

A primer on quantum RAM

Olivia Di Matteo

August 19, 2020

Abstract

This document is a work in progress and is being continuously edited.

Contents

1 Introduction and FAQ

Quantum RAM (qRAM) has gained some notoriety in the past few years, and attitudes toward it depend quite heavily on the circles you run in. The goal of this set of notes is to provide a general overview of the subject, as well as discuss some very specific implementations. It is meant to be a companion to the Q# qRAM libraries we are writing (cite the repo), but can also serve as a standalone reference.

In my experience, explaining qRAM to someone boils down to answering a handful of key questions. I'll provide some potentially unsatisfactory answers up front, but go into far more detail in the next few sections¹.

1. **Do I need a qRAM?** *Sometimes.* You'll need a qRAM, or some more general means of *quantum state preparation* in quantum machine learning (QML) algorithms that require you to load in classical data, or query an oracle that returns classical data. I've heard a number of stories of people working on QML being actively discouraged from doing so because "QML won't work without a qRAM". That's just not true, because *many QML algorithms do not need a qRAM*. Now, whether or not they yield any quantum advantage is a separate question, and won't be discussed here. The key point I want to make is that *some* QML algorithms need a qRAM, and they will potentially run into trouble as per the next question.
2. **Can we design an efficient qRAM?** *Maybe.* In this primer we'll take a look at proposals that will in principle run in polynomial depth, and others that scale far worse. There are some very interesting qubit-time tradeoffs one can explore, in particular if the data being stored has some sort of underlying structure. Regardless, even if we can design an efficient circuit, we'd also like something that is efficient in a fault-tolerant setting, and this is potentially very expensive.
3. **Can I build one?** *Maybe.* No one has actually done so, but there are a handful of hardware proposals that will be discussed in more detail in ??.

2 A brief history of the brief history of quantum machine learning

3 Encoding classical data on a quantum computer

There is a nice overview of the encodings below in [?].

¹Those details may not be satisfactory either.

3.1 Basis encoding

Give overview of the basis encoding, i.e.

$$\text{data } 01001 \rightarrow |01001\rangle \quad (1)$$

3.2 Quantum RAM / ROM

Query as bits in superposition:

$$\sum_i |i\rangle |0\rangle \rightarrow \sum_i |i\rangle |b_i\rangle \quad (2)$$

Query as phase in superposition (applications to Grover)

$$\sum_i |i\rangle \rightarrow \sum_i (-1)^{b_i} |i\rangle \quad (3)$$

3.3 Quantum state preparation (amplitude encoding)

Given some vector $\mathbf{a} = (a_0, \dots, a_{n-1})$, create the quantum state

$$|\psi\rangle = \sum_{i=0}^{n-1} a_i |i\rangle \quad (4)$$

Discuss how this relies on having a qROM as an underlying subroutine.

4 Bucket-brigade quantum RAM

Architectures for qRAM began to emerge roughly a decade ago with the bucket brigade RAM model of [?, ?]. The particular storage model of RAM that is used is a binary tree, where 2^n bits are located at the leaves of an n -level binary tree, as depicted in ???. We will first take a graphical look at the bucket brigade algorithm; following this, we will see it modelled as a quantum circuit.

4.1 Increasing efficiency of RAM with a bucket brigade

The motivation for bucket brigade lies in a less efficient type of RAM denoted as ‘fanout RAM’. In the fanout RAM, the nodes of this tree are transistors that, depending on their state, will route incoming current left or right towards the next level of the tree. Each level of the tree is associated to an address bit; the value of the address bit can set a level-specific switch that tells all transistors in the level to send current left, or send current right. This is shown in the right panel of ??, and has the effect creating only one complete path from the root of the tree to a leaf.

The issue with fanout RAM in this form is its efficiency - for an n -bit address, we are turning on $2^n - 1$ transistors, while only n of them are involved in the actual query. A further issue arises if we make a direct translation into qubits. Suppose we replace each node of the tree with a qubit; we can let its $|0\rangle$ state indicate ‘route left’, and its $|1\rangle$ state indicate ‘route right’. When we go to initialize the qubits to create the path, the k ’th address qubit will have to couple with 2^k node qubits. This is a very precarious superposition, especially in an era where coherence times are low and multi-qubit operations are significantly noisier than single-qubit ones.

The bucket-brigade model is an improvement that addresses both of these issues. Rather than using qubits at each node, we use *qutrits*, or three-state systems. In what follows we will discuss the quantum case, but the methodology is the same for the classical case (it uses trits instead of bits).

In the top left of ?? is our familiar binary tree. At each node sits a qutrit whose states are descriptively named $|\text{wait}\rangle, |\text{left}\rangle, |\text{right}\rangle$. All the qutrits begin in $|\text{wait}\rangle$. To perform a

