

# Rise of conditionally clean ancillae for optimizing quantum circuits

Tanuj Khattar and Craig Gidney

Google Quantum AI, Santa Barbara, California 93117, USA  
July 25, 2024

We argue by example that conditionally clean ancillae, recently described by [NZS24], should become a standard tool in the quantum circuit design kit. We use conditionally clean ancillae to reduce the gate counts and depths of several circuit constructions. In particular, we present:

- (a)  $n$ -controlled NOT using  $2n$  Toffolis and  $\mathcal{O}(\log n)$  depth given 2 clean ancillae.
- (b)  $n$ -qubit incrementer using  $3n$  Toffolis given  $\log_2^* n$  clean ancillae
- (c)  $n$ -qubit quantum-classical comparator using  $3n$  Toffolis given  $\log_2^* n$  clean ancillae.
- (d) unary iteration over  $[0, N)$  using  $2.5N$  Toffolis given 2 clean ancillae.
- (e) unary iteration via skew tree over  $[0, N)$  using  $1.25N$  Toffolis given  $n$  dirty ancillae.

We also describe a technique for laddered toggle detection to replace clean ancillae with dirty ancillae in all our constructions with a 2x Toffoli overhead. Our constructions achieve the lowest gate counts to date with sublinear ancilla requirements and should be useful building blocks to optimize circuits in the low-qubit regime of Early Fault Tolerance.

**Data availability:** The circuit constructions presented in this paper are available at <https://doi.org/10.5281/zenodo.12819218>

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Background &amp; History</b>	<b>2</b>
<b>3</b>	<b>Conditionally clean ancilla</b>	<b>3</b>
<b>4</b>	<b>Ladderred toggle detection when borrowing multiple dirty qubits</b>	<b>4</b>
<b>5</b>	<b>Application of conditionally clean ancilla to <math>n</math>-bit Toffoli circuits</b>	<b>6</b>
5.1	$n$ -bit Toffoli into $2n - 3$ Toffoli and $\mathcal{O}(n)$ depth using 1 clean ancilla . . . . .	8
5.2	$n$ -bit Toffoli into $2n - 3$ Toffoli and $\mathcal{O}(\log n)$ depth using 2 clean ancilla . . . . .	8
5.3	$n$ -bit Toffoli into $4n - 8$ Toffoli and $\mathcal{O}(n)$ depth using 1 dirty ancilla . . . . .	11
5.4	$n$ -bit Toffoli into $4n - 8$ Toffoli and $\mathcal{O}(\log n)$ depth using 2 dirty ancilla . . . . .	11
<b>6</b>	<b>Producing/Consuming all <math>n</math>-bit prefix/suffix ANDs using <math>3n</math> Toffoli and <math>\log_2^* n</math> clean ancilla</b>	<b>11</b>
6.1	$n$ -bit Incrementer into $3n$ Toffoli and $\mathcal{O}(n)$ depth using $\log_2^* n$ clean ancilla . . . . .	13
6.2	$n$ -bit LessThanConst into $3n$ Toffoli and $\mathcal{O}(n)$ depth using $\log_2^* n$ clean ancilla . . . . .	14
<b>7</b>	<b>Constructions for Unary Iteration and QROM</b>	<b>14</b>

Tanuj Khattar: Corresponding author: [tanujkhattar4@gmail.com](mailto:tanujkhattar4@gmail.com)  
Craig Gidney: Corresponding author: [craig.gidney@gmail.com](mailto:craig.gidney@gmail.com)

7.1	Unary Iteration as a tree traversal . . . . .	15
7.2	Unary iteration and QROM using conditionally clean ancillae . . . . .	15
7.3	Unary iteration and QROM using dirty ancilla . . . . .	16
<b>8</b>	<b>Conclusion</b>	<b>16</b>
<b>9</b>	<b>Contributions</b>	<b>16</b>
<b>10</b>	<b>Acknowledgements</b>	<b>16</b>

## 1 Introduction

Clean (or dirty) ancilla qubits are often used [Gid15a; Gid15b; Gid18] as temporary workspace when decomposing an arbitrary  $n$ -bit unitary into constant-sized gates like the Toffoli gate. When evaluating the efficiency of a decomposition, there are three main factors that one needs to compare: (a) number of additional ancilla qubits used as workspace (b) gate counts and (c) depth. When optimizing circuits for the fault tolerant regime using a surface code architecture, the target gateset is Clifford + T / Toffoli and cost of the decomposition is dominated by the number of T / Toffoli gates, which are significantly more expensive to execute as compared to Clifford gates [Lit19; Fow+12].

Several works have focused on coming up with efficient circuit constructions, with low T / Toffoli counts and reduced ancilla usage, for a wide variety of applications like quantum chemistry [Rub+23; Kim+22; Lee+21], quantum dynamics [Rub+24; Agr+24], combinatorial optimization [San+20], and quantum arithmetic circuits such as for Shor’s algorithm [Gid17; Lit23]. There are often tradeoffs where one can reduce the number of T / Toffoli gates in the decomposition by using a greater number of ancilla qubits as temporary workspace [Gid15a; Gid15b; Gid18; ZSL24].

We say an ancilla is “clean” when it is initialized into a known state (typically  $|0\rangle$ ) and can be discarded after use. By contrast, a dirty ancilla qubit is available for use but has an unknown state that can be temporarily perturbed but must be restored eventually. For example, a dirty ancilla qubit could be a qubit from somewhere else in the computation that happens to be currently idle. Clean ancillae are often more useful than dirty ancillae since they avoid the need for techniques like toggle detection, which generally adds a 2x gate count overhead [Gid15a; NZS24], and can be uncomputed using measurement based uncomputation [Jon13; Gid18].

In this work, we describe a trick where certain system qubits are in a conditionally known state. Their state is unknown, dirty, unless it is conditioned on a subset of other system qubits. Thus they are called “conditionally clean ancilla” qubits. For example, suppose that the control qubits of an AND gate are in an unknown state. In the subset of the superposition where the output qubit of the AND gate is ON, the control qubits must also be ON. The control qubits are clean when conditioned on the output; they are conditionally clean. Although conditionally clean ancilla qubits are technically dirty, they can be used in many respects as if they were clean. For example, although you cannot uncompute a conditionally clean ancilla using measurement based uncomputation, you can often avoid costs like repeating a circuit twice to perform toggle detection.

We use conditionally clean qubits to come up with new optimized circuit constructions for a variety of fault tolerant quantum primitives in the low ancilla usage regime: the  $n$ -control NOT, the  $n$ -qubit Incrementer, quantum-classical comparator circuits, and others (see Table 1). Our constructions improve upon the previously best known results, in terms of T / Toffoli counts and circuit depths, in the regime of sublinear ancilla usage. We believe these examples demonstrate that conditionally clean qubits are an obviously useful tool for optimizing quantum circuit constructions, and that there will be many other uses beyond the specific constructions that we provide.

## 2 Background & History

There have been several recent papers independently discovering or using conditionally clean qubits. It seems to be an example of an idea whose time is “due”.

Gate			Toffoli		T	
Type	Ancilla	Source	Cost	Depth	Cost	Depth
MCX <sub>n</sub>	One clean	[Gid15a]	6n	$\mathcal{O}(n)$	24n	$\mathcal{O}(n)$
	One dirty	[Gid15a]	8n	$\mathcal{O}(n)$	32n	$\mathcal{O}(n)$
	One dirty	[ZB24]	4n - 8	$\mathcal{O}(n)$	16n - 32	8n - 6
	One clean	[Cla+24]	$\mathcal{O}(n \log^4 n)$	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(n \log^4 n)$	$\mathcal{O}(\log^3 n)$
	One clean	[Nzs24]	$\mathcal{O}(n) (\approx 3n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
	One clean	Ours - Section 5.1	2n - 3	$\mathcal{O}(n)$	8n - 12	$\mathcal{O}(n)$
	Two clean	Ours - Section 5.2	2n - 3	$\mathcal{O}(\log n)$	8n - 12	$\mathcal{O}(\log n)$
	One dirty	Ours - Section 5.3	4n - 8	$\mathcal{O}(n)$	16n - 32	$\mathcal{O}(n)$
Two dirty	Ours - Section 5.4	4n - 8	$\mathcal{O}(\log n)$	16n - 32	$\mathcal{O}(\log n)$	
Incrementer <sub>n</sub>	One clean	[Gid15b]	32n	$\mathcal{O}(n)$	128n	$\mathcal{O}(n)$
	One clean	[Nzs24]	$\mathcal{O}(n) (\approx 64n)$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n) (\approx 256n)$	$\mathcal{O}(\log^2 n)$
	$\log_2^* n$ clean	Ours - Section 6.1	3n	$\mathcal{O}(n)$	12n	$\mathcal{O}(n)$
$U_{t \oplus (x < c)}$ (LessThanC <sub>n</sub> )	Two clean	[Gid17]	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
	One clean	[Yua+23]	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
	$\log_2^* n$ clean	Ours - Section 6.2	3n	$\mathcal{O}(n)$	12n	$\mathcal{O}(n)$
UnaryIteration (balanced tree)	$n = \log_2 N$ clean	[Bab+18]	N - 1	$\mathcal{O}(N)$	4N - 4	$\mathcal{O}(N)$
	$\log_2^* n$ clean	Ours - Section 7.2	2.5N - 1	$\mathcal{O}(N)$	10N - 4	$\mathcal{O}(N)$
	(skew tree) $\log_2^* n$ clean	Ours - Section 7.2	2.25N - 1	$\mathcal{O}(N)$	9N - 4	$\mathcal{O}(N)$
	(balanced tree) $n = \log_2 N$ dirty	Ours - Section 7.3	$1.5N + \mathcal{O}(n\sqrt{N})$	$\mathcal{O}(N)$	$6N + \mathcal{O}(n\sqrt{N})$	$\mathcal{O}(N)$
	(skew tree) $n = \log_2 N$ dirty	Ours - Section 7.3	$1.25N + \mathcal{O}(n\sqrt{N})$	$\mathcal{O}(N)$	$5N + \mathcal{O}(n\sqrt{N})$	$\mathcal{O}(N)$

Table 1: Comparison of prior work (shaded) to our constructions (not shaded). Note that  $\log_2^* n \leq 5$  for all practical purposes.

The papers we found that described or used conditionally clean qubits are:

- [Cla+24]. Figure 1 and Section II A divides the  $n$  controls into  $n/b$  blocks, each of size  $b$  and performs an MCX<sub>b</sub> with a single clean ancilla as the target to generate  $b$  conditionally clean ancilla, which are then used as targets to perform  $b$  different MCX<sub>n/b</sub> gates in parallel. Since their focus is on reducing the depth of the circuit, they obtain a decomposition with depth  $\mathcal{O}(\log^3 n)$  and Toffoli count  $\mathcal{O}(n \log^4 n)$  for one clean ancilla case.
- [Nzs24]. They explain the idea of conditionally clean qubits in Section-3 and note that it may be of interest to the community beyond their specific constructions. Since their focus is on getting constructions with optimal scaling, they give a construction for a MCX<sub>n</sub> in Figure 3 with  $\mathcal{O}(n)$  Toffoli count and  $\mathcal{O}(\log n)$  Toffoli depth but do not perform constant factor analysis. We give an implementation of their construction in the supplementary material for completeness and show that their constant factors for Toffoli count is  $3n$ , whereas our construction has the optimal Toffoli count of  $2n - 3$  while preserving the  $\log n$ -depth.

We also want to mention [CFS24], because our first realization of the concept of a conditionally clean qubit was triggered by trying to understand why Figure 2 of that paper worked.

### 3 Conditionally clean ancilla

A clean ancilla qubit is one which is initialized into a known state (typically  $|0\rangle$ ) and can be discarded after use by leaving it in the same known state once you are done. Clean ancilla qubits are often used as temporary workspace to store intermediate results and decompose multi qubit operations into smaller operations [Gid15a; Nzs24; ZB24].

Consider the operation AND, which allocates and initializes a qubit  $t$  to store the intersection of two input qubits  $a$  and  $b$ . That is to say, the AND gate satisfies

$$\sum_{x,y} \lambda_{x,y} \text{AND}_{a,b,t} \cdot |x\rangle_a |y\rangle_b = \sum_{x,y} \lambda_{x,y} |x\rangle_a |y\rangle_b |xy\rangle_t$$

Notice that, in the state  $|x\rangle_a |y\rangle_b |xy\rangle_t$ ,  $xy$  can only equal 1 if both  $x = 1$  and  $y = 1$ . Thus if  $t$  is storing  $|1\rangle$  then  $a$  and  $b$  must also be storing  $|1\rangle$ . Therefore  $a$  and  $b$  are conditionally clean qubits, with the condition being  $\langle Z_t \rangle = -1$ . Any computation that does nothing when  $\langle Z_t \rangle \neq -1$  can be compiled as if  $a$  and  $b$  happened to be  $|1\rangle$ .

