A general implementation of Shor's algorithm with two methods in IBMQ

Youjie, Li

Department of Electrical Engineering, National Taiwan University

b06901147@ntu.edu.tw

July 10, 2020

Abstract

Shor's algorithm[1] is a polynomial time quantum algorithm for integer factorization. As the security of RSA system based on the difficulties for factorization, Shor's algorithm gives a hope to break the system more efficiently than classical computer. Given a integer N, denote $n = logN$, we could construct two circuits for Shor's algorithm using $4n + 2$ and $2n + 3$[2] qubits respectively. In this report, I would start from adder, to modular adder, and then build the multiplier and $C - U_a$ where $U|x\rangle = |axmodN\rangle$. All code used in this paper could be found at my github https://github.com/alfa871212/shor_paper.

1 Introduction

In Shor's algortihm, we randomly choose a that is coprime with N, and try to find the order r *where* $a^r = 1\ mod\ N$.As the equivalence of factorization and order finding, if we obtain the order, then we could have a more probable guess for the factors of N. Here we will give the complete algorithm for factorizing N.

1. If N is even, return the factor 2.
2. Classically determine whether N is perfect power(p^q), if so p must be a factor of N.
3. Randomly choose a integer a, *which* $gcd(a, N) = 1$ *and* $1 < a < N$
4. Use the order finding quantum circuit to find the order r. In fig.1, we have two different methods of order finding circuit.
5. If r is odd, go back to step(3). Otherwise, compute $gcd(a^{\frac{r}{2}} - 1, N)$ *and* $gcd(a^{\frac{r}{2}} + 1, N)$ to find non-trivial factors of N.

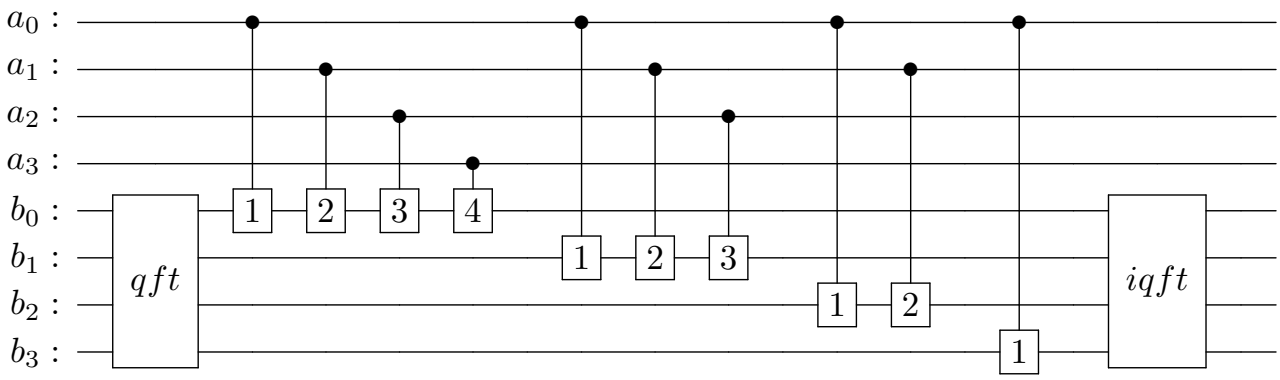
2 Circuits

In this section, I would follow the order of [2] to implement some elementary gates to realize Shor's algorithm.

2.1 Adder gate(ϕ ADD)

As the quantum addition[3] takes place in Fourier space, if we want to add the number a and b in n-bits, we should apply Quantum Fourier Transform on b *denote as* $\phi(b)$. Then expand a in binary representation to create proper phase shift gate to implement an adder. The circuit below is the demonstration of 4-bit number addition, where the gate with number k implement controlled phase shift gate.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} \tag{1}$$



In general, we should preserve additional one qubit to prevent overflow. The following is the result of my adder. For first experiment, I add 1+2=3 in 2-bit number, because the overflow doesn't occur, I can do this addition without adding the ancilla bit. While if I want to add 1+3=4 which exceeds the upper limit of 2-bit number(11=4 in decimal), overflow should occur. Thus, we need to add addition qubit to prevent overflow.

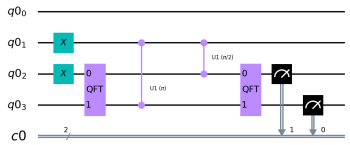


Figure 1: Adder with 1+2

==========

Executing adder with a=1, b=2, n=2...
Job Status: job has successfully run
Expect ans = 3, Measure res = 3
Result correct! Adder success!

==========

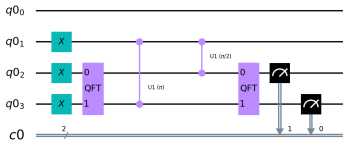


Figure 2: Adder with 1+3 (w/ overflow)

==========

Executing adder with a=1, b=3, n=2...
Job Status: job has successfully run
Expect ans = 4, Measure res = 0
Result wrong! Adder failed!
Overflow occurs!

==========

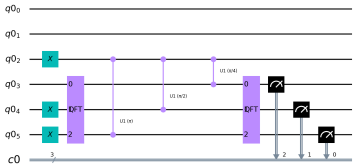


Figure 3: Adder with 1+3 (w/o overflow)

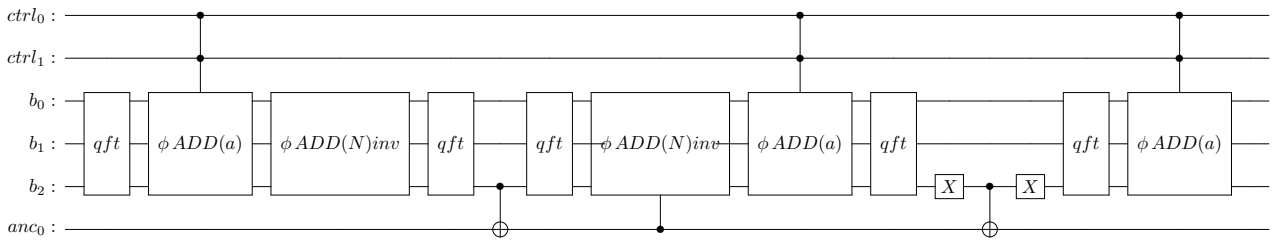
==========

Executing adder with a=1, b=3, n=3...
Job Status: job has successfully run
Expect ans = 4, Measure res = 4
Result correct! Adder success!

==========

In the following paper, we will introduce a notation $\phi ADD(a)$ to represent an adder gate with addition of number a, where ϕ denotes that the addition occurs in the Fourier space, so QFT application is required.

2.2 Modular adder($cc\phi ADDMOD$)



Now that we have a ϕADD(a) gate, we can use it to build a modular adder gate . For future use, two control qubits are included in the circuit. The ancilla bit needs to be preserve in state 0 in order to concatenate every modular adder. The derivation is clear in [2], so I would focus on the implementation. Noted that the cx gates and x gates are used to preserve the above property of circuit.

