Implementation of Shor’s Algorithm by Q# －Usage of IBM’s Quantum Computer－

Q#によるショアのアルゴリズムの実装
－IBM の量子コンピュータを使う－

万里川則亮*　　平田隆幸**

Implementation of Shor's Algorithm by Q#
－Usage of IBM's Quantum Computer－

Noriaki MARIKAWA* and Takayuki HIRATA**

(Received January 21, 2022)

Quantum computing is a scientific topic of people beyond computer researchers` community. Some companies offer service of usage of quantum computer to the people who want to use quantum computer. Various quantum computers are now working, which are open use for public users via online. Now we stand at the dawn of the utilization of quantum computers. There are some computer languages for quantum computing. We report the usage of Q# that was developed by IBM, and the implementation of Shor's algorithm on IBM's cloud computing.

Key Words **:** Quantum Computer, Q#, Shor's Algorithm, IBM's Cloud Computing.

1. はじめに

量子コンピュータは，計算機科学において新しいブレークスルーとなるものとして期待されている．また，科学者だけでなく多くの一般の人々の注目をも集めている[1]-[4]. 量子コンピュータは，量子力学が基礎となっているため，工学部の学生には敷居が高そうに思われているにもかかわらず，アンケート調査を実施したところ，工学部の学生は量子コンピュータに強い興味をもっているが分かった[5]. 現在，量子コンピュータの研究は，ハードウェア研究開発[6]のみならず，ソフトウェアの開発さらにアプリケーション開発がおこなわれるようになってきている[7].

さて，量子コンピュータの特徴を生かして発展が期待される分野を考えてみよう．量子コンピュータの計算は，超並列処理に特徴がある．この特徴を生かすことによって，現在利用されている機械学習や深層学習などにかかる処理時間を大幅に短縮できる可能性がある．というのは，量子コンピュータは複数の可能性を同時並行で探索できるため，超並列処理が可能となるからである．具体的な問題として，古典的なコンピュータでは，巡回セールスマン問題などの問題を解くのに莫大な計算時間を必要とした．それに対し，量子コンピュータは，超並列処理が可能であるため，巡回セールスマン問題などの NP 完全問題を短時間で解くことが期待されている．

現在，量子コンピュータの実用的な利用が現実的になってきたことにともない，どのように効率的なソフトウェアを開発するかに問題が移ってきた．同時に，量子コンピュータを利用し問題を効率良く計算を行うためには，量子計算のためのアルゴリズムを新しく作らなければならない．そこでは，どのようなアルゴリズムが信頼できて実用性があるのかを，実際の量子コンピュータを用いて評価しなければならない．これらのことが解決していかなければならない問題である．

ここで，量子コンピュータをもちいた量子計算の短所を振り返っておこう．量子コンピュータは処理の回数が多くなるほど解の信頼性が下がる．これは計算を行う際のノイズによって，計算結果を誤って

* 大学院工学研究科知識社会基礎工学専攻
* Fundamental Engineering for Knowledge-Based Society, Graduate School of Engineering
** 知能システム工学講座
** Department of Human and Artificial Intelligent Systems

しまう可能性があるからである．そのため，量子コンピュータの研究では，誤り訂正の研究も進められている．同時に，計算効率が良く，信頼性の高いアルゴリズムを作ることが求められている．本研究では，これらのことをふまえて，実際の量子コンピュータでショアのアルゴリズムを実行するために必要な乗算回路を Q#で製作した。また，C＃をもちいて古典的アルゴリズムによる素因数分解をおこなったので報告する．

2. 量子コンピュータをつかう

2.1 量子コンピュータをつかう環境

量子コンピュータを使える環境が整備されつつある．Amazon による AWS[8]，IBM による IBM Q[9]などである．ハードウェアだけでなく，同時にソフトウェアの開発が進んでいる．さらに，量子コンピュータを用いたプログラム環境も提供されている．

量子コンピュータのプログラミング実行環境には，現在, 2 種類の環境がある. 実際の量子コンピュータと用いるものと, エミュレータを用いる方法である. 2 つの環境には，メリットとデメリットが存在する．