A particularly interesting property of conditionally clean qubits is that they can catalyze their own production. **Figure 1a** shows an example of generating conditionally clean ancilla qubits when doing an AND gate on a clean ancilla as a target. **Figure 1b** shows an example of consuming the conditionally clean qubits  $a$  and  $b$  to store results of intermediate computations and further generate conditionally clean ancilla qubits conditioned over multiple control bits.

This idea of consuming a clean qubit to generate new conditionally clean qubits generalizes beyond the AND gate. For example, if you do an equality check of an  $n$ -bit register with a classical constant and store the result in new clean qubit, then conditioned on the clean qubit being in the  $|1\rangle$  state, you know the state of the input register is equal to the classical constant.

In general, Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  be any function and let  $U_f$  be a unitary that computes this function on a clean ancilla register of size  $m$  such that

$$|x\rangle^n |0\rangle^m \xrightarrow{U_f} |x\rangle^n |f(x)\rangle^m$$

If there exists a pair  $(x, f(x))$  such that  $x$  is the only element in the domain of  $f$  that maps to  $f(x)$ , then conditioned on the ancilla register being in the state  $f(x)$ , we know that the system register would be in the state  $x$ . Thus, we can use the system register as a conditionally clean register that is allocated in a known state  $x$ , use it as a temporary workspace to store intermediate computation results assuming we will consume those results conditioned on the ancilla being in state  $f(x)$ , and then uncompute the intermediate computation to restore the state of the system register to be in  $x$ . The system register in this case acts as a register of “conditionally clean” qubits.

Thus, there are conditions which must be satisfied for us to be able to use system qubits in our computation as conditionally clean qubits:

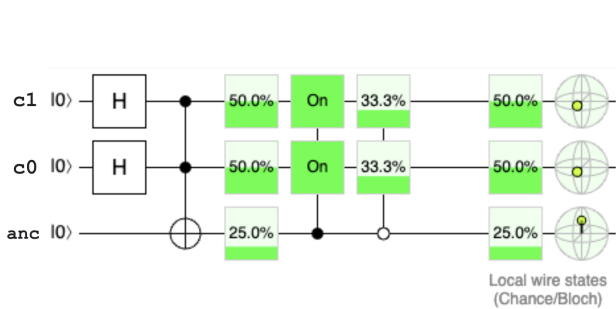
- We must consume (at least one) clean ancilla qubits to compute a function  $f(x)$  that has a unique inverse  $x$  such that conditioned on the ancilla register being in the state  $f(x)$ , we know the system register is in the state  $x$  and thus can be used as conditionally clean qubits to store intermediate results of subsequent computations.
- Any subsequent computation that uses the system register  $x$  as conditionally clean qubits to store intermediate results must consume these results conditioned on the ancilla register being in the state  $f(x)$ ; i.e. Let  $g(y)$  be the result of the computation that uses register  $x$  as a temporary workspace then we must consume  $f(x) \wedge g(y)$  as the final output.

In this paper, will mostly look at utilizing the conditionally clean qubits when we wish to compute a ladder of AND gates. The function  $f$  in this case is an AND operation and the pair  $(x = \{1\}^n, f(x) = 1)$  satisfies the criteria defined above. Note that AND operation also distributes nicely such that  $f(x, y) = f(x) \wedge f(y)$  and thus we can compute  $f(x)$  on a temporary qubit and use the register  $x$  as a register of “conditionally clean” qubits to store the result of computing  $f(y)$ , which is then consumed conditioned on the ancilla being in the state  $f(x)$  such that the final consumed output is  $f(x) \wedge f(y)$ . **Figure 1c** shows how to accumulate the AND of  $n$  ancillae on 2 qubits by incrementally generating and utilizing conditionally clean ancilla qubits.

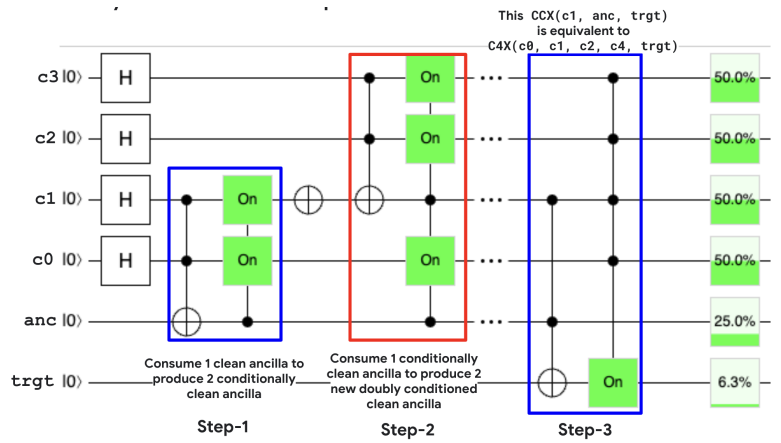
Note that an (obvious yet important) restriction of conditionally clean ancilla qubits is that they cannot be used as temporary workspace for computations that involve the control qubits which the conditionally clean ancilla is conditioned on. For example, in **Figure 1c** step-4 can use only  $c0$  as temporary workspace to accumulate controls  $c1$  and  $c3$  because because the remaining conditionally clean ancilla  $c2c4c5$  are all conditioned over  $c1$  and so cannot be used to as temporary workspace for computations involving  $c1$ .

## 4 Laddered toggle detection when borrowing multiple dirty qubits

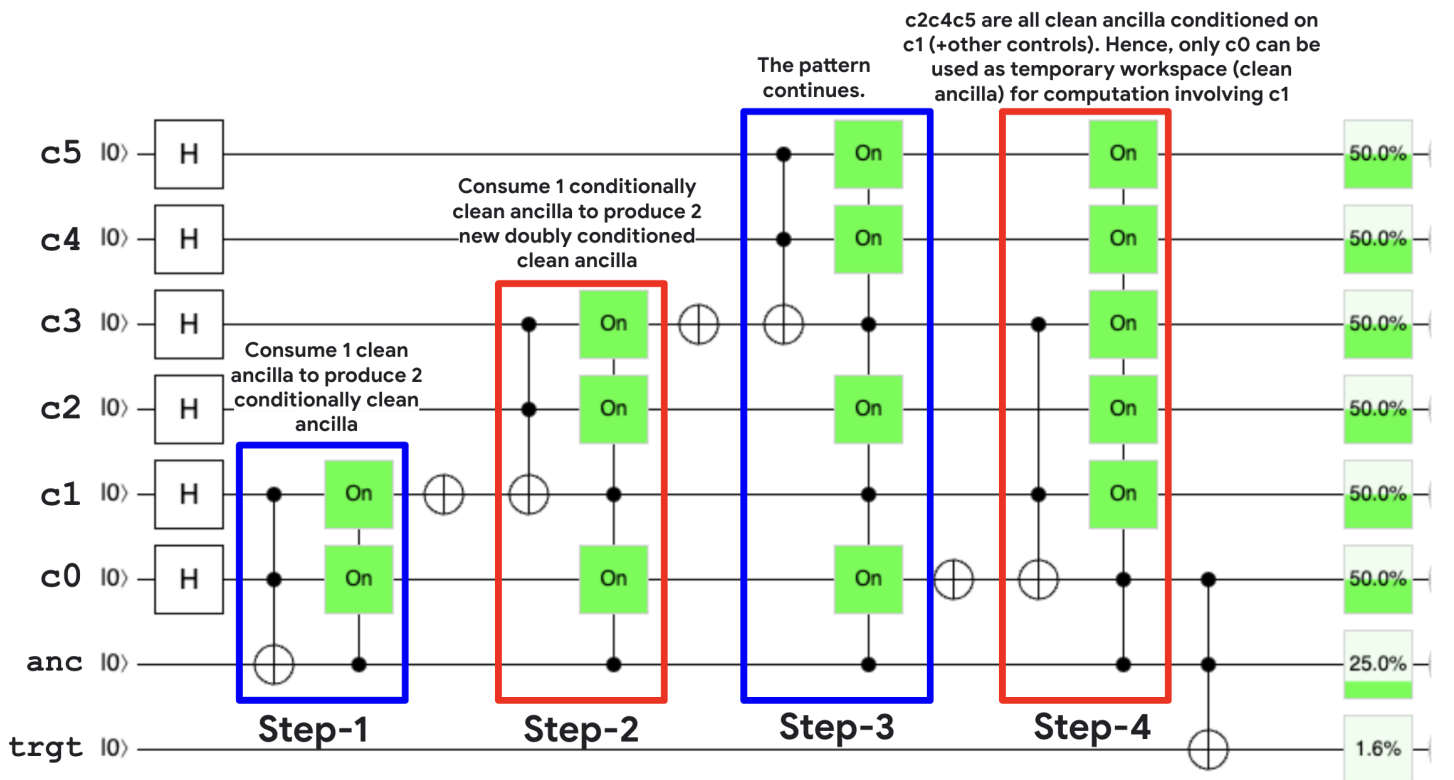
When implementing a controlled version of a self inverse operation  $U$ , one can replace the use of a clean ancilla qubit with a dirty ancilla qubit by repeating the self inverse controlled operation



(a) Application of a Toffoli gate on a clean ancilla qubit as the target generates two conditionally clean qubits. The percentage in the green boxes on each wire represent probability of that qubit being in the  $|1\rangle$  state. Conditioned on the ancilla qubit being in the  $|1\rangle$  state, the first two qubits are “conditionally clean”. These conditionally clean ancillae can be consumed from outside the sub-circuit in which the qubit is conditioned from. See [Figure 1b](#) for example. Here is [quirk link](#) for the circuit presented above.



(b) One of the conditionally clean ancilla from Step-1 is consumed to produce two more doubly conditioned clean ancilla in Step-2. Step-3 shows how  $CCX(c_1, anc, trg)$  is now equivalent to applying  $C^4X(c_0, c_1, c_2, c_3, trg)$  since anc stores  $c_0 \wedge c_1$  and conditioned on anc being True, i.e.  $c_1 = \text{True}$ ,  $c_1$  stores  $c_2 \wedge c_3$



(c) The staggered pattern continues so we accumulate  $n$  controls on  $n/2$  controls using only 1 ancilla by Step-3. Step-4 shows how conditionally clean ancilla can only be consumed in computations that do not involve the control qubits which the ancilla is conditioned on. To accumulate the result of open controls  $c_3 \wedge c_1$ , the only  $c_0$  can be used as a conditionally clean ancilla since  $c_2, c_4$  and  $c_5$  are all conditionally clean conditioned on  $c_1$  and hence cannot be used as temporary workspace for computations involving  $c_1$ . Here is [quirk link](#) for circuit presented above.

Figure 1: Observation and utilization of “conditionally clean ancilla” qubits

twice. If the control is True and dirty ancilla flips, the self inverse  $C - U$  gets applied on both the branches  $|0\rangle$  and  $|1\rangle$  of the dirty ancilla. If the control is False,  $C - U$  gets applied twice on the  $|1\rangle$  branch and zero times on the  $|0\rangle$  branch, effectively applying an identity operation. This well known trick of substituting a dirty ancilla in the place of a clean ancilla is commonly [Gid15a; NZS24] called “toggle detection” and we show how it works in Figure 2a.

If the implementation of  $C - U$  further borrows a clean ancilla qubit which we wish to replace with a dirty ancilla qubit, then a naive strategy would be to recursively repeat the toggle detection strategy described above. Every time we apply toggle detection, we incur a 2x overhead since we need to apply  $C - U$  twice. Thus, if we wish to replace  $n$  clean ancillae with  $n$  dirty ancillae, we can end up incurring an overhead that scales exponentially as  $2^n$ .

If the subsequent borrowed dirty ancillae are different from the control qubits used for toggle detection, the borrowed dirty ancillae can be used as clean ancillae and thus we can avoid incurring the 2x overhead for each subsequent clean ancilla replaced by a dirty ancilla. This construction is explained in Figure 2b.

We call this trick Laddered Toggle Detection and use it to come up with a new construction for decomposing  $n$ -bit Toffoli into  $4n - 8$  Toffoli using  $n - 2$  borrowed qubits in Figure 2c.

We note that all our circuit constructions presented below for the clean ancilla case can be updated to use dirty ancilla with a 2x Toffoli overhead using Laddered Toggle Detection.

