Vol.2021-QS-2 No.20
2021/3/29

# 超伝導回路を用いた量子誤り訂正向けオンライン復号器の提案

上野 洋典[1,a)]　近藤 正章[1,2]　田中 雅光[3]　鈴木 泰成[4]　田渕 豊[5]

**概要**：量子計算機の誤り訂正符号として Surface code が知られており，その復号プロセスはグラフの最小重み完全マッチング（MWPM: Minimum Weight Perfect Matching）問題に帰着される．超伝導量子ビットを元にした量子計算機において，量子ビットは極低温環境でのみ動作するという制約を持つ．それに対して既存のアーキテクチャでは MWPM を解く古典計算機は室温で動作するが，それらをつなぐ室温-極低温間の膨大な配線が，超伝導量子計算機のスケーラビリティを制限している．本研究では超伝導量子計算機のスケーラビリティ向上を目的として，極低温環境で動作する超伝導古典回路である SFQ 回路を用いて MWPM を近似的に解く復号器を構築した．数値シミュレーションにより，提案した復号器のしきい値を評価し，Surface code の観測プロセスに対して同時に復号プロセスを行うオンライン動作が可能であることも示した．また，提案した復号器の消費電力を評価し，極低温環境での動作に適していることも示した．

## 1. はじめに

超伝導素子による量子ビットを用いた量子計算機は近年盛んに研究されており，53 量子ビットの量子計算機による量子超越性の報告は記憶に新しい [1]．しかし実応用上で有用なタスク，例えば Shor のアルゴリズム [2] による 256-bit の RSA 暗号の解読を実行するためには数千量子ビットが必要であり，現在の開発状況とはまだ乖離が大きい．量子計算機の規模の増大を妨げている 1 つの要因として，デコヒーレンスにより量子ビットの重ね合わせ状態がすぐに破壊されてしまうことが挙げられる．デコヒーレンスにより量子ビットが量子性を保てなくなる時間はコヒーレンス時間と呼ばれるが，現状の超伝導量子ビットのコヒーレンス時間は数百 $\mu s$ 程度である．量子ビットをデコヒーレンスから守る手法が盛んに研究されており，量子誤り訂正（QEC: Quantum Error Correction）と呼ばれている．

QEC の手法の 1 つに，複数の物理量子ビットを冗長に組み合わせて 1 つの論理量子ビットを構成する量子誤り訂正符号が挙げられる．代表的な量子誤り訂正符号の 1 つに，Surface code（表面符号）がある [3]．Surface code は図 1 のように 2 次元平面上にデータ量子ビットと**スタビライザ**測定用の補助量子ビットが規則正しく並んだ構造を持つ．Surface code の復号はグラフの最小重み完全マッチング（MWPM: Minimum Weight Perfect Matching）問題に帰着できることが知られている [4]．スタビライザ測定がエラーなく実行される場合には，復号プロセスは 2 次元平面上のマッチング問題に帰着される．測定にエラーが発生する可能性がある場合の復号プロセスは，観測プロセスを複数回行い，得られた観測値を時間方向に積み上げることで 3 次元格子を構築し，その 3 次元格子上でのマッチング問題を解くことに帰着される（図 1(c)）．

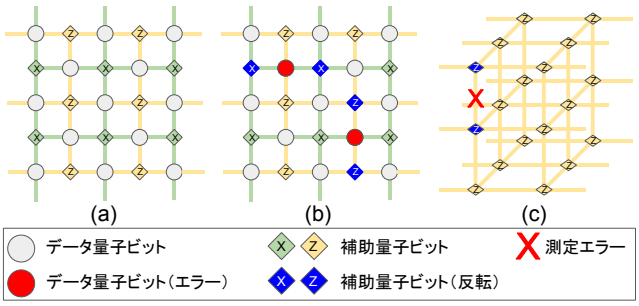


**図 1**　Surface code（表面符号）

測定にエラーが発生する場合，補助量子ビットの観測が複数回行われる必要がある．Blossom アルゴリズムによる復号を含めて，これまでに提案されている復号アルゴリズムの処理手順は，複数回の観測プロセスを行い 3 次元格子を完全に構築した後に復号プロセスを行うものが多い．本稿ではこの処理手順を**バッチ処理方式**と呼ぶ．それに対して，復号の実行時間の観点から観測プロセスと復号プロセスは同時に実行されることが望ましい．すなわち，1 回の観測プロセスが終了するたびに復号プロセスを行い，その