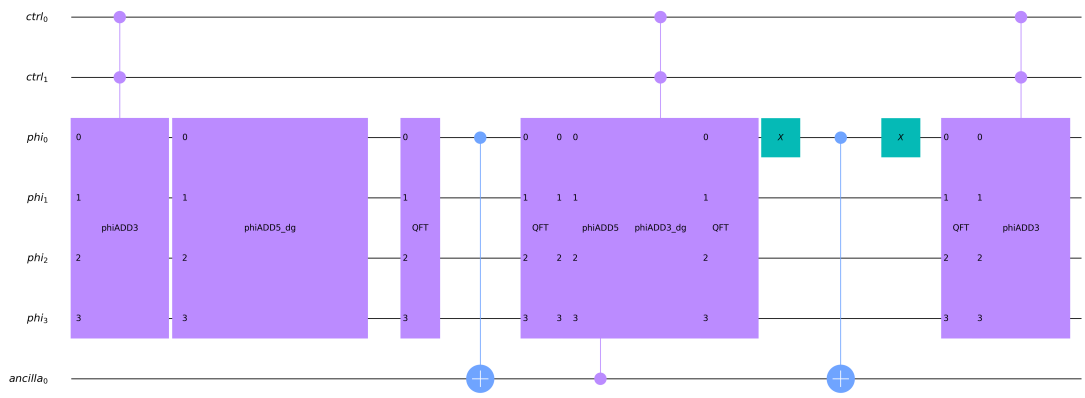


Figure 4: ccphiADDMOD composition

The experiment circuit should be 5 with a=3 b=3 N=5.

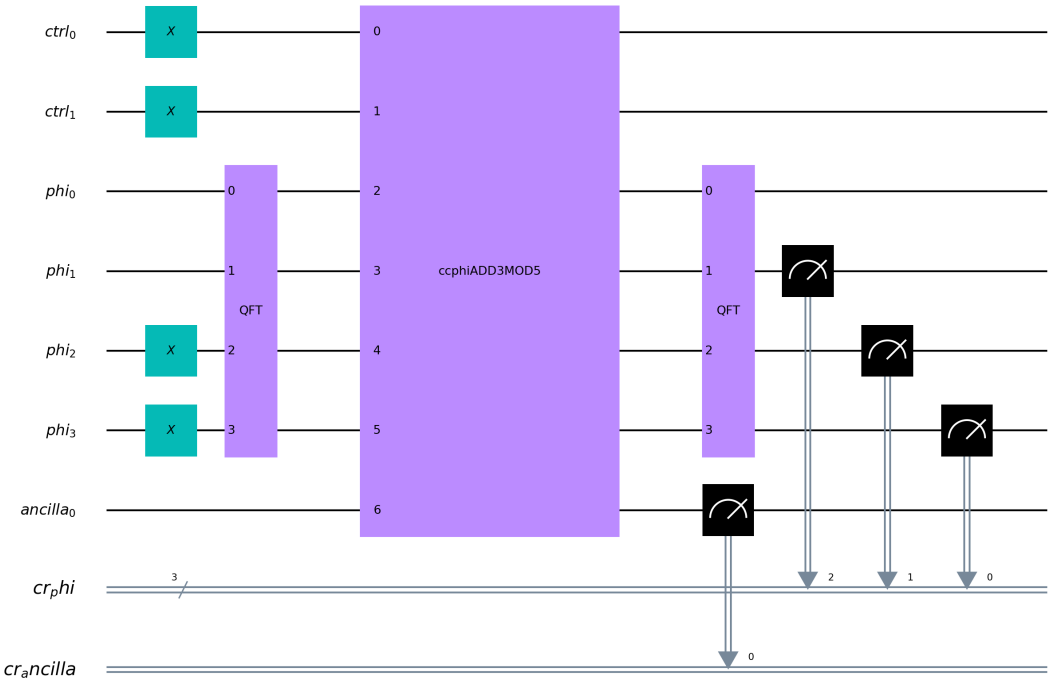


Figure 5: ccphiMOD experiment with a=3, b=3, N=5

---

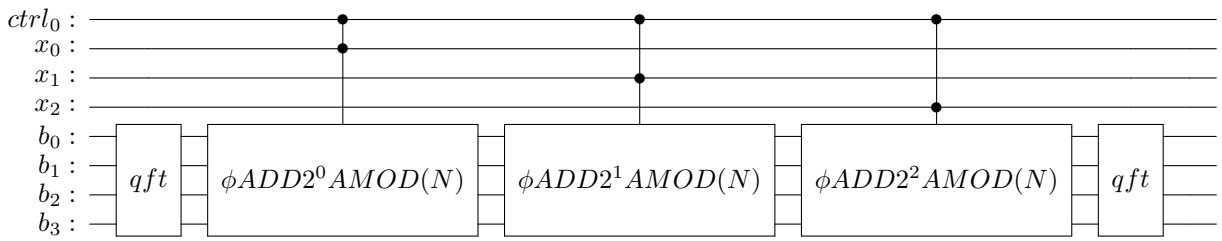
a=3, b=3, N=5, (a+b)mod N=1 (expect ans)
Job Status: job has successfully run
Ancilla bit correct!
The expect result is 3+3 mod 5 = 1
The measurement result is 0 001=1
Measure = Expect, Correct!

Some problems should be mentioned during the implementation. Numbers a and b should be both less than N, if they are equal to N, then they must be set to N. Next, my definition of bit order is to name the upper bit most significant bit, while the [2] isn't. It is important because the value we read

would be influenced.

2.3 Multiplier(CMULT)

In multiplier, our goal is to create $b + axmodN$, where ax perform multiplication. Therefore, we have three registers c, x, b, c is control register, and x, b are target registers. The schematic circuit should be the following figure.



Using the idendity, $(ax)modN = (...((2^0ax_0)modN + 2^1ax_1)modN + ... + 2^{n-1}ax_{n-1})modN$. We could construct multiplier by successive doubly controlled modular adder gates. The input should be $|c\rangle\,|x\rangle\,|b\rangle$ and the output will be$|c\rangle\,|x\rangle\,|b + axmodN\rangle$.

