depth 29, which is 3 less than 32 obtained from [1]. Finally, from these compact Clifford+$T$ circuits for $C^mX$, we can derive a minimal-width Clifford+$T$ circuit for $\mathcal{F}$.

## 5.2 A SAT-based Method

In this section, we propose a SAT-based method to find the minimal-width NCT implementation. Once we obtain the NCT implementation, we can obtain the minimal-width Clifford+$T$ implementation by decomposing Toffoli gates.

Suppose $\mathcal{F}$ is an invertible Boolean function with $n$ variables. Let $w=n+1$ if $\mathcal{F}$ is odd, and $w=n$ otherwise. We consider the following decision problem: *whether there exists an NCT circuit that implements $\mathcal{F}$ with width $w$ and Toffoli-count $k$.* We can encode this decision problem into a SAT problem, then solve this SAT problem by an off-the-shelf SAT solver. If the solver returns `SAT`, we can obtain a minimal-width NCT implementation with Toffoli-count $k$. If the solver returns `UNSAT`, we should increase $k$ by one and solve the decision problem for this new $k$. Here the critical problem is how to design an encoding scheme, which can convert the decision problem into a SAT problem that can be solved efficiently. In the following, we present our encoding scheme.

Due to the difficulty of solving large-scale SAT problems in practice, here we focus on the problem with $n \leq 5$. As some SAT-based algorithms for optimizing the multiplicative complexity of classical circuits [20, 51], we divide the entire NCT circuit into different affine layers and nonlinear layers. If $\mathcal{F}$ can be implemented by $k$ Toffoli gates, then $\mathcal{F}$ can be written as

$$\mathcal{S}_k \circ \mathcal{T}_k \circ \mathcal{S}_{k-1} \circ \cdots \circ \mathcal{S}_2 \circ \mathcal{T}_2 \circ \mathcal{S}_1 \circ \mathcal{T}_1 \circ \mathcal{S}_0, \tag{4}$$

where $\mathcal{S}_i$ is an invertible affine function corresponding to a circuit consisting of CNOT and NOT gates, and $\mathcal{T}_i$ is an invertible nonlinear function corresponding to a certain Toffoli gate. Let $\mathcal{T}_{i,j,k}$ denote the function corresponding to the Toffoli gate with the $i$-th and $j$-th wires being the controlled wires and the $k$-th wire being the target wire. Note that, for any $i, j, k$, we can find a set of rewire operations, denoted by $\mathcal{R}$, such that $\mathcal{T}_{i,j,k} = \mathcal{R} \circ \mathcal{T}_{1,2,3} \circ \mathcal{R}^{-1}$. Moreover, since the swap operation can be implemented by 3 CNOT gates, $\mathcal{R}$ can be implemented by a CNOT circuit. This means, $\mathcal{T}_{i,j,k} = \mathcal{L}' \circ \mathcal{T}_{1,2,3} \circ \mathcal{L}'^{-1}$ for some invertible linear function $\mathcal{L}'$. Then, by substituting each $\mathcal{T}_i$ with this expression in (4), we can deduce that

$$\mathcal{F} = \mathcal{L}_k \circ \mathcal{T}_{1,2,3} \circ \mathcal{L}_{k-1} \circ \cdots \circ \mathcal{L}_2 \circ \mathcal{T}_{1,2,3} \circ \mathcal{L}_1 \circ \mathcal{T}_{1,2,3} \circ \mathcal{L}_0, \tag{5}$$

for some invertible affine functions $\mathcal{L}_0, \ldots, \mathcal{L}_k$. In this way, we fix the the Toffoli gates, thus exclude numerous equivalent solutions within the solution space.

Now suppose the input state of the circuit is a computational basis state $|x_1, x_2, \cdots, x_n\rangle$ with $x_i \in \mathbb{F}_2$, then after each $\mathcal{L}_i$ and $\mathcal{T}_{1,2,3}$, the state on a wire can be represented as a Boolean function w.r.t. $\{x_1, x_2, \ldots, x_n\}$. We denote the ANF (algebraic normal form) of the quantum state on the $i$-th wire before $\mathcal{L}_j$, as $\mathsf{A}_{i,j}$ $(1 \leq i \leq w, 0 \leq j \leq k)$, and denote the ANF of the quantum state on the $i$-th wire after $\mathcal{L}_j$ as $\mathsf{B}_{i,j}$ $(1 \leq i \leq w, 0 \leq j \leq k)$. Then we have the following relations.

- $\forall i \in \{1, \ldots, w\}, \forall j \in \{0, \ldots, k\}$, $\mathsf{B}_{i,j} = c_{1,i}^{(j)}\mathsf{A}_{1,j} + \cdots + c_{w,i}^{(j)}\mathsf{A}_{w,j} + d_i^{(j)}$, for some Boolean variables $c_{1,i}^{(j)}, \ldots, c_{w,i}^{(j)}$ and $d_i^{(j)}$. These encode the affine layers.
- $\forall j \in \{1, \ldots, k\}$, $\mathsf{A}_{3,j} = \mathsf{B}_{3,(j-1)} + \mathsf{B}_{1,(j-1)} \cdot \mathsf{B}_{2,(j-1)}$, and $\mathsf{A}_{i,j} = \mathsf{B}_{i,(j-1)}$ if $i \neq 3$. These encode the Toffoli layers.
- $\forall i \in \{1, \ldots, n\}$, $\mathsf{A}_{i,0} = x_i$, $\mathsf{B}_{i,k} = f_i(x_1, x_2, \ldots, x_n)$. These encode that the inputs and outputs of circuits.
- If $w = n + 1$, $\mathsf{A}_{w,0} = 0$ and $\mathsf{B}_{w,k} = 0$. These encode that the ancilla qubit (for odd $\mathcal{F}$), is in the state $|0\rangle$ at the beginning and the end of the circuit.

The next step is generating Boolean polynomial equations from these relations. Here we use the technique proposed in [53], which can accelerate the subsequently SAT-problem solving process as announced. Specifically, in these relations, we replace $\mathsf{A}_{i,j}$ and $\mathsf{B}_{i,j}$ by Boolean polynomials that contain all $2^n$ monomials with distinct Boolean variables as their coefficients, then compare the monomial coefficients of the two polynomials on both sides of these relations and generate equations about the coefficient variables. After obtaining the Boolean polynomial equations, we can use an ANF-to-CNF converter, such as Bosphorus [19], to convert them into a CNF, which is the canonical input of a SAT solver.

**Meet-in-the-Middle**. Since a quantum circuits is reversible, we can divide the entire circuit into two parts. The forward part transforms $(x_1, x_2, \ldots, x_n)$ to $(g_1, g_2, \ldots, g_n)$, while the backward part transforms $(f_1, f_2, \ldots, f_n)$ to $(g_1, g_2, \ldots, g_n)$, where $g_i$ is a Boolean function with $2^n$ Boolean variables as its coefficients. In this way, we can use the determination of $f_i$ to reduce the search space, which can greatly increase the efficiency of our method. Note that, to ensure this strategy works, we should add the equations: $\forall j \in \{0, \ldots, k\}$, $\det(c_{s,t}^{(j)})_{w \times w} = 1$, to guarantee that each affine layer remains invertible.

Experiments show that our method can efficiently find a minimal-width NCT circuit for $\mathcal{F}$ with $n \leq 4$, and the circuit achieved can also have the minimal Toffoli-count. When $n = 5$ and $\mathcal{F}$ is even, this method can find the minimal-width NCT circuits for certain practical problems (see Section 5.3) in a reasonable time.

*Remark 3.* Compared to the SAT-based methods recently proposed in [18, 38], our method is based on a new encoding scheme for a different decision problem. Specifically, our encoding scheme applies several techniques to accelerate the subsequent solving process, enabling our method to efficiently construct circuits for certain 5-bit S-boxes.

## 5.3 Applications in Implementations of AES and SHA3

**Implementing the AES S-box with 9 qubits.** The AES S-box corresponds to an odd permutation on $\mathbb{F}_2^8$, hence its NCT implementation has the minimal width of 9. In [23], Grassl et al. mentioned a method to obtain a 9-qubit NCT implementation based on the stabilizer chains of permutation groups, but only upper bounds for the numbers of $T$ gates and Clifford gates were given.

We demonstrate how to construct a 9-qubit NCT circuit and a 9-qubit Clifford+$T$ for the AES S-box by using the methods introduced before. We first apply the tensor decomposition-based method to find an 8-qubit MCT implementation. That is finding some MCT gates, which corresponding permutations $M_1, M_2, \ldots, M_s$ on $\mathbb{F}_2^8$, such that $M_1 \circ M_2 \circ \cdots \circ M_s = \mathcal{P}_{AES}$. To further improve our implementation, we attempt to find consecutive $M_j, M_{j+1}, \cdots, M_{j+k}$ such that $\mathcal{P} = M_j \circ M_{j+1} \circ \cdots \circ M_{j+k}$ is a permutation on $\mathbb{F}_2^4$ or an even permutation on $\mathbb{F}_2^5$, which means the NCT circuit for $\mathcal{P}$ has minimal-width $\leq 5$. For such $\mathcal{P}$, by using our SAT-based method, we can get a minimal-width NCT implementation with smaller Toffoli-count.