[1] 東京大学 大学院情報理工学系研究科
[2] 理化学研究所 計算科学研究センター
[3] 名古屋大学 大学院工学研究科
[4] NTT セキュアプラットフォーム研究所
[5] 理化学研究所 創発物性科学研究センター
[a)] ueno@hal.ipc.i.u-tokyo.ac.jp

時点で訂正可能なエラーを訂正する処理手順である．本稿ではこの処理手順を**オンライン処理方式**と呼ぶ．本稿ではオンライン処理方式の Surface code の復号アルゴリズムを提案する．

また，超伝導量子ビットは 20 mK 程度の極低温環境でのみ動作するという制約があり，量子計算機システムは希釈冷凍機を用いて構成されることが一般的である．一方で，QEC を行う古典計算機は通常は室温環境で動作するため，量子ビットと古典計算機を繋ぐ異なる温度環境間の配線が膨大となり，量子計算機システムのスケーラビリティは制限されている．スケーラビリティ向上のためには QEC も極低温環境で行われることが望ましいが，希釈冷凍機の中で許容される消費電力は量子ビットの動作する 20mK 層では 10〜100 $\mu$W，4 K 層では 1 W 程度と極端に小さい．誤り耐性量子計算機の実現のためには高速・超低消費電力で極低温環境で動作する復号器が必要不可欠である．

本研究では Surface code のオンライン復号アルゴリズム **QECOOL** (Quantum Error COrrection by On-Line algorithm) を提案し，その実行に特化した復号器を超伝導素子を用いた古典回路である SFQ (Single Flux Quantum) 回路を用いて実装する．QECOOL は，Surface code の補助量子ビットに 1 対 1 で対応する “Unit” と呼ばれる素子同士の信号のやりとりで実行される並列分散型の特徴を持っており，CMOS に比べて集積度が低く大容量 RAM の構築が難しい SFQ 回路での実装にも適していると考えている．提案アルゴリズムのエラー訂正性能およびそれを実行する回路の消費電力の観点から，実装した復号器が極低温環境での動作に適していることを示す．

## 2. 背景知識

### 2.1 SFQ 回路

SFQ 回路は超伝導素子を用いた古典計算を行うディジタル回路である．図 2 のよう超伝導体で構成されたループを通る磁束は量子化されるという性質がある．この磁束量子は $\Phi_0 \approx 2.068 \times 10^{-15}$ Wb として表される．SFQ 回路ではこの磁束量子の有無をディジタルの ‘0’ と ‘1’ にそれぞれ割り当てて情報処理を行う．超伝導ループを構成するジョセフソン接合（JJ: Josephson junction）はスイッチング素子としてはたらき，CMOS におけるトランジスタに対応する．4 K での動作を想定して設計された SFQ 回路において，JJ の 1 回のスイッチングで消費されるエネルギーは $10^{-19}$ J 程度であり，トランジスタの 10000 分の 1 程度である．

SFQ 回路は超伝導現象を利用して計算を行うため，4 K 程度の極低温環境でのみ動作するという制約があり，冷却コストなどの問題から計算機への実用化にはまだ多くの課題がある．一方で超伝導量子ビットも同様に極低温環境でのみ動作するため，超伝導量子計算機の制御回路に SFQ 回路を用いる手法が盛んに研究されている [5], [6], [7]．もともと量子ビットの冷却が必要であるため，SFQ 回路の冷却も同じ冷凍機で行うことが可能となる．以上の背景を踏まえ，本研究では超伝導回路を用いて量子誤り訂正を行うシステムを提案する．

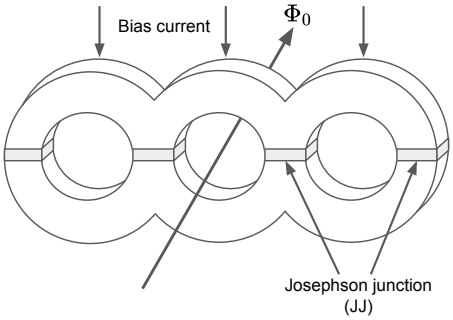


**図 2** SFQ 回路

### 2.2 Surface code

Surface code は高いエラー訂正能力を持つ量子誤り訂正符号の 1 種であり，量子状態を保持するデータ量子ビットと，データ量子ビットのエラーを検出するために測定される補助量子ビットを格子状に並べて構築される．図 1(a) は符号化距離 $d = 3$ の Surface code の概要を表しており，データ量子ビットと補助量子ビットがそれぞれ丸と四角で示されている．各補助量子ビットは隣接する 4 つまたは 3 つのデータ量子ビットと相互作用しており，その観測値は隣接する 4 つのデータ量子ビットのうち，エラーが生じているものの個数のパリティを表す．補助量子ビットの観測値の集合のことをエラーシンドロームと呼ぶ（図 1(b)）．Surface code には $X$ と $Z$ の 2 種類の補助量子ビットがあり，それぞれが Pauli-$X$ エラー（ビット反転エラー）および Pauli-$Z$（位相反転エラー）を検出する．