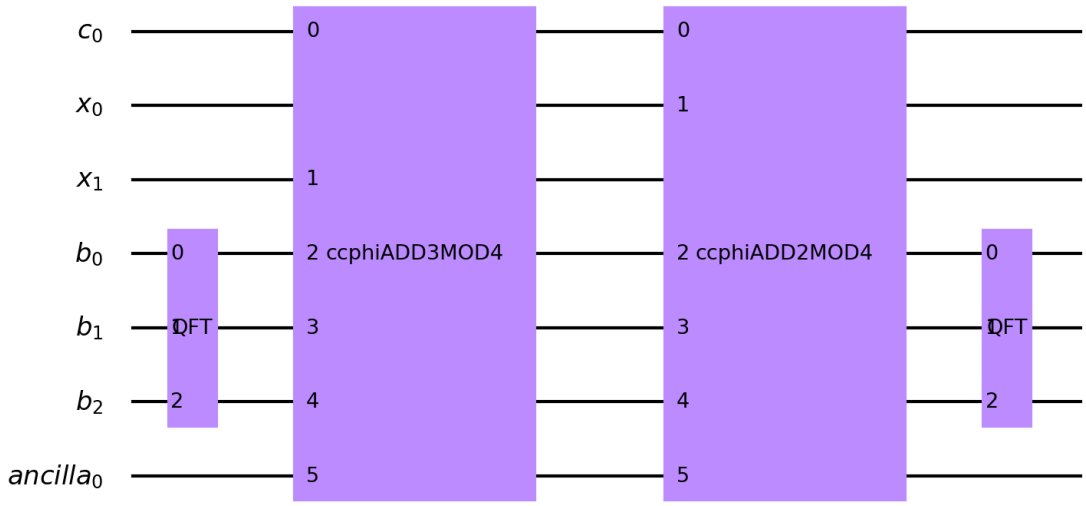


Figure 6: Qiskit implementation for multiplier

The following experiment is based on $x = 3, b = 5, a = 3, N = 4$. Noted that the actual gate we want is multiplier and modular, we would not get the actual multiplication, instead, we get the result after modular operation. Thus, when $b = 5$ exceeds N, we will do the modular first, which could effectively prevent the overflow. In 7, we initialize the register b with number $1 = 5mod4$.

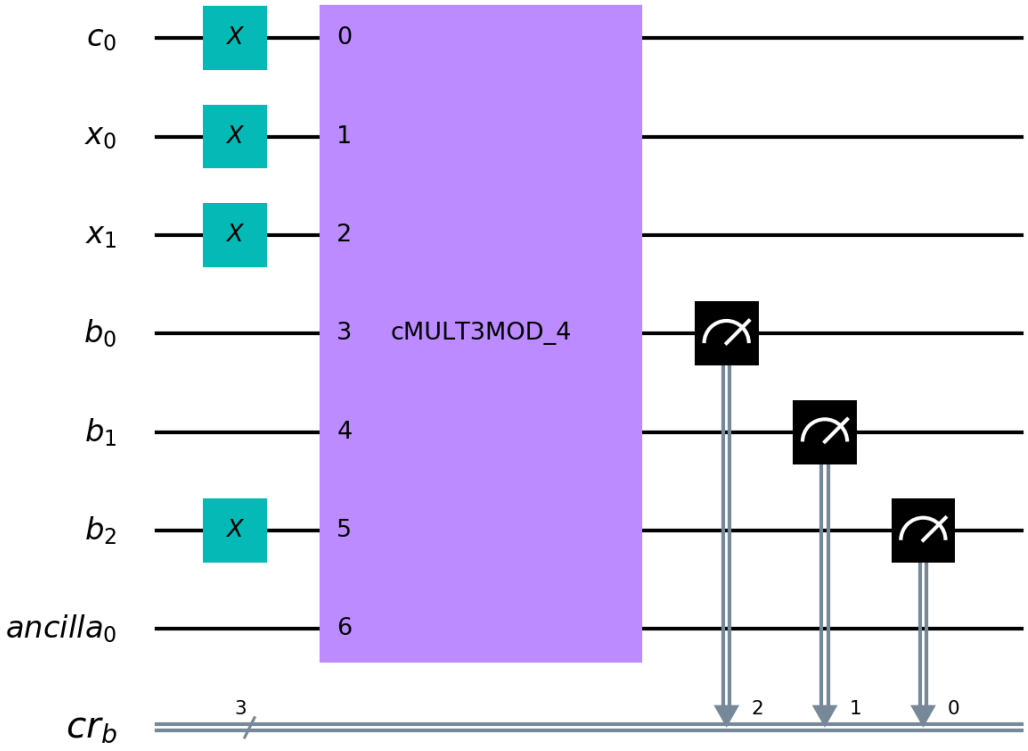
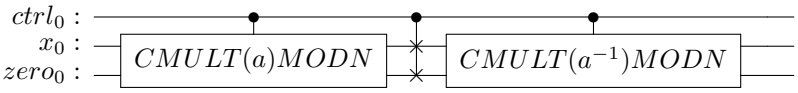


Figure 7: Multiplier Experiment of $x = 3, b = 5, a = 3, N = 4$
Job Status: job has successfully run
The x remain the same! Correct!
Ancilla bit correct!
x=3, b=5, a=3, N=4, b+ax mod N=2
Expect = Measure = 2
Multiplier correct!

2.4 Controlled Unitary gate(CU_a)



We finally move on to the target gate constructing Shor's algorithm. In CU_a, we would implement a gate whose input is $|c\rangle |x\rangle |0\rangle$ and output is $|c\rangle |axmodN\rangle |0\rangle$. From the result state of multiplier, we know that the unchanging register is the final register x. But to construct Shor's algorithm with convenience, I would swap the order of the last two register. By setting $b = 0$, we could easily obtain the result state $axmodN$ which is exactly we want in the algorithm. Noted that the last register should keep in state 0 due to the future concatenation.The a^{-1} in the gate is the modular inverse of number a. It means that $aa^{-1} = 1modN$.

Here is the result of the follow experiment.
x=3,a=5,N=4,ax mod N = 3
Job Status: job has successfully run
Expect = Measure = 3
CU correct!