まず, 実際の量子コンピュータを使うものとして，IBM や AWS が提供しているサービスからみていこう. 第一の特徴は, IBM や AWS(Amazon Web Services) では, 借りることが可能な qubit が非常に少ないであるがである．2016 年に IBM がオンラインサービスを開始したときは，5 量子ビットであった．現在，2020 年 9 月時点で，65 量子ビット まで拡張されている量子プロセッサ「Hummingbird」を公開している．さらに，IBM は 2023 年には 1000 量子ビット超えの量子コンピュータを計画している[10]．Amazon も，AWS で, 超伝導量子ビットやイオントラップを用いて構築された量子アニーラーやゲートベースのシステムなど，数種類のハードウェアをオンラインで利用することができるサービスを開始している．

現在は，IBM や AWS などによる量子コンピュータ（ハードウェア）のオンライン利用サービスだけでなく，量子コンピュータのエミュレータを提供するサービスも始まっており，オフラインによる量子計算のプログラミングも可能となっている. 例えば，Microsoft が提供する Visual Studio2019 では，ライブラリとして Q＃の拡張ができ，Q＃による量子コンピュータをつかったプログラミングが可能となっている．ここで注意すべき点は，量子計算をエミュレーションで実行しても速くはならないということである. なぜなら, Q#のコードが C#のコードに翻訳され，実行されるためにステップ数が増えるためである．それゆえ，量子プログラムだから速くなるということにはならないことに留意しよう．

では，どのような意味があるのかを考える．エミュレーションによる方法は，量子コンピュータが一般的に利用可能になった時のために，プログラミングの練習を行うことができるという意味合いが大きい．実際の量子コンピュータとエミュレータを使用した場合の比較を以下でおこなう．

１）実際の量子コンピュータを使ったとき

一般的に，実際の量子コンピュータを使うのは，オンライン上で提供されているサービスを利用することになる．また，多くの量子ビットを扱えるわけではないため複雑な回路を組むことは難しい．

２）量子コンピュータのエミュレータを使ったとき

量子コンピュータのエミュレータを使って実行するには visual studio を拡張したものやその他のソフトを用いて，仮想の量子ビットを扱うことになる．この方法で得られる計算結果は理論的に実行されるもので，後で触れるように実際の量子コンピュータを実行した結果とは異なる可能性がある．

将来的には，一般の人々でも１）の実際の量子コンピュータの利用が可能になることが期待されるが，現在は２）が主流である．実際の量子コンピュータを利用できる時代に備え，仮想的な量子コンピュータでプログラムや経験を積み重ねている段階といえる．そのため、量子コンピュータのエミュレータが開発されている。例えば，QCEngine などの量子コンピュータのエミュレータである[7]．QCEngine は JavaScript で書かれたサンプルプログラムをブラウザ上で動かすことができる．なお，QCEngine の特徴は，量子コンピュータでのプログラミングに必要な知識を学ぶ環境が提供されていることである．

量子コンピュータには，他に量子アニーリング方式がある．量子アニーリング方式では，D-Wave 社[11]のシステムのように大規模な量子ビットをあつかえるという利点を持っている．なお，D-Wave 社の advantage という名前のシステムは 5000qubit を扱うことができるが、ゲート式の量子コンピュータとは異なり汎用的な使い方はできない．しかし、組み合わせ最適化問題を解くための量子アルゴリズムを扱うことに特化して，商用として利用されている．このように，日本人研究者が先駆的な役割を果たした量子アニーリングもすでに実用化されている. なお，本論文では，量子アニーリングについてこれ以上詳しくは触れないことにする．

2.2 量子コンピュータで扱われている問題

現時点で，量子コンピュータの強みを生かして，量子コンピュータで扱うことが期待されている問題

をみていこう．ネットを介しての通信および取引の発展にともない, 公開鍵暗号が重要性を増してきた．例として，Shor の素因数分解は，量子コンピュータで計算時間が大幅に短縮できるものとして挙げることできる．Shor のアルゴリズムは、古典アルゴリズムでは困難な RSA 暗号解読への応用が期待されている．

さらに，量子コンピュータは，分子・物質の性質解析への期待も高まっている．自然界でも，宇宙空間のような極低温・超真空の極限状態では，分子の一部分では重ね合わせの状態が実現している場合がある．対象とする分子が増えると，考慮しなければならない分子のとり得る分子状態数が爆発的に増える。このことに伴い，重ね合わせの状態の計算に必要な計算時間が増大する．それゆえ，量子コンピュータを用いた高速化が研究されている．他にも HHL の行列積計算を量子コンピュータで行うことで物理現象，機械学習，金融といった分野で活躍することが期待されている．

