Multiple Control X Gate Synthesis

2023 Studentship

Enda Harrigan

20348273

E.HARRIGAN2@universityofgalway.ie

October 12, 2023

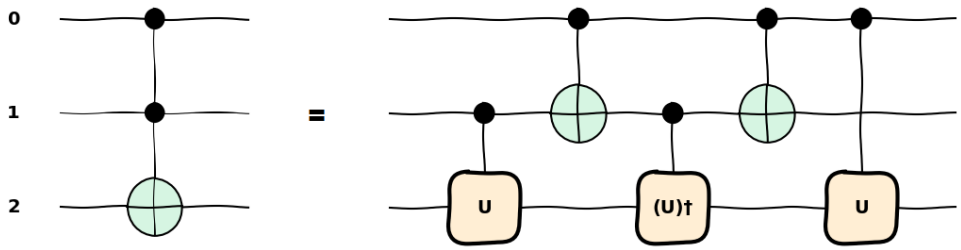
Contents

1 The Toffoli

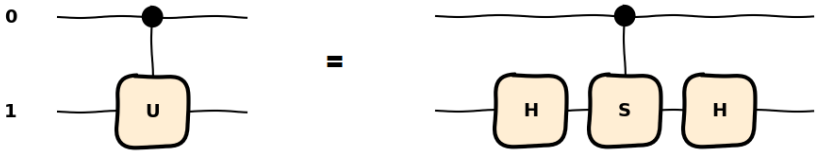
1.1 From the Ground Up

In order to understand how all this worked I started at the very basics; how to build a Toffoli gate. I needed to use the following equivalences:

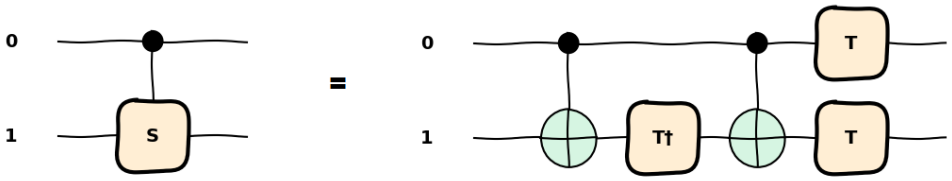
The Toffoli can be decomposed to two $CNOT$ gates and three controlled $\sqrt{X}$ gates as shown below. The program I was using did not support the use of the $\sqrt{X}$ symbols



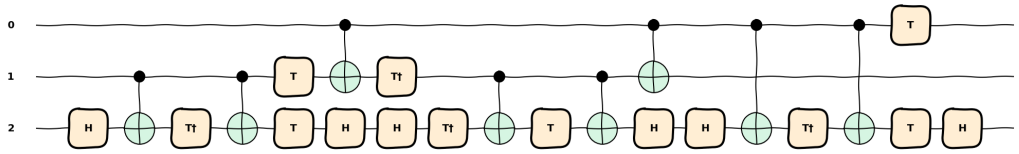
The controlled $\sqrt{X}$ gate is decomposed into two Hadamard gates and a controlled S gate as follows:



The controlled S gate is then broken down further into three T gates and two $CNOT$ gates:



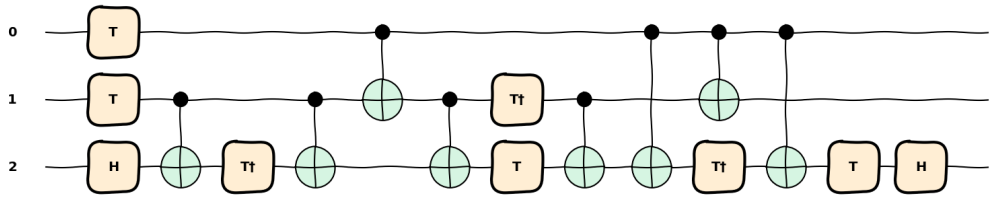
Putting all these equivalences together you get the rather large and expensive circuit below:



1.2 Toffoli Optimisation

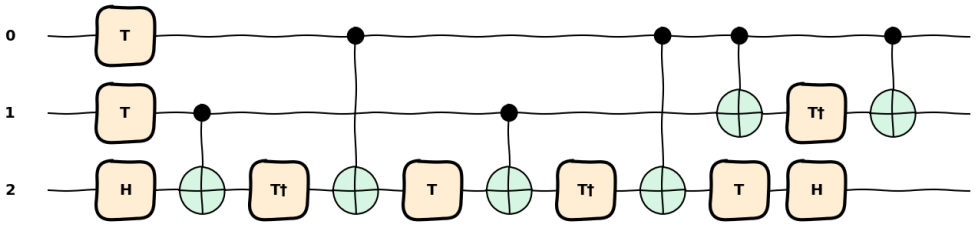
We can cut down on gates by knowing that Hadamard gates are unitary and are their own inverses, so cancel each other out. Any T gates next to a $T^\dagger$ gate will also cancel out, as will any matching $CNOT$ gates next to one another. Additionally It should be noted that some of the operators can commute; $(T \otimes I)(CNOT) = (CNOT)(T \otimes I)$

This gives us:



There are a good few $CNOT$ gates side by side, and luckily there are tricks involving them to further cut down the gate count; $(I \otimes CNOT)(CNOT \otimes I) = (CNOT \otimes I)(C \otimes I \otimes NOT)(I \otimes CNOT)$

Using this new rule we can cancel some more $CNOT$ and T gates and reorder a bit to give;



Here we have the furthest optimised form of the Toffoli gate without providing any extra wires called ancillae. It contains seven T, six $CNOT$ and two Hadamard gates.

2 Three Controlled X Gate

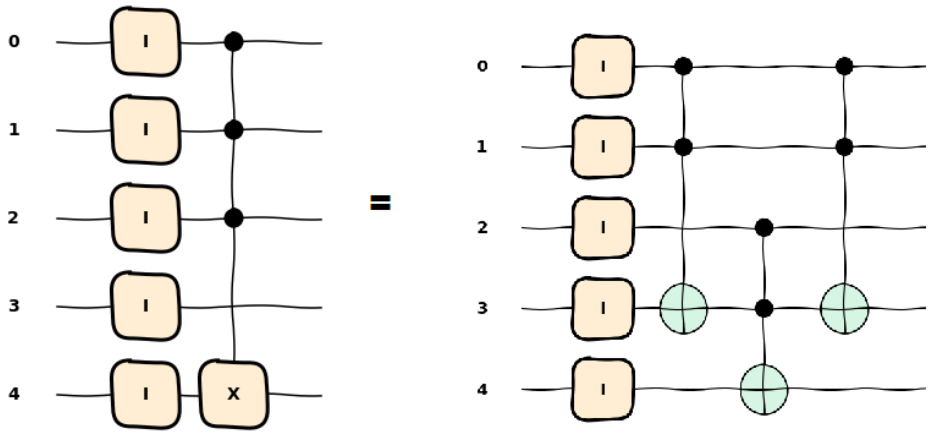
2.1 Discussion

The Three Controlled X Gate is, as the name suggests, an X gate controlled by three other wires. It can be constructed in a number of ways, but I will be focusing on employing Toffoli gates and one ancilla. This ancilla can be either 'clean' or 'dirty', which I will get into later.