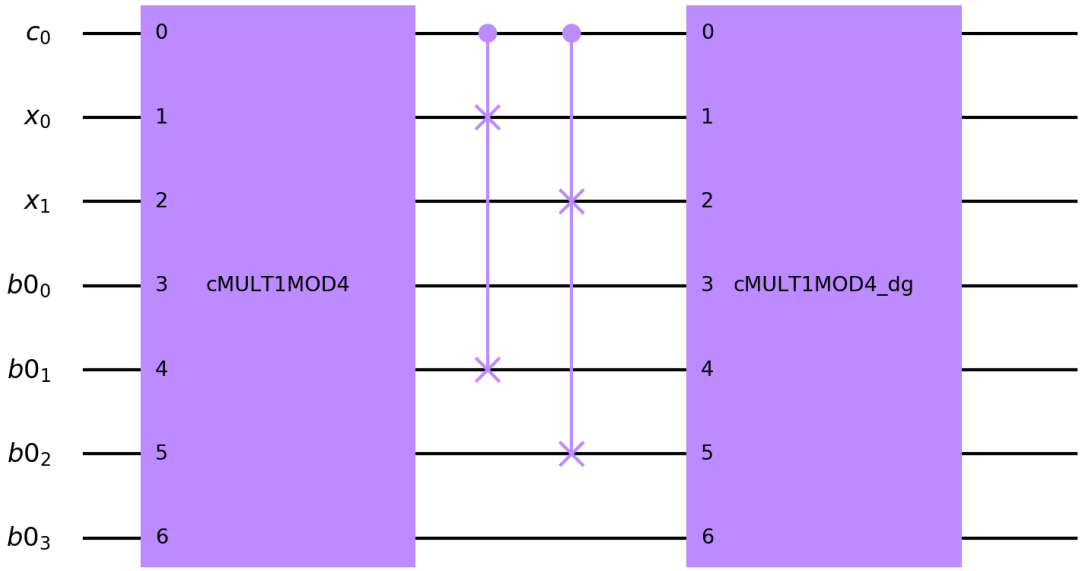


Figure 8: Qiskit implementation for CU_a

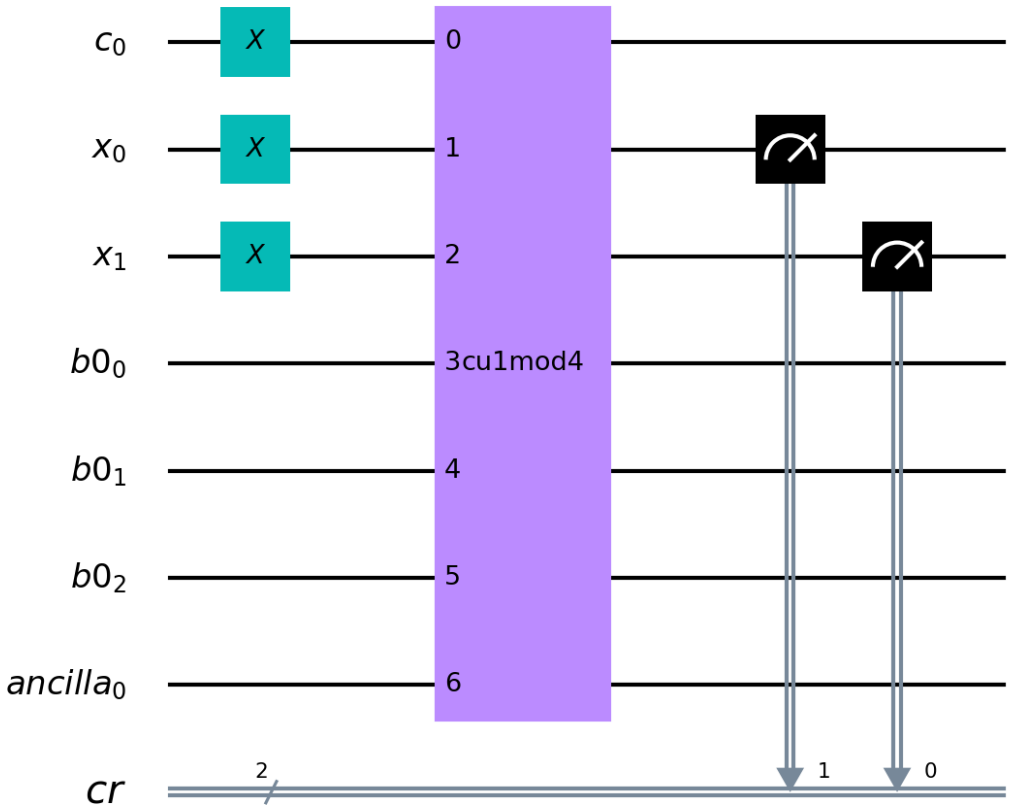
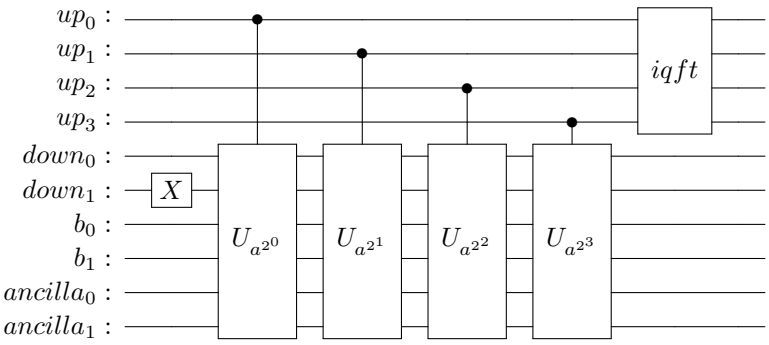


Figure 9: CU_a experiment with $x = 3, a = 5, N = 4$

3 Normal Shor's algorithm

In the normal form of Shor's algorithm, we need $2n+3$ qubits to implement(the figure in [2] was wrong).



In my experiment, I use classical algorithm to find such a that has smaller multiplicative order. By doing this preprocess, we can focus on whether the function of my circuit could work, though in real situation, we should randomly choose a which is coprime with N. The circuit is too big, so the figure is attached at the end of paper.

We could use continuous fraction algorithm(CF) to approximate our result divide 2^n in order to obtain possible multiplicative order.