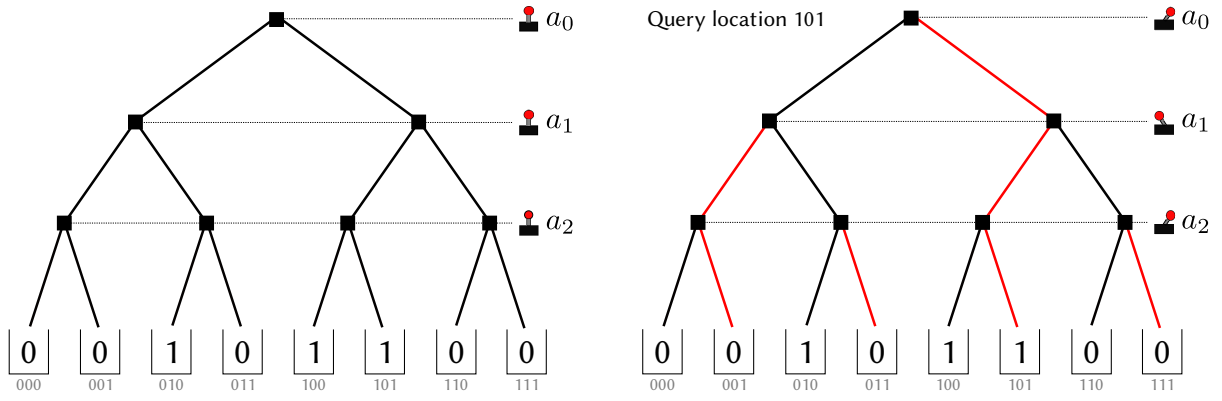


Figure 1: (Left) Schematic of a ‘fan-out RAM’, whose inefficiencies were the motivation for the bucket brigade RAM. For an n -bit address, an n -level binary tree stores the memory contents at its leaves. At each node of the tree is a transistor that can direct current right, or left. Within each level of the tree, all transistors are controlled by a switch that can either direct them all to route current left, or all right. (Right) To query the memory contents, the directions of the switches are set according to the address bits (one bit per level of the tree). This creates a single path from the root node to the desired cell.

query, *address qubits* are introduced to the tree one by one; the k ’th address qubit will effect an operation in the k ’th level of the tree to carve a path to the desired memory cell.

The key to the bucket brigade model is to define a unitary transformation that allows address qubits to alter the state of the qutrit nodes depending on their own state. Specifically, we need to find some unitary that performs:

$$U(|0\rangle |wait\rangle) \rightarrow |s\rangle |left\rangle \quad (5)$$

$$U(|1\rangle |wait\rangle) \rightarrow |s\rangle |right\rangle \quad (6)$$

Here, $|s\rangle$ refers to some arbitrary state; we don’t particularly care what it is, as U is reversible and will be undone later on.

When an address qubit reaches a qutrit in the $|wait\rangle$, U is performed. If it instead encounters $|left\rangle$ or $|right\rangle$ it ‘passes through’ the node without altering it, and continues traveling along the path until it reaches its particular level. This is shown in the top right of ??.

The bottom half of ?? describes how the contents of the memory are retrieved. After all the address qubits are sent, a ‘quantum bus’ qubit traverses the path, couples to the desired memory cell to gather the data, and is reflected back the way it came. As it passes upwards through the tree, it re-initializes the qutrits back to $|wait\rangle$ by running U^\dagger . Note that such an algorithm allows us to query in superposition, as all operations are unitary.

To see how efficiency is improved let’s count the number of unitary operations. The first address qubit performs one operation by coupling with the root qutrit. The second qubit performs two, the root qutrit and then the qutrit in the second layer. The third performs three, and so on. This means to set the qutrits takes $n(n+1)/2$ operations, more succinctly represented as $O(n^2)$. The bus qubit performs only $2n+1$ operations, which does not worsen this complexity. As this is polynomial in the address bit size, the bucket brigade qRAM appears to be efficient. It is also claimed that, since only n of the qutrits are involved in the query (rather than all $2^n - 1$), it is potentially possible to get away with a worse error rate $O(1/n^2)$, and leave the idle qutrits uncorrected. **I have to go check my notes, this paragraph needs to be improved.**

1. **What happens when we query in superposition?** *I don’t have a satisfactory answer for this.* The claimed advantage comes partially from the fact that we only need to tinker with n of the qutrits; but if we query in superposition, don’t we still have to handle all $2^n - 1$ of them anyways?
2. **Can we actually get away without error correction?** *Potentially.* It has been shown that in cases where there are a superpolynomial amount of queries to the qRAM (for example, in Grover’s algorithm), the error rate of the qRAM must be superpolynomially small [?], in which case we will in fact need error correction [?]. That said, in situations