To obtain a 9-qubit NCT circuit for the AES S-box, we decompose all MCT gates into NCT gates according to Lemma 7 and Lemma 8. Then we utilize our quantum resource estimator to identify simple cancellations of consecutive gates and obtain the layer structure of the optimized 9-qubit NCT circuit.In Table 4, we present the costs of this NCT circuit.

**Table 4.** Costs of the 9-qubit NCT circuit for the AES S-box.

| #NOT | #CNOT | #Toffoli | Width | Toffoli-depth | Full Depth |
|---|---|---|---|---|---|
| 233 | 885 | 833 | 9 | 793 | 1594 |

To achieve a 9-qubit Clifford+$T$ circuit, we can use the optimized Clifford+$T$ decomposition for each MCT gate, then assemble these Clifford+$T$ sub-circuits by our quantum resource estimator. In the assembling process, 262 $T$ gates, 162 CNOT gates, and 374 1-qubit Clifford gates can be eliminated by simple cancellation. We also employ the quantum circuit optimizer T-par, as proposed in [3], for further optimization. Using T-par, we can obtain a new circuit with reduced $T$-depth and $T$-count, although it incurs higher costs in other metrics. In this way, we achieve two 9-qubit Clifford+$T$ circuits for the AES S-box, whose costs are presented in Table 5. This is the *first time* that detailed Clifford+$T$ circuits, with their layer structures, for the AES S-box are presented, achieving the minimal width 9. Based on these circuits, we can induce a 276-qubit Clifford+$T$ implementations for AES with $T$-depth being 12740 or 15010.

**Table 5.** Costs of different 9-qubit Clifford+$T$ circuits for the AES S-box.

| #Clifford (CNOT, 1qClifford) | #$T$ | Width | $T$-depth | Full Depth | Source |
|---|---|---|---|---|---|
| $\leq$ 12631( - , - ) | $\leq$ 9295 | 9 | - | - | [23] |
| **7465 (6028, 1437)** | **3783** | **9** | **1501** | **7180** | This work |
| **13008 (10633, 2375)** | **3447** | **9** | **1274** | **9954** | This work (T-par) |

**Implementing a pair of AES S-boxes with 16 qubits.** In one round of AES encryption, one needs to apply 16 S-boxes in `ByteSub` of the round function and 4 S-boxes in `SubByte` of key expansion. If we group every two S-boxes into a pair, there are 8 pairs of S-boxes in `ByteSub` and 4 pairs of S-boxes in `SubByte`. For a pair of AES S-boxes $(X_1, X_2) \to (S_1(X_1), S_2(X_2))$, it can be seen as an even permutation on $\mathbb{F}_{2^{16}}$, hence should have a 16-qubit NCT implementation. To construct a such minimal-width circuit, we can use a qubit allocated for implementing $S_2$ as the dirty ancilla qubit when implementing $S_1$, and vice versa. By this trick, we can easily achieve a 16-qubit circuit for a pair of AES S-boxes, based on our 8-qubit MCT circuit for the AES S-box. Here we should decompose each MCT gate by using one dirty ancilla qubit. Table 6 presents the costs of the corresponding 16-qubit NCT implementation and 16-qubit Clifford+$T$

implementation. Apparently, these 16-qubit Clifford+$T$ circuits can easily lead to 256-qubit Clifford+$T$ implementation for AES with $T$-depth being 29490 or 27740, which *achieves the theoretical minimum width*.

**Table 6.** Costs of the 16-qubit quantum circuits for a pair of AES S-boxes.

| **Type** | #NOT | #CNOT | #Toffoli | Width | Toffoli-depth | Full Depth |
|---|---|---|---|---|---|---|
| NCT | 502 | 1770 | 2140 | 16 | 1714 | 2990 |
| **Type** | #1qClifford | #CNOT | #$T$ | Width | $T$-depth | Full Depth |
| Clifford+$T$ | 3628 | 14786 | 9008 | 16 | 2949 | 15253 |
| Clifford+$T$ (T-par) | 5066 | 23976 | 8360 | 16 | 2774 | 18883 |

**Implementing the $\chi$ function of SHA3 with 5 qubits.** For SHA3, its only nonlinear component is the $\chi$ function. It corresponds to a 5-bit even permutation, therefore its NCT implementation has the minimal width 5. By utilizing the proposed SAT-based method, we can achieve a 5-qubit NCT implementation with Toffoli-count 7. It is easy to prove that without any constraint on the width, the minimal number of Toffoli gates required for implementing the $\chi$ function is 5. Therefore, this NCT circuit not only achieves the minimal width but also has an almost minimal Toffoli-count. Furthermore, we can decompose all Toffoli gates to obtain a 5-qubit Clifford+$T$ implementation. Then, for the *first time*, we obtain the quantum circuits for the $\chi$ function that achieving the minimal width 5. In Table 7, we present the costs of our circuits, and compare our Clifford+$T$ circuit with the out-of-place circuit[6] from [49].

**Table 7.** Costs of the 5-qubit quantum circuits for the $\chi$ function of SHA3.

| **Type** | #NOT | #CNOT | #Toffol | Width | Toffoli-depth | Full Depth | Source |
|---|---|---|---|---|---|---|---|
| NCT | 12 | 0 | 7 | 5 | 7 | 10 | This work |
| **Type** | #CNOT | #1qClifford | #$T$ | Width | $T$-depth | Full Depth | Source |
| Clifford+$T$ | 79 | 24 | 70 | 12 | 30 | 103 | [49] |
| | **49** | **24** | **49** | **5** | **21** | **66** | This work |

## 6 Conclusions

We resolve the problem of constructing an out-of-place circuit with the minimal $T$-depth in two steps. First, we propose an algorithm for achieving an efficient $T$-depth-s quantum circuit from an AND-depth-s classical circuit. Then, we present a general approach for addressing the problem of constructing classical circuits with the minimal AND-depth. In particular, we introduce a new

[6] In [49], only a circuit mapping $|x_1, \ldots, x_5\rangle|0\rangle$ to $|\chi(x_1, \ldots, x_5)\rangle|x_1, x_2\rangle$ was given. Here copy and uncomputation are added to obtain an out-of-place implementation.

concept called the parallel addition chain, which helps us design the minimal AND-depth circuit for the multiplicative inversion in $\mathbb{F}_{2^n}$. Based on these, we achieve a highly compact $T$-depth-3 circuit for the AES S-box. We propose an MCT decomposition-based method and a SAT-based method for constructing the minimal-width Clifford+$T$ circuits for Boolean functions. As applications, we achieve a 9-qubit Clifford+$T$ circuit for the AES S-box, and a 5-qubit Clifford+$T$ circuit for the $\chi$ function of SHA3, by which the minimal-width circuits for AES and SHA3 can be constructed. As all the methods and techniques developed in this paper are general, they can be utilized to derive quantum circuits for diverse symmetric-key ciphers with low costs of quantum resources. At `https://github.com/hzy-cas/Minimal_T-depth_Width`, new applications, including compact minimal-T-depth and minimal-width circuits for the S-boxes of SKINNY and ASCON, are presented to demonstrate the versatility of our approaches. Moreover, our methods for constructing low AND-depth and AND-count circuits may potentially be used in constructing low-latency and low-cost masking in masked hardware implementations, and reducing the latency and throughput in FHE, MPC, or ZK evaluation of a cipher.

## References

1. Abdessaied, N., Amy, M., Soeken, M., Drechsler, R.: Technology mapping of reversible circuits to Clifford+T quantum circuits. In: 46th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2016, Sapporo, Japan, May 18-20, 2016. pp. 150–155. IEEE Computer Society (2016)
2. Almazrooie, M., Samsudin, A., Abdullah, R., Mutter, K.N.: Quantum reversible circuit of AES-128. Quantum Inf. Process. **17**(5), 112 (2018)
3. Amy, M., Maslov, D., Mosca, M.: Polynomial-time $T$-depth optimization of clifford+$T$ circuits via matroid partitioning. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(10), 1476–1489 (2014)
4. Amy, M., Matteo, O.D., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.M.: Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In: SAC 2016. LNCS, vol. 10532, pp. 317–337. Springer (2016)
5. Banegas, G., Bernstein, D.J., van Hoof, I., Lange, T.: Concrete quantum cryptanalysis of binary elliptic curves. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 451–472 (2021)
6. Barenco, A., Bennett, C.H., Cleve, R., Divincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H.: Elementary gates for quantum computation. Physical Review A **52**(5) (1995)
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 313–314. Springer (2013)
8. Bilgin, B., De Meyer, L., Duval, S., Levi, I., Standaert, F.X.: Low AND depth and efficient inverses: a guide on S-boxes for low-latency masking. IACR Transactions on Symmetric Cryptology **2020**(1), 144–184 (2020)
9. Biswal, L., Bhattacharjee, D., Chattopadhyay, A., Rahaman, H.: Techniques for fault-tolerant decomposition of a multicontrolled toffoli gate. Physical Review A **100**(6), 062326 (2019)

10. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Y., Schrottenloher, A.: Quantum attacks without superposition queries: The offline Simon's algorithm. In: ASIACRYPT 2019, Kobe, Japan, December 8-12, 2019, Proceedings, Part I. pp. 552–583 (2019)
11. Bonnetain, X., Leurent, G., Naya-Plasencia, M., Schrottenloher, A.: Quantum linearization attacks. In: Advances in Cryptology - ASIACRYPT 2021, Singapore, December 6-10, 2021, Proceedings, Part I. pp. 422–452 (2021)
12. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: On quantum slide attacks. In: SAC 2019. pp. 492–519 (2019)
13. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symmetric Cryptol. **2019**(2), 55–93 (2019)
14. Bonnetain, X., Schrottenloher, A., Sibleyras, F.: Beyond quadratic speedups in quantum attacks on symmetric schemes. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 315–344. Springer (2022)
15. Boyar, J., Peralta, R.: A small depth-16 circuit for the AES s-box. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) IFIP International Information Security Conference. pp. 287–298. Springer (2012)
16. Canright, D.: A very compact s-box for AES. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 441–455. Springer (2005)
17. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography. In: ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. pp. 211–240 (2017)
18. Chen, J., Liu, Q., Fan, Y., Wu, L., Li, B., Wang, M.: New sat-based model for quantum circuit decision problem: Searching for low-cost quantum implementation. IACR Commun. Cryptol. **1**(1), 31 (2024)
19. Choo, D., Soos, M., Chai, K.M.A., Meel, K.S.: Bosphorus: Bridging ANF and CNF solvers. In: Teich, J., Fummi, F. (eds.) Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019. pp. 468–473. IEEE (2019)
20. Courtois, N., Mourouzis, T., Hulme, D.: Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits. Int. J. Adv. Intell. Syst **6**(3), 165–176 (2013)
21. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999)
22. Dobraunig, C., Kales, D., Rechberger, C., Schofnegger, M., Zaverucha, G.: Shorter signatures based on tailor-made minimalist symmetric-key crypto. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 843–857. ACM (2022)
23. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying grover's algorithm to AES: quantum resource estimates. In: Takagi, T. (ed.) Post-Quantum Cryptography - PQCrypto 2016. LNCS, vol. 9606, pp. 29–43. Springer (2016)
24. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, 1996. pp. 212–219. ACM (1996)
25. Hosoyamada, A., Sasaki, Y.: Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In: EUROCRYPT 2020, Part II. vol. 12106, pp. 249–279. Springer

26. Hosoyamada, A., Sasaki, Y.: Cryptanalysis against symmetric-key schemes with online classical queries and offline quantum computations. In: CT-RSA 2018, Proceedings. pp. 198–218 (2018)
27. Hosoyamada, A., Sasaki, Y.: Quantum Demiric-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions. In: SCN 2018. pp. 386–403 (2018)
28. Huang, Z., Sun, S.: Synthesizing quantum circuits of AES with lower $T$-depth and less qubits. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022. Lecture Notes in Computer Science, vol. 13793, pp. 614–644. Springer (2022)
29. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases. Information and computation **78**(3), 171–177 (1988)
30. Jang, K., Baksi, A., Kim, H., Song, G., Seo, H., Chattopadhyay, A.: Quantum analysis of aes. Cryptology ePrint Archive (2022)
31. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing Grover oracles for quantum key search on AES and lowmc. In: Advances in Cryptology - EUROCRYPT 2020. LNCS, vol. 12106, pp. 280–310. Springer (2020)
32. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: Advances in Cryptology - CRYPTO 2016 Proceedings. pp. 207–237. Springer (2016)
33. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2016**(1), 71–94 (2016)
34. Langenberg, B., Pham, H., Steinwandt, R.: Reducing the cost of implementing AES as a quantum circuit. IACR Cryptol. ePrint Arch. p. 854 (2019), https://eprint.iacr.org/2019/854
35. Lee, H., Jung, K.C., Han, D., Kim, P.: An algorithm for reversible logic circuit synthesis based on tensor decomposition. CoRR **abs/2107.04298** (2021), https://arxiv.org/abs/2107.04298
36. Li, Z., Gao, F., Qin, S., Wen, Q.: New record in the number of qubits for a quantum implementation of AES. Frontiers in Physics **11**, 1171753 (2023)
37. Lin, D., Xiang, Z., Xu, R., Zhang, S., Zeng, X.: Optimized quantum implementation of AES. Cryptology ePrint Archive (2023)
38. Lin, D., Yang, C., Xu, S., Tian, S., Sun, B.: On the construction of quantum circuits for s-boxes with different criteria based on the SAT solver. IACR Cryptol. ePrint Arch. p. 565 (2024), https://eprint.iacr.org/2024/565
39. Liu, Q., Preneel, B., Zhao, Z., Wang, M.: Improved quantum circuits for AES: Reducing the depth and the number of qubits. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. pp. 67–98. Springer Nature Singapore, Singapor (2023)
40. Microsoftt Q#: Quantum development, https://devblogs.microsoft.com/qsharp/
41. Naya-Plasencia, M., Schrottenloher, A.: Optimal merging in quantum k-xor and k-xor-sum algorithms. In: EUROCRYPT 2020, Part II. vol. 12106, pp. 311–340. Springer
42. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2016)
43. NIST: Digital signature standard (2013), available at https://csrc.nist.rip/publications/detail/fips/186/4/final
44. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), available at https://csrc.nist.gov/projects/post-quantum-cryptography

45. NIST: Post-quantum cryptography: Digital signature schemes (2022), available at https://csrc.nist.gov/Projects/pqc-dig-sig/standardization/call-for-proposals
46. Shende, V.V., Prasad, A.K., Markov, I.L., Hayes, J.P.: Synthesis of reversible logic circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **22**(6), 710–722 (2003)
47. Shi, H., Feng, X.: Quantum circuits of AES with a low-depth linear layer and a new structure. In: Advances in Cryptology - ASIACRYPT 2024. LNCS, vol. 15491, pp. 358–395. Springer (2024)
48. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303–332 (1999)
49. Song, G., Jang, K., Seo, H.: Improved low-depth SHA3 quantum circuit for fault-tolerant quantum computers. IACR Cryptol. ePrint Arch. p. 211 (2023), https://eprint.iacr.org/2023/211
50. Steiger, D.S., Häner, T., Troyer, M.: Projectq: an open source software framework for quantum computing. Quantum **2**, 49 (2018)
51. Stoffelen, K.: Optimizing S-box implementations for several criteria using SAT solvers. In: International Conference on Fast Software Encryption. pp. 140–160. Springer (2016)
52. Xiang, Z., Zeng, X., Lin, D., Bao, Z., Zhang, S.: Optimizing implementations of linear layers. IACR Trans. Symmetric Cryptol. **2020**(2), 120–145 (2020)
53. Zhang, F., Huang, Z.: Optimizing S-box implementations using SAT solvers: Revisited. Cryptology ePrint Archive (2023)
54. Zhang, M., Shi, T., Wu, W., Sui, H.: Optimized quantum circuit of AES with interlacing-uncompute structure. IEEE Transactions on Computers (2024)
55. Zhu, C., Huang, Z.: Optimizing the depth of quantum implementations of linear layers. In: Deng, Y., Yung, M. (eds.) Information Security and Cryptology - 18th International Conference, Inscrypt 2022, Beijing, China, December 11-13, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13837, pp. 129–147. Springer (2022)
56. Zou, J., Wei, Z., Sun, S., Liu, X., Wu, W.: Quantum circuit implementations of AES with fewer qubits. In: Advances in Cryptology - ASIACRYPT 2020. pp. 697–726. Springer (2020)

## Appendix

## A Clifford+$T$ Implementations for Toffoli, QAND, and QAND$^\dagger$

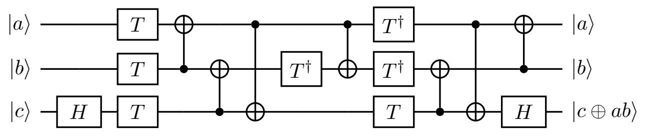


**Fig. 1.** A Clifford+$T$ implementation of the Toffoli gate with width 3 and $T$-depth 3.

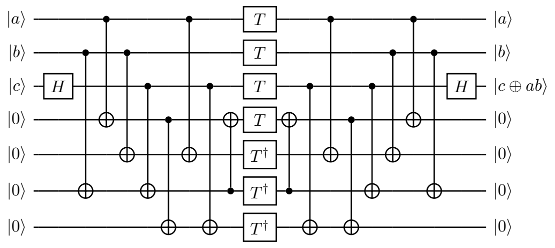
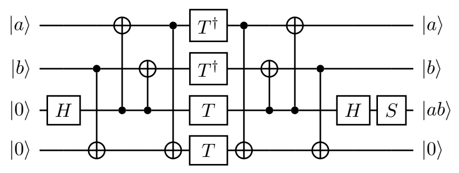
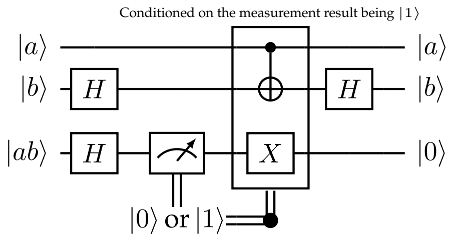


**Fig. 2.** A Clifford+$T$ implementation of the Toffoli gate with width 7 and $T$-depth 1.



(a) Quantum AND gate



Conditioned on the measurement result being $|1\rangle$
$|a\rangle$
$|b\rangle$
$|ab\rangle$
$H$
$X$
$|0\rangle$ or $|1\rangle$
$|0\rangle$

(b) Quantum AND$^\dagger$ gate

**Fig. 3.** The quantum AND gate together with its adjoint. This implementation for QAND$^\dagger$ is from the latest eprint version of [31].