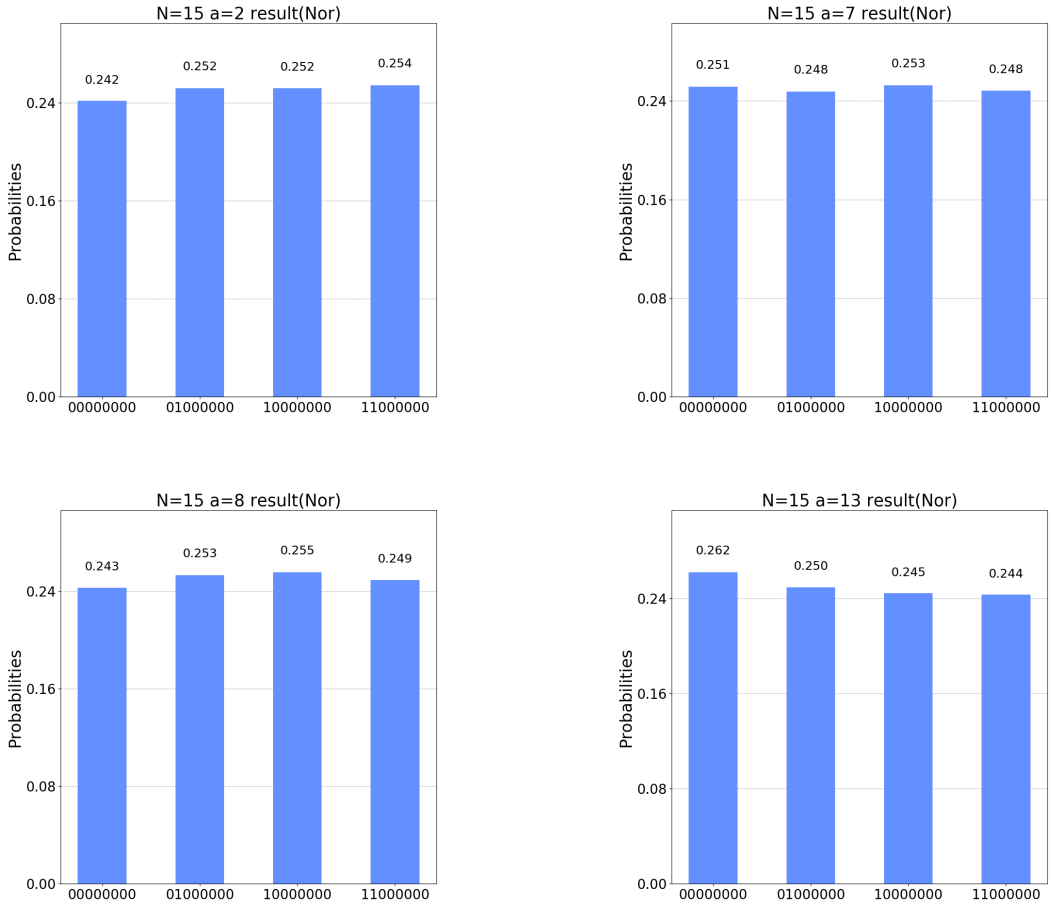


Figure 10: Result of normal Shor's algorithm with four kinds of possibility

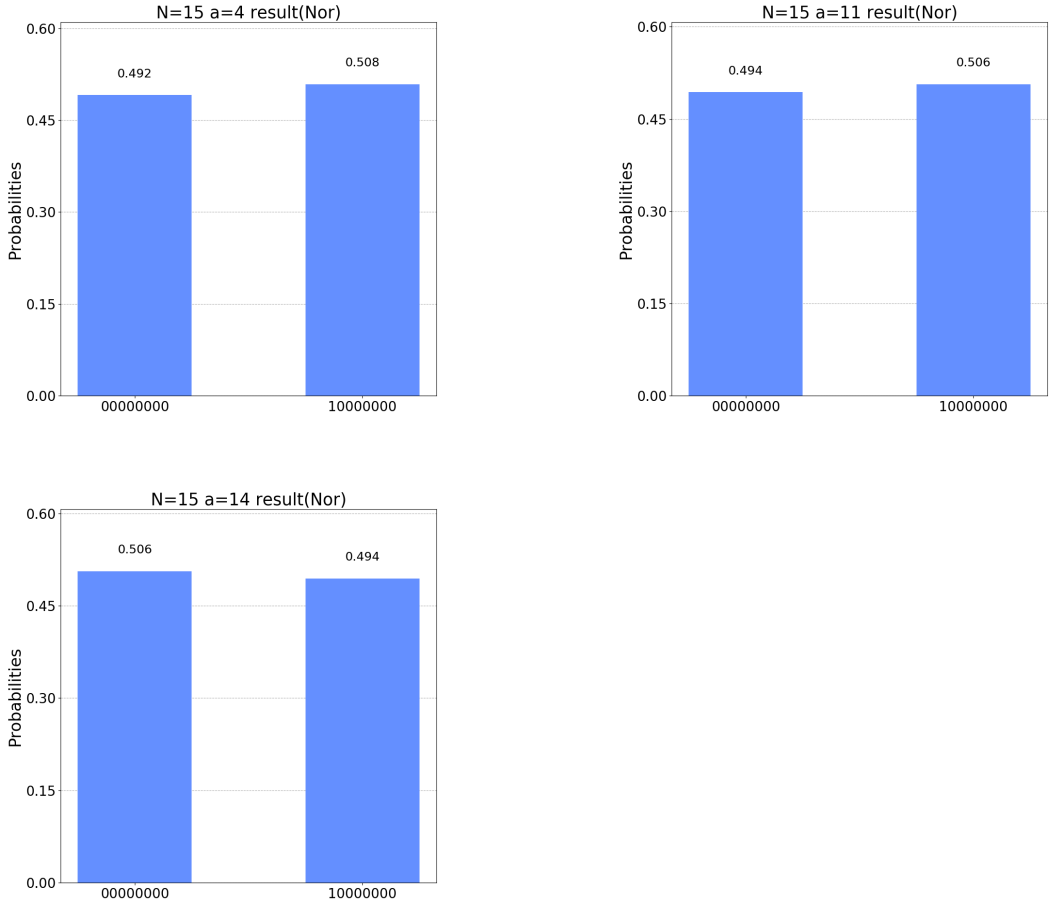
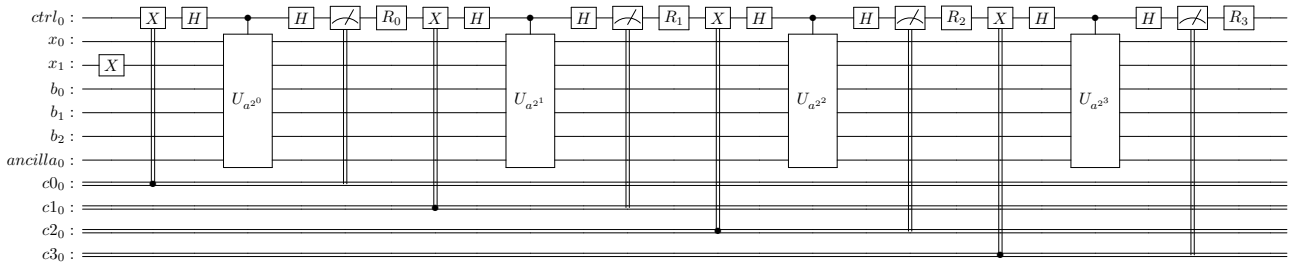


Figure 11: Result of normal Shor's algorithm with two kinds of possibility

4 Sequential Shor's algorithm



4.1 Semi-classical QFT

QFT could be implemented in sequential manner, that is, semi-classical[4] QFT should rely on previous classical measurement. Using the circuit and gate in [5], we create a R_j gate whose matrix representation

$$\begin{bmatrix} 1 & 0 \\ 0 & \phi_j^{'} \end{bmatrix} \tag{2}$$

where

$$\phi_j^{'} = e^{-2\pi i \Sigma \frac{m_{j-k}}{2^k}} \tag{3}$$

which means that this phase shift gate is different from the normal QFT one. The angle should be determined by previous measurement. This semi-classical modification preserves the probabilities of all measurement results. Taking this further we need only insist on one control qubit and the remaining qubits as we can recycle the control qubit after each measurement.

4.2 2n+3 qubits

In this part, the result is similar to the normal circuit, which is matched our expectation. Therefore, the semi-classical QFT is appropriate to apply in Shor's algorithm.