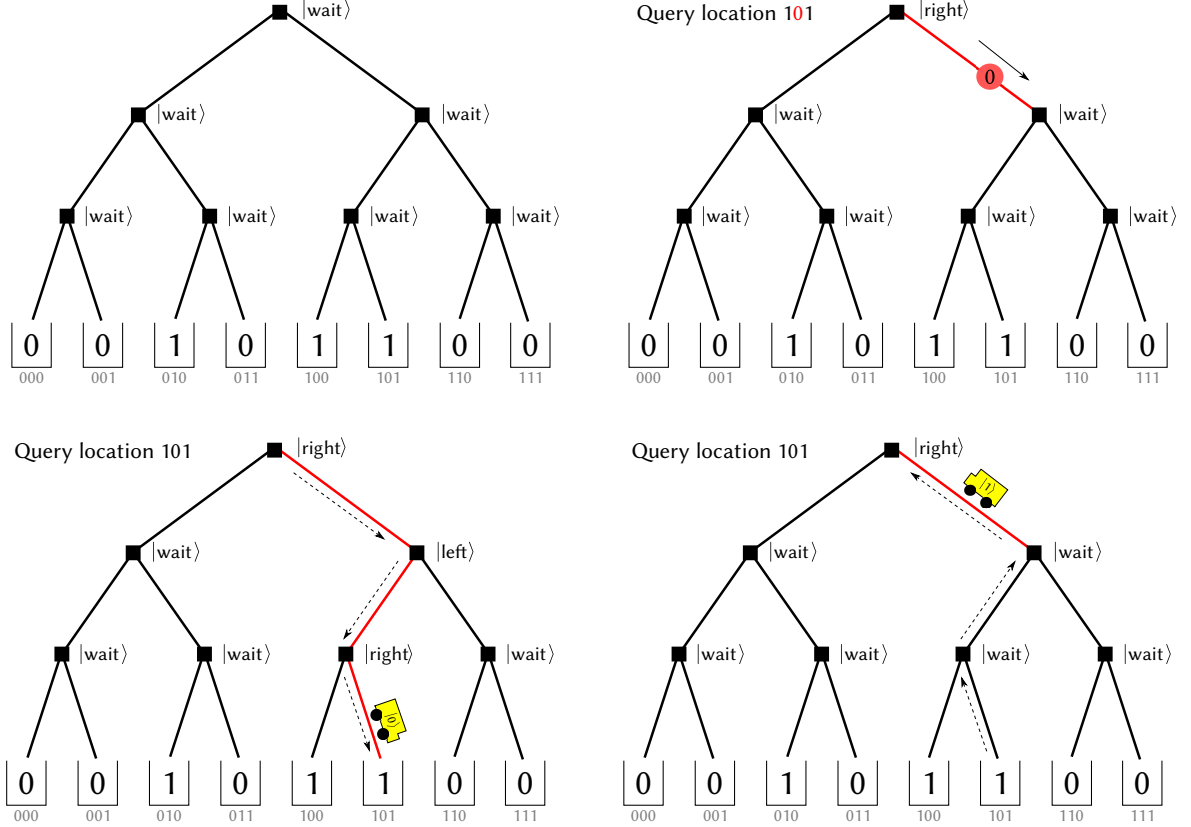


Figure 2: Schematic of a bucket brigade quantum RAM query. (Top left) Memory contents are stored in the leaves of a binary tree, whose nodes are denoted by qutrits with possible states $|wait\rangle$, $|left\rangle$, and $|right\rangle$. (Top right) A bucket brigade RAM query is set up by sequentially sending in address qubits to change the qutrit states until a path to the desired cell is created. (Bottom left) To extract the contents of the memory cell, a ‘bus’ photon is used. The photon is directed to the desired cell by the qutrits, where it couples to the memory cell and copies its contents. (Bottom right) The bus photon travels back through the qRAM, resetting the qutrit states to $|wait\rangle$ on its way out.

where the number of queries is small, it may be possible yet to get away without it, in which case bucket brigade provides a substantial improvement in the number of operations that must be performed.

4.2 Bit query circuit model

While conceptually simple, the bucket brigade model as described in the previous section does not lend itself very well to implementation on our present day machines. For one, it requires both qutrits and qubits; furthermore, the operations are not specified in the usual parlance of the circuit model - how do we implement this U , and the routing?

A circuit model for bucket brigade qRAM was presented in [?]. ?? contains a re-drawn version of these circuits in order to highlight the structure of the algorithm as well as the patterns of gates that are used to perform the query.

The bucket brigade circuit model consists of four registers of qubits. To query an n -bit address, we require:

- An address register, a , of n qubits
- A memory register m with 2^n qubits containing the stored values (i.e. qubits in $|0\rangle$ or $|1\rangle$).
- An auxiliary register τ with 2^n qubits to trigger readout of the memory register
- An output register

To perform a query, the address register is first ‘fanned out’ onto the auxiliary register τ in such a way that it produces a ‘one-hot’ encoding of the addresses to be queried. For example, if we want to query at address $|i\rangle$, the fanout will set the i ’th qubit of τ to $|1\rangle$. This then couples to the i ’th qubit of the memory register using a Toffoli, and changes the output bit depending on the contents. Finally, note that for full reversibility, the fanout must be run in reverse to reset the auxiliary qubits back to $|0\rangle$.

This can also be generalized to the case where the memory contents are multi-bit values. A circuit for this case is presented in ??. The structure of the address and auxiliary registers is the same, but the memory and output registers differ. If the 2^n memory cells contain k -bit values, the output register changes from 1 qubit to k qubits. The memory becomes $(k + 1)2^n$ qubits wide. Essentially, the memory register is split into 2^n subregisters where k qubits contain the memory contents, and the last qubit is used as an indicator to trigger readout. The trigger register τ works on this set of indicator qubits (shown in red in ??). The indicator qubits then couple with the memory contents and copy the values to the output.

4.2.1 Resource estimation

One of the primary reasons for designing qRAM libraries in Q# was to get better resource estimates for qRAM circuits using the functionality afforded to us by both the `ResourcesEstimator` and the `ToffoliSimulator`. Note that all the gates in ?? and ?? are Toffolis, and so bucket brigade qRAMs are really just fancy Boolean circuits. Plenty of optimization tools (cite some) exist for Boolean circuits, and so we anticipate that we’ll be able to do reasonably well. But first, let’s do some resource estimation ‘by hand’ to get an idea of the worst we can expect.