2.2 Clean Ancilla Build

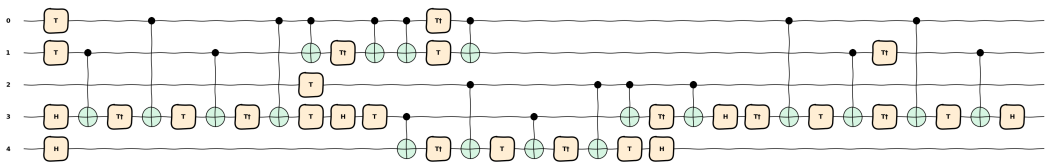
A clean ancilla is an extra wire added to a circuit that must start in a $|0\rangle$ state, and end in a $|0\rangle$ once computations have ended. This is to allow the ancilla to be used later on in potentially another circuit, and just because it is good practice to return ancillae to their previous states after use.

The Three Controlled X Gate (henceforth referred to as C^3X) is constructed with a clean ancilla as shown below:

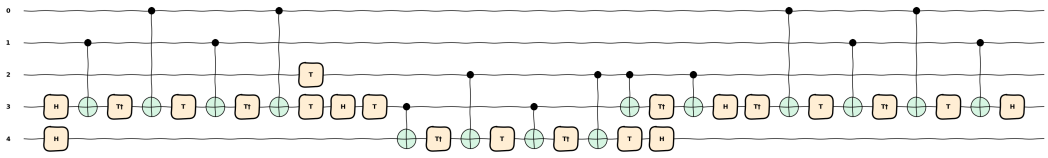


The first Toffoli gathers the information from wires 0 and 1 and stores it in the ancilla located at wire 3. The second Toffoli then uses the information on wire 2 and the ancilla to control wire 4, our target wire. Finally the third Toffoli uncomputes the ancilla, returning it to the $|0\rangle$ state.

I have already demonstrated the way in which one builds the Toffoli gates, so I will dive straight into substituting them in for the circuit above. I will however note that I will use a backwards facing Toffoli for the last one, as this will allow me to more effectively cancel out some gates.



Straight away you should be able to see plenty of gates that can be removed, particularly every gate situated on wires 0 and 1. This leaves us with the final circuit diagram shown below:



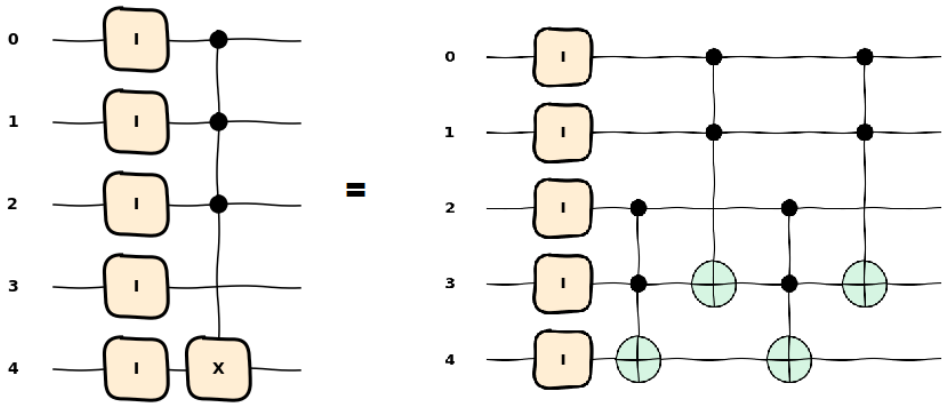
The image to the right shows the various inputs and resulting outputs possible for the circuit above. As expected, the last bits are changed only if the first three bits are 1, and the second last bit remains at 0 the entire way across. It acts exactly as a C^3X would with a $|0\rangle$ state wire present.

```
00000 --> 00000
00001 --> 00001
00100 --> 00100
00101 --> 00101
01000 --> 01000
01001 --> 01001
01100 --> 01100
01101 --> 01101
10000 --> 10000
10001 --> 10001
10100 --> 10100
10101 --> 10101
11000 --> 11000
11001 --> 11001
11100 --> 11101
11101 --> 11100
```

2.3 Dirty Ancilla Build

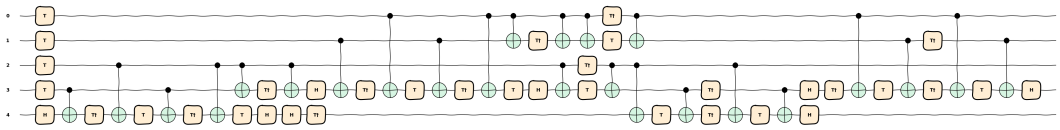
Similar to the clean ancilla, a dirty ancilla is an extra wire that must start and end in any arbitrary state $|\phi\rangle$. Gates built to utilise dirty ancillae are very versatile as they can be inserted into the middle of any circuitry without the fear of any unwanted effects to the wiring.

The C^3X is constructed with a dirty ancilla as shown below:

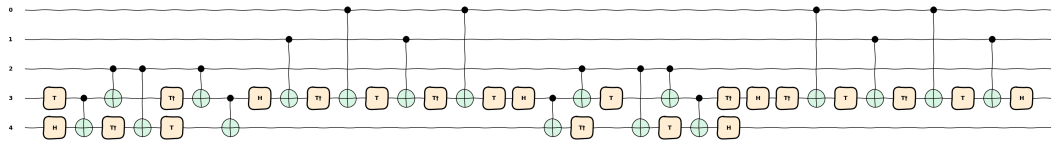


The construction works much in the same way as the clean ancilla build, except there is an extra level of uncomputation to make sure the ancilla and the target wires are not left in an unwanted state.

Once again, I will be substituting in the Toffoli gates from the previous section. I will use two forwards and two backwards facing Toffoli gates in that order to better cancel out some components.



Deleting unnecessary gates gives us:



The image to the right shows the various inputs and resulting outputs possible for the circuit above. As expected, the last bits are changed only if the first three bits are 1, and the second last bit remains at whatever arbitrary state it began at. It acts exactly as a C^3X would with an arbitrary $|\phi\rangle$ state wire present.

```
00000 --> 00000 same     10000 --> 10000 same
00001 --> 00001 same     10001 --> 10001 same
00010 --> 00010 same     10010 --> 10010 same
00011 --> 00011 same     10011 --> 10011 same
00100 --> 00100 same     10100 --> 10100 same
00101 --> 00101 same     10101 --> 10101 same
00110 --> 00110 same     10110 --> 10110 same
00111 --> 00111 same     10111 --> 10111 same
01000 --> 01000 same     11000 --> 11000 same
01001 --> 01001 same     11001 --> 11001 same
01010 --> 01010 same     11010 --> 11010 same
01011 --> 01011 same     11011 --> 11011 same
01100 --> 01100 same     11100 --> 11101 different
01101 --> 01101 same     11101 --> 11100 different
01110 --> 01110 same     11110 --> 11111 different
01111 --> 01111 same     11111 --> 11110 different
```

2.4 Results

I was successful in implementing the clean and dirty ancilla builds for the C^3X gate. The clean build contained 15 T, 14 $CNOT$ and 6 Hadamard gates. The dirty build contained 16 T, 18 $CNOT$ and 6 Hadamard gates.

3 Four Controlled X Gate

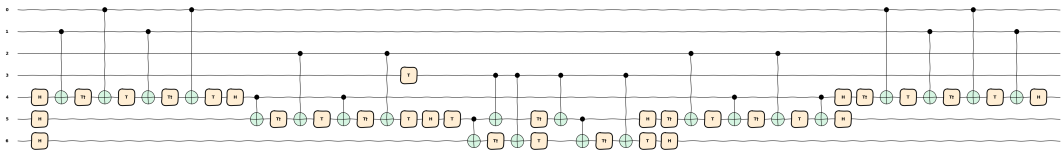
3.1 Discussion

Following on from the last section, the C^4X gate will require four control wires, two ancillae and the target wire, so seven wires overall. There is however another way of implementing these multi-controlled gates with just one ancilla, which I will touch on later. The images are becoming quite unwieldy so I will be cutting down on using them. All of my workings can be found in the various python scripts I will provide alongside this report.