## 5 Application of conditionally clean ancilla to $n$ -bit Toffoli circuits

In this section, we will present a number of new circuit constructions for decomposing  $n$ -bit Toffoli’s into Toffoli gates using conditionally clean qubits. We first reduce the problem of constructing circuit decomposition assisted by conditionally clean ancilla into the following abstract computer science problem and show that each strategy for solving the question listed below can lead to a circuit construction for decomposing  $n$ -bit Toffoli gates.

**Question 1.** Given an array  $A$  with  $n + 1$  elements, such that  $A = [1, 0, 0, \dots, 0]$  i.e. initially  $A[0] = 1$  and  $A[i] = 0$  for  $0 < i \leq n$ . In each step, you can perform the following operation:

- Chose indices  $t, x, y$  such that  $t < x < y$  and  $A[t] = 1, A[x] = A[y] = 0$
- Flip the values of  $A[x], A[y], A[t]$ ; i.e. set  $A[t] = 0, A[x] = A[y] = 1$

Our objective is to perform a sequence of operations to minimize the number of unmarked elements (0’s) in the array. We can characterize any valid scheme of performing the operations using the following 3 parameters -

- $K$ : the number of unmarked elements (i.e.  $i$  such that  $A[i] = 0$ ) at the end of the procedure
- $T$  be the number of operations used
- $D$  be the depth of the sequence of chosen operations, where two operations can be performed parallelly if their  $(x, y, t)$  tuples are disjoint

Our goal is to minimize  $K, T$  and  $D$ .

**Theorem 5.1.** Each solution to Question 1 defined above can be mapped to a circuit decomposition for accumulating the AND of  $n$ -qubits into  $K$  qubits using exactly  $T$  Toffoli gates, Toffoli depth of  $D$  and 1 clean ancilla qubit

*Proof.* We can map a sequence of operations satisfying the constraints to a circuit decomposition as follows:

- For  $n$  bit toffoli, each of the  $n$  system qubits can be mapped to indices  $i = 1..n$  in the array  $A$ .
- The index  $i = 0$ , which is initially marked, corresponds to the 1 clean ancilla qubit required by the decomposition.

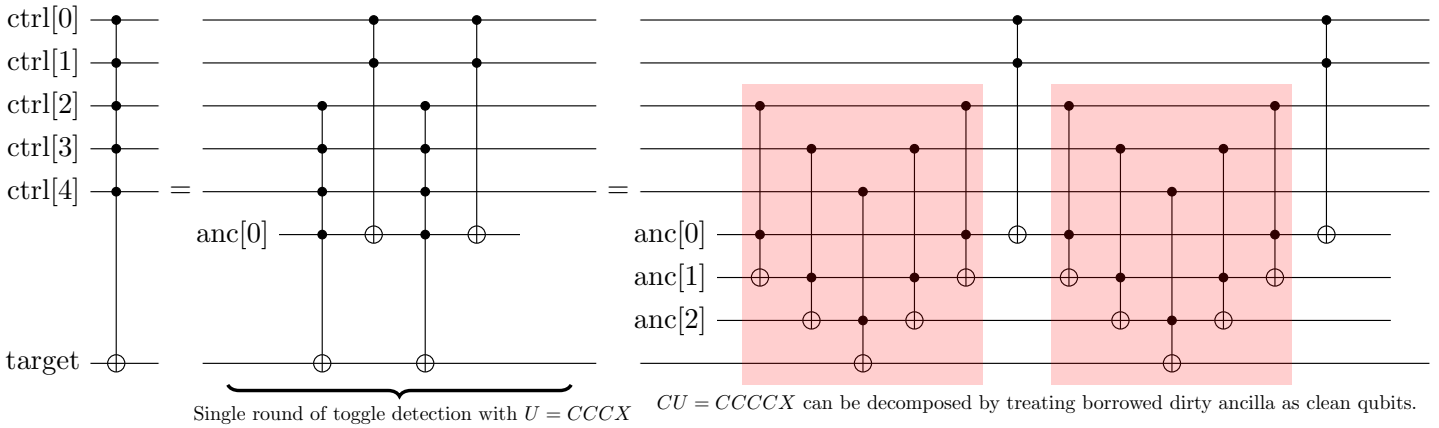
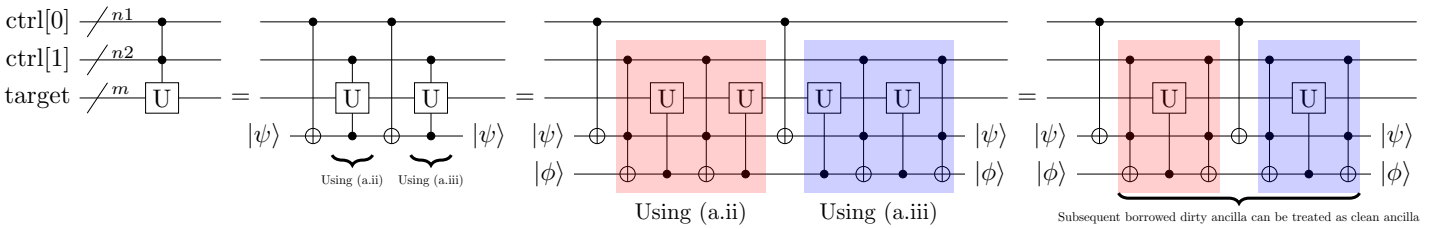
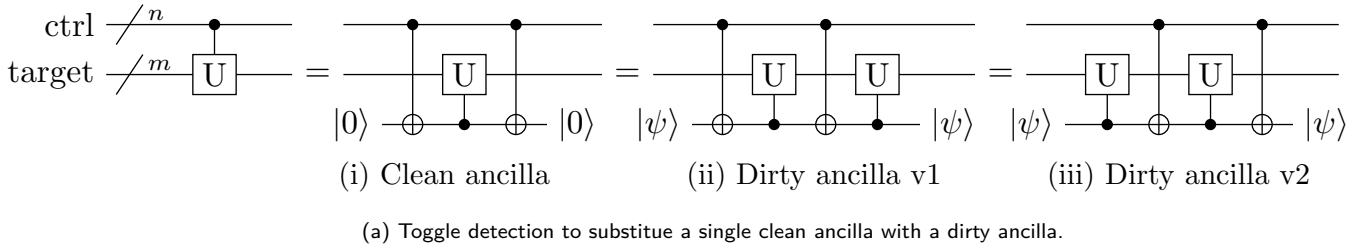


Figure 2: Laddered Toggle detection can be used to substitute multiple clean ancillae with dirty ancillae with a single round of toggle detection.  $U^2 = I$

- During the course of the procedure, each index  $i$  such that  $A[i] = 1$  corresponds to a conditionally clean ancilla qubit which can be consumed as a resource to accumulate the AND of qubits on the right of it.
- Each operation on a tuple  $(x, y, t)$  can be replaced with a gate sequence  $[\text{Toffoli}(x, y, t), X(t)]$  if  $t$  is a conditionally clean qubit (i.e.  $t > 0$ ). If  $t$  is a clean qubit (i.e.  $t = 0$ ), then you can use an  $\text{AND}(x, y, t)$  gate instead. Thus, each operation computes  $x \wedge y$  and saves the result on the conditionally clean qubit  $t$ . Thus, after the operation is performed,  $x, y$  are available as conditionally clean qubits and  $t$  is flipped from a conditionally clean qubit to a system qubit.
- The constraint that for each operation  $t < x < y$  ensures that at any point in time, a conditionally clean qubit  $t$  is clean conditioned on a set of control qubits to the left of it and is used as temporary workspace to store results of computations for qubits on the right of it. This invariant ensures that property for conditional cleanliness is always satisfied.

□

We will now look at specific circuit constructions by first describing solutions to the problem above and then mapping those solutions to circuits via the procedure described above.

### 5.1 $n$ -bit Toffoli into $2n - 3$ Toffoli and $O(n)$ depth using 1 clean ancilla

One solution to Question 1 can be obtained via a greedy strategy where in each step, you pick the rightmost marked index  $t$  such that  $A[t] = 1$  and there are at least two indices  $x$  and  $y$  such that  $t < x < y$  and  $A[x] = A[y] = 0$ . Pick the leftmost such pair of  $x$  and  $y$  and apply the operation on the tuple  $(x, y, t)$ .

This greedy procedure gives us a solution with  $K = 2, T = n - 2, D = n - 2$  and we can map it to a circuit for decomposing  $n$ -qubit Toffoli into  $2n - 3$  Toffoli and  $O(n)$  depth using 1 clean ancilla as shown in Figure 3

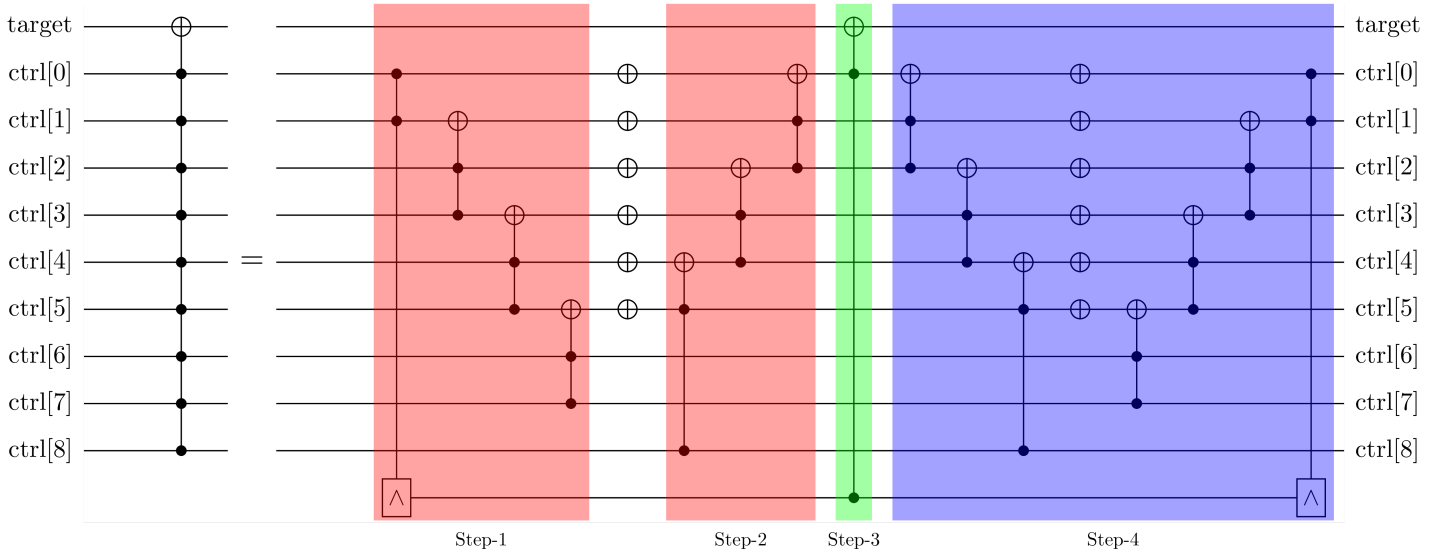
### 5.2 $n$ -bit Toffoli into $2n - 3$ Toffoli and $O(\log n)$ depth using 2 clean ancilla

Another solution to Question 1 that minimizes depth can be described as follows:

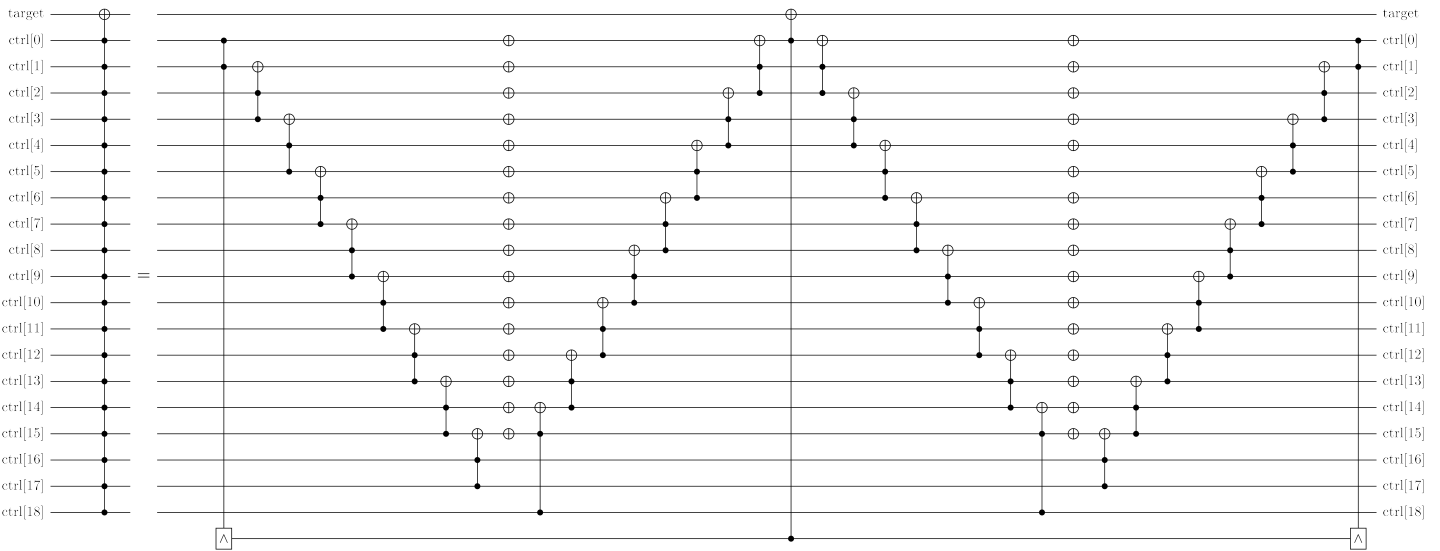
- At the  $i$ 'th timestep, we maintain the invariant that  $A$  has leftmost  $i$  elements in the 0 state, the next  $2^i$  elements in the 1 state and all the remaining elements in the 0 state. Thus, the procedure ends in  $\log n$  such timesteps and  $K = \log n$  unmarked elements remain in the array at the end of the procedure. For example, over the course of the algorithm, the array  $A$  for  $n = 36$  would look like:
  - At  $i = 0$ ,  $A = 10000000000000000000000000000000$
  - At  $i = 1$ ,  $A = 01100000000000000000000000000000$
  - At  $i = 2$ ,  $A = 00111100000000000000000000000000$
  - At  $i = 3$ ,  $A = 00011111110000000000000000000000$
  - At  $i = 4$ ,  $A = 000011111111111111110000000000000000$
  - At  $i = 5$ ,  $A = 000001111111111111111111111111111111$
- In the  $i$ 'th timestep, we utilize the leftmost 1 of the  $2^i$  marked elements as the target and remaining  $2^i - 1$  marked elements as a temporary workspace to flip the next  $2^i + 1$  elements by  $i$  consecutive sequence of operations of the form  $(2k, 2k + 1, k)$  such that they are all parallelizable and can be performed in depth 1. Thus, we end up with  $(2^i - 1) + (2^i + 1) = 2^{i+1}$  marked elements using  $2^i - 1$  operations.

This procedure gives us a solution with  $K = \log n, T = n - \log n, D = \log n$ . In order to map it to a circuit for decomposing  $n$ -qubit Toffoli into  $2n - 3$  Toffoli and  $O(\log n)$  depth, we can recursively invoke the linear depth procedure from Section 5.1 and thus obtain a  $O(\log n)$  depth decomposition using 2 clean ancilla as shown in Figure 4



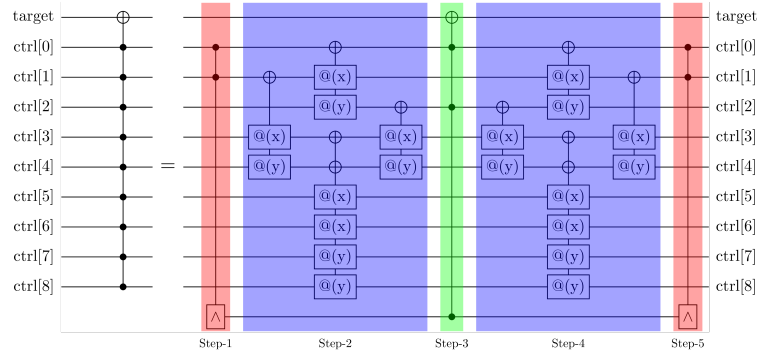
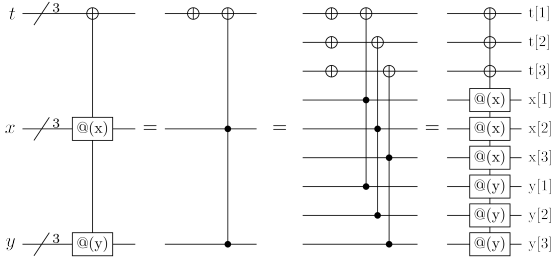


(a) Step-1 is a “up” Toffoli ladder that uses  $\lfloor \frac{n}{2} \rfloor$  Toffoli’s to accumulate the  $n$  control values on  $\lfloor \frac{n}{2} \rfloor$  controls using conditionally clean ancillae. Step-2 then uses a “down” Toffoli ladder with  $n - \lfloor \frac{n}{2} \rfloor - 2$  Toffoli’s to accumulate the AND of all controls on conditionally clean ancilla ctrl0. Step-3 then uses 1 Toffoli, which doesn’t appear in a compute/uncompute pair, to apply the  $n$ -qubit  $C^n X$  gate on the target qubit. Step-4 uncomputes the “up” and “down” Toffoli ladders to return the clean ancilla and intermediate conditional ancilla to their original state. Note that the uncomputation takes 1 less Toffoli since Toffoli on clean ancilla (AND gate) can be uncomputed using measurement based uncomputation and only clifford gates. [Gid18]



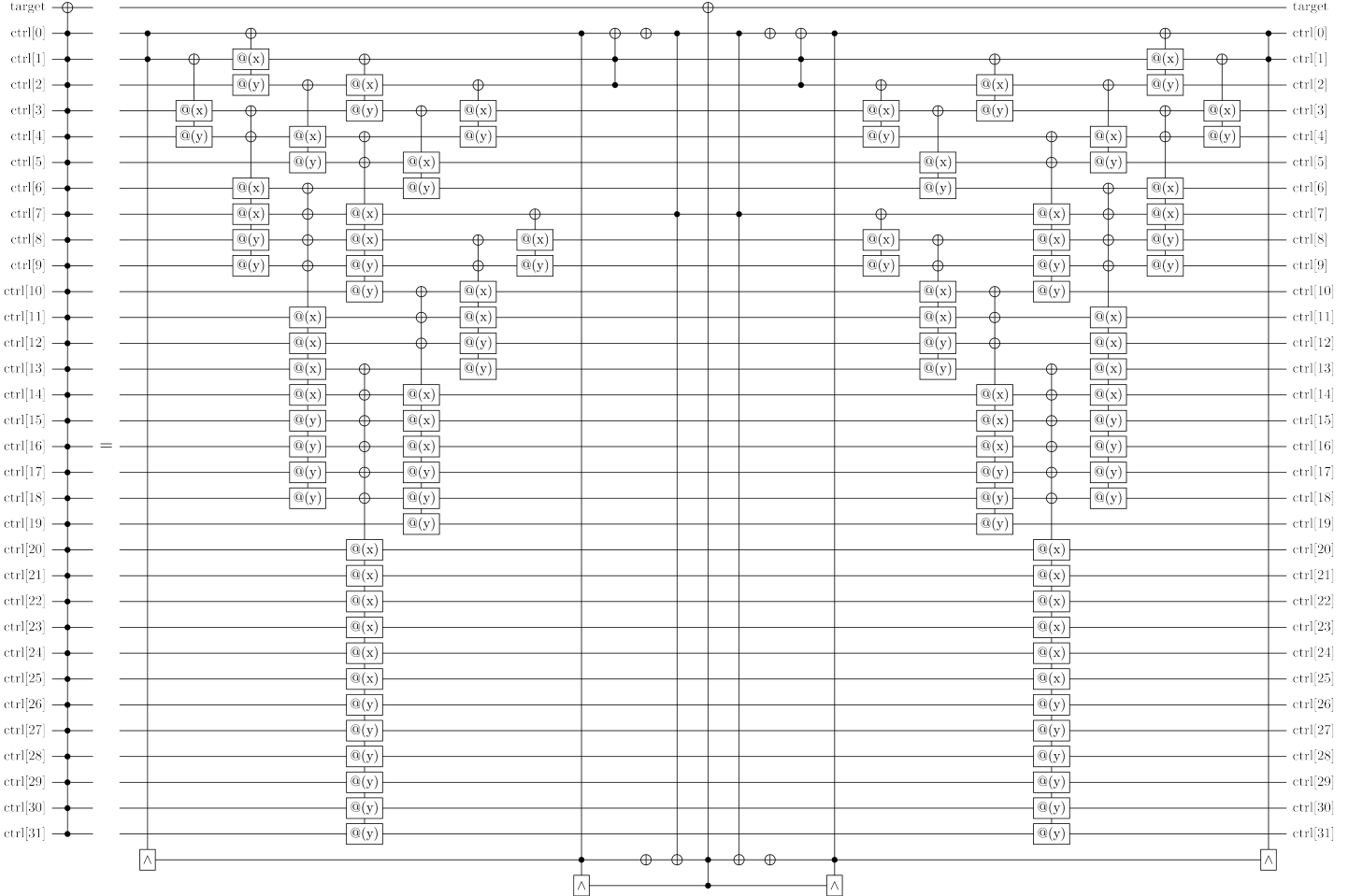
(b) Example circuit for decomposing a 19-bit Toffoli into 33 Toffoli + 1 And/And<sup>†</sup> pair using 1 clean ancilla.

Figure 3: Decomposition of  $n$ -qubit Toffoli into  $2n - 3$  Toffoli using 1 clean ancilla.



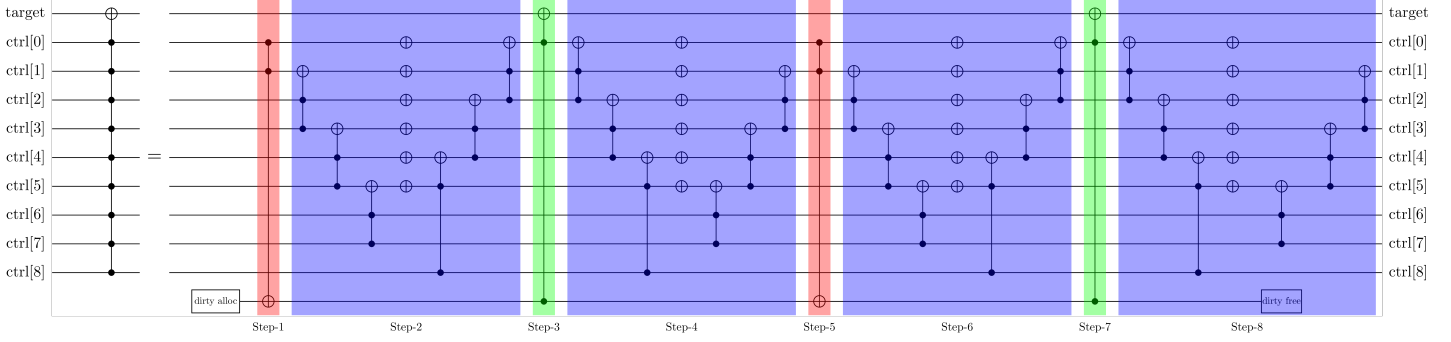
(b) Step-1 uses the one clean ancilla to generate two conditionally clean ancillae. Step-2 uses a log-depth ladder of  $n - \log n - 2$  Toffoli's to accumulate the AND of the remaining  $n - 2$  controls on  $\log n$  qubits. Step-3 uses a  $(\log n + 1)$ -bit Toffoli to accumulate the AND of all  $n$  bits on the target qubit. Step-4 and 5 uncompute Step-2 and 1 respectively. Step-3 can be implemented by recursively invoking a linear depth construction using 1 additional clean ancilla from Section 5.1

(a) A circuit primitive used in our construction which first toggles the  $n$  conditionally clean ancilla qubits and then applies  $n$  Toffoli gates in parallel. The depth of this primitive is 2.

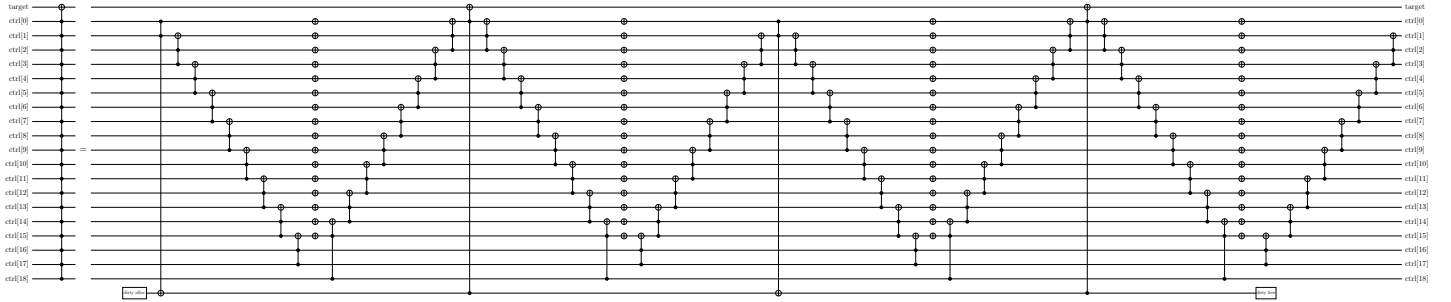


(c) Decomposition of 32-bit Toffoli into 61 Toffolis and  $O(\log_2 n)$  depth using 2 clean ancilla.