We will do resource estimation at the *logical level*, i.e. not delve into the fault-tolerant level and estimate things like physical qubits and time required. Instead, we will focus on the following set of parameters:

- Number of qubits N_q (or, circuit width)
- Circuit depth D
- T -depth D_T
- T -count N_T
- CNOT-count N_{CX}
- non-CNOT Clifford count N_{HS}

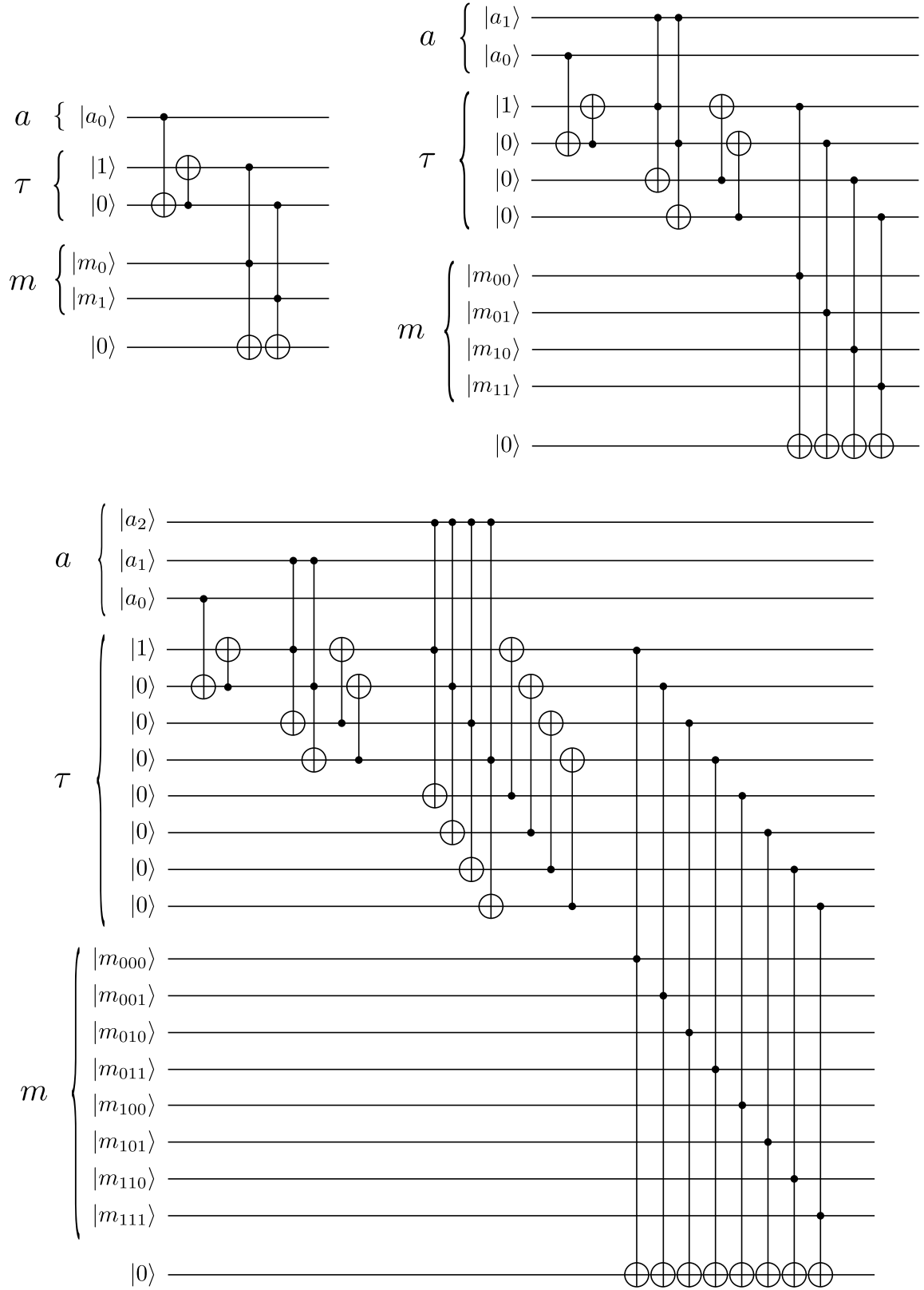


Figure 3: **For pedagogical purposes; not most optimal.** (Top left) A bucket brigade circuit for a 1-qubit address (2 memory cells). (Top right) A bucket brigade circuit for a 2-qubit address. The first 6 gates are a fanout of the address bits to the auxiliary ‘trigger’ register, denoted by τ . A cascade of Toffolis then couples the memory cells to the output. To make the query fully reversible, the fanout component must be repeated. The subscripts on the memory cells are ordered from top to bottom of the address register, i.e. $a_{n-1}, a_{n-2}, \dots, a_0$. (Bottom) A bucket brigade circuit for a 3-qubit address, highlighting the pattern that emerges in the construction of this circuit. The fanout component is recursive, with the 1- and 2-qubit fanout portions appearing.