*Remark 4.* Note that, in this QAND$^\dagger$ implementation, the conditioned CNOT and X gates should be applied after the measurement on the third qubit. To ensure correct full depth, one must prevent this CNOT from being moved forward during resource estimation. This can be achieved by setting a 3-qubit barrier after the measurement (as we did in our Clifford+$T$ circuits in the QASM format) or by using a 3-qubit operation instead of this measurement (as we did when using ProjectQ).

## B The Proof of Lemma 2

Lemma 2 can be easily deduced from the result of the following lemma.

**Lemma 9.** *Let $|x_1, x_2, \dots, x_n\rangle$ be the input of an $n$-qubit quantum register, where each $x_i$ is a Boolean variable. Suppose $L_1(x_1, \dots, x_n), L_2(x_1, \dots, x_n), \dots, L_m(x_1, \dots, x_n)$ are $m$ linear functions, and the rank of $L_1, L_2, \dots, L_m$ is $k$. Then to output the state $|L_1, L_2, \dots, L_m\rangle$ using a CNOT circuit, $m-k$ additional qubits are necessary and sufficient.*

*Proof.* We denote the coefficient vector of each $L_i$ as $\mathbf{a}_i = (a_{i1}, a_{i2}, \dots, a_{in})$. Since $k \le n$, we consider two cases: $k = n$ and $k < n$.

If $k = n$, to store $|L_1, L_2, \dots, L_m\rangle$, we need at least $m - n$ additional qubits. With $m - n$ additional qubits, using CNOT gates to output $|L_1, L_2, \dots, L_m\rangle$ is equivalent to transforming

$$\begin{bmatrix} 1\,0\cdots\,0 \\ 0\,1\cdots\,0 \\ \ddots \\ 0\,0\cdots\,1 \\ 0\,0\cdots\,0 \\ \vdots \\ 0\,0\;\;0\;\;0 \end{bmatrix} \quad \text{to} \quad A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \\ a_{(n+1)1} & a_{(n+1)2} & \dots & a_{(n+1)n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

by applying row addition operations. This can be addressed by performing Gaussian elimination to $A$.

If $k < n$, without loss of generality, suppose $\{L_1, L_2, \dots, L_k\}$ is the maximal linearly independent set for $\{L_1, L_2, \dots, L_m\}$. Assume with $s$ additional qubits where $s < m - k$, we can output $\{L_1, L_2, \dots, L_m\}$. It means that by applying row addition operations to the matrix

$$U = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix}_{(n+s)\times n},$$

we can obtain $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m$ from the rows of $U$. Note that $\mathrm{rank}(U) = n$, indicating that, after row addition operations, there will always be $s$ rows that are linearly dependent on others. However, within the set $\{\mathbf{a}_i\}_{1\leq i\leq m}$, there are $m-k$ vectors that are linearly dependent on others, leading to a contradiction. This implies we need at least $m-k$ additional qubits. Now suppose we have $m-k$ additional qubits. Obviously, we can find $n-k$ unit vectors $\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \ldots, \mathbf{e}_{i_{n-k}}$ such that the matrix $V = (\mathbf{a}_1, \ldots, \mathbf{a}_m, \mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_{n-k}})^T$ has rank $n$. Then, by some row addition operations, we can transform $V$ into the identity matrix with order $n+m-k$. These row addition operations induce a CNOT circuit that outputs $|L_1, L_2, \ldots, L_m\rangle$ as part of its final state. This proves the lemma. $\square$

## C The Algorithm `ClassicalToQuantum`

**Algorithm 2:** `ClassicalToQuantum`

```
input : An $n$-bit input and $m$-bit output classical circuit $\mathcal{C}$ for a Boolean
        function $\mathcal{F}$ with AND-depth $s$ and under its algebraic expression
        form.
output: A forward $\mathfrak{C}^0$-circuit for $\mathcal{F}$ with Toffoli-depth $s$.
1  Set $a = 0$, $dim = n$;
2  $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_s, \mathcal{R}$ be the output of ExtractLinear($\mathcal{C}$) ;   /* Extract the
     linear expressions */
3  Let $w$ and $w_a$ be the outputs of MinWidth($n, \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_s, \mathcal{R}$);
4  Set $S_0 = (x_1, x_2, \ldots, x_n, \mathbf{0})_w$, where $\mathbf{0}$ denotes some zeros;   /* $(\cdot)_w$ means a
     sequence containing $w$ elements */
5  for $i$ from 1 to $s-1$ do
6      Let $g_i = |\mathcal{B}_i|/2$, $a_i = |\mathcal{B}_i| - \mathrm{rank}(\mathcal{B}_i)$ ;   /* $a_i + g_i$: number of ancilla
         qubits required in this layer */
7      if $|\mathbf{0}_{\in S_{i-1}}| \geq g_i + a_i$ then
8          $S_i = ((S_{i-1} \setminus \mathbf{0})^*, \mathcal{B}_i, \mathbf{0})_w$, where $(S_{i-1} \setminus \mathbf{0})^*$ are $|S_{i-1} \setminus \mathbf{0}| - rank(\mathcal{B}_i)$
             elements selected from $S_{i-1} \setminus \mathbf{0}$ such that $rank(S_i) = dim$;
9      end
10     else
11         $S_i = ((S_{i-1} \setminus \mathbf{0})^\#, \mathcal{B}_i, \mathbf{0})_w$, where $(S_{i-1} \setminus \mathbf{0})^\#$ are $dim - rank(\mathcal{B}_i)$
             elements selected from $S_{i-1} \setminus \mathbf{0}$ such that $rank(S_i) = dim$
12     end
13     $w_0 = |S_i \setminus \mathbf{0}|$
14     Find an in-place CNOT circuit $\mathsf{C}_i$ that maps $|S_{i-1}\rangle$ to $|S_i\rangle$ with the input
         and output wires labeled as $\{y_1, y_2 \ldots, y_w\}$.
15     $\mathsf{T}_i = \{\mathsf{Toffoli}_{w_0-2g_i, w_0-2g_i+1, w_0+1}, \ldots, \mathsf{Toffoli}_{w_0-1, w_0, w_0+g_i}\}$ ;
16     $X_i = (x_{dim+1}, x_{dim+2}, \ldots, x_{dim+g_i})$, $S_i = (S_i \setminus \mathbf{0}, X_i, \mathbf{0})_w$, $dim = dim + g_i$;
17 end
18 $g_s = |\mathcal{B}_s|/2$, $S_s = (S^*_{s-1}, \mathcal{B}_s, \mathbf{0})_w$, where $S^*_{s-1}$ are $w - g_s - |\mathcal{B}_s|$ elements
     selected from $S_{s-1}$ such that $rank(S_s) = dim$;
19 Find an in-place CNOT circuit $\mathsf{C}_s$ that maps $|S_{s-1}\rangle$ to $|S_s\rangle$ with the input
     and output wires labeled as $\{y_1, y_2 \ldots, y_w\}$;
20 $\mathsf{T}_s = \{\mathsf{Toffoli}_{w-3g_s+1, w-3g_s+2, w-g_s+1}, \ldots, \mathsf{Toffoli}_{w-g_s-1, w-g_s, w}\}$ ;
21 $X_s = (x_{dim+1}, x_{dim+2}, \ldots, x_{dim+g_s})$, $dim = dim + g_s$, $w = w + w_a$;
22 $S_s = (S^*_{s-1}, \mathcal{B}_s, X_s, \mathbf{0})_w$, $S_{s+1} = (S^*_{s-1}, \mathcal{B}_s, X^*_s, \mathcal{R})_w$, where $X^*_s$ are elements
     selected from $X_s$ such that $rank(S_{s+1}) = dim$;
23 Find an in-place CNOT circuit $\mathsf{C}_{s+1}$ that maps $|S_s\rangle$ to $|S_{s+1}\rangle$ with the input
     and output wires labeled as $\{y_1, y_2 \ldots, y_w\}$;
24 return $\{\mathsf{C}_1, \mathsf{T}_1, \mathsf{C}_2, \mathsf{T}_2, \ldots, \mathsf{C}_s, \mathsf{T}_s, \mathsf{C}_{s+1}\}$.
```

Here we explain the main steps of Algorithm 2:

- Function `ExtractLinear` is used to obtain $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_s, \mathcal{R}\}$, which are $s$ sequences of linear expressions corresponding to the inputs of AND gates

```
Function: ExtractLinear
input  : A $n$-bit input classical circuit $\mathcal{C}$ with AND-depth $s$ and under its
         algebraic expession form.
output: $s$ sequences of linear functions, which correspond to the inputs of
         AND gates with different AND-depth, and a linear function
         corresponding to the circuit outputs.
1  Calculate the AND-depth of each gate, and regroup them into different sets:
     $\mathcal{L}_0, \ldots, \mathcal{L}_s, \mathcal{A}_1, \ldots, \mathcal{A}_s$, where $\mathcal{L}_i$ consists of XOR gates with AND-depth $i$,
     and $\mathcal{A}_i$ consists of AND gates with AND-depth $i$ ;
2  Set the circuit inputs as $\{x_1, \ldots, x_n\}$, $N = n$, vars $= \{x_1, \ldots, x_n\}$;
3  for $i$ from 1 to $s$ do
4      Set $\mathcal{B}_i = \emptyset$, $g_i = |\mathcal{A}_i|$, $\texttt{vars}_0 = \emptyset$;
5      for $j$ from 1 to $g_i$ do
6          Let $G_j$ be the $j$-th AND gate in $\mathcal{A}_i$;
7          For the two input nodes of $G_j$, compute their linear expressions w.r.t.
             vars based on the algebraic expressions in $\mathcal{L}_{i-1}$, then append these
             two linear expressions to $\mathcal{B}_i$ ;
8          Set the value of the output node of $G_j$ to be $x_{N+j}$;
9          $\texttt{vars}_0 = \texttt{vars}_0 \cup \{x_{N+j}\}$;
10     $\texttt{vars} = \texttt{vars} \cup \texttt{vars}_0$;
11     $N = N + |\texttt{vars}_0|$;
12 Set $\mathcal{R}$ to be the sequence containing the linear expressions for the circuit
     outputs w.r.t. vars;
13 return $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_s, \mathcal{R}\}$.
```

in $s$ AND layers and a sequence of linear expressions corresponding to the circuit outputs. Suppose there are $g_i$ AND gates with AND-depth $i$, then the $g_i$ outputs of these AND gates are denoted as

$$X_i = \{x_{n+g_1+\cdots+g_{i-1}+1}, \ldots, x_{n+g_1+\cdots+g_{i-1}+g_i}\}.$$

Hence, after the $i$-th AND layer, the dimension of the linear space increases by $g_i$.

- Function `MinWidth` outputs $w$, the number of qubits required to implement the first $s$ AND layers, and $w_a$, the number of extra qubits required to store some output bits. Lemma 2 guarantees that $w$ and $w_a$ are minimal if we don't increase the Toffoli-count of the forward circuit[7]. In these $w$ qubits, $k = rank(\mathcal{R}') = m - w_a$ target qubits of the last Toffoli layer are used to store $k$ output bits after applying the last CNOT sub-circuits. In this way, we don't need to apply the reverse of these $k$ Toffoli gates in the uncomputation process, hence can reduce the Toffoli-count by $k$. Obviously, the width of the entire circuit is $w + w_a$.

[7] Sometimes, we can clean up the target qubits of some previous Toffoli gates, by adding some additional Toffoli gates in the subsequent Toffoli layer, hence obtain some clean ancilla qubits.

```
Function: MinWidth
input  : a number $n$, and $s+1$ sequences of linear functions $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_s, \mathcal{R}$
         w.r.t $N$ variables $x_1, x_2 \ldots, x_N$.
output:  $w$, the number of qubits required for implementing the first $s$ AND
         layers, and $w_a$, the number of extra qubits used to store some output
         bits.
1  Set $w = n$, $c = 0$ ;  /* $w$ is the number of used qubits, $c$ is the number
     of qubits that can be returned to $|0\rangle$ */
2  for $i = 1$ to $s-1$ do
3      $a_i = |\mathcal{B}_i| - rank(\mathcal{B}_i)$, $b_i = |\mathcal{B}_i|/2$;   /* $a_i + b_i$ qubits are required to
         store the inputs and outputs of the $i$-th AND layer */
4      if $c < a_i + b_i$ then  $w = w + a_i + b_i - c$;
5      $c = w - \sum_{j=1}^{i} b_j - n$;
6  $g = |\mathcal{B}_s|/2$, $k = N - g$;
7  Let $\mathcal{R}'$ be the sequence of linear functions (including zero functions) obtained
     from $\mathcal{R}$ by removing all terms involving the variables $x_1, x_2, \ldots, x_k$;
8  $a_s = |\mathcal{B}_s| - rank(\mathcal{B}_s)$, $w = w + rank(\mathcal{R}')$;   /* $rank(\mathcal{R}')$ new allocated
     qubits are required to store a part of the final outputs */
9  if $c + rank(\mathcal{R}') < a_s + g$ then $w = w + a_s + g - c - rank(\mathcal{R}')$;
10 $w_a = |\mathcal{R}| - rank(\mathcal{R}')$;
11 return $w$, $w_a$
```

- $\mathsf{C}_i$ in Step 14 is the CNOT sub-circuit that generates $|\mathcal{B}_i\rangle$ based on a sequence of linear expressions $S_{i-1}$. Step 5-13 are constructing the input $|S_{i-1}\rangle$ and the output $|S_i\rangle$ of $\mathsf{C}_i$. At this point, the dimension of the linear space is $dim = n + g_1 + \cdots + g_{i-1}$, and $S_{i-1}$ has the full rank. After determining $|S_{i-1}\rangle$ and $|S_i\rangle$, we can construct an invertible Boolean matrix that maps $|S_{i-1}\rangle$ to $|S_i\rangle$. We call this matrix the *state transform matrix*. Finding an efficient $\mathsf{C}_i$ is equivalent to finding an efficient CNOT circuit that implements the state transform matrix, and this can be addressed by some existing algorithms. For example, the heuristic algorithm proposed in [52]. Sometimes, after $\mathsf{C}_i$, some previously used qubits are restored to $|0\rangle$, and these qubits will be used to store the outputs of the following AND gates.
- After executing $\mathsf{C}_i$, the $2g_i$ inputs of the $g_i$ AND gates with AND-depth $i$ are all stored in $2g_i$ different qubits. Then, we can obtain the $g_i$ AND outputs by applying $g_i$ Toffoli gates in parallel. These outputs are stored in all qubits previously taking the state $|0\rangle$.
- The process for generating the input of the last AND layer and the circuit output is a little different. As we mentioned before, we use $k = rank(\mathcal{R}')$ target qubits of the last Toffoli layer $\mathsf{T}_s$ to store $k$ output bits. We should ensure that these $k$ qubits are not involved in $\{\mathsf{C}_1, \mathsf{T}_1, \mathsf{C}_2, \ldots, \mathsf{T}_{s-1}\}$, since uncomputation will return the qubits involved in $\{\mathsf{C}_1, \mathsf{T}_1, \mathsf{C}_2, \ldots, \mathsf{T}_{s-1}\}$ to their initial states. Finally, the algorithm returns a forward NCT circuit for $\mathcal{F}$. With this forward circuit, we can construct an entire out-of-place circuit

for $\mathcal{F}$: $\{\mathsf{C}_1, \mathsf{T}_1, \ldots, \mathsf{C}_s, \mathsf{T}_s, \mathsf{C}_{s+1}, \mathsf{T}_s^*, \mathsf{C}_s^\dagger, \ldots, \mathsf{T}_1^\dagger, \mathsf{C}_1^\dagger\}$. Here $\mathsf{T}_s^*$ is a part of $\mathsf{T}_s^\dagger$, which cleans up the qubits that are not used to store the final output.

- The quantum state before and after each sub-circuit can be illustrated as follows. Here $X_0$ denotes $x_1, \ldots, x_n$, $X_s^*$ is a subset of $X_s$, and $S_i^*$ is a subset of $S_i$.

$$|S_0\rangle_w \xrightarrow{\mathsf{C}_1} |S_0^*, \mathcal{B}_1, 0\rangle_w \xrightarrow{\mathsf{T}_1} |S_1\rangle_w = |\mathcal{B}_1, X_0^*, X_1, 0\rangle_w \xrightarrow{\mathsf{C}_2} |S_1^*, \mathcal{B}_2, 0\rangle_w \xrightarrow{\mathsf{T}_2} |S_2\rangle_w = |\mathcal{B}_2, S_1^*, X_2, 0\rangle_w \xrightarrow{\mathsf{C}_3} \cdots \xrightarrow{\mathsf{C}_s} |S_{s-1}^*, \mathcal{B}_s, 0\rangle_w \xrightarrow{\mathsf{T}_s} |S_{s-1}^*, \mathcal{B}_s, X_s, 0\rangle_w \xrightarrow{\mathsf{C}_{s+1}} |S_{s-1}^*, \mathcal{B}_s, X_s^*, \mathcal{R}\rangle_w$$

A SageMath implementation for this algorithm is available at https://github.com/hzy-cas/Minimal_T-depth_Width. Its outputs are the state transform matrices and the algebraic forms of each Toffoli layer. From these state transform matrices, one can construct the corresponding CNOT sub-circuits by using the heuristic algorithm proposed in [52] or the SAT-based method proposed in [28]. The depth of these CNOT sub-circuits can be further reduced by using the method proposed in [55]. Note that, if the outputs of an obtained CNOT sub-circuit form a permutation of the rows of the state transform matrix, we should renumber the qubits for the subsequent sub-circuits.