さて、現時点の暗号技術と量子コンピュータとの関係をふりかえってみよう。現在，ビットコインなどのインターネット上にある資産は RSA 暗号等で保護されている．しかし，RSA 暗号は，前述したように量子コンピュータによって暗号が解かれる可能性がでてきた．逆に，量子コンピュータによって資産を守る方法が考えられている．量子暗号は，情報のやりとりを盗み見られたかを判断できるアルゴリズムである．それゆえ，もし見られていても．通信を途中からやり直すという方法で盗み見を回避できる．以上のように，量子コンピュータは現在の暗号技術を根底から覆す可能性を持っている．

2.3. 量子コンピュータのプラットフォーム

一部の研究者を除き，計算機研究者にとっても現実の量子コンピュータを直接触ることができる機会は多くない．しかし，ネットワークを介して，量子コンピュータをつかえる様々なプラットフォームが公開されている．また，量子プログラミング（量子アルゴリズムによる表現をプログラムする）には，ネットワークを通して比較的少ない量子ビット(qubit)を借りる方法と仮想量子ビット(qubit)を用いてプログラミングする方法がある．

現実の量子コンピュータの qubit を借りて利用する方法について説明する．IBM や AWS で，量子コンピュータを使うときは，アカウントを持つ必要がある．アカウントを作成するためには，いくつかの必要事項（メールアドレス等）を入力する過程がある．しかし，簡単かつ無料で利用可能であるため，利用しやすい. 多くの qubit を借りることはできないが，現実の量子コンピュータで，量子ゲートを用いた回路の計算ができるようになる．一方，仮想量子ビット(qubit)を用いたプログラミングでは，現実の量子コンピュータを利用するよりも多くの qubit を仮想的に使うことができる．Visual studio や Python をつかうことで, 無料で環境を整えることができる．なお，仮想量子ビット(qubit)を借りることは多くの qubit を扱うことが出来るメリットがあるが，実際に量子コンピュータで同じプログラムを動かすとうまくいかない可能性がある．今回の実験では，素因数分解を行うプログラムを行うため, 利用する qubit の数が多くなる．

今回の実験では Visual studio による仮想量子ビット(qubit)を利用する．また，今回利用したローカルな計算環境は AMD Ryzen 7 3700X 8-Core Processor 3.60 GHz である．

2.4 Quantum 開発キット

Quantum 開発キット(以降 QDK)には, Microsoft のオープンソースプログラミング言語である Q#や量子シミュレータ，他のプログラミング環境用の拡張機能，API ドキュメントが用意されている[12]．QDK では，Python，C#，F#の要素を取り入れた Q#を用いている．Q#の特徴として，量子固有の構造と演算が可能なデータ構造が導入されている．なお，Q#では量子演算を確認するために，量子アルゴリズムを記述する必要がある． QDK 内のライブラリを使用することで，簡単な演算シーケンスを設計する必要がなく，複雑な量子演算が可能となる．

3. ショアのアルゴリズム

ショアのアルゴリズムは，整数の素因数分解をおこなうアルゴリズムである．N の因数について調べるとき，次の手順で因数を探すことができる．はじめに，ユークリッド互除法で a と N の最大公約数を探す．a と N が互いに素であるならば，$f(x)=a^x \bmod N$ の周期 r を求める．最後に周期 r から因数を求めることができる．

3.1. 量子コンピュータを用いた計算

量子コンピュータはショアのアルゴリズムのうち，周期 r を求めることに用いられる．重ね合わせの状態と量子コンピュータで行われるフーリエ変換によって従来では計算しきれないものを理論上では可能にしている．

3.2 量子フーリエ変換

量子フーリエ変換(QFT)は，qubit の相対位相と大きさの中にあるパターンや情報を得るための変換である[2]．QFT は，量子ビットの相対位相を利用してパターンの情報を得ている．なお，離散フーリエ変換は，デジタルビットの値で周波数の情報を得るための変換である． 4qubit の QFT を図 1 に示す．左のゲートから順に 4qubit に作用させることで周期の情報を得る．また，逆量子フーリエ変換は図 1 のゲートを右から順に作用させることで利用できる．これはユニタリ変換の特徴によるものである．

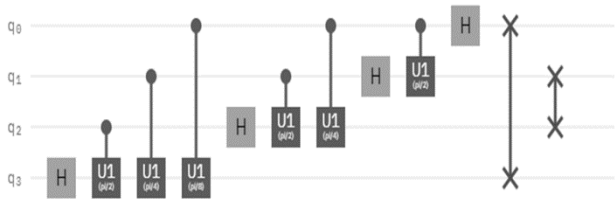

図 1　4qubit の QFT.