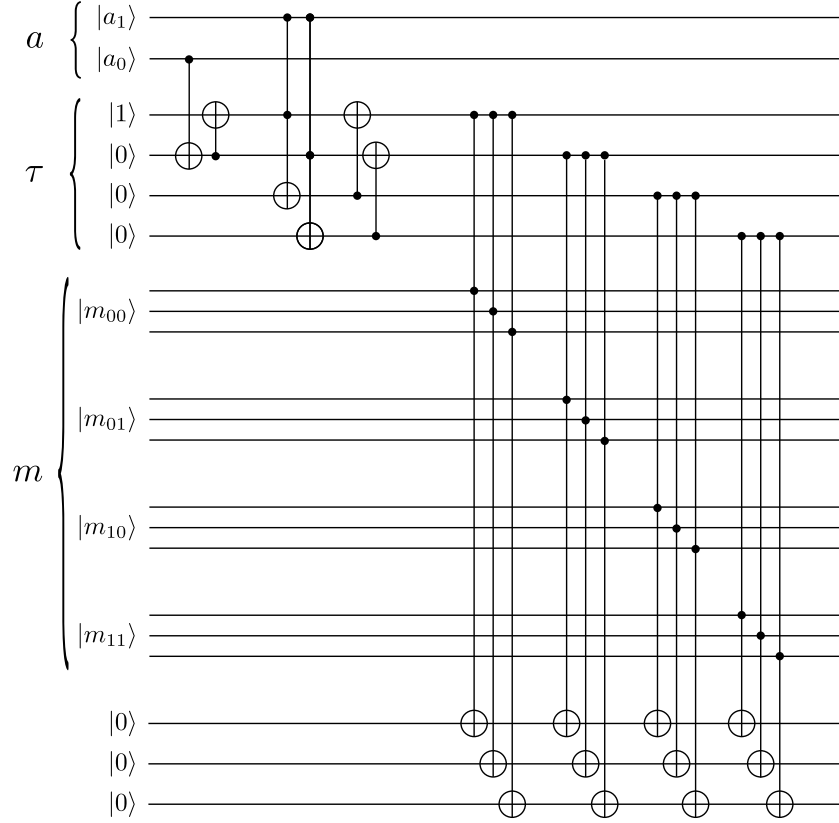


Figure 4: **For pedagogical purposes; not most optimal.** Extension of the bucket brigade circuit model to the case where each memory cell contains multiple output bits (in this case, 3). The structure of the circuit is largely the same, except now each trigger qubit must perform a Toffoli for every bit in the cell.

Auxiliary qubits	T -depth	Reference
0	3	MITM
1	2	Cite
3	1	[?]

Table 1: A Toffoli gate can be decomposed over Clifford+ T . While 7 T gates total are required, the T -depth is flexible if one chooses to add some additional qubits. [Verify citations](#)

The value of N_{HS} is often not included in lower-level resource estimation. For one, on a present-day machine, CNOTs are much noisier and harder to implement, so reducing the number of CNOTs is far more important than reducing the number of other types of gates. Furthermore, of the single-qubit gates, the T gate is the one that will give us the most trouble in a fault-tolerant setting. The H and S operations are generally considered ‘easy’, but we still include it in our analysis for completeness.

4.2.1.1 Single-bit output Before we begin, we need to make some assumptions. All of the gates in a BB circuit are either CNOTs or Toffolis. Assuming we choose the typical universal gate of Clifford+ T , it is necessary to further decompose the Toffolis into elementary gates. There are a number of ways of doing so, and they differ in that one needs to make a choice about whether to use auxiliary qubits. They all require a total of 7 T gates, but the T -depth can be decreased with extra qubits. There are three ‘standard’ decompositions, and their respective resource counts are shown in ??.

In what follows, we will take the simplest route and assume that no auxiliary qubits are added, but note that the resource counts for the T -depth 1 case are listed in [?]. We will use the decomposition of Figure 13 in [?], which is reproduced here in ??.

The number of qubits required was explained previously, but let’s total them up. Suppose

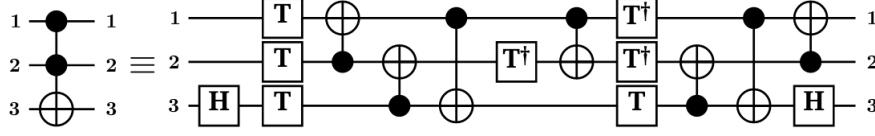


Figure 5: A T -depth 3 version of the Toffoli gate over Clifford+ T . This circuit has full depth 9. Figure reproduced from Figure 13 of [?]. remake as vector graphic

that our memory addresses are n bits. Then:

Register	Number of qubits
a	n
τ	2^n
m	2^n
Output	1
Total N_q	$2^{n+1} + n + 1$

(7)

Next, let's take a closer look at the gate counts. In the fanout portion, we see there are $2^n - 2$ Toffolis, and 2^n CNOTs. In the case where the auxiliary register is being re-used, the fanout portion must be run twice (depending on how the library handles aux registers, we may not actually need this), yielding $2^{n+1} - 4$ Toffolis and 2^{n+1} CNOTs. As for the query portion, we see there are 2^n sequential Toffolis.

Let us begin by computing the T -count N_T . Each Toffoli has 7 T -gates, so we have

$$\begin{aligned}
 N_T &= 2 \cdot N_T(\text{fanout}) + N_T(\text{query}) \\
 &= 2 \cdot 7 \cdot (2^n - 2) + 7 \cdot 2^n \\
 &= 21 \cdot 2^n - 28
 \end{aligned}
 \tag{8}$$

Next, the T -depth of all the Toffolis. Since none of the Toffolis are parallelized, it is just the sum of the T -depth of the individual gates:

$$\begin{aligned}
 D_T &= 2 \cdot D_T(\text{fanout}) + D_T(\text{query}) \\
 &= 2 \cdot 3 \cdot (2^n - 2) + 3 \cdot 2^n \\
 &= 9 \cdot 2^n - 12
 \end{aligned}
 \tag{9}$$

The number of non- $CNOT$ Cliffords is also straightforward; only the Toffolis have any such gates: 2 Hadamards per Toffoli. Thus,

$$\begin{aligned}
 N_{HS} &= 2 \cdot N_{HS}(\text{fanout}) + N_{HS}(\text{query}) \\
 &= 2 \cdot 2 \cdot (2^n - 2) + 2 \cdot 2^n \\
 &= 6 \cdot 2^n - 8
 \end{aligned}
 \tag{10}$$