## D Improving Techniques for the Top-down Approach

**(1) Performing Gaussian elimination.** Before Step 4 of Algorithm 1, we can perform Gaussian elimination on $\mathcal{G} = \{g_1, g_2, \ldots, g_m\} \subseteq \mathbb{M}_s$ by regarding distinct monomials as distinct variables. Through this process, we can simplify these $g_i$'s, and identify linear dependencies among $ML_s(g_1), ML_s(g_2), \ldots, ML_s(g_m)$. If a $ML_s(g_k)$ can be written as the linear combination of some other $ML_s(g_i)$'s, then after Gaussian elimination, $g_k$ is converted into a polynomial in $\mathbb{M}_{\leq s-1}$. This technique can improve the efficiency of finding covers, and prevent the case that the cover of $g_k$ includes polynomials not present in other covers, which leads to a higher AND-count.

**(2) Using linear combinations of input polynomials.** In Algorithm 1, we find the max-depth cover $\mathcal{C}_i$ of each $g_i \in \mathcal{G}$ individually. Actually, $\cup_i \mathcal{C}_i$ forms a max-depth cover of $\mathcal{G}$. A max-depth cover of a polynomial set $\mathcal{G}$ is defined to be a set of Boolean polynomials $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ satisfying the condition that for any $g_i \in \mathcal{G}$ there is a subset of $\mathcal{C}$ that serves as a cover of $g_i$. Obviously, if we can find a cover $\mathcal{C}$ of $\mathcal{G}$ with $|\mathcal{C}| < |\cup_i \mathcal{C}_i|$, we can construct a circuit with lower AND-count. For example, let $\mathcal{G} = \{g_1, g_2\} = \{x_1x_3 + x_2x_4, x_1x_4 + x_2x_3\}$. Then $\{(x_1 + x_2)(x_3 + x_4), x_1x_3, x_2x_4\}$ is a cover of $\mathcal{G}$, since $\{x_1x_3, x_2x_4\}$ is a cover of $g_1$, and $\{(x_1 + x_2)(x_3 + x_4), x_1x_3, x_2x_4\}$ is a cover of $g_2$. In contrast, if we individually find the minimal-size covers of $g_1$ and $g_2$, we can not obtain a cover of $\mathcal{G}$ with size 3.

To directly find a cover of $\mathcal{G}$, we can modify the SAT-based method introduced before. For each $g_j$, we have the relation $g_j + \sum_{i=1}^{k} a_{j,i} \mathsf{D}_i^1 \cdot \mathsf{D}_i^2 + \mathsf{R}_j = 0$, where $\mathsf{D}_i^1, \mathsf{D}_i^2, \mathsf{R}_j$ are defined as Equation (2), and $a_{j,i}$ is a Boolean variable indicating whether $\mathsf{D}_i^1 \cdot \mathsf{D}_i^2$ is chosen as an element in the cover of $g_j$. Nevertheless, this encoding way will increase the degree of the equations, which significantly reduces the efficiency of the subsequent solving process.

Here, we consider an alternative approach. Suppose $\mathcal{G} = \{g_1, g_2, \ldots, g_m\} \subseteq \mathbb{M}_s$. We assume $ML_s(g_1), ML_s(g_2), \ldots, ML_s(g_m)$ are linearly independent, otherwise, we can apply the technique of performing Gaussian elimination on $\mathcal{G}$. Let $\mathcal{G}' = \{L_1(g_1, \ldots, g_m), \ldots, L_m(g_1, \ldots, g_m)\}$, where $L_1, L_2, \ldots, L_m$ are $m$ independent linear functions. Then a cover of $\mathcal{G}'$ can be converted into a cover of $\mathcal{G}$. To find a cover of $\mathcal{G}'$, we still individually find a cover of each polynomial in $\mathcal{G}'$. By considering all possible $L_1, L_2, \ldots, L_m$, we can obtain a cover of $\mathcal{G}$ with lower size. For the above example, let $\mathcal{G}' = \{g_1' = g_1 + g_2, g_2' = g_2\}$, then $\{(x_1 + x_2)(x_3 + x_4)\}$ is a cover of $g_1'$, and $\{x_1x_4, x_2x_3\}$ is a cover of $g_2'$. From these two covers, we can construct a cover of $\mathcal{G}$ with size 3.

**(3) Finding covers of two m-layers.** Suppose $\mathcal{C}_1$ and $\mathcal{C}_2$ are two covers of $f$, and they have the same size. It is obvious that the sizes of the covers for the factors and remainders of $\mathcal{C}_1$ and $\mathcal{C}_2$ may be different. Therefore, to reduce the total AND-count, we can consider simultaneously finding a cover of $f$ and covers of the corresponding factors and remainder. For the SAT-based method, we can set the sizes of these covers to be different values, then solve the following equations,

$$\left\{\begin{array}{l} \mathrm{EQN}_c(f, d, k) \\ \mathrm{EQN}_c(\mathsf{D}_1^1, d-1, t_1), \mathrm{EQN}_c(\mathsf{D}_1^2, d-1, t_2) \\ \qquad \cdots \\ \mathrm{EQN}_c(\mathsf{D}_k^1, d-1, t_{2k-1}), \mathrm{EQN}_c(\mathsf{D}_k^2, d-1, t_{2k}) \\ \mathrm{EQN}_c(\mathsf{R}, d-1, t_{2k+1}) \end{array}\right\}. \tag{6}$$

Here $\mathsf{D}_\mathsf{j}^1, \mathsf{D}_\mathsf{j}^2, \mathsf{R}$ are defined as in Equation 2.

## E The Degree of the Boolean Function Corresponding to the Multiplicative Inversion in $\mathbb{F}_{2^n}$

For a number $m$, its 2-Hamming weight is defined as the number of 1's in its binary expression.

**Lemma 10.** *Suppose $\{\beta_1, \beta_2, \ldots, \beta_n\}$ is a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, and $X = x_1\beta_1 + x_2\beta_2 + \cdots + x_n\beta_n$ is an element in $\mathbb{F}_{2^n}$. Let $F(X)$ be an univariate polynomial about $X$ with $\deg(F) < 2^n$. Assume the largest 2-Hamming weight of the degrees of terms in $F(X)$ is $d$. If we represent $F(X)$ as $f_1(x_1, x_2, \ldots, x_n)\beta_1 + f_2(x_1, x_2, \ldots, x_n)\beta_2 + \cdots + f_n(x_1, x_2, \ldots, x_n)\beta_n$, then $\deg(f_i) \le d$ for $1 \le i \le n$. Moreover, $\deg(f_s) = d$ for some $s$.*

*Proof.* For any $k$, $X^{2^k} = x_1\beta_1^{2^k} + x_2\beta_2^{2^k} + \cdots + x_n\beta_n^{2^k}$. Since $\beta_1^{2^k}, \beta_2^{2^k}, \ldots, \beta_n^{2^k}$ are fixed elements in $\mathbb{F}_{2^n}$, we have $(\beta_1^{2^k}, \beta_2^{2^k}, \ldots, \beta_n^{2^k}) = (\beta_1, \beta_2, \ldots, \beta_n)B$, where $B$

is an $n \times n$ matrix over $\mathbb{F}_2$. Then

$$X^{2^k} = (\beta_1, \beta_2, \ldots, \beta_n) B (x_1, x_2, \ldots, x_n)^T$$

Hence $X^{2^k}$ can be written as $L_1(x_1, x_2, \ldots, x_n)\beta_1 + L_2(x_1, x_2, \ldots, x_n)\beta_2 + \cdots + L_n(x_1, x_2, \ldots, x_n)\beta_n$, for some linear functions $L_1, L_2, \ldots, L_n$.

Consider the term $X^{2^{t_1}+2^{t_2}+\cdots+2^{t_d}}$ whose degree has the largest 2-Hamming weight $d$. Since $X^{2^{t_1}+2^{t_2}+\cdots+2^{t_d}} = X^{2^{t_1}} X^{2^{t_2}} \cdots X^{2^{t_d}}$, it obvious that, if we represent $X^{2^{t_1}+2^{t_2}+\cdots+2^{t_d}}$ as $g_1(x_1, x_2, \ldots, x_n)\beta_1 + g_2(x_1, x_2, \ldots, x_n)\beta_2 + \cdots + g_n(x_1, x_2, \ldots, x_n)\beta_n$, we have $\deg(g_1), \deg(g_2), \ldots, \deg(g_n) \leq d$. Similarly, for other terms, the total degrees of the corresponding multivariate polynomials are all bounded by $d$, and $f_1, f_2, \ldots, f_n$ are sums of these multivariate polynomials, hence have total degrees not bigger than $d$.