Surface code の復号の目的は，データ量子ビットに生じたエラーの種類と箇所をエラーシンドロームから特定することである．エラーシンドロームを再現する最も尤度の高いエラーパターンを推定することが望ましいが，そのような処理は一般に NP 困難であるため，多項式時間で解ける問題に近似的に帰着することが一般的である．そのうち最も一般的なものが，グラフのマッチング問題への帰着である [4].

補助量子ビットの観測にエラーが生じる可能性がある場合は，観測を複数回おこなってから復号を行う必要がある．すなわち，2 次元平面状のエラーシンドロームを時間方向に重ねていくことで，図 1(c) のように 3 次元構造のエラーシンドロームを構築し，その上でマッチング問題を解く必要がある．

## 3. 関連研究

MWPM よりも効率的な Surface code の復号方法とし

て，Union-Find アルゴリズムを元にした手法が注目されており [8]，FPGA や ASIC をターゲットとしたハードウェア実装についても提案されている [9]．Das らの提案するハードウェア実装では，高度にパイプライン化されたアーキテクチャが Surface code の復号プロセスの高速化に有効であると述べている．

SFQ 回路や CryoCMOS をはじめとした極低温コンピューティングを用いて，超伝導量子ビットの制御を行う手法についても盛んに研究が行われている [5], [6], [7]．Holmes らのグループは SFQ 回路を用いて Surface code の復号を行う Approximate Quantum Error Correction (AQEC) という手法を提案した [5]．彼らの実装した復号器は Surface code の各データ量子ビットおよび補助量子ビットに 1 対 1 対応した複数のユニットから成り，そのユニット間の信号のやりとりを通してエラーを検出・訂正するという並列分散処理方式で構成されている．各ユニットは完全に並列に動作し，物理量子ビットのエラー発生確率 $p$ や Surface code の符号化距離 $d$ にほぼ依存せずに高速に動作する．しかし，各ユニットが並列に動作して 1 つのノードを同時にマッチングしようとした場合に必要となる “調停” の処理が複雑で，回路面積を増加させている．また，AQEC で解けるのは 2 次元平面グラフのマッチング問題のみであり，2.2 節で述べた補助量子ビットの測定にエラーがある場合，すなわち 3 次元格子状グラフのマッチング問題に関しては言及されていない．

## 4. 提案手法

### 4.1 復号アルゴリズム

本節では，MWPM の貪欲法的な近似アルゴリズム [10] に基づく Surface code の復号アルゴリズム QECOOL を提案し，2 次元平面上のマッチング問題の解法を例にアルゴリズムの詳細を説明する．構成要素として，各補助量子ビットに 1 対 1 に対応する “Unit” とそれらを制御する “Controller” を導入する．各 Unit は補助量子ビットと同様に隣接する 4 つの Unit と接続し，信号のやり取りを行う．Controller は 1 つの論理ビットにつき 1 つ設けられ，論理ビット内の全 Unit の制御を行う．

Algorithm 1 に QECOOL アルゴリズムを示す．このアルゴリズムは 3 次元構造上のマッチング問題を解くためのものであるが，格子の高さを表す $N_{depth}$ を 1，後述する $th_v$ を-1 と設定することで，2 次元平面上のマッチング問題へと適用できる．各 Unit は論理ビット内の一番左上の Unit を $(0,0)$ として，Row と Column についての ID を持っている．測定プロセスにおいて，各補助量子ビットのシンドローム値は対応する Unit の持つ “Reg” に保持される（Algorithm 1 の **MeasureEachUnit**）．復号プロセスにおいて，Controller は “Token” を左上の Unit から順に渡していく．Algorithm 1 の **RestartUnit** の処理は以下