For N_{CX} , we have

$$\begin{aligned}
 N_{CX} &= 2 \cdot N_{CX}(\text{fanout}) + N_{CX}(\text{query}) \\
 &= 2 \cdot (7 \cdot (2^n - 2) + 2^n) + 7 \cdot 2^n \\
 &= 23 \cdot 2^n - 28
 \end{aligned}
 \tag{11}$$

Finally, let's compute the depth. Note that in the fanout portion, the CNOTs can be run in only $n + 1$ layers of depth because many of the cascades can be parallelized. The depth of the Toffolis, however, is fixed at 9, so we obtain:

$$\begin{aligned}
 D &= 2 \cdot D(\text{fanout}) + D(\text{query}) \\
 &= 2 \cdot (9 \cdot (2^n - 2) + n + 1) + 9 \cdot 2^n \\
 &= 27 \cdot 2^n + 2n - 34
 \end{aligned}
 \tag{12}$$

The final (unoptimized) resource counts for a one-bit bucket brigade qRAM are:

$$\begin{aligned}
N_q &= 2^{n+1} + n + 1 \\
D &= 27 \cdot 2^n + 2n - 34 \\
D_T &= 9 \cdot 2^n - 12 \\
N_T &= 21 \cdot 2^n - 28 \\
N_{CX} &= 23 \cdot 2^n - 28 \\
N_{HS} &= 6 \cdot 2^n - 8
\end{aligned} \tag{13}$$

VERIFY and cross-check with `ResourcesEstimator` and our library code.

4.2.1.2 Multi-bit output Let us now perform the same analysis for the situation where the memory cells may store multiple bits. We'll suppose that addresses are still n bits, but store k bits each. Consider the differences between the top right panel of ??, and ?? (i.e. a 2-bit address, so 4 memory cells). The address fanout portion is *exactly the same*; what differs is the query itself, as now we need to couple with each of the memory cells to copy the information to the output. The number of qubits of course increases:

Register	Number of qubits
a	n
τ	2^n
m	$k \cdot 2^n$
Output	k
Total N_q	$(k+1)2^n + n + k$

(14)

Then each of the 2^n Toffolis of the query portion is replaced with k Toffolis, one per output bit. Note, however, that many of these Toffolis can in fact be done in parallel. We will not consider that when estimating resources here by hand, but note that once we implement the circuits in Q# and use the Toffoli simulator, the circuit depth will likely decrease significantly. Calculating by hand, the resources for the multi-output bit case are as follows:

$$\begin{aligned}
N_q &= (k+1)2^n + n + k \\
D &= (18 + 9k)2^n + 2n - 34 \\
D_T &= (6 + 3k)2^n - 12 \\
N_T &= (14 + 7k)2^n - 28 \\
N_{CX} &= (16 + 7k)2^n - 28 \\
N_{HS} &= (4 + 2k)2^n - 8
\end{aligned} \tag{15}$$

Verify and cross-check with `ResourcesEstimator`

4.3 Improving T depth with the CCZ gate

Looking at the circuits of ??, one notes that there are many instances of sequential Toffolis. In [?], a parallelized version of the circuit was proposed. In the fanout, multiple copies of the address bits were made so the Toffolis could be performed in parallel. The query portion was parallelized by giving each Toffoli a different target qubit, and then taking the parity of those bits using CNOT gates. This works fine, and has the effect of reducing the T -depth from $O(2^n)$ to $O(n)$. However, it comes at a significant cost: the number of auxiliary qubits that must be added is $O(2^n)$.

These sections are rough, and will be re-written as I make my way through [?]. The authors of [?] found a more efficient way to parallelize this by taking advantage of some circuit identities that can be applied when two neighbouring Toffolis share a control bit. In fact, this gives a significant advantage - the T -depth will go from $O(2^n)$ to $O(n)$ *without* adding any auxiliary qubits!

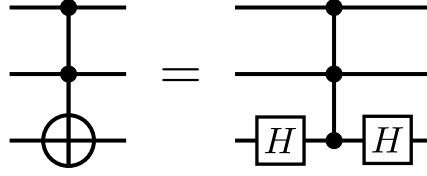


Figure 6: A Toffoli can be re-written as a controlled-controlled- Z gate with its target qubit conjugated by Hadamards.

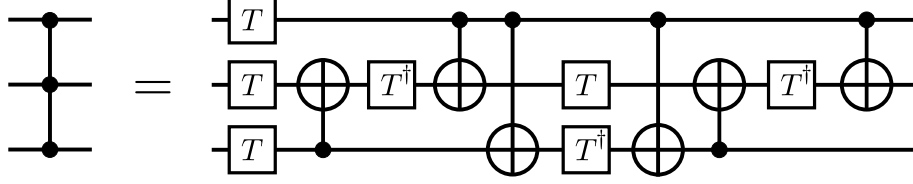


Figure 7: Decomposition of CCZ over Clifford+ T gate set. It requires 7 T gates, has T depth 4, and total depth 10.