3.2 Two Clean Ancillae Build

The clean C^4X can be built in a similar manner to the clean C^3X, substituting the first and third Toffoli gates with C^3X gates. It is important to note the C^3X gates I will be using are the clean ancilla builds I created in the last section.

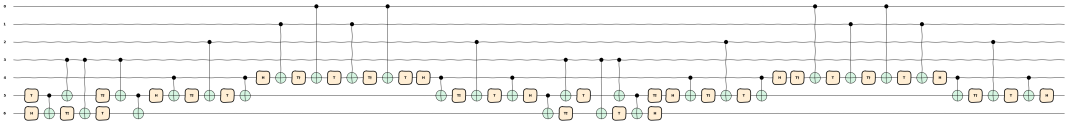
Substituting the respective builds and cancelling gates gives the following circuit:



This build contains 23 T, 22 $CNOT$ and 10 Hadamard gates.

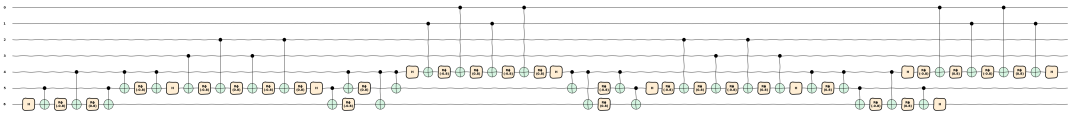
3.3 Two Dirty Ancillae Build

Again, the dirty C^4X can be built in a similar manner to the dirty C^3X, substituting the second and fourth Toffoli gates with dirty C^3X gates. This produces the below circuit:



This build contains 24 T, 30 $CNOT$ and 10 Hadamard gates.

Another method for building it is sandwiching a C^3X with two Toffoli. It produces a worse version with 26 T, 34 $CNOT$ and 10 Hadamard gates, but I will not exclude it just yet.

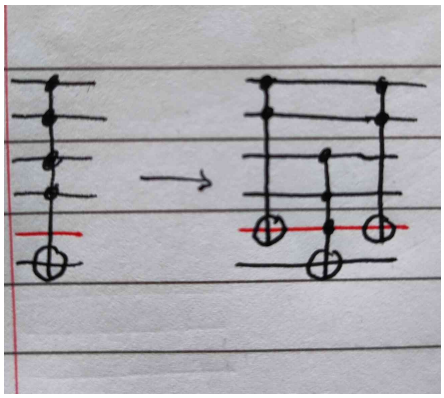


3.4 Single Clean Ancilla Build

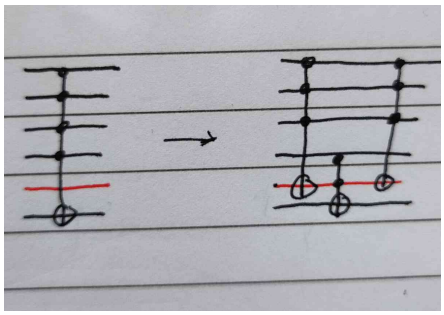
As stated in the Discussion subsection earlier, we can build any C^nX gate with only one ancilla, using the other control wires as what I will refer to as "pseudo-ancillae". These circuits are bulkier and more awkward than the other builds I have discussed thus far, but they allow us to create circuits that use far less wires, which is a requirement for the present day quantum computers.

I was able to come up with four different designs for the single clean ancilla build for the C^4X, which I will display and discuss now.

- The first design uses two Toffoli gates sandwiching a dirty C^3X gate. The C^3X gate requires the use of an ancilla, so I used wire 0 as the pseudo-ancilla.
- The second design utilises the same gates as the first, but the pseudo-ancilla for the C^3X is on wire 1.



- The third design consists of a Toffoli gate sandwitched by two dirty C^3X gates. The pseudo-ancilla for both C^3X gates is wire 3
- The fourth and final design is the same as the third, except the pseudo-ancilla is wire 5, the target wire.



Before diving into the results of these four designs, I will discuss my early intuitions on them. Out of the first two, I believed the second would work the best, as I figured some of the gates could be blocked against cancelling out by the pseudo ancillae on wire 0. I had no idea which of the last two would fare better, as the nature of the pseudo-ancillae were still unknown to me. And finally of the two groups, I reasoned the first two would result in less T gates, as the preliminary T-count for each were 30 vs. 39.

The first two designs were surprisingly (to me anyways) exactly the same. I suppose it shouldn't have been a surprise, as it is just a slight reshuffling of the wires. Both designs from group one used 24 T, 26 $CNOT$ and 10 Hadamard gates.

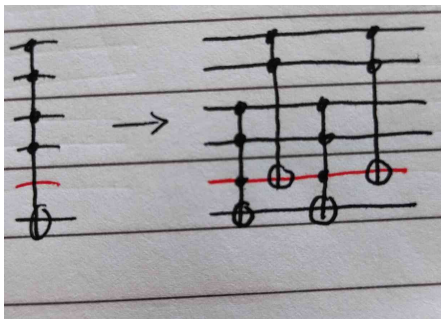
The second group also contained the same amount of T and $CNOT$ gates, being 31 and 34 respectively. However, after having decomposed all these builds to Toffoli gates only, I noticed the last build allowed for two of the Toffoli gates to cancel out. This resulted in the last build having only 10 Hadamard gates, as opposed to the 14 Hadamard gates the third design had. I believe this is due to the pseudo-ancilla being on the target wire itself, which is worth taking note of for further builds.

So my preconceptions of the designs were somewhat correct. I expected the first two to fare better than the last two. Showing this is useful, and I hope to further grow my intuition for larger circuits.

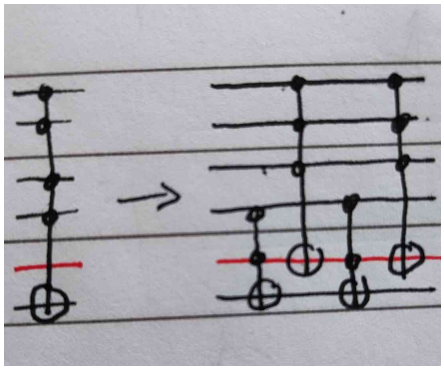
3.5 Single Dirty Ancilla Build

Same as for the clean ancilla builds, I was able to come up with four different designs for the single dirty ancilla build for the C^4X.

- The first design is the same as the group one design for the single clean ancilla build, with a dirty C^3X tacked onto the front. Both C^3X will use wire 0 as the pseudo-ancilla.
- The second design is the same as above, except the pseudo-ancilla is on wire 1



- The third design, again, is the same as group two from the single clean ancilla build section, with another Toffoli attached to the front. The pseudo-ancilla is on wire 3
- The fourth and final design is the same as the third, with the pseudo-ancilla on wire 5, the target wire.



My predictions on these builds are less interesting than those for the clean builds. Preliminary T count for all of these are the same, at 46. I believed the first two would be the same, bar minor positioning differences in the gates. I expected the last one to have two Toffoli gates that can cancel out.

The first two were, as expected, the same bar minor positioning differences. They both contained 36 T, 44 $CNOT$ and 14 Hadamard gates.

The last two were more interesting than I had guessed. I managed to whittle the third one down to only 34 T, 40 $CNOT$ and 14 Hadamard gates. A small decrease, but useful for bigger circuits. The last design ended up at 36 T, 40 $CNOT$ and 12 Hadamard gates. Again, this was achieved through the deletion of two entire Toffoli gates.