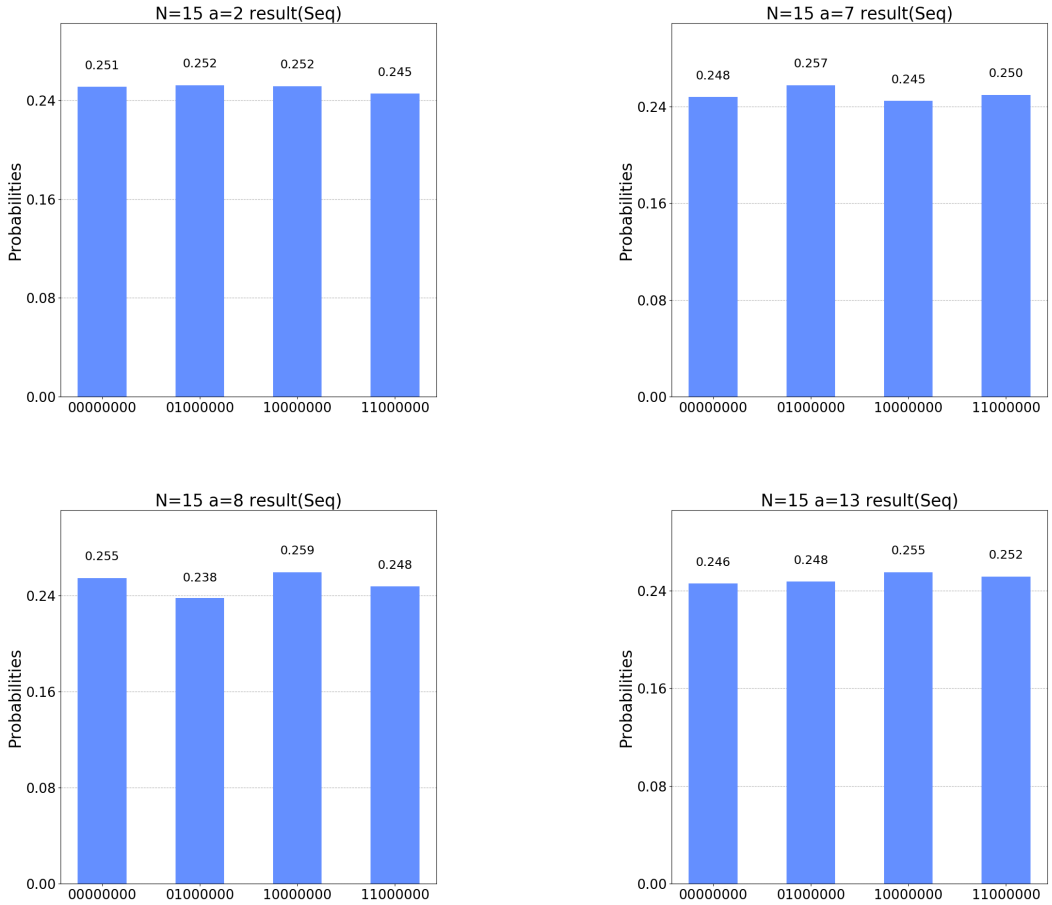


Figure 12: Result of sequential Shor's algorithm with four kinds of possibility

4.3 Factorizing result

Using the module provided by *sympy.ntheory*, the CF algorithm could be implemented easily.

measurement result	possible order	2	4	7	8	11	13	14
11000000	4	success	✗	success	success	✗	success	✗
01000000	4	success	✗	success	success	✗	success	✗
10000000	2	success	success	success	success	success	success	fail
00000000	✗	fail	fail	fail	fail	fail	fail	fail

5 Discussion

A detailed complexity derived in [2], as we know the number of qubit could be reduce if we apply semiclassical QFT, the advantage should appear when N is larger. However, other point should be focused on are error and experiment execution. The wider the circuit is(more qubit), the depth should somehow reduce too. So in the real device, sequential method should accumulate more error. The recent simulation on the IBMQ devices still not support post-measurement result recycle to next level of operations, which means that we could build the semi-classical QFT with real experiment. Unfortunately,

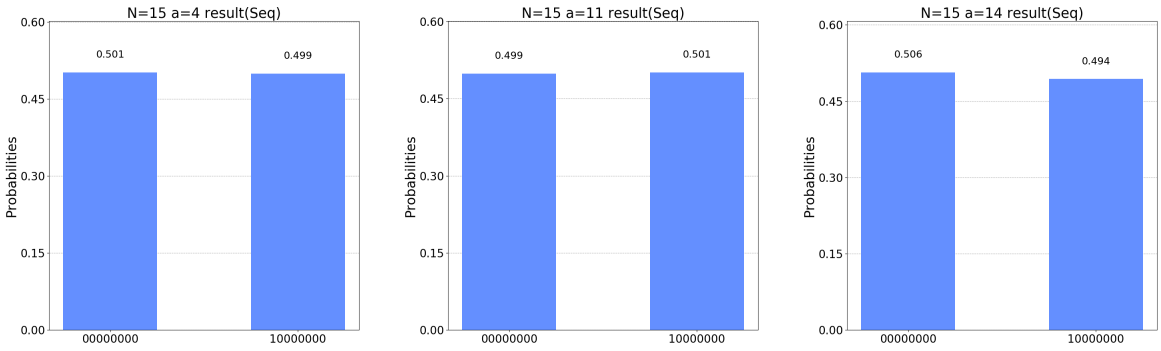


Figure 13: Result of sequential Shor's algorithm with two kinds of possibility

the normal Shor's algorithm circuit also can't run on real device. I infer the number of gate operations is too big, so IBM doesn't allow the experiment. I only run some test with adder on real device, but the result is far away from the result we expected. The mission now maybe should focus on developing more powerful while flawless algorithm, or we should concentrate on enhancing gate error.

References

[1] Shor, Peter W. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." SIAM review 41.2 (1999): 303-332.

[2] Beauregard, Stephane. "Circuit for Shor's algorithm using 2n+ 3 qubits." arXiv preprint quant-ph/0205095 (2002).

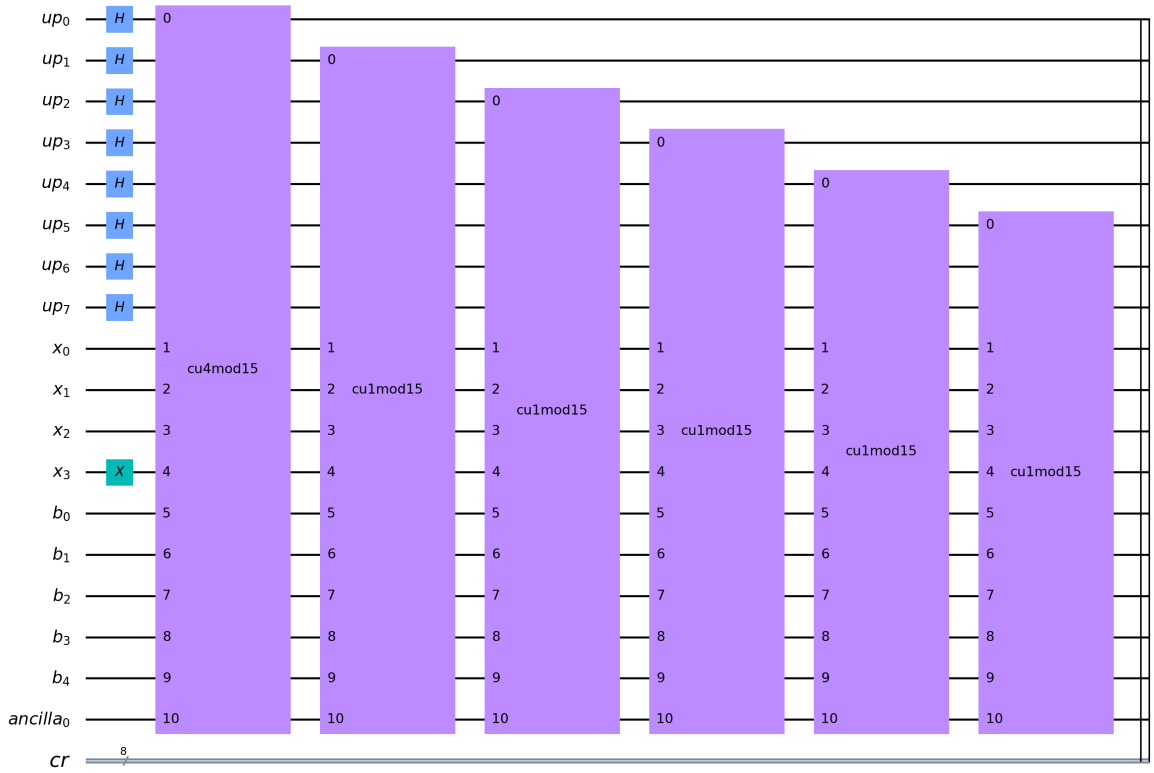
[3] Draper, Thomas G. "Addition on a quantum computer." arXiv preprint quant-ph/0008033 (2000).