The trick involves replacing the Toffoli gates with controlled-controlled- Z (CCZ) gates, as per the identity shown in ???. The CCZ gate perform the operation

$$CCZ |b_0 b_1 b_2\rangle = (-1)^{b_0 b_1 b_2} |b_0 b_1 b_2\rangle \quad (16)$$

In other words, the phase of the state is flipped only if all three qubits are in $|1\rangle$. Note that due to this symmetry, qubits in a CCZ lose the distinction of target/control. The CCZ can be done with the qubits in ‘any order’, as the only time the state is affected is when all qubits are in the same state anyways. Thus, every Toffoli in the bucket brigade circuits can be turned into a CCZ by simply conjugating the target by Hadamard gates.

To see how the T -depth is reduced when working in terms of CCZ, we refer to the Clifford+ T decomposition shown in ???. The T -count of this circuit is 7 (as is the Toffoli), and its T depth is 4. The magic happens when two adjacent CCZs share a control bit. An example is shown in ???. Using the decomposition of ??, we can rewrite the cascades of Toffolis that occur from the address bits to the auxiliary register. Note that this *still* has only T depth 4. Also, it has T -count 12 rather than 14 - the T on the shared control qubit combine to form an S gate, which is a Clifford. (Identities and notation for this circuit are shown in ??.)

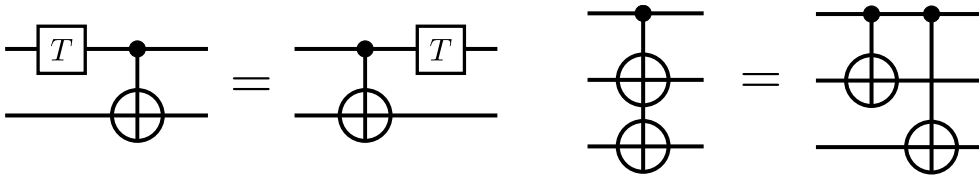


Figure 8: (Left) When simplifying the circuit for two CCZs sharing a control bit, we make use of the fact that a T gate commutes with a CNOT when applied to the control qubit (note that this is not true for a T gate applied to the target). (Right) Notation for a fanout of CNOTs. This notation indicates that the same qubit is the control for multiple target qubits.

The identity in ??? can be applied throughout the entire bucket brigade circuit. Full examples for the case of 2 and 3 address bits are shown in ?? and ?? respectively. In the query portion, recall that the initial set of Toffolis share a common target, rather than a control bit. However once we conjugate these by Hadamards (all but the outer two cancel), we are left again with a sequence of CCZs sharing a ‘control’ (the former target), and can make use of the same identity.

In these figures, the savings in depth becomes apparent. While the T -count is reduced only slightly, the T s have been nicely parallelized. The query portion in particular has now only T depth 4. The fanout portion has T -depth $4(n - 1)$: $n - 1$ of the address bits contain adjacent Toffolis, and each has depth 4. Taking into account the reverse of the fanout, the total T -depth is $8n - 4$, a significant reduction over the $9 \cdot 2^n - 12$ that was naively computed in the previous section. In the case where the memory cells store multiple data bits, the query portion is

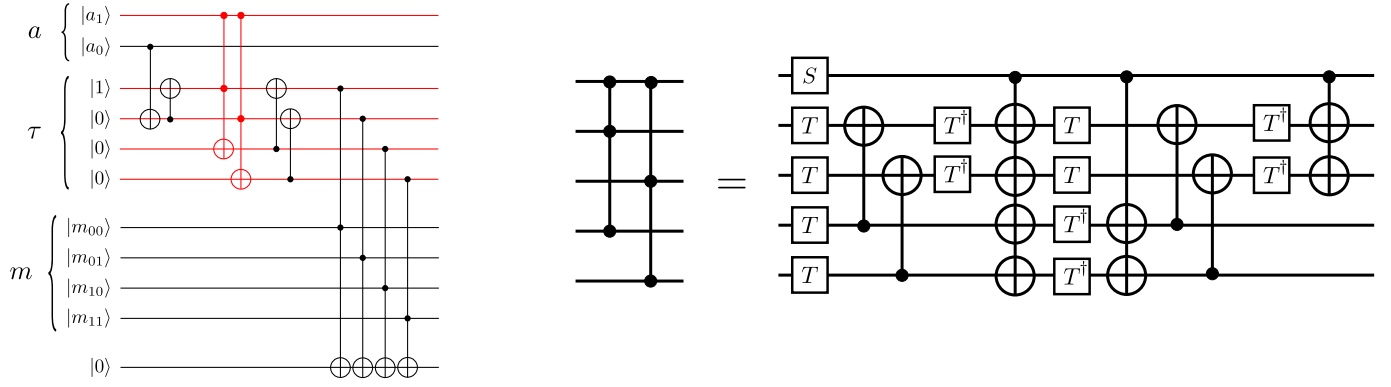


Figure 9: If we apply the decomposition of ?? to portions of the address fanout, we can significantly reduce the T depth and circuit depth. We can even reduce the T count by 2, since the T gates on the shared control qubit combine to form an S . Rather than having 14 T gates, T depth 8, and total depth 20 as one naively would expect for two CCZs in sequence, it has T count 12, T -depth 4, and total depth 13.

repeatedly identically for each target bit with the same auxiliary register, but coupling to the associated memory cell. For k -bit output, the T -depth is thus $8n - 8 + 4k$.

Add discussion of circuit depth and complexity of ‘quantum fanout’.