3.3 C#による因数分解プログラミング

量子コンピュータによるショアのアルゴリズムの計算と C#を使った古典的なアルゴリズムによるものとを比較する．ここでは，C#で総当たりによる因数分解を行うプログラムを作成した．このプログラムのアルゴリズムは，入力した値に対して小さい値から順番に因数の候補として除算を行う．余りが 0 の時，因数として記録する．その後，入力した値から因数を割った商に対して同様に小さい値から除算を行う．このプログラムでは入力を素数としたときに計算量が最大となる．最大 $N^{1/2}$ の除算を繰り返し行うプログラムである．実際に素数を入力することで計算にかかる時間を調べた．このプログラムによる結果を表 1 に示す.

表 1　C#を使った因数分解にかかった時間

入力した素数	処理時間(s)
1000000000000037	0.848
46116646144580591	4.240
100000000000000003	5.573
4611664614458057389	39.131

このように，従来のコンピュータによる計算方法では計算時間が指数関数的に延びていくことが分かる．これに対して量子コンピュータでの計算時間は qubit が増えるにつれて線形的に増えていくと考えられている.

3.4. a^xmod N の実装

ショアのアルゴリズムを量子コンピュータで実行するには，重ね合わせの状態を保ったまま a^xmod N の計算を行う必要がある．参考にした教科書[7]では a=2 の場合について紹介をしていた．a=2 の場合は swap ゲートを上の位から順に作用させることで簡単に実装できる．また，a^xmod N の量子回路は既に考えられている．しかし，今回の実験では作成した乗算の回路がオンライン上で借りた量子コンピュータで実際に動くかを確かめる. 多くの qubit を借りることは出来ないため，2 以外の比較的簡単な乗算を考えた.

3.4.1 フェルマー数

本研究では，回路を作るに際して，フェルマー数に注目した．フェルマー数とは 2^n+1 である数を指す．また，素数であるフェルマー数はフェルマー素数と呼ばれる．フェルマー素数の乗算回路がステップ数として少なくなるかは問題となる

3.4.2 (2^n+1)の乗算回路

図 2 は入力 2qubit に対して 3 を乗算する回路である．2^n+1 の乗算を行う際，4 つのステップにわけられる. ①繰り上がりする位の記録, ②$2^n$ 回のシフト，③元の状態との同期，④繰り上がりによるインクリメントである．また，繰り上がりを記録する qubit を演算が終わるたびに|0>の状態へリセットすることで乗算を繰り返すことができる.

図 2　(2^n+1)の乗算回路(n は自然数)．n=1 の例.

4．実験と分析

4.1. IBM での乗算回路の実行

IBM のオンラインサービスで提供されている量子コンピュータを使用して，3 の乗算を行う回路を製作した．表 2 は，実行した際に得られた実行までの過程と所要時間をまとめたものである．次に，個々に詳細に見ていこう．Created は自身が作成した量子回路を IBM へ送った時である．今回は図 3 の内容を

送信した．2qubit にアダマール変換を行った後に 3 の乗算を行う回路である(つまり入力は|0>～|3>). IBM へ量子回路を送ると Transpiling が行われる．すると, 図 4 のように変換された量子回路が得られる．その後，Validating が行われ，キューに加えられる．IBM では複数のユーザーが量子コンピュータを使うために順番を待たなければいけない．この待ち時間が In queue の時間である．

今回の実行では，比較的にキューの待ち時間が短かった．なお，長いときは 2 時間ほどかかることもある．そして，この回路を量子コンピュータで実行すると約 11 秒かかったことがわかった．

表 2　実行までの過程*と所要時間

動作	所要時間
Created	—
Transpiling	1.4 s
Validating	0.992 s
In queue	557.4 s
Running	11.1 s
Completed	—

*(IBM Quantum Experience)

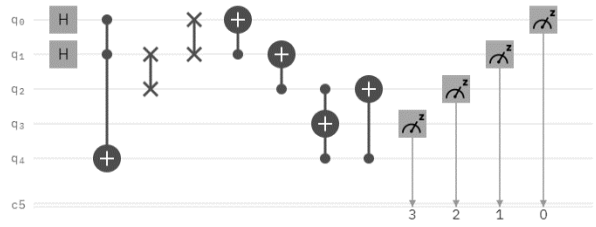

図 3　3 の乗算回路．IBM の量子コンピュータへ送信した回路(Transpiling 前).