3.6 Results

I will list the results of all the builds from this section here for convenience.

Ancillae	Design	T	$CNOT$	Hadamard
2	Clean	23	22	10
2	Dirty1	24	30	10
2	Dirty2	26	34	10
1	Clean 1 & 2	24	26	10
1	Clean 3	31	34	14
1	Clean 4	31	34	10
1	Dirty 1 & 2	36	44	14
1	Dirty 3	32	40	14
1	Dirty 4	36	40	12

You can clearly see the two clean ancillae build is the most optimal. It is followed closely, almost jointly, by the two dirty ancillae build and the first and second design for the single clean ancilla build. At this stage the T-count is not too high and doesn't differ by much between builds, so I would say it is safe to use whatever build you like.

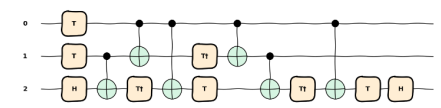
4 Modularity

4.1 Introduction

Throughout my experimenting with these circuits I have noticed a good bit of patterning. I have developed a few "modules" that tend to arise a lot in my builds.

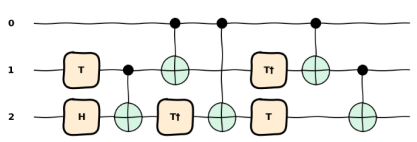
- The fist and most obvious module is the Toffoli itself, which I have dubbed the "**Toff**", for notation's sake.

```
def Toff(wires):
    qml.T(wires[0])
    qml.T(wires[1])
    qml.Hadamard(wires[2])
    qml.CNOT([wires[1],wires[2]])
    qml.CNOT([wires[0],wires[1]])
    qml.adjoint(qml.T)(wires[2])
    qml.CNOT([wires[0],wires[2]])
    qml.adjoint(qml.T)(wires[1])
    qml.T(wires[2])
    qml.CNOT([wires[0],wires[1]])
    qml.CNOT([wires[1],wires[2]])
    qml.adjoint(qml.T)(wires[2])
    qml.CNOT([wires[0],wires[2]])
    qml.T(wires[1])
    qml.Hadamard(wires[2])

qml.draw_mpl(Toff, decimals=1, style="sketch", wire_order=[0,1,2])([0,1,2])
plt.show()
```



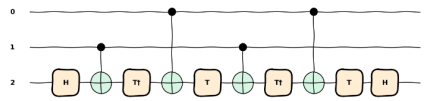
- The second module I have named the "**ToffHalf**". It is a Toff with a few gates taken off the end of it.

```
def ToffHalf(wires):
    qml.T(wires[1])
    qml.Hadamard(wires[2])
    qml.CNOT([wires[1],wires[2]])
    qml.CNOT([wires[0],wires[1]])
    qml.adjoint(qml.T)(wires[2])
    qml.CNOT([wires[0],wires[2]])
    qml.adjoint(qml.T)(wires[1])
    qml.T(wires[2])
    qml.CNOT([wires[0],wires[1]])
    qml.CNOT([wires[1],wires[2]])

qml.draw_mpl(ToffHalf, decimals=1, style="sketch", wire_order=[0,1,2])([0,1,2])
plt.show()
```



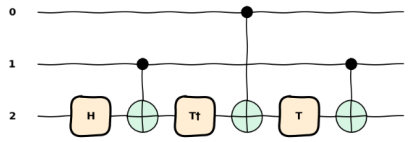
- The third module is a Toff that has the gates not on the target wire removed. I have named it a "Line", due to how it appears.

```
def Line(wires):
    qml.Hadamard(wires[2])
    qml.CNOT([wires[1],wires[2]])
    qml.adjoint(qml.T)(wires[2])
    qml.CNOT([wires[0],wires[2]])
    qml.T(wires[2])
    qml.CNOT([wires[1],wires[2]])
    qml.adjoint(qml.T)(wires[2])
    qml.CNOT([wires[0],wires[2]])
    qml.T(wires[2])
    qml.Hadamard(wires[2])

qml.draw_mpl(Line, decimals=1, style="sketch", wire_order=[0,1,2])([0,1,2])
plt.show()
```



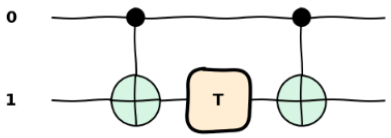
- The fourth module is the "**LineHalf**", as just like the ToffHalf, it is a Line with some of the gates taken from it. You could also see it as the ToffHalf with all the gates bar those on the target wire removed, if you can even call it a target wire anymore.

```
def LineHalf(wires):
    qml.Hadamard(wires[2])
    qml.CNOT([wires[1],wires[2]])
    qml.adjoint(qml.T)(wires[2])
    qml.CNOT([wires[0],wires[2]])
    qml.T(wires[2])
    qml.CNOT([wires[1],wires[2]])

qml.draw_mpl(LineHalf, decimals=1, style="sketch", wire_order=[0,1,2])([0,1,2])
plt.show()
```



- The fifth, final and probably silliest module is the "**Basket**". It is just a T gate sandwiched by two $CNOT$ gates. Note, however that the ToffHalf can be built by combining a LineHalf, one T gate, and a Basket†.

```
def Basket(wires):
    qml.CNOT([wires[0],wires[1]])
    qml.T(wires[1])
    qml.CNOT([wires[0],wires[1]])

qml.draw_mpl(Basket, decimals=1, style="sketch")([0,1])
plt.show()
```

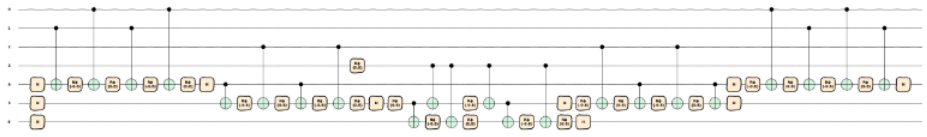


4.2 Discussion

Being able to recognise these five modules has greatly improved my my ability to cancel out gates in my builds. In fact, I have been continuously discovering little tricks relating to these modules throughout this process, and I keep having to double back on myself to further optimise builds I had thought I had completed. I have no doubt there are plenty other methods of optimisation I have not found, and therefore my builds are not the most optimised they could be. But I hope to find as many as I can.

Another upside for these modules is that I can call them at the beginning of my python notebook and refer to them while building my circuits. For instance, I can now create a C^4X gate using three Lines, one Toff and three Line†s, as follows:

```
nwires=7
def C4Xclean(wires):
    Line([wires[0],wires[1],wires[4]])
    Line([wires[2],wires[4],wires[5]])
    Toff([wires[3],wires[5],wires[6]])
    qml.adjoint(Line)([wires[2],wires[4],wires[5]])
    qml.adjoint(Line)([wires[0],wires[1],wires[4]])

compiler(C4Xclean, nwires)
#MatLocator(C4Xclean, nwires)
```



```
defaultdict(<class 'int'>, {'Hadamard': 10, 'CNOT': 22, 'PhaseShift': 23})
```

4.3 Multiple Ancillae Pattern

Speaking of building my circuits with these modules, I believe I have found a pattern when it comes to the multiple ancillae builds.

Any clean C^nX build (as of me completing C^5X, which I will discuss in the next section) can be created with $n-2$ clean ancillae, $n-2$ Lines, one Toff, and $n-2$ Line†s. Counting the individual gates inside each of these modules, I suggest a clean C^nX can be created with $n-2$ clean ancillae, $8n-9$ T gates, $8n-10$ $CNOT$ gates and $4n-6$ Hadamard gates.