Figure 4: Decomposition of  $n$ -qubit Toffoli into  $2n - 3$  Toffoli and  $O(\log n)$  depth using 2 clean ancilla. Note that the 2 Toffoli's acting on clean ancilla qubits are replaced with 2 AND/AND<sup>†</sup> pairs.



(a) Step 1 and 5 are applications of  $CCX$  on dirty ancilla  $anc_0$  as target for toggle detection. The “up” and “down” Toffoli ladders in between at steps 2, 4, 6, 8 are responsible to compute/uncompute the AND of all remaining controls on conditionally clean  $ctrl[0]$ . Step-3 and 7 flip the target when all accumulated control bits are ON. When all controls are ON, the target is flipped once in Step-3 if the  $anc_0$  is initially OFF and once in Step-7 if the  $anc_0$  is initially ON. Each of step 2, 4, 6 and 8 requires  $n - 3$  Toffoli so the construction requires a total of  $4n - 8$  Toffoli.



(b) Example circuit for decomposing a 19-bit Toffoli into 68 Toffoli using 1 dirty ancilla.

Figure 5: Decomposition of  $n$ -qubit Toffoli into  $4n - 8$  Toffoli using and  $O(n)$  depth using 1 dirty ancilla.

### 5.3 $n$ -bit Toffoli into $4n - 8$ Toffoli and $O(n)$ depth using 1 dirty ancilla

The key thing to observe here is that the ancilla qubit used during the clean ancilla decomposition described in Section 5.1 only becomes the target of a single  $CCX$  gate and stores  $c_0 \wedge c_1$  for the first 2 controls. Thus, we can use the toggle detection trick from Section 4 to repeat the Toffoli “up” and “down” ladders twice to get a construction where the ancilla qubit can be a dirty ancilla qubit. This is shown in Figure 5.

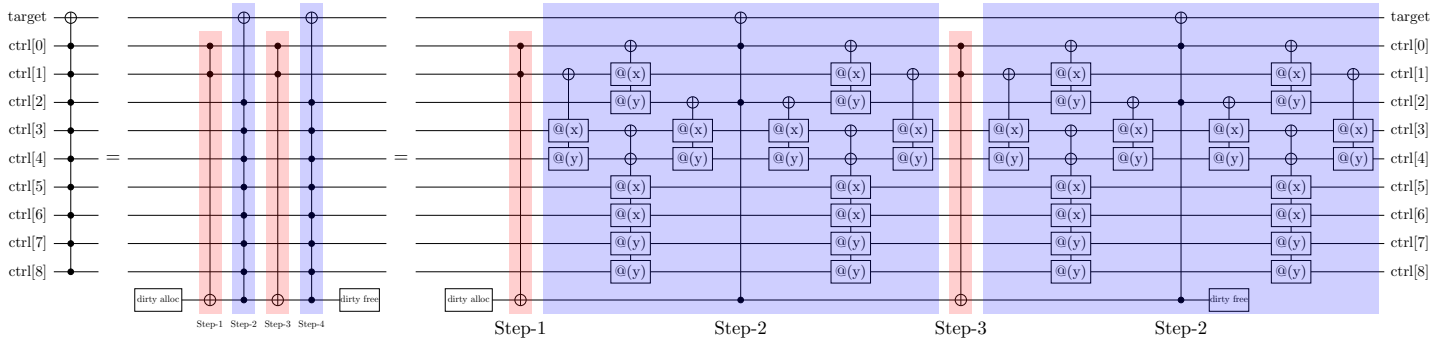
### 5.4 $n$ -bit Toffoli into $4n - 8$ Toffoli and $O(\log n)$ depth using 2 dirty ancilla

For the first round of decomposition from Section 5.2, we use the standard toggle detection trick to replace the clean qubit with a dirty qubit. The recursive decomposition using the linear depth procedure and a second clean ancilla can treat a borrowed dirty ancilla as a clean ancilla, using the laddered toggle detection trick from Section 4, and thus the recursive decomposition remains identical to the clean ancilla case. This is shown in Figure 6

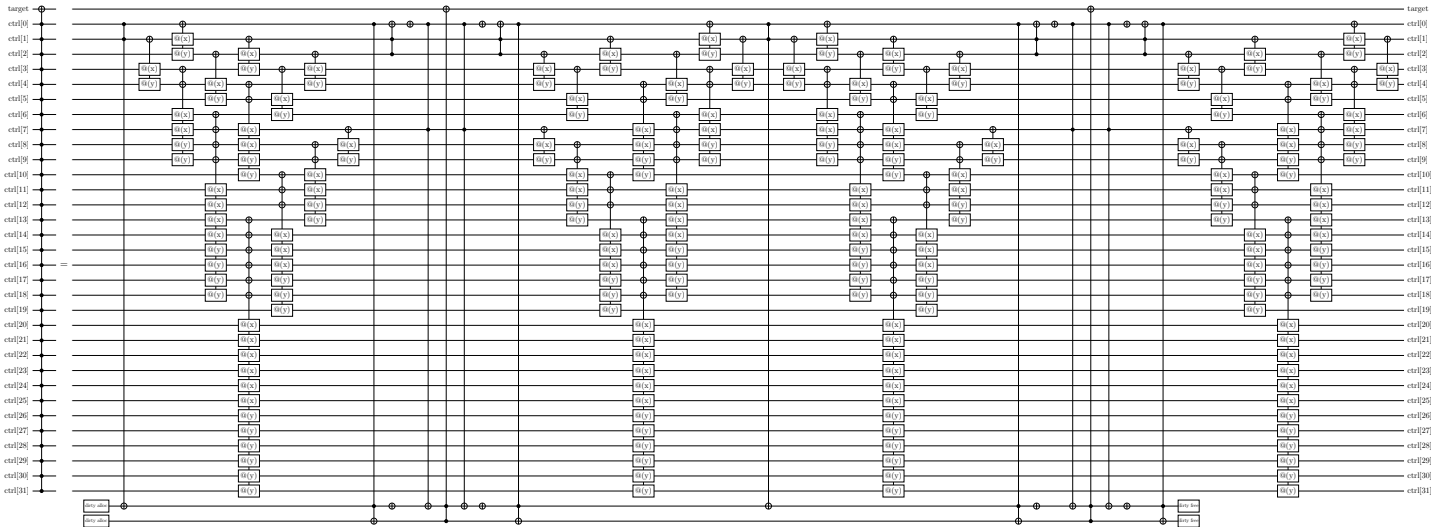
## 6 Producing/Consuming all $n$ -bit prefix/suffix ANDs using $3n$ Toffoli and $\log_2^* n$ clean ancilla

If we had access to  $n$  clean ancilla qubits, one could build a ladder of  $n$  AND/AND $^\dagger$  gates such that the  $i$ 'th ancilla qubit stores the prefix / suffix AND of first / last  $i$  qubits. Then, we can consume each prefix / suffix AND using a CNOT gate controlled on the  $i$ 'th ancilla. This gives us a decomposition which uses  $2n$  Toffoli (or  $n$  pairs of AND/AND $^\dagger$ ) gates to compute / uncompute every prefix / suffix AND and we can consume them using a single CNOT gate.

With the help of conditionally clean ancilla qubits, we will aim to achieve the same thing - i.e. compute and uncompute the prefix / suffix AND of all of the  $n$  qubits on  $n$  different conditionally clean qubits using  $2n$  Toffoli gates and consume each prefix/suffix AND using a Toffoli gate. This



(a) Step-1 and 3 implement toggle detection using the first borrowed ancilla. Steps-2 and 4 use the laddered toggle detection trick from Figure 2 to decompose an  $(n - 1)$ -bit Toffoli into  $2n - 5$  Toffoli using 1 dirty ancilla, by treating the dirty ancilla as clean and using decomposition from Figure 4. The overall Toffoli complexity is  $4n - 8$



(b) Example circuit for decomposing a 19-bit Toffoli into 68 Toffoli using 1 dirty ancilla and  $O(\log n)$  depth.

Figure 6: Decomposition of  $n$ -qubit Toffoli into  $4n - 8$  Toffoli using and  $O(\log n)$  depth using 2 dirty ancilla.

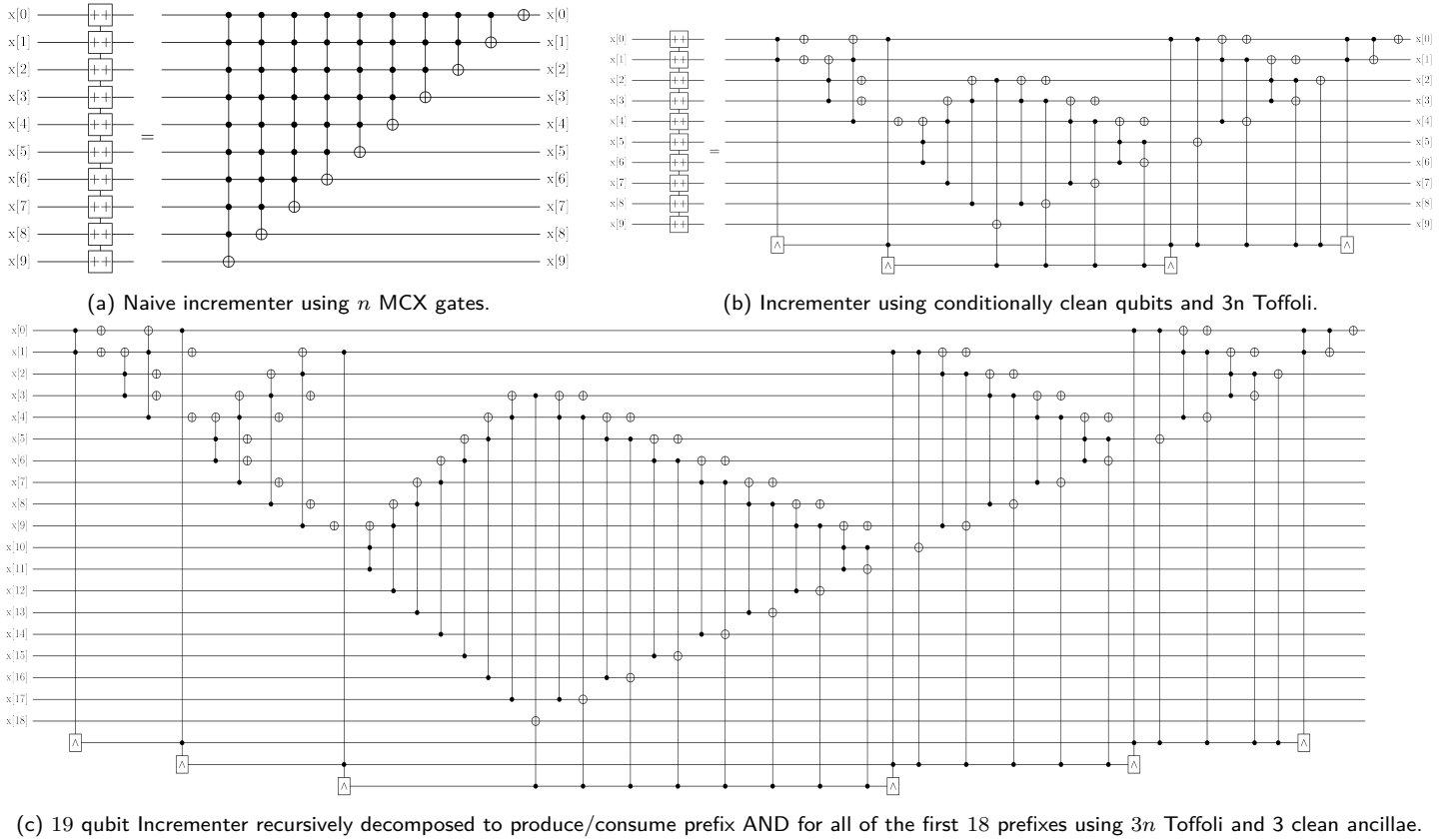


Figure 7: Decomposition of  $n$ -bit Incrementer into  $3n$  Toffoli and  $O(n)$  depth using  $\log_2^* n$  clean ancilla.

gives us an overall Toffoli complexity of  $3n$  instead of  $2n$ , like the clean ancilla case described above, because consuming a prefix AND stored on a conditionally clean ancilla requires a Toffoli gate instead of a CNOT gate.