この回路で整数 0～3 の重ね合わせの状態で乗算の計算が可能であるかを試した．図 3 の回路においても表 3 に示された結果が得られることを期待した．量子コンピュータは誤差があるため，表 3 のように正確に確率に則った結果を出すことはない. しかし，今回実行した結果である図 5 と比較してみると，予想と結果に違いがでていることがわかる．例えば 00000 と 00011 は，予想では高い数値にならないはずの 00001 の半分ほどの確率となった．問題点を探すため，繰り上がりのない回路を実行した．その結果，出力が 000 となったのが 49.9%，110 となったのが 38.1%という値が得られた．また，その他の割合は 0.6～2.9%と結果が得られた. このことから今回考案した繰り上がりの演算に，回路もしくは量子コンピュータ自身に何かしら問題があると考えられる．

表 3　重ね合わせの入力に対して期待される結果

入力	結果	確率
00000	00000	25%
00001	00011	25%
00010	00110	25%
00011	11001	25%
	その他	0%

4.2. Q#での乗算回路の実行

図 3 の 3 の乗算回路をエミュレータ用プログラミング言語 Q#を使ってプログラムして演算させた．考案した演算が予想通りに動くのかをエミュレータを用いて検証した．その結果，すべての状態が出力され，表 3 と同じ結果が得られた．

次に(2^n+1)の乗算の他の例として 9(n=3)と 17(n=4)の乗算を行う．9 の乗算を行う理由として，3 の乗算回路を 2 回繰り返す演算と 9 の乗算回路を 1 度だけ行う回路ではどちらが信頼でき，計算が早い回路であるかを試すべきだと考えたからである．また，17 を試す理由は 3 の次に小さいフェルマー素数であることがあげられる．2^n+1 の乗算回路が演算として安定した結果を残せるのであれば，ショアのアルゴリズムにおける a の値として(2^n+1)は回路が組みやすく，効率的であると考えることができる．

17 といった大きな値は実際の量子コンピュータではできないため，Q#を用いる．そのため信頼性についての評価はしていない．Q#では用意する仮想 qubit の数が計算時間に影響がある可能性があるため，17 の乗算に必要な qubit をそれぞれの計算に用意した．3 の乗算回路を 2 回繰り返す演算，9 の乗算

図 4　量子コンピュータで実行された量子回路 (0-3 の整数の重ね合わせの状態に 3 の乗算を行う) ．

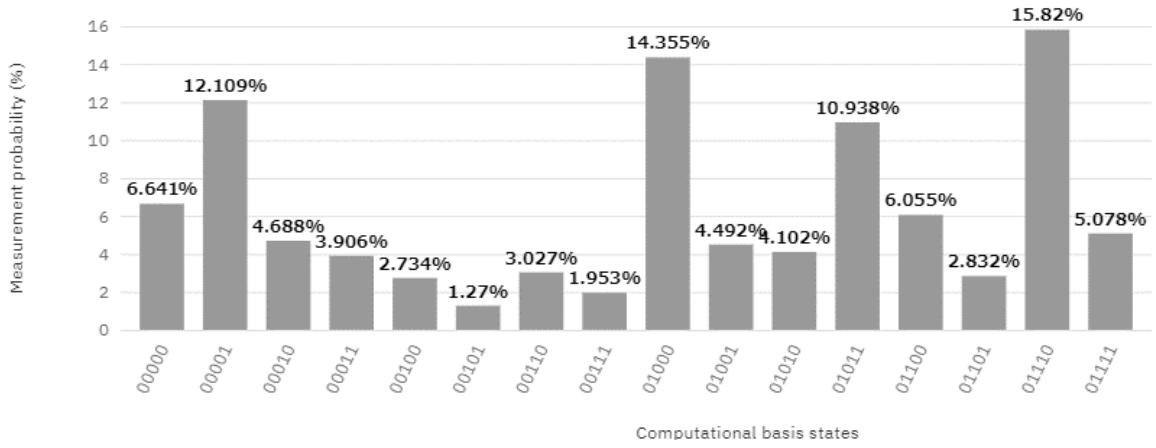

図 5　2048 回実行した際にでた計算結果の分布.