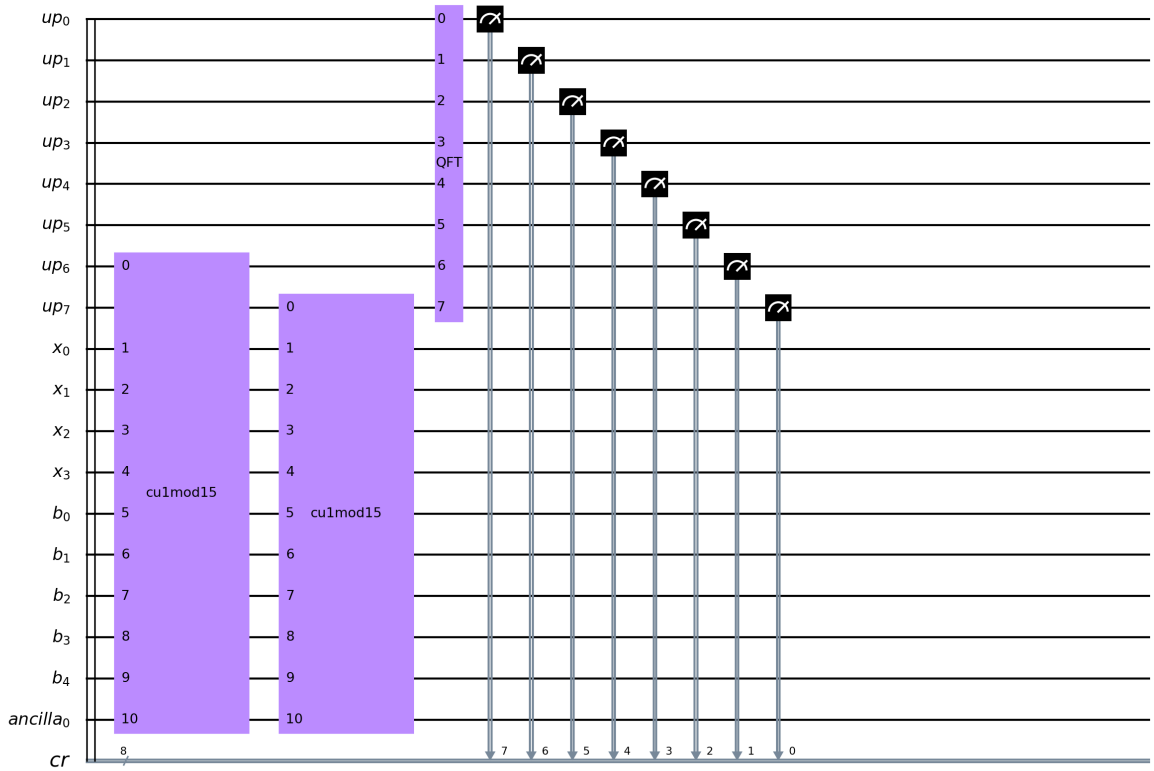
[4] Griffiths, Robert B., and Chi-Sheng Niu. "Semiclassical Fourier transform for quantum computation." Physical Review Letters 76.17 (1996): 3228.

[5] Parker, S., and Martin B. Plenio. "Efficient factorization with a single pure qubit and log N mixed qubits." Physical review letters 85.14 (2000): 3049.

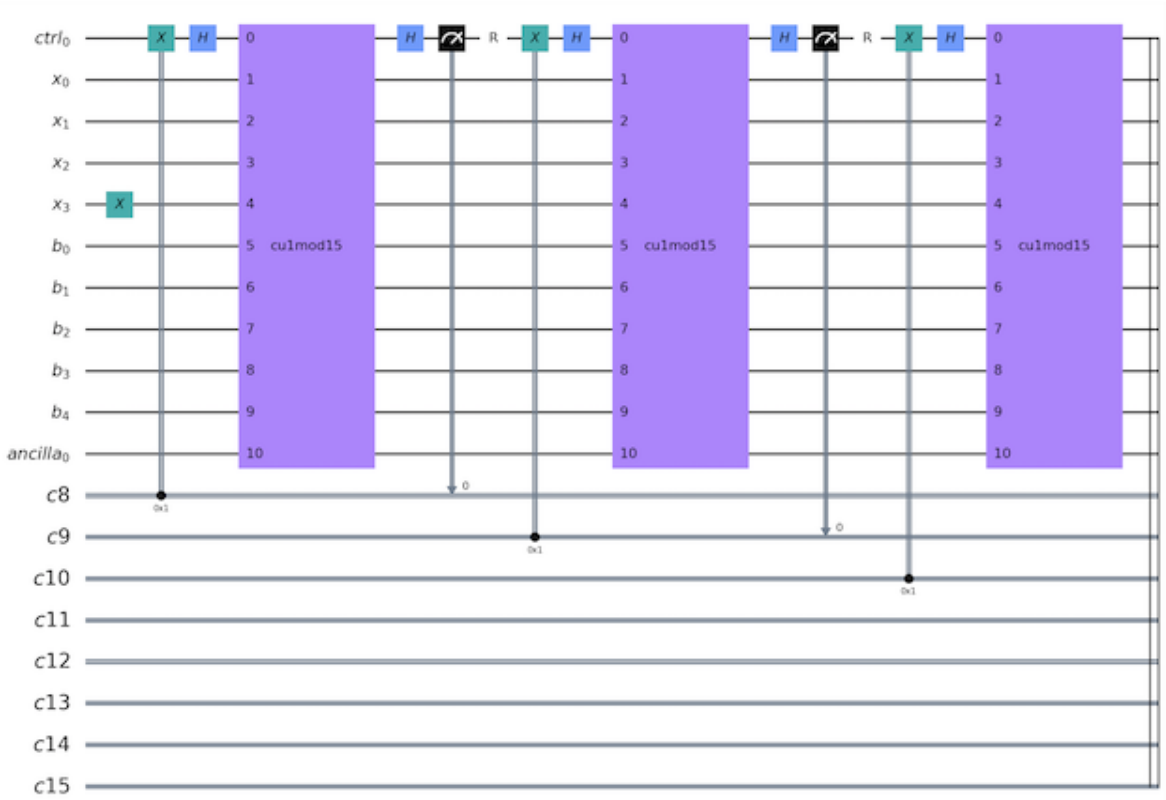
6 Appendix: Qiskit circuits for two methods