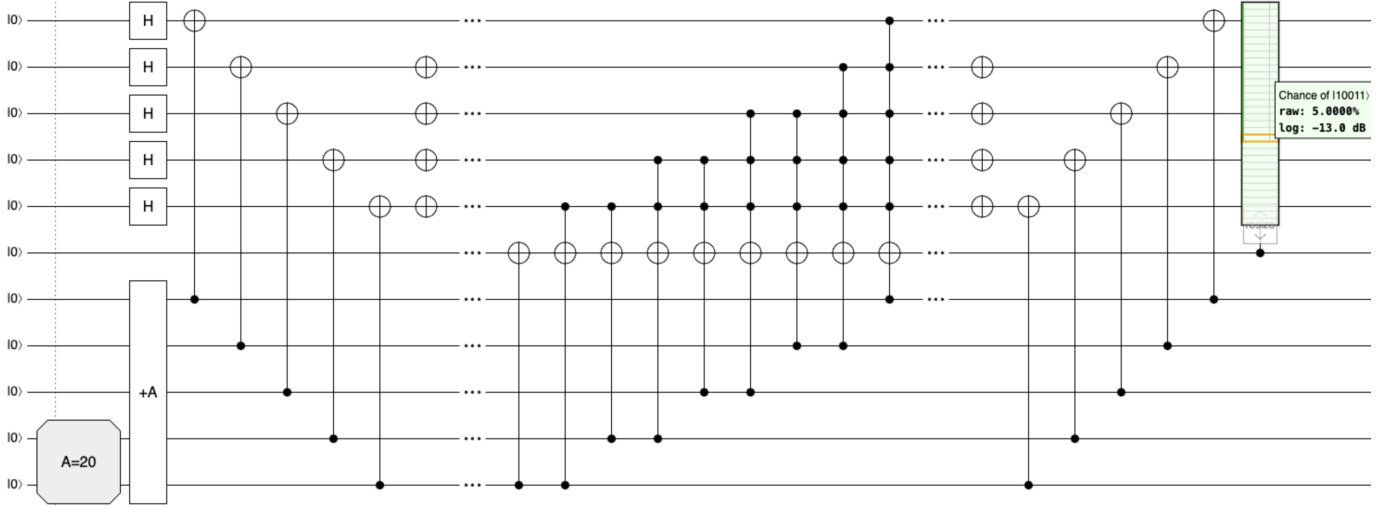
The construction is shown in Figure 7 and follows closely from the the construction described in Section 5.2 but instead of processing the  $i$ 'th batch of size  $2^i$  in log-depth, we now process it in linear depth and compute the prefix AND using  $2^i - 1$  conditionally clean ancilla qubits as temporary workspace.

Also, this time we need to recursively apply the same decomposition that computes the prefix AND over the  $K = \log n$  unmarked items obtained after the first decomposition because we wish to consume the prefix ANDs sequentially and thus we want to build the structure recursively. Figure 7b shows the circuit structure after only the first level of decomposition where consuming the prefix ANDs require you to apply a  $\log n$ -controlled MCX instead of a Toffoli. Figure 7c shows the full recursive decomposition where the  $\log n$ -controlled MCX are recursively decomposed until  $\log_2^* n \leq 1$  using the same strategy so that we can access the prefix AND of every prefix by controlling on at-most 2 qubits using a Toffoli gate.

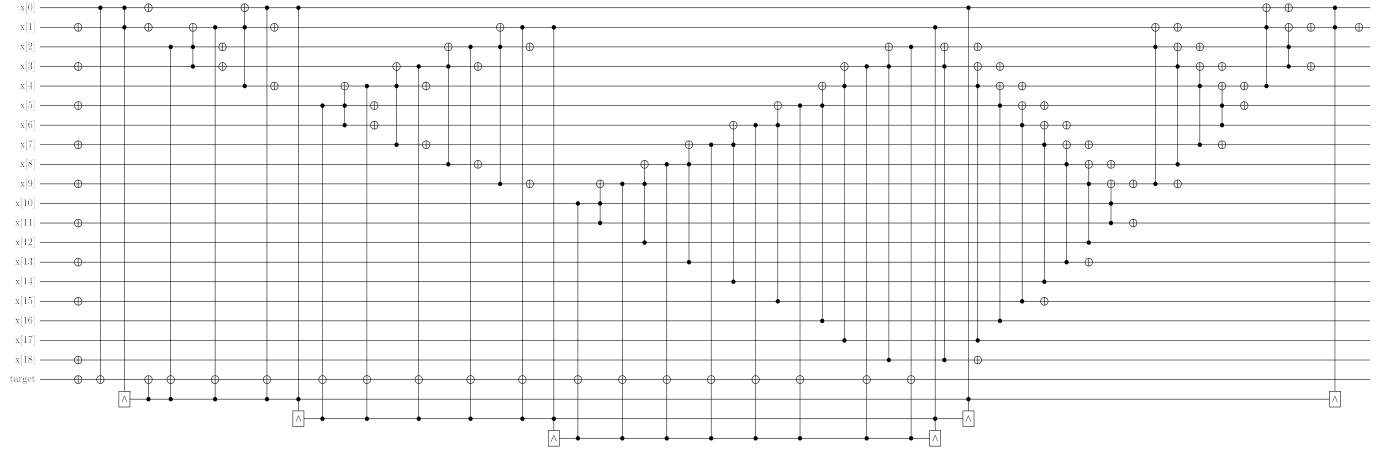
For all practical purposes, the required number of ancilla qubits is 5. Requiring six ancilla qubits would mean you were attempting to increment an integer with more digits than atoms in the observable universe.

## 6.1 $n$ -bit Incrementer into $3n$ Toffoli and $O(n)$ depth using $\log_2^* n$ clean ancilla

Figure 7a shows a naive decomposition of an Incrementer gate into  $n$  different MCX gates, which was described in [Gid15b]. The key thing to observe here is that we need to compute and consume the prefix AND of all of the first  $i$  qubits one by one. We now use the technique described above to decompose this ladder of prefix AND's into  $3n$  Toffoli's using  $\log_2^* n$  clean ancillae. The decomposition is explained in Figure 7



(a) Naive LessThanConst using  $n$  MCX gates. Here is a [quirk link](#) of a circuit which shows this reduction.



(b) 19-bit LessThanConst for constant  $c = 349525$  that has a binary representation 10101010101010101. The quantum-classical comparator uses 3 clean ancillae and 47 Toffoli gates.

Figure 8: Decomposition of  $n$ -bit LessThanConst into  $3n$  Toffoli and  $O(n)$  depth using  $\log_2^* n$  clean ancilla.

## 6.2 $n$ -bit LessThanConst into $3n$ Toffoli and $O(n)$ depth using $\log_2^* n$ clean ancilla

A LessThanConst primitive implements a Quantum-Classical comparison of the form

$$\text{LessThanConst}_c |x\rangle |t\rangle \rightarrow |x\rangle |t \oplus (x < c)\rangle$$

In [Figure 8a](#) we describe a way of reducing the classical-quantum comparison to consuming a ladder of suffix AND's where the number of suffix ANDs to consume depends upon the bits of the classical constant  $c$ . In the worst case, we consume all  $n$  suffix AND's.

After this reduction, we directly apply the technique described above to decompose the ladder of suffix AND's into  $3n$  Toffolis using  $\log_2^* n$  clean qubits. The complete decomposition for a specific constant  $c$  is shown in [Figure 8b](#)

## 7 Constructions for Unary Iteration and QROM

Unary iteration is a technique for applying indexed (aka multiplexed) operations on a target register controlled on the selection register. The technique was introduced in [\[Bab+18\]](#) and since then has seen widespread use in compiling quantum algorithms [\[Lee+21; ZSL24; Rub+24; Yua+23\]](#). The Controlled Unary Iteration construction introduced by [\[Bab+18\]](#) uses  $n$  clean ancilla qubits to

iterate on  $N = 2^n$  indices with a Toffoli cost of  $N - 1$ . In this section, we will describe two new constructions for the low ancilla regime where we either consume a constant number of clean ancillae and use the system qubits as conditionally clean qubits, or we borrow  $n$  dirty qubits from the system.

## 7.1 Unary Iteration as a tree traversal

Unary iteration can be viewed as a tree traversal where each node of the tree corresponds to one or more elements of the range we wish to iterate upon and traversing the tree in a DFS order produces a circuit that corresponds to unary iteration.

In [Figure 9](#) we explain how one can view the standard unary iteration circuit as a tree traversal of a balanced binary tree. If we are given  $n$  clean ancillae, we can use the circuit elements from [Figure 9c](#) to arrive at the construction with a toffoli cost of  $N - 1$ , presented in [\[Bab+18\]](#).

In [Figure 9d](#) and [Figure 9e](#) we show how we can modify the construction to avoid measurement based uncomputation with a Toffoli cost of  $1.5N - 1$ . This construction is identical to the one described in Appendix G.4 of [\[Chi+18\]](#). Avoiding measurement based uncomputation will become important when we describe constructions using conditionally clean ancillae or dirty ancillae - both of which cannot be cleaned up using measurement based uncomputation.

In situations where it's cheap to apply the inverse of operations we are indexing over, we can use a different tree than balanced binary trees. In [Figure 10a](#), we give a recursive definition of a "Skew tree" and in [Figure 10b](#) we show how tree traversal of a Skew tree can be translated into unary iteration. The basic idea of the skewed tree is to replace circuits of the form "if  $\neg C$  then do  $A$ ; if  $C$  then do  $B$ " with circuits of the form "do  $A$ ; if  $C$  then do  $A^{-1} \cdot B$ ". Instead of doing  $A$  conditionally, we do  $A$  unconditionally but undo  $A$  in addition to doing  $B$  when  $C$  is true. This optimization works best when  $A^{-1} \cdot B$  is efficient to apply. This is the case in QROM reads, where  $A$  and  $B$  are both a product of Pauli X gates. The skewed tree has better behavior when  $L$  is not a power of 2. For example, the lopsided tree over  $L$  items has  $\lceil \log_2 L \rceil$  levels, whereas the binary tree has  $\lfloor \log_2 L \rfloor$  levels. This saves an ancilla qubit for domain sizes  $L$  that are not a power of 2. The skewed tree associates outputs with every node, instead of only associating outputs with leaf nodes. This halves the size of the tree, which reduces the size of the circuit. In [Figure 10d](#), we show how one can use a Skew tree to perform unary iteration of  $N$  elements using  $1.25N - 1$  Toffoli gates and still avoid measurement based uncomputation (compared to  $1.5N - 1$  using balanced binary trees).

Here is a [quirk link](#) to a QROM circuit that loads the first 16 natural numbers as data = [1, 2, ..., 16] using a skew tree construction. Many of the controlled reads are 0 because for a skew tree construction, the data to be loaded needs to be modified as follows to account for the undoing the unconditional read done at a previous index which affects the current index.

```
skew_data = [0] * N
for i in range(N):
    for j in range(i, N):
        if i & j == i:
            skew_data[j] ^= data[i]
```

## 7.2 Unary iteration and QROM using conditionally clean ancillae

We can use the tricks developed in [Section 6](#) to consume a constant number of clean qubits and use the generated conditionally clean qubits to produce / consume a prefix AND ladder and combine it with unary iteration constructions given in [Figure 9d](#) and [Figure 10d](#) to get a constant ancilla version of unary iteration using both balanced binary trees and skewed trees. The main overhead with this approach is that consuming a prefix AND stored on a conditionally clean ancilla requires a Toffoli instead of a CNOT. Therefore, we get an  $N$  Toffoli overhead in both the approaches described above.

Therefore, a constant number of clean ancillae and conditionally clean qubits, we can do unary iteration over  $N$  elements using  $2.5N$  Toffoli gates via balanced binary trees and  $2.25$  Toffoli gates via skewed trees.

Here is a [quirk link](#) to a circuit where we iterate on  $N = 16$  elements using 37 Toffolis ( $2.5N$ ) via balanced binary trees and conditionally clean qubits.