表 4　エミュレータを使った Q#での実行時間 a)

	実行時間(s)
3 の乗算	49
3 の乗算 2 回	62
9 の乗算	53
17 の乗算	68

a)　AMD Ryzen 7 3700X 8-Core Processor 3.60 GHz の PC 使用

を行う演算，17 の乗算を行う演算を Q#で実行した結果を表 4 に示す．3 と 9 の乗算には計算時間に大きな差は見られないようにみえる．それに比べて 3 の乗算を 2 回行った場合には 9 の乗算を行った時間よりも時間を要した．このことから，乗算を行う際は，素数の乗算を複数回組み合わせて乗算して行うよりも，一般化された乗算回路のほうが時間の効率が良い可能性がある．しかし，エミュレータ上での計算であるため，量子コンピュータで行う計算にかかる時間とは強く関係するとは言えないことに注意しなければならない．また，エミュレータ上で実行に時間がかかる場合は回路が複雑である可能性があげられる．17 の乗算に注目する．3 の乗算を 2 回行った場合よりも多くの時間を費やしている．3,9,17 の乗算の結果から，乗算する値に対して時間が指数関数的に増えていく可能性がある．これは乗算する値によって回路を変えるために起こる現象だと考えられる．一般化した乗算回路であれば，乗算する値が変わることよって計算時間に変化は生まれないと考えられる．

4.3. 奇数の乗算回路

(2^n+1)の乗算回路を発展させることで, 奇数の乗算回路を作ることを考えた．奇数と偶数の掛け算の組み合わせですべての乗算が可能になる．偶数の乗算は値をシフトするだけで良いので，奇数の乗算回路が完成するとすべての乗算回路を作ることができる．また，完成した乗算回路を逆から作用させることで除算の計算が可能であるかを考察した．

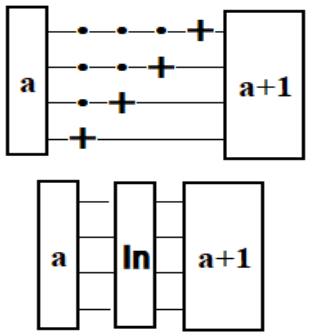

図 6　加算回路.

奇数の乗算は，$(2^a+2^b+\ldots2^z+1)$の乗算として考えられる．図 3 の回路を発展させることで、加算回路(図 6 参照)の組み合わせによる奇数の乗算回路を実現させることができた．奇数の乗算回路の例として 4qubit 同士の回路を図 7 に示す．これは入力を任意の値 a に対して，奇数 b を乗算する回路である．実行した結果，重ね合わせの状態同士の乗算に成功したことを確認できた．また，利用 qubit の数を増やすほど計算時間が指数関数的に増える結果となった．図 7 に示されたように加算回路が階段状に並んでいることがわかる．このアルゴリズムはオペランドや筆算による計算手順と異なる．計算内容は筆算やデジタル計算と量子回路による計算内容は同じである

が，量子回路は計算の順序が変わるだけで結果が変わってしまうので，古典的なアルゴリズムを量子回路に転用することはできない．その理由は，量子回路に用いられる各ゲートがユニタリ行列であることからわかる．

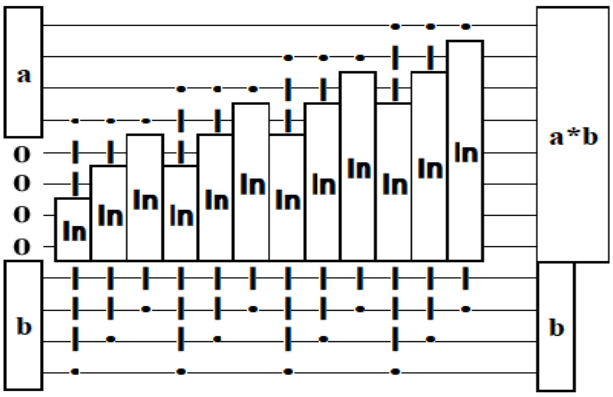

図 7　奇数の乗算回路．

4.4. 奇数の除算回路

奇数の除算回路が可能であるかについて考える．量子フーリエ変換と逆量子フーリエ変換との関係のように，今回の奇数の乗算回路は逆から作用させることで奇数の除算回路として成り立たない．乗算回路とは異なり，除算には解が整数とならない場合を考えなければならない．この除算回路では割り切れない場合，予測できないような値が出力されてしまう．割り切れない場合を含めた除算回路の完成が今後の課題として残った．