Now we prove at least one $f_s$ has total degree $d$. Assume $\deg(f_i) < d$ for all $1 \leq i \leq n$. We have the following relation between $\{X, X^2, \ldots, X^{2^{n-1}}\}$ and $\{x_1, x_2, \ldots, x_n\}$:

$$A \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} X \\ X^2 \\ \vdots \\ X^{2^{n-1}} \end{bmatrix}, \text{ where } A = \begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_n \\ \beta_1^2 & \beta_2^2 & \cdots & \beta_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_1^{2^{n-1}} & \beta_2^{2^{n-1}} & \cdots & \beta_n^{2^{n-1}} \end{bmatrix}.$$

Obviously, $A$ is invertible, otherwise we can obtain a univariate polynomial which has degree $2^{n-1}$, but vanishes for all $2^n$ points in $\mathbb{F}_{2^n}$, which leads to a contradiction. Hence $(x_1, x_2, \ldots, x_n)^T = A^{-1}(X, X^2, \ldots, X^{2^{n-1}})^T$. This means $x_i = t_{i,1}X + t_{i,2}X^2 + t_{i,n}X^{2^{n-1}}$, for some fixed elements $t_{i,j}, t_{i,2}, \ldots, t_{i,n} \in \mathbb{F}_{2^n}$. Then, a monomial $x_{i_1}x_{i_2}\cdots x_{i_k}$ with total degree $k$ is equal to

$$T(X) = (\sum_{j=1}^{n} t_{i_1,j} X^{2^{j-1}})(\sum_{j=1}^{n} t_{i_2,j} X^{2^{j-1}}) \cdots (\sum_{j=1}^{n} t_{i_k,j} X^{2^{j-1}})$$

It is easy to see that the degrees of terms in $T(X)$ have 2-Hamming weight not larger than $k$. In this way, since $\deg(f_i) < d$, we can convert $f_i(x_1, x_2, \ldots, x_n)$ into some $g_i(X)$ having the property that the degrees of terms in $g_i(X)$ have 2-Hamming weight less than $d$. Furthermore, since $F(X) = f_1\beta_1 + f_2\beta_2 + \cdots + f_n\beta_n$, the degrees of terms in $F(X)$ should have 2-Hamming weight less than $d$, a contradiction. This proves that at least one $f_s$ has total degree $d$. □

We should notice that in Lemma 10, some $f_i$ may have degree less than $d$.

*Example 6.* Let $X \in \mathbb{F}_{2^2}$ and $X = x_1\beta + x_2$, where $\beta$ is a root of the irreducible polynomial $y^2 + y + 1$ over $\mathbb{F}_2$. Obviously, $f(X) = X^3 + X^2$ has 2-Hamming weight 2. We can easily deduce that $f(X)$ can be represented as $x_1\beta + x_1x_2$. Then the first coordinate Boolean polynomial $x_1$ has degree 1, which is less than the 2-Hamming weight of $f(X)$.

Now suppose $\{\beta_1, \beta_2, \ldots, \beta_n\}$ is a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Let $X = x_1\beta_1 + x_2\beta_2 + \cdots + x_n\beta_n$ be an element in $\mathbb{F}_{2^n}$. Since

$$X^{-1} = X^{2^n-2} = X^{2^{n-1}+2^{n-2}+\cdots+2},$$

its degree has 2-Hamming weight $n-1$. According to Lemma 10, if we write $X^{-1}$ as $f_1(x_1, x_2, \ldots, x_n)\beta_1 + f_2(x_1, x_2, \ldots, x_n)\beta_2 + \cdots + f_n(x_1, x_2, \ldots, x_n)$, then $\deg(f_i) \leq n-1$, for all $1 \leq i \leq n$, and $\deg(f_s) = n-1$ for some $s$. This gives the degree of the Boolean function corresponding to the multiplicative inversion in $\mathbb{F}_{2^n}$.

## F The AND-count for Implementing the Multiplication in $\mathbb{F}_{2^n}$

### F.1 The Proof of Lemma 3

**Lemma 3.** *If $k = 2^r s$, for some positive number $r$ and odd number $s$, then the multiplication in $\mathbb{F}_{2^k}$ can be implemented by one AND layer and $3^r s^2$ AND gates.*

*Proof.* For two elements $a = a_1\beta_1 + a_2\beta_2 + \cdots a_n\beta_k$ and $b = b_1\beta_1 + b_2\beta_2 + \cdots b_k\beta_k$ in $\mathbb{F}_{2^k}$. A trivial way to implement $a \cdot b$ is first simultaneously implementing $a_i b_j$ for all $1 \leq i, j \leq k$, then using XOR gates to achieve the final output. This requires 1 AND layer and $k^2$ AND gates

If $k = 2s$, then $a$, $b \in \mathbb{F}_{2^{2s}}$ can be expressed as $a = a_1\gamma_1 + a_2\gamma_2$, and $b = b_1\gamma_1 + b_2\gamma_2$, where $a_1, a_2, b_1, b_2 \in \mathbb{F}_{2^s}$ and $\{\gamma_1, \gamma_2\}$ is a basis of $\mathbb{F}_{2^{2s}}$ over $\mathbb{F}_{2^s}$. We have $a \cdot b = a_1b_1\gamma_1^2 + (a_1b_2 + a_2b_1)\gamma_1\gamma_2 + a_2b_2\gamma_2^2$. Apparently, from $a_1b_1$, $(a_1 + a_2)(b_1 + b_2)$, $a_2b_2$, we can construct $a_1b_2 + a_2b_1$ by linear operations, and subsequently, construct $a \cdot b$ by linear operations. Therefore, we only need 3 multiplications in $\mathbb{F}_{2^s}$. If we implement the multiplication in $\mathbb{F}_{2^s}$ by the above trivial way, then the 3 multiplications in $\mathbb{F}_{2^s}$ can be implemented by one AND layer and $3s^2$ AND gates. By this technique, we reduce the AND-count from $(2s)^2$ to $3s^2$. Furthermore, if $s = 2t$ for some $t$, then we can apply the above technique again and further reduce the number of AND gates. Recursively, we can prove the lemma. $\square$

### F.2 A Generalization of Lemma 5

Suppose $b = a2^{st}$ and $n = sk$ for some integers $s$, $t$, and $k$. Then $\alpha^a$ can be expressed as $\alpha_1\beta_1 + \alpha_2\beta_2 + \cdots + \alpha_k\beta_k$, where each $\alpha_i \in \mathbb{F}_{2^s}$ and $\{\beta_1, \beta_2, \ldots, \beta_k\}$ is a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_{2^s}$.

Since $\alpha_i^{2^{st}} = \alpha_i$, we have $\alpha^b = (\alpha^a)^{2^{st}} = \alpha_1^{2^{st}}\beta_1^{2^{st}} + \alpha_2^{2^{st}}\beta_2^{2^{st}} + \cdots + \alpha_k^{2^{st}}\beta_k^{2^{st}} = \alpha_1\beta_1^{2^{st}} + \alpha_2\beta_2^{2^{st}} + \cdots + \alpha_k\beta_k^{2^{st}} = L_1(\alpha_1, \ldots, \alpha_k)\beta_1 + L_2(\alpha_1, \ldots, \alpha_k)\beta_2 + \cdots +$

$L_k(\alpha_1, \ldots, \alpha_k)\beta_k$, for some linear polynomials $L_1, L_2, \ldots, L_k$. Therefore, $\alpha^a \cdot \alpha^b$ is equal to

$$\begin{aligned}(\alpha_1\beta_1 + \cdots + \alpha_k\beta_k)(L_1(\alpha_1, \ldots, \alpha_k)\beta_1 + \cdots + L_k(\alpha_1, \ldots, \alpha_k)\beta_k) \\ = G_1(\alpha_1, \ldots, \alpha_k)\beta_1 + \cdots + G_k(\alpha_1, \ldots, \alpha_k)\beta_k,\end{aligned}$$

where $G_1, G_2, \ldots, G_k$ are some quadratic polynomials. Obviously, if we have already obtain $\alpha_i^2$, $\alpha_i\alpha_j$ for all $1 \leq i \neq j \leq k$, then we can generate $\alpha^a \cdot \alpha^b$ by some linear operations. Note that, generating $\alpha_i^2$ does not need AND gates, and we need $k(k-1)/2$ multiplications in $\mathbb{F}_{2^s}$ to generate all $\alpha_i\alpha_j$'s. Apparently, these multiplication can be implemented by one AND layer. Therefore, we have the following Lemma.

**Lemma 11.** *If $n = sk$ and $\alpha$ is an element in $\mathbb{F}_{2^n}$. Then the multiplication of $\alpha^a$ and $\alpha^b$, with $b = a2^{st}$ for some $t$, can be implemented by one AND layer and $\omega(s)(k(k-1)/2)$ AND gates.*

It is easy to see that, for any $n$, we can set $s = 1$ to obtain an implementation of $\alpha^a \cdot \alpha^{a2^t}$ with one AND layer and $n(n-1)/2$ AND gates, which reduces the AND gates by a ratio of $\frac{n+1}{2n}$ compared to the trivial implementation. With $s$ increasing, this technique can reduce more AND gates.

## G A Classical Circuit for the AES S-box with AND-depth-3, AND-count 42, and XOR-count 88

For this circuit, the inputs are $X_0, X_1, \ldots, X_7$, and the outputs are $Y_0, Y_1, \ldots, Y_7$.