### 7.3 Unary iteration and QROM using dirty ancilla

If we had access to  $n$  dirty qubits instead of clean qubits. We can divide the selection register of size  $n$  into a top half of size  $k$  and a bottom half of size  $n - k$ . To iterate on the top half, we execute  $K = 2^k$   $k$ -bit Toffoli gates using constructions described in Section 5. For each of the  $K$  iterations on the top half, we execute a perform a dirty QROM read on the bottom  $n - k$  qubits using  $n - k$  borrowed dirty ancillae. Each of these QROM reads has the same tree shape and is of size  $N/K$ . Since we the ancilla qubits this time are borrowed and can be in an unknown state, we also need to perform a round of laddered toggle detection as described in Section 4. Because all the  $K$  QROM trees are of identical shape with differing data elements, we can do the toggle detection via just 1 more QROM read of the same size but where for each leaf node  $i$ , we load all data elements from the  $i$ 'th leaf nodes of each of the  $K$  QROM trees.

The overall cost of the procedure is  $(K + 1) \times \text{QROMDirtyCost}(N/K) + \mathcal{O}(K * k)$ . Setting  $K = \sqrt{N}$  gives us a cost of  $1.5N + \mathcal{O}(n\sqrt{N})$  for dirty QROM via balanced binary tree and  $1.25N + \mathcal{O}(n\sqrt{N})$  for dirty QROM via the skewed trees case.

Here is a [quirk link](#) to a circuit that shows a QROM read using dirty qubits with a single round of toggle detection where just 1 additional dirty QROM of size  $N/K$  suffices to perform toggle detection for  $K$  dirty QROMs because each of the  $K$  dirty QROMs have identical tree structure.

## 8 Conclusion

In this work, we describe a trick for optimizing circuit compilations by observing and utilizing the presence of “conditionally clean ancilla” qubits and used it to improve upon the previously best known constructions of  $n$ -bit Toffoli and  $n$ -bit Incrementer circuits in the sub-linear ancilla regime. One of the reasons why our constructions are not optimal in terms of T/Toffoli counts when compared to linear clean ancilla case is because uncomputing a clean ancilla qubit can often be done with only measurement + clifford operations and requires no T / Toffoli gates. However, we do not have a way to cheaply uncompute the conditionally clean ancilla qubits we use as part of our constructions. This gives us a 2x overhead in terms of T / Toffoli gate counts. Its an open question to figure out whether one can extend the ideas of measurement based uncomputation to cheaply uncompute the conditionally clean ancilla qubits, similar to the clean ancilla case.

We expect the “conditionally clean ancilla” trick will be useful for circuit optimizations beyond the specific circuit constructions we provide and should be part of a circuit optimization toolkit for compilers and researchers.

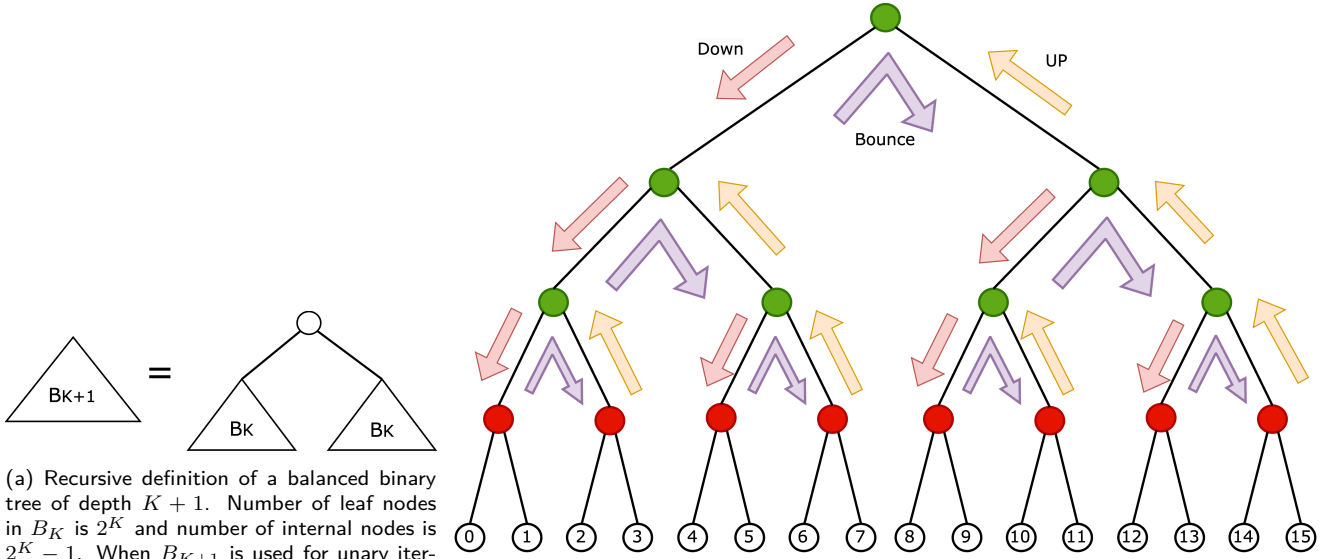
## 9 Contributions

Craig guided the project and came up with some initial constructions. Tanuj improved the constructions, found additional ones, and wrote the paper as well as the accompanying code.

## 10 Acknowledgements

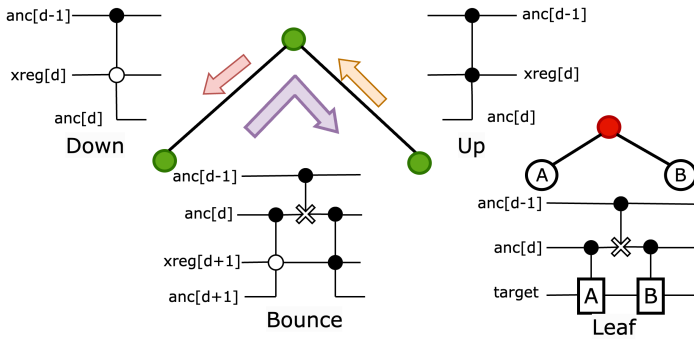
We thank Robin Kothari for helpful discussions on the abstract computer science problem presented in Question 1. We thank Hartmut Neven for creating an environment where this research was possible.



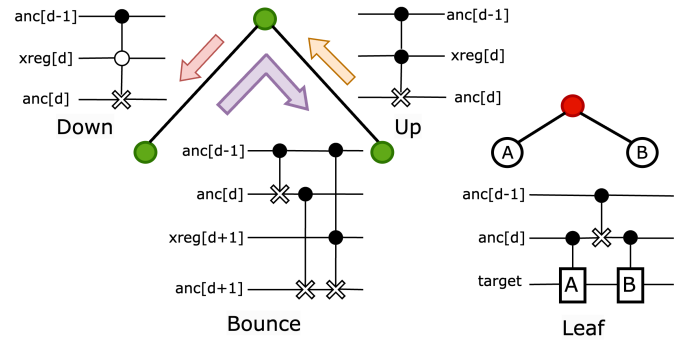


(a) Recursive definition of a balanced binary tree of depth  $K + 1$ . Number of leaf nodes in  $B_K$  is  $2^K$  and number of internal nodes is  $2^K - 1$ . When  $B_{K+1}$  is used for unary iteration, the left subtree corresponds to indices in the range  $[0, 2^K)$  (i.e.  $K + 1$ 'th bit is 0) and the right subtree corresponds to indices in the range  $[2^K, 2^{K+1})$  (i.e.  $K + 1$ 'th bit is 1). Outputs are associated only with the leaf nodes.

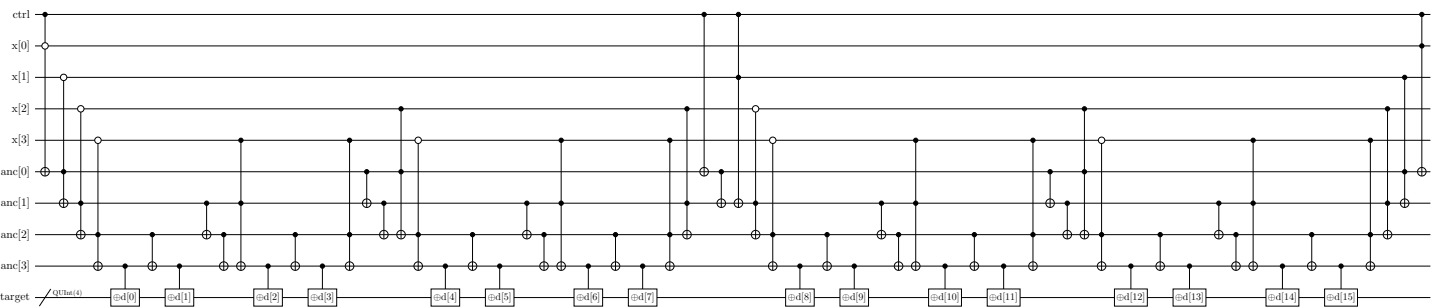
(b) When used for unary iteration on  $N$  elements, a balanced binary tree has  $N/2$  leaf nodes (marked in red) and  $N/2 - 1$  internal nodes (marked in green). A DFS traversal of the tree yields  $N/2 - 1$  DOWN moves,  $N/2 - 1$  BOUNCE moves and  $N/2 - 1$  UP moves. For controlled unary iteration, the number of UP / DOWN moves is  $N/2$  (an edge comes in to the root node). Each move corresponds to a circuit element as shown in Figure 9c and Figure 9d



(c) Circuit elements produced during tree traversal of a balanced binary tree with clean ancilla. Every DOWN and BOUNCE traversal as a Toffoli cost of 1. UP traversal has Toffoli cost of 0 since the AND gate can be uncomputed using measurement based uncomputation [Gid18]. LEAF traversal to consume data has a Toffoli cost of 0. Thus, Controlled Unary iteration over  $N = 2^n$  elements using  $n$  clean ancilla has a Toffoli cost of  $N - 1$ . This yields the unary iteration construction from [Bab+18]

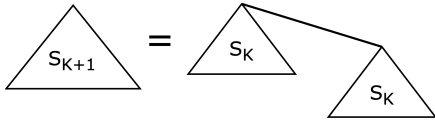


(d) Circuit elements produced during tree traversal of a balanced binary tree with (potentially dirty) ancilla where measurement based uncomputation is not allowed. Every DOWN, BOUNCE and UP traversal now has a Toffoli cost of 1. LEAF traversal to consume data has a Toffoli cost of 0. Controlled Unary iteration over  $N = 2^n$  elements using  $n$  ancilla (without measurement based uncomputation) has a Toffoli cost of  $1.5N - 1$ . This yields the unary iteration construction from [Chi+18]

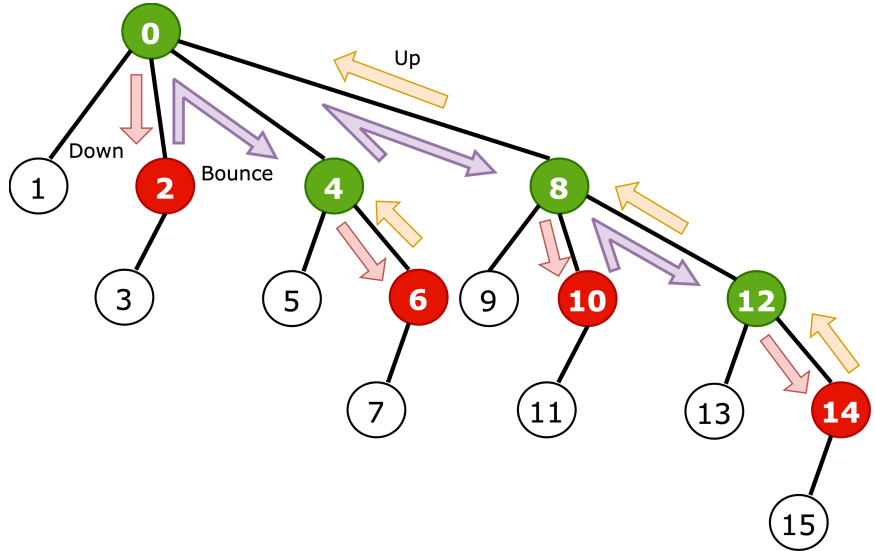


(e) QROM circuit to load  $N = 16$  data elements using 4 ancilla qubits (without measurement based uncomputation) and  $1.5N - 3 = 23$  Toffoli gates. Uses circuit elements described in Figure 9d and forms the basis of (i) unary iteration using only 2 clean ancilla and  $2.5N$  Toffoli AND (ii) unary iteration using  $n = \log_2 N$  dirty ancilla and  $1.5N + \mathcal{O}(n\sqrt{N})$  Toffoli