Similarly, I found the best build for a dirty C^nX (as of starting C^6X) can be created with $n-2$ dirty ancillae, one ToffHalf, $n-3$ LineHalfs, one Line, $n-3$ LineHalf†s, one ToffHalf†, $n-3$ LineHalfs, one Line† and $n-3$ LineHalf†s. A dirty C^nX can be built with $n-2$ clean ancillae, $8n-8$ T gates, $12n-18$ $CNOT$ gates and $4n-6$ Hadamard gates.

The clean and dirty builds are extremely close gatewise, despite the dirty builds requiring nearly double the amount of Toffoli gates to produce. They differ by only 1 T gate, which is very important due to its expensive nature. They even have the same amounts of Hadamard gates.

This pattern holds thus far for the C^nX, $n = 3, 4, 5$ gates, and I predict it should hold for further values of n.

I hope to figure out a suitable pattern for the single ancilla builds. We will see.

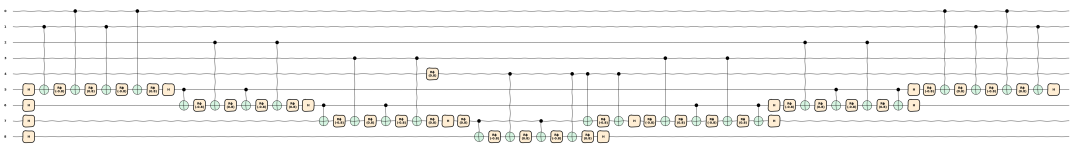
5 Five Controlled X Gate

5.1 Discussion

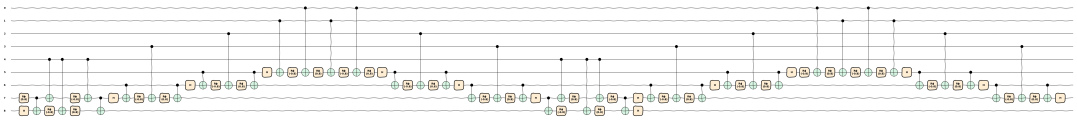
As I have gotten the hang of the multiple ancillae builds I will be discussing them less and just showing my resulting circuitry and gate numbers. The single ancilla builds are going to become more complicated, with many more options to create them opening up.

5.2 Three Ancillae Builds

The clean C^5X can be built with 31 T, 30 $CNOT$ and 14 Hadamard gates.



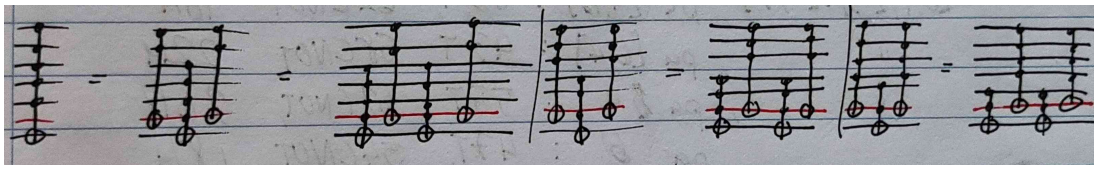
The best dirty C^5X design can be built with 32 T, 42 $CNOT$ and 14 Hadamard gates. There were three others using different versions and orientations of gates, which can be found in the python notebooks accompanying this. They however contained more T gates so I will not describe them in detail here.



5.3 Single Clean Ancilla Builds

The single clean ancilla C^5X can be divided into three main builds. These three builds can be further specialised by altering the combination of pseudo ancillae or types of gates in use. Overall the number of builds possible is around 32. Thankfully, similar to C^4X, some of the possible builds are just slight reshufflings of control wires that result in the same number of constituent gates in the end. I will be grouping these builds together for brevity's sake.

The form of the main builds for a single clean (or dirty) ancilla are as follows:



The first build for the group one consisted of two Toffolis and a multiple dirty ancilla C^4X. The pseudo-ancillae for the C^4X were wires 0 and 1. The ordering didn't matter for this particular one. Using the better version of the dirty C^4X gate from the 4 Controlled X Gate section used only 32 T, 38 $CNOT$ and 14 Hadamard gates.

The next builds used single dirty ancilla C^4X gates. Using wires 0 or 1 as a pseudo-ancilla gave the same result. The best version was produced using the third single dirty ancilla design. It used 42 T, 48 $CNOT$ and 18 Hadamard gates.

Group two was built using three C^3X gates. The first six of these were the same. The first and third C^3X gates used wires 3 or 4 as their pseudo-ancilla, while the middle C^3X could use wires 0, 1 or 2 as its pseudo-ancilla. These all gave builds consisting of 40 T, 46 $CNOT$ and 18 Hadamard gates.

The last three designs for group 2 all had the same outcome. They all had wire 6 as the first and third C^3X pseudo-ancilla. If the middle C^3X had its pseudo ancilla on wire 0, 1 or 2, the build had 40 T, 46 $CNOT$ and 14 Hadamard gates. Again, we see placing a pseudo-ancilla onto the target wire gives interesting results, and allows cancellations of elemental Toffoli gates.

Group three builds were interesting too for the fact that pseudo-ancillae sat on or near the target wire. The first iteration had two multiple dirty ancilla C^4X gates and one Toffoli. The C^4X's pseudo-ancillae were on wires 4 and 6, in that order. This allowed for the cancellation of four elemental Toffoli gates, giving us 47 T, 54 $CNOT$ and 18 Hadamard gates. If the pseudo-ancillae order was flipped, the result was 47 T, 58 $CNOT$ and 22 Hadamard gates.

The next iteration used single dirty ancilla C^4X gates. Again, I tested each of the three different designs from the C^4X section, and found the best one. If the pseudo-ancilla was on wire 4, I found

the fourth design to be the most efficient, using 55 T, 62 $CNOT$ and 26 Hadamard gates. With the pseudo-ancilla on wire 6, the third design was the better option once again, using 47 T, 54 $CNOT$ and 18 Hadamard gates.

I am not surprised much by the results above. The best version had the least preliminary T count. I will tabulate my results after the discussion of the dirty builds below.

5.4 Single Dirty Ancilla Builds

The single dirty ancilla C^5X follows much the same pattern as the clean versions, just with a single extra main gate adhered to the front. The groups are pictured in the above subsection. I expect a lot more uniformity in this group of gates, thanks to the similar preliminary T counts and higher cancellation potential.

The first iteration of the first group used multiple dirty ancilla C^4X, and had pseudo-ancillae on wires 0 and 1, in either order. This gave 52 T, 68 $CNOT$ and 22 Hadamard gates.

Next we used the different single dirty ancilla designs again. The pseudo-ancilla could be on either 0 or 1, it didn't matter. Design 4 gave the best build, with 60 T, 72 $CNOT$ and 28 Hadamard gates.

The second group, similar to the clean builds, had six iterations that didn't differ in gate number. Those were the ones that used any of wires 0, 1 or 2 for the first and third C^3X gate pseudo ancilla, and either wires 3 or 4 as pseudo-ancilla for the other two C^3X. These produced 52 T, 64 $CNOT$ and 22 Hadamard gates.

Now for the three group 2 builds that differed. They all allowed for a cancellation of two Toffoli gates, and had the second and last C^3X pseudo-ancilla on wire 6. If the other pseudo-ancilla was on wire 0,1 or 2, we got 52 T, 64 $CNOT$ and 20 Hadamard gates.

The first iteration of group 3 used multiple dirty ancilla C^4X, and had pseudo-ancillae on wires 4 and 6, in that order. This gave four Toffoli deletions, 52 T, 60 $CNOT$ and 20 Hadamard gates. If the order was flipped, there were no Toffoli deletions, but we still got 52 T, 64 $CNOT$ and 24 Hadamard gates.