5.　おわりに

古典的なアルゴリズムによって素因数分解をおこなう場合と、実際の量子コンピュータを使った量子アルゴリズム（ショアのアルゴリズム）によって素因数分解をおこなった場合との比較を目指した．しかし，量子コンピュータを使った計算は，途中の乗算回路の設計までしか実行できなかった．エミュレーションでは，Q#を用いてショアのアルゴリズムの素因数分解に用いるための汎用的な奇数の乗算回路を製作できた．

量子演算によって解を求めると，確率によって解が得られることを確かめた．Q＃などのエミュレータによって，プログラミングすると，確率的な揺らぎは生じず，実際に量子コンピュータを使った場合とは，異なることに注意しなければならない．

ショアのアルゴリズムにおける累乗の計算を，乗算回路を複数回利用することで計算できることをエミュレータ上で確かめた．偶数の乗算(奇数の乗算と 2^n の乗算を組み合わせた演算)が可能であることを少数 qubit の場合(3qubit までの乗算)に確かめた．なお，Quantum supremacy(量子超越性)を議論するためには，揺らぎを考慮しなければならないので，エミュレータを使う場合は留意しなければならないだろう．累乗の計算を複数回の乗算で行うことは，ノイズの影響を多く受けてしまう可能性があり，結果の信頼性が低くなる恐れがある．

ハードウェアとしての量子コンピュータの現実性について述べよう．IBM は，「スーパー冷凍機」の開発をおこなうことにより，「フォールトトレラントな（ノイズがあっても正確な量子計算ができる）量子コンピュータは今後 10 年以内に達成できる目標だと感じている」と考えており[10]，揺らぎの問題も解決されるかもしれないと予想している．

次にソフトウェアとしての量子コンピュータの課題について述べよう．ハードウェアとしての量子コンピュータが完成しても，実際にプログラミングする場合，習得の難度が高い可能性がある．例えば，古典的な乗算回路のアルゴリズムは量子アルゴリズムには転用できなかったのをみてきた．量子アルゴリズムには，量子コンピュータにおける知識が必要不可欠である．いくつかの回路を実際の量子コンピュータで実行することにより，量子コンピュータに関する理解を深めることができるかもしれない．しかし，実際の量子コンピュータで複雑な回路を実行するのに多くの qubit を利用することはまだ難しい．そのため現時点では，量子プログラミング言語の習得やアルゴリズムの開発に，エミュレータを利用することは有効であると考えられる．

謝　辞

本論文を執筆するにあたり，議論および有益なコメントをしてくださった髙田宗樹教授，および研究室のメンバーに感謝いたします．

参考文献

[1] F. Arute et al.: Quantum supremacy using a programmable superconducting processor, Nature, 574, 505-511 (2019).

[2] M. Brooks: Before the quantum revolution with decades still to go until the first general-purpose quantum computers, the race is on to make today’s systems useful, Nature, 574, 19-21 (2019).

[3] E. Gibney: Google publishes landmark quantum supremacy claim, Nature, 574, 461-462 (2019).

[4] William D. Oliver: Quantum computing takes flight, Nature, 574, 487-488 (2019).
[5] 平田隆幸:なぜ工学部の学生は量子コンピュータを学ぶべきなのかー量子コンピュータへ至る計算機の歩みー, 福井大学大学院工学研究科報告(2020).
[6] 宮野健次郎, 古澤明: 量子コンピュータ入門第２版, 日本評論社, pp.165 (2016).
[7] Eric R.Johnston,Nic Harrigan,Mercedes Gimeno-Segovia : 動かして学ぶ量子コンピュータプログラミング, 北野明章訳, 丸山耕司技術監修, オライリー・ジャパン, pp. 309, (2020).
[8] Amazon AWS<https://aws.amazon.com/jp/braket/> (2021 年 11 月).
[9]IBMQ<https://www.ibm.com/services/jpja/strategy/quantum/>(2021 年 11 月).
[10] ITmedia NEWS >科学・テクノロジー>2023 年に 1000 量子ビット超えの IBM 製量子コンピュータ <https://www.itmedia.co.jp/news/articles/2009/17/news095.html>(2021 年 11 月).
[11] DWave<https://dwavejapan.com/>(2021 年 11 月).
[12] Azure Quantum documentation <https://docs.microsoft.com/en-us/azure/quantum/>(2021 年 11 月).