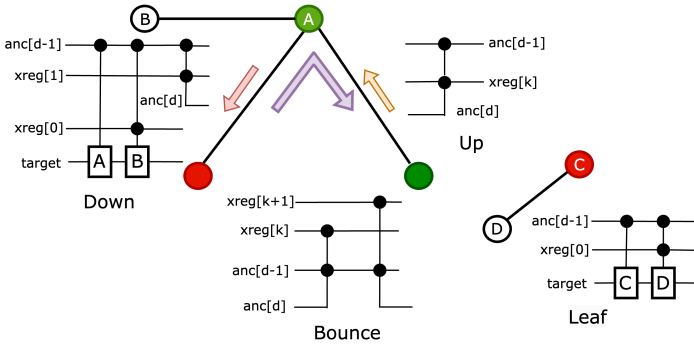
Figure 9: Unary iteration using balanced binary trees



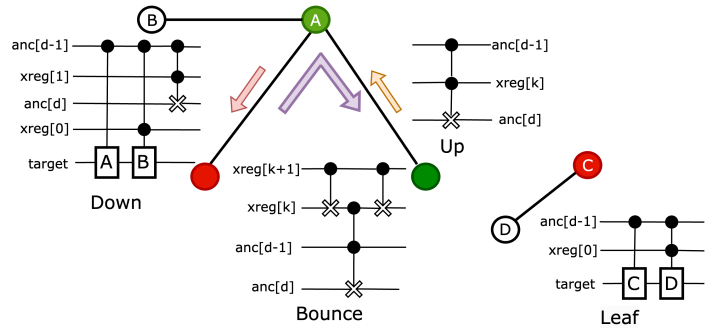
(a) Recursive definition of a Skew tree of depth  $K + 1$ . The size of  $S_k$  is  $2^k$ , thus when doing unary iteration a skew tree associates an output with every node. The basic idea of using skewed trees for unary iteration is to replace circuits of the form “if  $\neg C$  then do  $A$ ; if  $C$  then do  $B$ ” with circuits of the form “do  $A$ ; if  $C$  then do  $A^{-1} \cdot B$ ”. Instead of doing  $A$  conditionally, we do  $A$  unconditionally but undo  $A$  in addition to doing  $B$  when  $C$  is true. This optimization works best when  $A^{-1} \cdot B$  is efficient to apply, which is the case for QROM



(b) When used for unary iteration on  $N$  elements, a skew tree has  $N/4$  leaf nodes (marked in red) and  $N/4$  internal nodes (marked in green). A DFS traversal of the tree yields  $N/4$  DOWN moves,  $N/4 - 1$  BOUNCE moves and  $N/4$  UP moves. Each move corresponds to a circuit element as shown in Figure 10c and Figure 10d.



(c) Circuit elements produced during tree traversal of a skew tree with clean ancilla. DOWN traversal has a Toffoli cost of 2, BOUNCE traversal has a Toffoli cost of 1, UP traversal has a Toffoli cost of 0 since the AND gate can be uncomputed using measurement based uncomputation [Gid18]. LEAF traversal to consume data has a Toffoli cost of 1. Thus, Controlled Unary iteration over  $N = 2^n$  elements using  $n$  clean ancilla has a Toffoli cost of  $N - 1$ .



(d) Circuit elements produced during tree traversal of a skew tree with (potentially dirty) ancilla where measurement based uncomputation is not allowed. DOWN traversal has a Toffoli cost of 2. BOUNCE, UP and LEAF traversals have a Toffoli cost of 1. Thus, Controlled Unary iteration over  $N = 2^n$  elements using  $n$  ancilla (without measurement based uncomputation) using Skew trees has a Toffoli cost of  $\frac{5}{4}N - 1 = 1.25N - 1$ .

Figure 10: Optimized Unary iteration using skew trees. The optimization can be applied only when  $A^{-1} \cdot B$  is efficient to apply where  $A$  and  $B$  are multiplexed unitaries applied via unary iteration. For example - QROM satisfies this criteria.

## References

- [Agr+24] Anjali A. Agrawal, Joshua Job, Tyler L. Wilson, S. N. Saadatmand, Mark J. Hodson, Josh Y. Mutus, Athena Caesura, Peter D. Johnson, Justin E. Elenewski, Kaitlyn J. Morrell, and Alexander F. Kemper. *Quantifying fault tolerant simulation of strongly correlated systems using the Fermi-Hubbard model*. 2024. DOI: [10.48550/ARXIV.2406.06511](https://doi.org/10.48550/ARXIV.2406.06511). URL: <https://arxiv.org/abs/2406.06511>.
- [Bab+18] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. “Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity”. In: *Physical Review X* 8.4 (Oct. 2018). ISSN: 2160-3308. DOI: [10.1103/physrevx.8.041015](https://doi.org/10.1103/physrevx.8.041015). URL: <http://dx.doi.org/10.1103/PhysRevX.8.041015>.
- [CFS24] Clémence Chevalignard, Pierre-Alain Fouque, and André Schrottenloher. *Reducing the Number of Qubits in Quantum Factoring*. Cryptology ePrint Archive, Paper 2024/222. <https://eprint.iacr.org/2024/222>. 2024. URL: <https://eprint.iacr.org/2024/222>.
- [Chi+18] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. “Toward the first quantum simulation with quantum speedup”. In: *Proceedings of the National Academy of Sciences* 115.38 (Sept. 2018), pp. 9456–9461. ISSN: 1091-6490. DOI: [10.1073/pnas.1801723115](https://doi.org/10.1073/pnas.1801723115). URL: <http://dx.doi.org/10.1073/pnas.1801723115>.
- [Cla+24] Baptiste Claudon, Julien Zylberman, César Feniou, Fabrice Debbasch, Alberto Peruzzo, and Jean-Philip Piquemal. “Polylogarithmic-depth controlled-NOT gates without ancilla qubits”. In: *Nature Communications* 15.1 (July 2024). ISSN: 2041-1723. DOI: [10.1038/s41467-024-50065-x](https://doi.org/10.1038/s41467-024-50065-x). URL: <http://dx.doi.org/10.1038/s41467-024-50065-x>.
- [Fow+12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. “Surface codes: Towards practical large-scale quantum computation”. In: *Physical Review A* 86.3 (Sept. 2012). ISSN: 1094-1622. DOI: [10.1103/physreva.86.032324](https://doi.org/10.1103/physreva.86.032324). URL: <http://dx.doi.org/10.1103/PhysRevA.86.032324>.
- [Gid15a] Craig Gidney. *Constructing Large Controlled Nots*. <https://algassert.com>. 2015. URL: <https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html>.
- [Gid15b] Craig Gidney. *Constructing Large Increment Gates*. <https://algassert.com>. 2015. URL: <https://algassert.com/circuits/2015/06/12/Constructing-Large-Increment-Gates.html>.
- [Gid17] Craig Gidney. *Factoring with  $n+2$  clean qubits and  $n-1$  dirty qubits*. 2017. DOI: [10.48550/ARXIV.1706.07884](https://doi.org/10.48550/ARXIV.1706.07884). URL: <https://arxiv.org/abs/1706.07884>.
- [Gid18] Craig Gidney. “Halving the cost of quantum addition”. In: *Quantum* 2 (June 2018), p. 74. ISSN: 2521-327X. DOI: [10.22331/q-2018-06-18-74](https://doi.org/10.22331/q-2018-06-18-74). URL: <https://doi.org/10.22331/q-2018-06-18-74>.
- [Jon13] Cody Jones. “Low-overhead constructions for the fault-tolerant Toffoli gate”. In: *Physical Review A* 87.2 (Feb. 2013). ISSN: 1094-1622. DOI: [10.1103/physreva.87.022328](https://doi.org/10.1103/physreva.87.022328). URL: <http://dx.doi.org/10.1103/PhysRevA.87.022328>.
- [Kim+22] Isaac H. Kim, Ye-Hua Liu, Sam Pallister, William Pol, Sam Roberts, and Eunseok Lee. “Fault-tolerant resource estimate for quantum chemical simulations: Case study on Li-ion battery electrolyte molecules”. In: *Physical Review Research* 4.2 (Apr. 2022). ISSN: 2643-1564. DOI: [10.1103/physrevresearch.4.023019](https://doi.org/10.1103/physrevresearch.4.023019). URL: <http://dx.doi.org/10.1103/PhysRevResearch.4.023019>.
- [Lee+21] Joonho Lee, Dominic W. Berry, Craig Gidney, William J. Huggins, Jarrod R. McClean, Nathan Wiebe, and Ryan Babbush. “Even More Efficient Quantum Computations of Chemistry Through Tensor Hypercontraction”. In: *PRX Quantum* 2.3 (July 2021). ISSN: 2691-3399. DOI: [10.1103/prxquantum.2.030305](https://doi.org/10.1103/prxquantum.2.030305). URL: <http://dx.doi.org/10.1103/PRXQuantum.2.030305>.

- [Lit19] Daniel Litinski. “A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery”. In: *Quantum* 3 (Mar. 2019), p. 128. ISSN: 2521-327X. DOI: [10.22331/q-2019-03-05-128](https://doi.org/10.22331/q-2019-03-05-128). URL: <http://dx.doi.org/10.22331/q-2019-03-05-128>.
- [Lit23] Daniel Litinski. *How to compute a 256-bit elliptic curve private key with only 50 million Toffoli gates*. 2023. DOI: [10.48550/ARXIV.2306.08585](https://doi.org/10.48550/ARXIV.2306.08585). URL: <https://arxiv.org/abs/2306.08585>.
- [Nzs24] Junhong Nie, Wei Zi, and Xiaoming Sun. *Quantum circuit for multi-qubit Toffoli gate with optimal resource*. 2024. DOI: [10.48550/ARXIV.2402.05053](https://doi.org/10.48550/ARXIV.2402.05053). URL: <https://arxiv.org/abs/2402.05053>.
- [Rub+23] Nicholas C. Rubin, Dominic W. Berry, Fionn D. Malone, Alec F. White, Tanuj Khattar, A. Eugene DePrince, Sabrina Siculo, Michael Küehn, Michael Kaicher, Joonho Lee, and Ryan Babbush. “Fault-Tolerant Quantum Simulation of Materials Using Bloch Orbitals”. In: *PRX Quantum* 4.4 (Oct. 2023). ISSN: 2691-3399. DOI: [10.1103/prxquantum.4.040303](https://doi.org/10.1103/prxquantum.4.040303). URL: <http://dx.doi.org/10.1103/PRXQuantum.4.040303>.
- [Rub+24] Nicholas C. Rubin, Dominic W. Berry, Alina Kononov, Fionn D. Malone, Tanuj Khattar, Alec White, Joonho Lee, Hartmut Neven, Ryan Babbush, and Andrew D. Baczewski. “Quantum computation of stopping power for inertial fusion target design”. In: *Proceedings of the National Academy of Sciences* 121.23 (May 2024). ISSN: 1091-6490. DOI: [10.1073/pnas.2317772121](https://doi.org/10.1073/pnas.2317772121). URL: <http://dx.doi.org/10.1073/pnas.2317772121>.
- [San+20] Yuval R. Sanders, Dominic W. Berry, Pedro C.S. Costa, Louis W. Tessler, Nathan Wiebe, Craig Gidney, Hartmut Neven, and Ryan Babbush. “Compilation of Fault-Tolerant Quantum Heuristics for Combinatorial Optimization”. In: *PRX Quantum* 1.2 (Nov. 2020). ISSN: 2691-3399. DOI: [10.1103/prxquantum.1.020312](https://doi.org/10.1103/prxquantum.1.020312). URL: <http://dx.doi.org/10.1103/PRXQuantum.1.020312>.
- [Yua+23] Yewei Yuan, Chao Wang, Bei Wang, Zhao-Yun Chen, Meng-Han Dou, Yu-Chun Wu, and Guo-Ping Guo. “An improved QFT-based quantum comparator and extended modular arithmetic using one ancilla qubit”. In: *New Journal of Physics* 25.10 (Oct. 2023), p. 103011. ISSN: 1367-2630. DOI: [10.1088/1367-2630/acfd52](https://doi.org/10.1088/1367-2630/acfd52). URL: <http://dx.doi.org/10.1088/1367-2630/acfd52>.
- [ZB24] Ben Zindorf and Sougato Bose. *Efficient Implementation of Multi-Controlled Quantum Gates*. 2024. DOI: [10.48550/ARXIV.2404.02279](https://doi.org/10.48550/ARXIV.2404.02279). URL: <https://arxiv.org/abs/2404.02279>.
- [ZSL24] Shuchen Zhu, Aarthi Sundaram, and Guang Hao Low. *Unified Architecture for a Quantum Lookup Table*. 2024. DOI: [10.48550/ARXIV.2406.18030](https://doi.org/10.48550/ARXIV.2406.18030). URL: <https://arxiv.org/abs/2406.18030>.