Using the single dirty ancilla designs, the fourth design was best for pseudo-ancilla on wire 4. It contained 58 T, 68 $CNOT$ and 26 Hadamard gates. With the pseudo-ancilla on wire 6, the third design came out on top again, with 52 T, 60 $CNOT$ and 20 Hadamard gates.

Overall, we seem to have a bit of an upset. The best build, with the least amount of T, $CNOT$ and Hadamard gates, is the group three, design 3, single dirty ancilla C^4X, pseudo-ancilla on wire 6 iteration (quite the mouthful, I may need to come up with a better naming convention). This design had a preliminary T count of 82, while there were builds there with preliminary T counts of 62.

5.5 Results

I will list the results of all the builds here.

Ancillae	Design	Pseudo-A Location	T	$CNOT$	Hadamard
3	Clean		31	30	14
3	Dirty1		32	42	14
3	Dirty2		44	54	18
3	Dirty3		34	46	14
3	Dirty4		44	56	14
1	Clean 1	$[0,1] \vee [1,0]$	32	38	14
1	Clean 1	$[0 \vee 1]$	42	48	18
1	Clean 2	$[3 \vee 4], [0 \vee 1 \vee 2]$	40	46	18
1	Clean 2	$[6], [0 \vee 1 \vee 2]$	40	46	14
1	Clean 3	$[4,6]$	47	54	18
1	Clean 3	$[6,4]$	47	58	22
1	Clean 3	$[4]$	55	62	26
1	Clean 3	$[6]$	47	54	18
1	Dirty 1	$[0,1] \vee [1,0]$	52	68	22
1	Dirty 1	$[0 \vee 1]$	60	72	28
1	Dirty 2	$[3 \vee 4], [0 \vee 1 \vee 2]$	52	64	22
1	Dirty 2	$[6], [0 \vee 1 \vee 2]$	52	64	20
1	Dirty 3	$[4,6]$	52	60	20
1	Dirty 3	$[6,4]$	52	64	24
1	Dirty 3	$[4]$	58	68	26
1	Dirty 3	$[6]$	52	60	20

As much as it pains me, I am going to have to try intuit which builds to ignore from now on, as the amount of possible single builds is growing rapidly. For instance, the preliminary T count has a large effect on the final product when dealing with the clean single ancilla.

6 Six Controlled X Gate

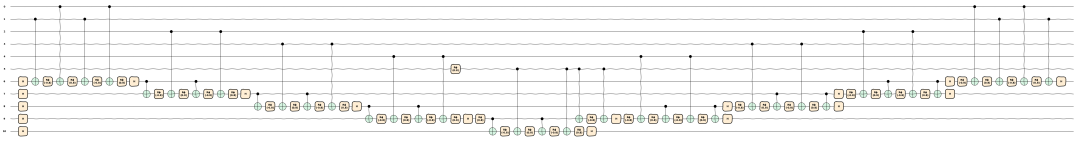
6.1 Discussion

The multiple ancillae clean and dirty builds for the C^6X gates are be much the same as the other multiple ancilla builds, using four ancillae this time. This brings us up to eleven wires for these builds, which my computer does not enjoy simulating in python. This is no surprise, as the matrix these circuits produce are 2048·2048 in size.

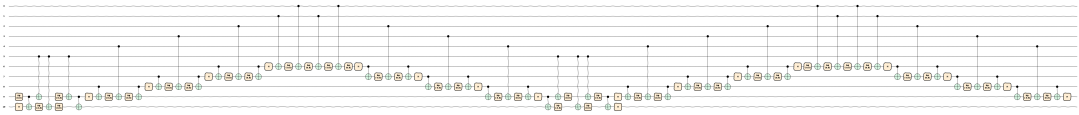
The single ancilla builds are becoming far more numerous, however I believe I have come up with a method that allows me to pick and choose which ones I should bother building. Unfortunately it has come very close to the end of this studentship, so I haven't had much of a chance of fully investigating this method's workings. I will discuss it further before the single clean ancilla builds subsection.

6.2 Four Ancillae Builds

The clean C^6X can be built with 39 T, 38 $CNOT$ and 18 Hadamard gates.



The best dirty C^6X design uses 40 T, 54 $CNOT$ and 18 Hadamard gates.



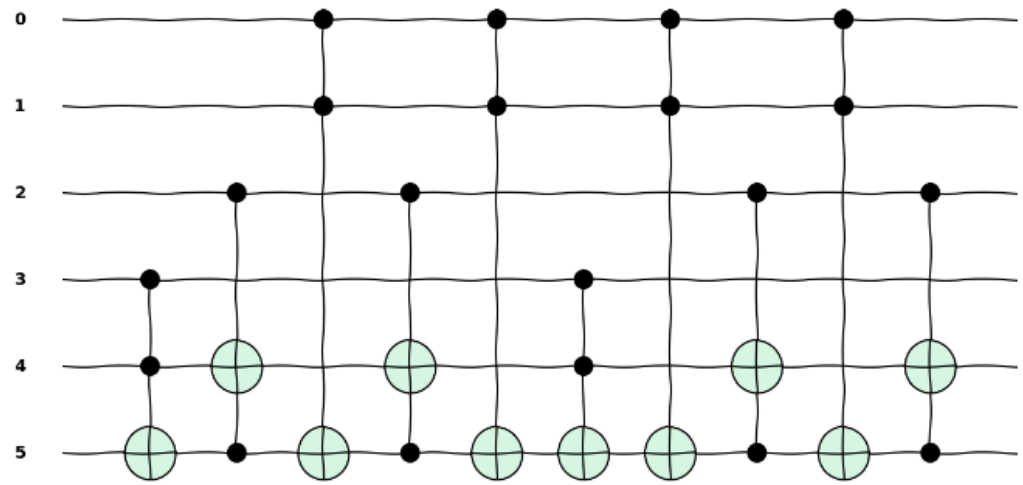
6.3 Lower-Bounds

Throughout my investigations of all of the different builds I can create, I have noticed a couple things I haven't yet mentioned. Firstly, there are sections of my aforementioned "modules" that are more likely to be cancelled out than others. When used in a build, a Toff is likely to cancel down into a Line, and a ToffHalf likely cancels to a LineHalf. Secondly, individual modules are sometimes able to be cancelled out themselves. This depends on the orientations of the constituent gates and the ancillae used. The best way to investigate if modules can be gotten rid of is to look at a gate's elemental Toffoli build.

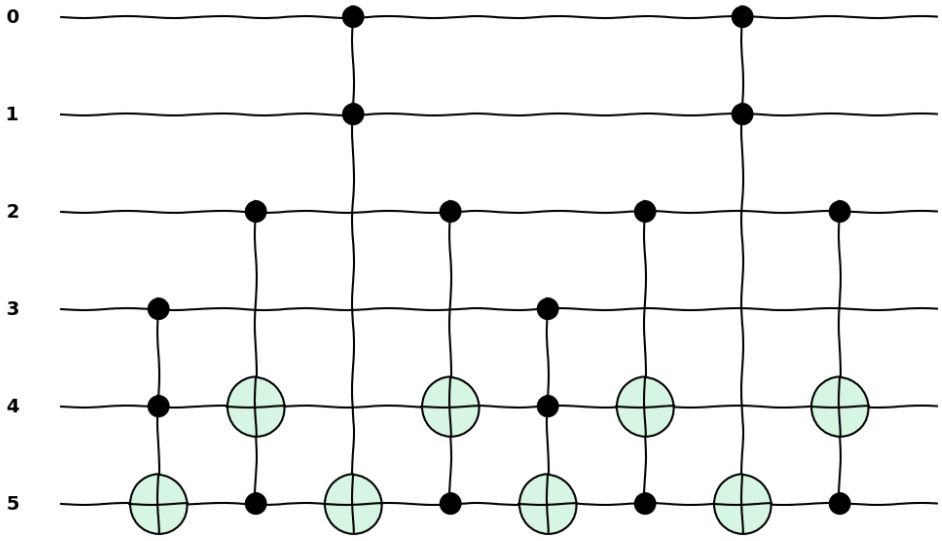
These two points allow me to calculate a sort of "lower-bound" for each gate created up to this point, and for any further gates. The way I do this is to build the gates using just Toffolis, seeing how many can be gotten rid of, and then creating the new gate, excluding the modules that matched up with the Toffolis I removed. I will demonstrate using one of the single dirty ancilla builds for C^4X:

It is possible to build a single dirty C^4X using two dirty C^3X and two Toffoli gates. The C^3X gates are controlled by wires 0, 1 and 2, acts on wire 4, and has wire 5 as its pseudo ancilla. The Toffolis are controlled by wires 3 and 4, and act on wire 5. Below is a code representation of this information, which produces the elemental Toffoli build for the C^4X gate.

```
#3:2, 5 pa
nwires=6
def C4X1dirty4(wires):
    qml.Toffoli([wires[3],wires[4],wires[5]])
    C3Xdirty([wires[0],wires[1],wires[2], wires[5], wires[4]])
    qml.Toffoli([wires[3],wires[4],wires[5]])
    qml.adjoint(C3Xdirty)([wires[0],wires[1],wires[2], wires[5], wires[4]])

compiler(C4X1dirty4, nwires)
```



The fifth and seventh Toffoli gates in the image above can cancel out with one another, leaving the following build:



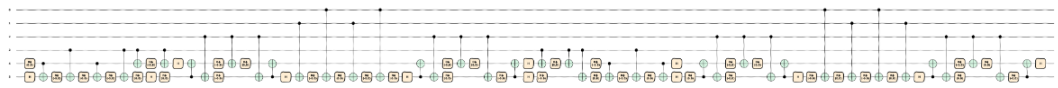
The Toffoli deletions is the same as deleting the last module that makes up the C^3X, giving us:

```
def C3Xdirty(wires):
    ToffHalf([wires[2],wires[3],wires[4]])
    Line([wires[0],wires[1],wires[3]])
    qml.adjoint(ToffHalf)([wires[2],wires[3],wires[4]])
    qml.adjoint(Line)([wires[0],wires[1],wires[3]])

def C3Xdirtyedit(wires):
    ToffHalf([wires[2],wires[3],wires[4]])
    Line([wires[0],wires[1],wires[3]])
    qml.adjoint(ToffHalf)([wires[2],wires[3],wires[4]])
```

And to prove that all of this is true, I created the C^4X using the old and altered version of the C^3X, and they both came out to be the same:

```
nwires=6
pa=5
def tester1(wires):
    for i in wires:
        qml.Identity(i)
    Toff([wires[3],wires[4],wires[5]])
    C3Xdirty([wires[0],wires[1],wires[2],wires[pa],wires[4]])
    qml.adjoint(Toff)([wires[3],wires[4],wires[5]])
    qml.adjoint(C3Xdirty)([wires[0],wires[1],wires[2],wires[pa],wires[4]])

def tester2(wires):
    for i in wires:
        qml.Identity(i)
    Toff([wires[3],wires[4],wires[5]])
    C3Xdirtyedit([wires[0],wires[1],wires[2],wires[pa],wires[4]])
    qml.adjoint(Toff)([wires[3],wires[4],wires[5]])
    qml.adjoint(C3Xdirtyedit)([wires[0],wires[1],wires[2],wires[pa],wires[4]])

print(CheckMat(tester1, tester2, nwires))
compiler(tester2, nwires)
```

```
True
```



```
defaultdict(<class 'int'>, {'PhaseShift': 36, 'Hadamard': 12, 'CNOT': 40})
```

Now to calculate the "lower-bound" of this C^4X gate. This gate is built using four ToffHalfs, two Lines and two Toffs. Using this gate in later builds could offer up the opportunity of cancelling all the Toffs to Lines and ToffHalfs to LineHalfs, meaning the lowest T-count this particular gate could be cancelled to is 24.

There is a slight issue I have found with this method; I have come across a few Toffoli-builds that allow some of the elemental Toffolis to be cancelled out, and I don't understand why. Again, I have come close to the end of my studentship, so I don't think I have the time to investigate this further, but I believe if I looked into the logic behind this I might find clues as to why this happens. Instead, the Toffoli-builds are small enough that I can find the cancellable Toffolis by trial and error. I know this method is not sustainable however, and I will investigate the logic further if I get the chance to.

Going back through all my builds up to now (for the hundredth time), I was able to find the lower-bound for each, and improve some of my builds along the way. I will create a table below of all the gates that are relevant for building other higher control gates:

Gate	Design	Actual T-count	Lower-Bound
C^2X	Tofoli	7	4
C^3X	Clean	15	12
	Dirty	16	12
C^4X	Clean	23	20
	Dirty	24	20
	1Dirty1	36	32
	1Dirty3	32	28
	1Dirty4	36	24
C^5X	Clean	31	28
	Dirty	32	28
	1Dirty1	52	48
	1Dirty2	52	48
	1Dirty3	52	40
	1Dirty4	52	40
	1Dirty5	52	48
	1Dirty6	52	40
	1Dirty7	52	48

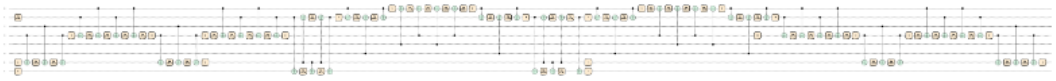
Note that there are far more possible C^5X builds than I have listed here. I have omitted the builds that had lower-bounds greater than the lowest actual T-count 52, as I see no point in continuing to use them. There may be an argument for continuing their use, as it is possible to cancel out some of their elemental Toffolis, but this happens so rarely I feel like I am safe to omit them. Perhaps if I figure out the underlying logic behind the Toffoli cancellations I can consider them again, but only time will tell.

6.4 Single Clean Ancilla Builds

It is relatively simple to deduce the most effective build for the single clean ancilla C^6X. A single clean ancilla C^nX gate is always constructed using two identical C^mX gates sandwiching a C^pX gate, where $n > m, p$, and $m + p = n + 1$. The C^pX gate is often completely unchanged, as it has no pair to cancel with. Therefore to find the best build we need to look at what can and cannot change.

The first group of builds I looked at were those created with dirty C^3X and C^4X gates. The C^4X with the least T-count that we can use here is the multiple dirty ancilla build, containing 24 T gates. Therefore sandwiching this gate with two C^3X and cancelling gives the following circuit:

```
nwires=8
def C3Xdirtyedit(wires):
    LineHalf([wires[2],wires[3],wires[4]])
    Line([wires[0],wires[1],wires[3]])
    qml.adjoint(LineHalf)([wires[2],wires[3],wires[4]])
    qml.adjoint(Line)([wires[0],wires[1],wires[3]])

def C6X1clean1(wires):
    C3Xdirtyedit([wires[0],wires[1],wires[2], wires[3], wires[6]])
    C4Xdirty([wires[3],wires[4],wires[5],wires[6], wires[0],wires[1], wires[7]])
    qml.adjoint(C3Xdirtyedit)([wires[0],wires[1],wires[2], wires[3], wires[6]])

compiler(C6X1clean1, nwires)
```



```
defaultdict(<class 'int'>, {'Hadamard': 22, 'CNOT': 58, 'PhaseShift': 48})
```

Using any other C^4X gate here will result in an increased T-count, as again the C^4X is unchangeable. As such there is no need to investigate this particular combination further.