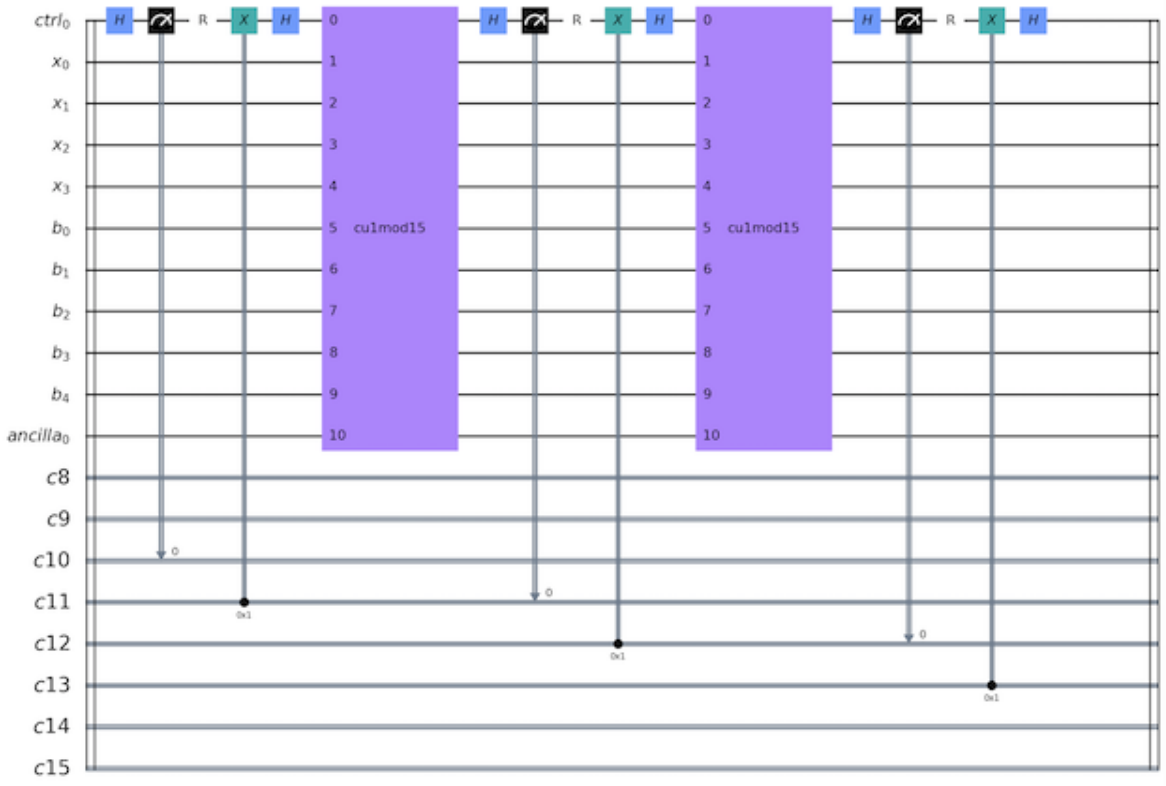
6.1 Normal circuit

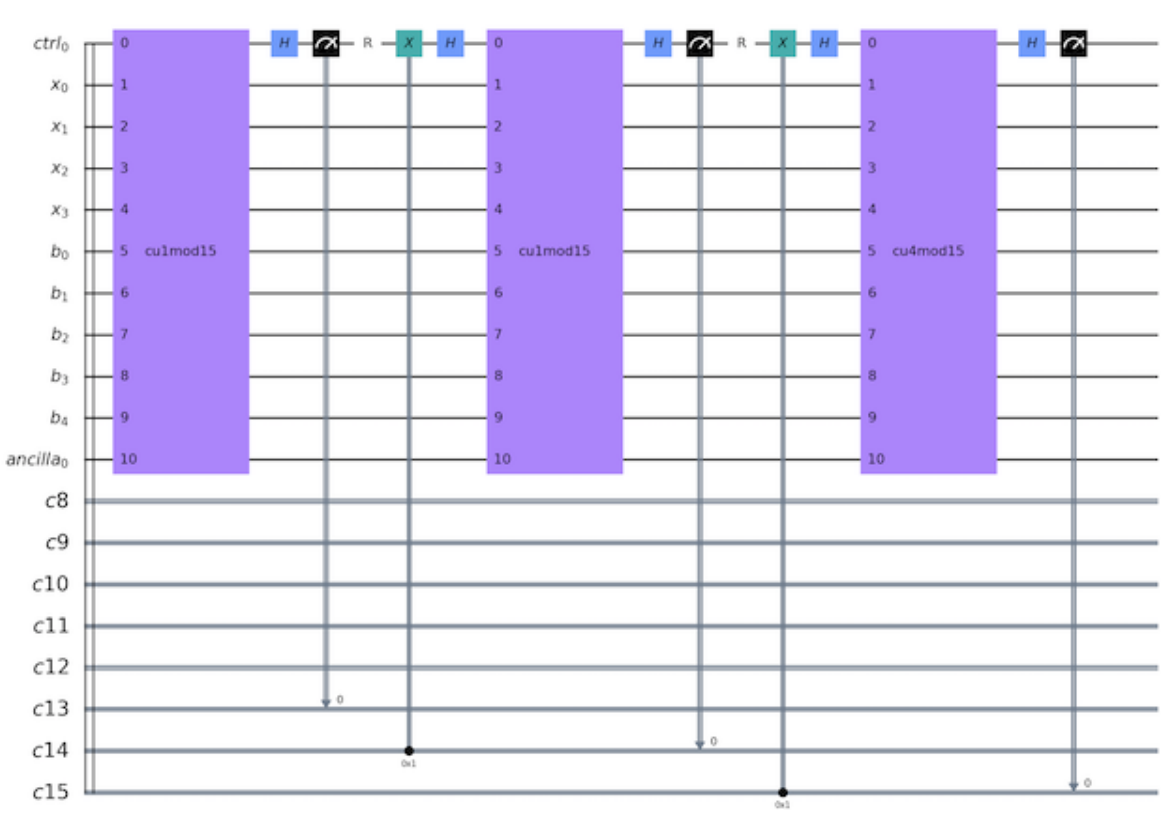




6.2 Sequential circuit


ctrl0
x0
x1
x2
x3
b0
b1
b2
b3
b4
ancilla0
c8
c9
c10
c11
c12
c13
c14
c15
X
H
R
culmod15
0
1
2
3
4
5
6
7
8
9
10
0x1


ctrl0
x0
x1
x2
x3
b0
b1
b2
b3
b4
ancilla0
c8
c9
c10
c11
c12
c13
c14
c15
H
R
X
culmod15
0
1
2
3
4
5
6
7
8
9
10
0x1


ctrl0
x0
x1
x2
x3
b0
b1
b2
b3
b4
ancilla0
c8
c9
c10
c11
c12
c13
c14
c15
cu1mod15
cu1mod15
cu4mod15
H
R
X