`Linear Layer 1:`
$L_0 = X_0 + X_3$, $L_1 = X_0 + X_6$, $L_2 = X_0 + X_5$, $L_3 = L_{0+}L_2$, $L_4 = L_{1+}L_3$, $L_5 = X_4 + L_4$, $L_6 = X_5 + L_5$, $L_7 = X_7 + L_6$, $L_8 = X_1 + X_2$, $L_9 = X_7 + L_8$, $L_{10} = X_3 + L_9$, $L_{11} = L_{4+}L_{10}$, $L_{12} = L_{2+}L_{11}$, $L_{13} = L_{1+}L_{12}$, $L_{14} = L_{7+}L_9$, $L_{15} = X_1 + L_0$, $L_{16} = L_{5+}L_{15}$, $L_{17} = X_7 + L_{16}$, $L_{18} = L_{9+}L_{17}$, $L_{19} = L_{14} + L_{16}$

`AND Layer 1:`
$M_0 = L_7 \cdot L_{11}$, $M_1 = X_7 \cdot L_{10}$, $M_2 = L_9 \cdot L_{12}$, $M_3 = L_{13} \cdot L_{17}$, $M_4 = L_4 \cdot L_6$, $M_5 = L_0 \cdot L_{16}$, $M_6 = L_1 \cdot L_{18}$, $M_7 = L_2 \cdot L_{14}$, $M_8 = L_3 \cdot L_{19}$,

`Linear Layer 2:`
$L_{20} = X_2 + M_0$, $L_{21} = L_{1+}M_2$, $L_{22} = L_{15} + M_4$, $L_{23} = L_{22} + M_6$, $L_{24} = L_{20} + L_{21}$, $L_{25} = L_{23} + L_{24}$, $L_{26} = M_{1+}M_3$, $L_{27} = L_{23} + L_{26}$, $L_{28} = L_{25} + L_{27}$, $L_{29} = L_{18} + M_5$, $L_{30} = X_0 + M_7$, $L_{31} = L_{21} + M_8$, $L_{32} = L_{29} + M_6$, $L_{33} = L_{31} + L_{32}$, $L_{34} = L_{25} + L_{33}$, $L_{35} = L_{30} + M_3$, $L_{36} = L_{32} + L_{35}$, $L_{37} = L_{27} + L_{36}$, $L_{38} = L_{34} + L_{37}$, $L_{39} = L_{33} + L_{36}$,

`AND Layer 2:`
$M_9 = L_{33} \cdot L_{34}$, $M_{10} = L_{36} \cdot L_{37}$, $M_{11} = L_{38} \cdot L_{39}$, $M_{12} = L_{11} \cdot L_{33}$, $M_{13} = L_{10} \cdot L_{36}$, $M_{14} = L_{12} \cdot L_{34}$, $M_{15} = L_{13} \cdot L_{37}$, $M_{16} = L_4 \cdot L_{39}$, $M_{17} = L_0 \cdot L_{27}$, $M_{18} = L_1 \cdot L_{38}$, $M_{19} = L_2 \cdot L_{25}$, $M_{20} = L_3 \cdot L_{28}$, $M_{21} = L_7 \cdot L_{33}$, $M_{22} = X_7 \cdot L_{36}$, $M_{23} = L_9 \cdot L_{34}$, $M_{24} = L_{17} \cdot L_{37}$, $M_{25} = L_6 \cdot L_{39}$, $M_{26} = L_{16} \cdot L_{27}$, $M_{27} = L_{18} \cdot L_{38}$, $M_{28} = L_{14} \cdot L_{25}$, $M_{29} = L_{19} \cdot L_{28}$,

`Linear Layer 3:`
$L_{40} = M_{10} + M_{11}$, $L_{41} = L_{28} + L_{40}$, $L_{42} = L_{27} + M_9$, $L_{43} = L_{42} + M_{10}$, $L_{44} = L_{41} + L_{43}$, $L_{45} = M_{19} + M_{20}$, $L_{46} = M_{17} + M_{20}$, $L_{47} = L_{46} + M_{18}$, $L_{48} = L_{47} + M_{14}$, $L_{49} = L_{45} + M_{15}$, $L_{50} = L_{47} + L_{49}$, $L_{51} = L_{48} + L_{50}$, $L_{52} = L_{46} + M_{16}$, $L_{53} = L_{52} + M_{12}$, $L_{54} = L_{45} + M_{13}$, $L_{55} = L_{52} + L_{54}$, $L_{56} = L_{53} + L_{55}$, $L_{57} = M_{28} + M_{29}$, $L_{58} = M_{26} + M_{29}$, $L_{59} = L_{58} + M_{27}$, $L_{60} = L_{59} + M_{23}$, $L_{61} = L_{57} + M_{24}$, $L_{62} = L_{59} + L_{61}$, $L_{63} = L_{60} + L_{62}$, $L_{64} = L_{58} + M_{25}$, $L_{65} = L_{64} + M_{21}$, $L_{66} = L_{57} + M_{22}$, $L_{67} = L_{64} + L_{66}$, $L_{68} = L_{65} + L_{67}$,

`AND Layer 3:`
$M_{30} = L_{41} \cdot L_{53}$, $M_{31} = L_{44} \cdot L_{55}$, $M_{32} = L_{43} \cdot L_{56}$, $M_{33} = L_{41} \cdot L_{48}$, $M_{34} = L_{44} \cdot L_{50}$, $M_{35} = L_{43} \cdot L_{51}$, $M_{36} = L_{41} \cdot L_{65}$, $M_{37} = L_{44} \cdot L_{67}$, $M_{38} = L_{43} \cdot L_{68}$, $M_{39} = L_{41} \cdot L_{60}$, $M_{40} = L_{44} \cdot L_{62}$, $M_{41} = L_{43} \cdot L_{63}$,

`Affine Layer:`

$L_{69} = M_{30} + M_{32}$, $L_{70} = L_{69} + M_{39}$, $L_{71} = M_{37} + M_{38}$, $L_{72} = L_{71} + M_{35}$, $L_{73} = M_{36} + M_{38}$, $L_{74} = M_{40} + M_{41}$, $L_{75} = L_{72} + L_{74}$, $Y_0 = L_{70} + M_{41}$, $Y_1 = L_{69} + L_{73}$, $Y_2 = L_{72} + M_{34}$, $Y_3 = L_{73} + Y_0$, $Y_7 = L_{75} + M_{33}$, $L_{76} = L_{71} + Y_3$, $L_{77} = L_{76} + Y_7$, $L_{78} = M_{32} + Y_2$, $L_{79} = L_{78} + Y_0$, $Y_4 = L_{74} + L_{76}$, $Y_5 = L_{79} + M_{31}$, $Y_6 = L_{77} + Y_1$, $Y_1 = Y_1 + 1$, $Y_2 = Y_2 + 1$, $Y_6 = Y_6 + 1$, $Y_7 = Y_7 + 1$

## H The Costs of the Grover Oracle and Encryption Oracle for AES

We present the costs of the Grover Oracle and Encryption Oracle for AES based on our new $T$-depth-3 AES S-box circuit. In these tables, as in [54], the widths and $T$-depths are from manual estimation, and the values of other metrics are from the resource estimator of ProjectQ (The ProjectQ codes are available at `https://github.com/hzy-cas/Minimal_T-depth_Width`). The full depths presented here are slightly different from those presented in [54]. The reason is that in their ProjectQ code for QAND$^\dagger$, the conditioned CNOT gate can be applied before the measurement, hence will induce a wrong full depth. We modified this by replacing the measurement with a Toffoli gate during resource estimation.

**Table 8.** The Costs of Grover Oracles based on the Pipeline Structure.

| #CNOT | #1qClifford | #$T$ | #M | $T$-depth | Full Depth | Width | Source |
|---|---|---|---|---|---|---|---|
| 456040 | 179200 | 105600 | 26400 | 60 | 1802 | 3796 | [54] |
| **353160** | **119200** | **67200** | **16800** | 60 | **1782** | **3156** | This work |

**Table 9.** The Costs of Encryption Oracles based on the Interlacing-Uncompute Structure.

| #CNOT | #1qClifford | #$T$ | #M | $T$-depth | Full Depth | Width | Source |
|---|---|---|---|---|---|---|---|
| 364360 | 144584 | 84480 | 21120 | 33 | 1078 | 4128 | [54] |
| **281896** | **96584** | **53760** | **13440** | 33 | **1066** | **3104** | This work |

## I Implementing $C^7X$ and $C^5X$ according to Lemma 7 and Lemma 8

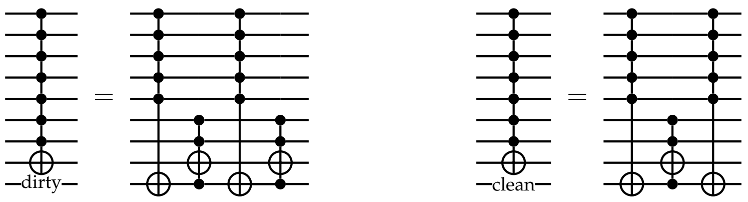


**Fig. 4.** Implementing a $C^7X$ gate with one ancilla qubit.

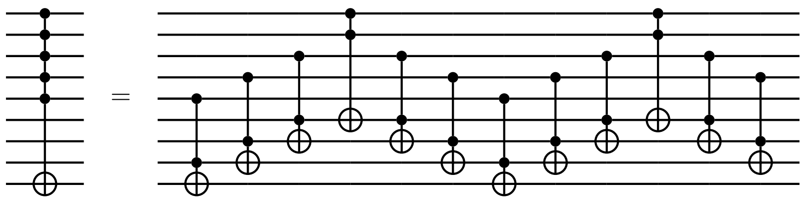


**Fig. 5.** Implementing a $C^5X$ gate with three dirty ancilla qubits.