Switching to looking at two C^4X gates sandwiching a C^3X gate now. The only thing that can change is the C^4X gates. Even with choosing the pseudo-ancillae such that the C^4X gates experience elemental Toffoli annihilation, the lower-bound for all possible builds are higher than 48, and therefore won't be much use to us.

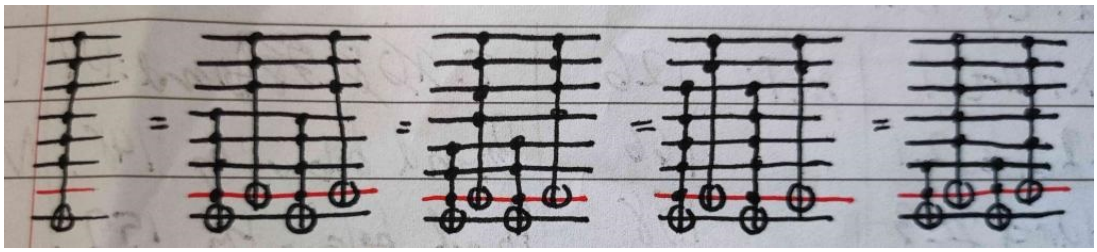
The builds created using C^5X and C^2X gates tell a similar story. I cannot use the three dirty ancilla C^5X gate, as there aren't enough pseudo-ancillae accommodate it. Therefore I must use single ancilla C^5X gates, all of which have a T-count of 52. This means a C^2X, C^5X, C^2X build is pointless to look at, as its lower bound will always be 60. Perhaps if I looked into creating C^5X gates that require only two ancillae I could get better results, but that would require me straying off topic once again and I don't have much time left.

The C^5X, C^2X, C^5X builds on the other hand don't offer enough Toffoli cancellations to achieve a lower-bound less than 48. As such, this concludes my look into the single clean ancilla C^6X builds.

6.5 Single Dirty Ancilla Builds

I will use the lower-bounds method for finding the best single dirty ancilla C^6X gates too. I expect many of these different possible builds will be far too large to even bother considering, so I will focus a lot of my attention on the amount of elemental Toffoli deletions possible. For each pair of Toffoli deletions there is a max decrease of the lower-bound by 8. Therefore the lower-bound of a design will be $\geq \Sigma$(lower-bounds of constituent gates) $- 8 \cdot$ (elemental Toffoli pair deletions). If this lower-bound estimate is less than the lowest T-count for a design I have already found, I will investigate this design further.

I will use the following builds:



Starting with the first design above, we could place the pseudo-ancilla for the dirty C^4X on wires 0, 1 or 2. Placing the pseudo-ancilla for the C^3X on wires 3, 4 or 5 had the same effect, while placing it on wire 7 often allowed for more elemental Toffoli deletions. I will tabulate the results below:

C^4X design	C^3X pseudo-ancilla	Toff pair deletions	Lower-Bound estimate	Lower-Bound	T-count
C^4Xdirty	3,4 or 5	0	64	64	68
C^4Xdirty	7	1	56	56	68
C^4X1dirty1	3,4 or 5	2	72		
C^4X1dirty1	7	3	64	76	
C^4X1dirty3	3,4 or 5	2	64	68	76
C^4X1dirty3	7	3	56	60	76
C^4X1dirty4	3,4 or 5	0	72		
C^4X1dirty4	7	1	64	64	88

Looking at the second design, we can place the C^3X pseudo-ancilla on any of the wires 0, 1, 2 or 3 without any change in the outcome. The outcome is changed however depending on the choice of pseudo-ancillae for the C^4X gate.

C^4X design	C^4X pseudo-ancilla	Toff pair deletions	Lower-Bound estimate	Lower-Bound	T-count
C^4Xdirty	4 and 5	0	64	64	68
C^4Xdirty	4 and 7	1	56	56	68
C^4X1dirty1	4	0	88		
C^4X1dirty1	7	2	72		
C^4X1dirty3	4	1	72		
C^4X1dirty3	7	4	48	56	68
C^4X1dirty4	4	0	72		
C^4X1dirty4	7	2	56	64	84

The third design must use the single dirty ancilla build for C^5X, as there aren't enough possible pseudo-ancillae for the multiple ancilla build. Once again a two-ancilla build for the C^5X could do better, but time is running short. Pseudo-ancilla choice does not matter for this design.

C^5X design	Toff pair deletions	Lower-Bound estimate	Lower-Bound	T-count
C^5X1dirty1	2	88		
C^5X1dirty2	3	80		
C^5X1dirty3	2	72		
C^5X1dirty4	2	72		
C^5X1dirty5	4	72		
C^5X1dirty6	0	88		
C^5X1dirty7	0	104		

The fourth and final design will depend on pseudo-ancilla location. I don't expect these to be much better than the third design.

C^5X design	C^5X pseudo-ancilla	Toff pair deletions	Lower-Bound estimate	Lower-Bound	T-count
C^5X1dirty1	5	1	96		
C^5X1dirty1	7	5	64	80	
C^5X1dirty2	5	1	96		
C^5X1dirty2	7	5	64	72	
C^5X1dirty3	5	0	88		
C^5X1dirty3	7	2	72		
C^5X1dirty4	5	1	80		
C^5X1dirty4	7	3	64	72	
C^5X1dirty5	5	2	88		
C^5X1dirty5	7	6	56	80	
C^5X1dirty6	5	0	88		
C^5X1dirty6	7	3	64	72	
C^5X1dirty7	5	0	104		
C^5X1dirty7	7	3	80		

The lowest T-count I found for the single dirty ancilla C^6X is 68, and they are all in the first and second designs. The third and fourth designs all had lower-bounds far too large to consider completing.

7 Final Remarks

This project has been very interesting and I had a lot of fun exploring the intricacies of quantum computing. I have learned a huge amount of techniques and skills for working with quantum circuitry.

If I were to continue this project I would investigate the optimisation of elemental Toffoli builds, in an effort to improve the designs and to depend on my previous designs much less. I believe my leaning on previous gate designs may have limited me a lot.

A lot of my time was spent on revising all my previous builds once I discovered a new optimisation technique, which became quite the headache.

Moving on to the C^7X gates could have proven difficult. The multiple ancilla C^6X gates were created with 11 wires, resulting in matricies of size $2048 \cdot 2048$. My laptop was struggling to compute these matricies, so a C^7X gate requiring 13 wires, and therefore a $8192 \cdot 8192$ matrix may have set my laptop on fire.

As mentioned in the Modularity section, I was able to find a pattern for the T-count of my multiple ancilla builds. A C^nX could be created with $n-2$ clean ancillae, and $8n-9$ T gates, or with $n-2$ dirty ancillae, and $8n-8$ T gates. This is not as efficient as other approaches that can work with $4n-6$ T gates, however they used techniques I would require further reading and experimentation in python to understand. I was unable to create enough single ancilla builds to find a suitable pattern for them. Additionally I fear the builds I did create may not be the most efficient possible, and so may not have shown a pattern even if I had a sufficient number. I would need to investigate the elemental Toffoli builds further to ensure I have found the most efficient builds possible.

As mentioned throughout this document, I have many python scripts that show my workings. I also have a copy in which I did the majority of my working out by hand, however all the relevant information is found within this document.

Finally, I would like to thank Dr. Mark Howard for providing the opportunity to complete this studentship. He has provided me with a great amount of materials and guidance with which I was able to further my understanding of quantum computing.