4.4 Phase query circuit model

In some situations, most notably in Grover’s algorithm, one is concerned with retrieving the memory contents not in bit form, but as a phase. Explicitly,

$$|i\rangle \rightarrow (-1)^{m_i} |i\rangle \quad (17)$$

Such a query can be done using slight modifications from the bucket brigade model. In particular, all that must be done is to start with the target in the state $|1\rangle$, and replace the Toffolis in the query with CCZs. An example is shown in ?? for a 2-bit address. This circuit can be parallelized using the same techniques as shown in ?. In fact, the only difference is that the target qubit starts in $|1\rangle$, and the Hadamards are removed.

What should this look like for multioutput bit? Should it be the product of the memory contents?

4.4.1 Resource estimation

5 Quantum ROM

5.1 Quantum ROM

5.2 Efficient state preparation with a qROM

Discuss [?]

5.3 Optimizing quantum ROMs

5.4 Problem-specific qROMs

Talk about the qROM in [?]

6 Hardware proposals for qRAM

7 Outlook



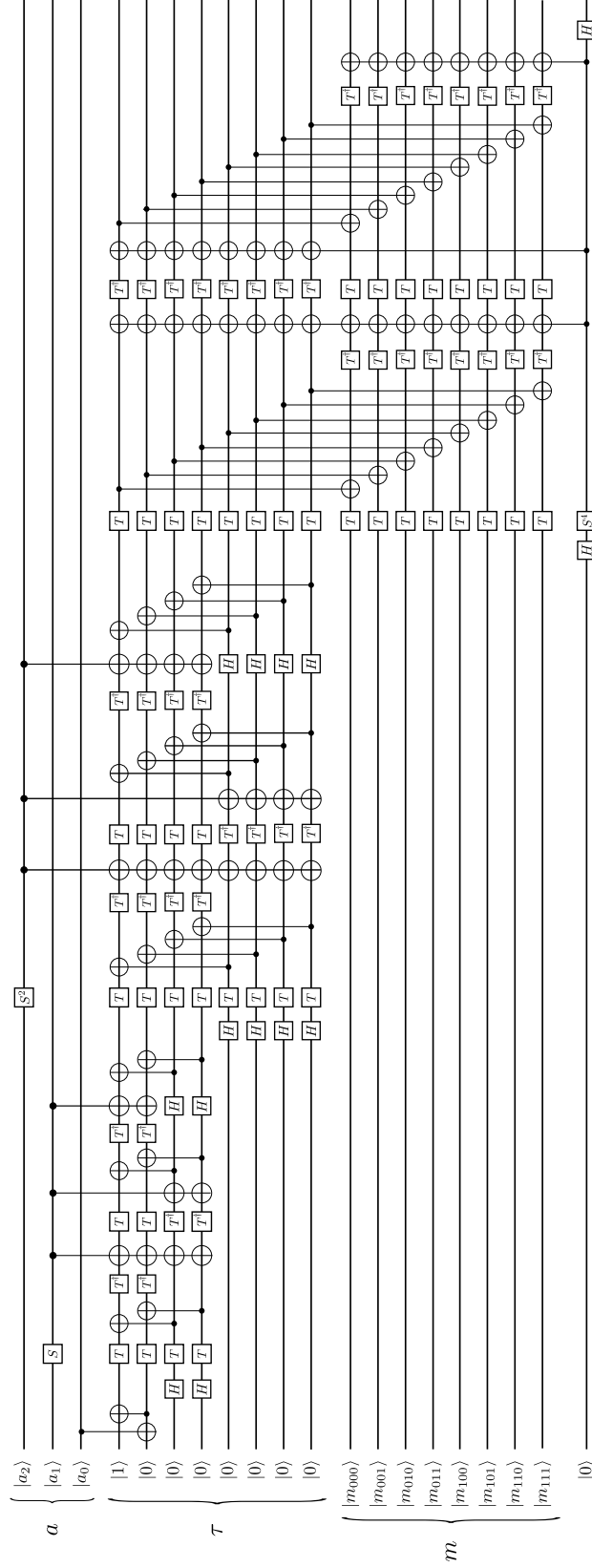


Figure 11: 3-address bucket brigade circuit with Toffoli cascades as CCZs. This figure should eventually be removed because it's enormous, but leaving here for now as a reference. Note that the S^4 on the target register is actually 1; it's been left here for clarity.

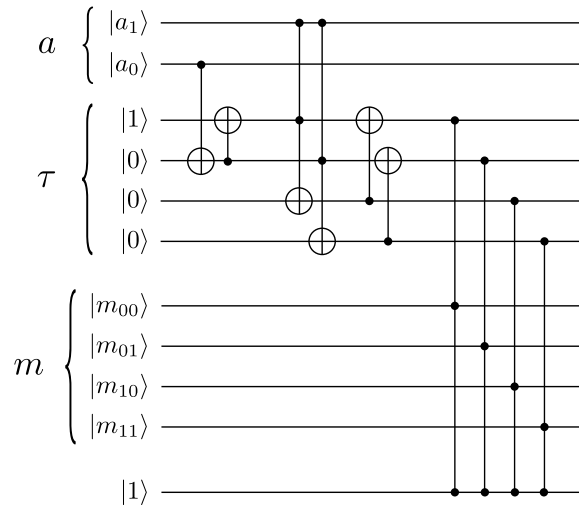


Figure 12: Bucket-brigade style circuit that performs the query $|i\rangle \rightarrow (-1)^{m_i} |i\rangle$, where m_i is the contents of the memory at address $|i\rangle$. Note that the target qubit starts in $|1\rangle$ allow for phase kickback with the CCZ; the CCZ triggers only if all of the auxiliary, memory, and targets are in $|1\rangle$.