**Algorithm 1** QECOOL のアルゴリズム

```
1: MeasureEachUnit:
2: m = 0
3: while true do
4:  A = checkAncilla()
5:  if m == 0 then
6:   Reg[0] = A
7:  else
8:   Reg[m] = Reg[m-1] ⊕ A
9:  end if
10:  m = m + 1
11:  Sleep(Mcycle)
12: end while

1: Controller:
2: start_loop:
3: for C = 1 to N_limit do
4:  for b = 0 to N_depth do
5:   shift = true
6:   for i = 0 to N_row do
7:    currentRow = i
8:    for j = 0 to N_col do
9:     if m − b > th_v then
10:      giveToken(i,j)
11:      RestartUnit(b)
12:      while !getFinish()
13: && !Timeout()
14:     end if
15:
    shift&=!Unit(i,j).Reg[0]
16:    end for
17:   end for
18:   sendResetFlag()
19:   if shift then
20:    SHIFTREG()
21:    goto start_loop
22:   end if
23:  end for
24: end for

1: procedure
2: SPIKE(row, flag)
3:  if row == currentRow
    then
4:   if flag == 1 then
5:    sendSpikeEast();
6:   else
7:    sendSpikeWest();
8:   end if
9:  else
10:  if flag == 1 then
11:   sendSpikeSouth();
12:  else
13:   sendSpikeNorth();
14:  end if
15:  end if
16: end procedure

1: procedure SHIFTREG
2:  if m > 0 then
3:   for i = 0 to N_depth − 2 do
4:    Reg[i] = Reg[i+1]
5:   end for
6:   m = m − 1
7:  end if
8: end procedure

1: RestartUnit(Input: b)
2: if Token == 1 then
3:  FlagToken = 1
4:  if Reg[b] == 1 then
5:   requestSpike()
6:   for t = b to N_depth do
7:    if (S = getSpike()) != NULL
8:  then
9:     Dir = rotate(S)
10:     correctQubit(Dir)
11:     sendSyndrome(Dir)
12:    end if
13:    if t != b && Reg[t] == 1
14:  then
15:     sendController(”Finish”)
16:    end if
17:   end for
18:  else
19:   sendController(”Finish”)
20:  end if
21: else
22:  for t = b to N_depth do
23:   if Reg[t] == 1 then
24:    SPIKE(self.row,FlagToken)
25:    if getCorrect() then
26:     Reg[t] = 0
27:     sendController(”Finish”)
28:    end if
29:   else
30:    if (S = getSpike()) != NULL
31:  then
32:     Dir = rotate(S)
33:     SPIKE(self.row,FlagToken)
34:     if getCorrect() then
35:      correctQubit(Dir)
36:      sendSyndrome(Dir)
37:     end if
38:    end if
39:   end if
40:  end for
41: end if
```

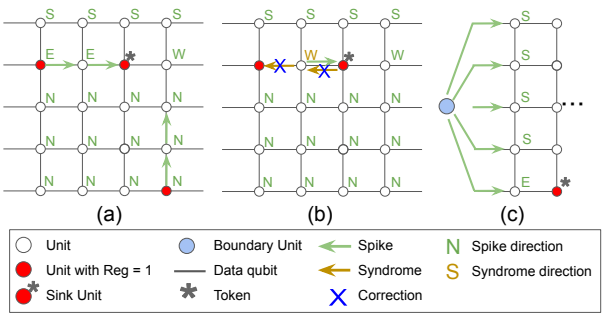


**図 3** 提案アルゴリズム QECOOL の処理の概要

のようにまとめられる．

(1) Unit が Token を受け取ると，Unit は自身の Reg をチェックする．もし値が 1 であれば Unit は “Sink

Unit” となり，他の Unit に対して “Spike” 信号の送信を要求する．Sink Unit でなく，Reg の値が 1 である他の Unit は Sink Unit の方に向けて Spike 信号を送信し，Sink Unit は最初の Spike の到着を待つ．所定のサイクル以内に Spike が Sink Unit に到着しなければ，Sink Unit は次の Unit に Token を渡す．

(2) Sink Unit が他の Unit に対して Spike 信号を要求すると，他の Unit は自身の Reg をチェックし，値が 1 であれば Sink Unit に向けて Spike を送信する（図 3 (a)）．Spike の送信方向は自身の行 ID と Controller の設定する “*currentRow*” の比較，および Token をすでに受け取ったかどうかを表す “*FlagToken*” の値に応じて決定される（Algorithm 1 の **SPIKE** を参照）．

(3) Sink Unit でなく Reg の値が 0 であった Unit は，Spike 信号を隣接する Unit から受け取る可能性がある．その場合は Spike が Sink Unit へと届くように他の隣接する Unit へと Spike を送信する．Spike の送信方向はステップ (2) と同様に行われる．また，その Unit は Spike を受け取った方向を 180° 回転させることにより，Spike を送信してきた Unit の位置を指し示す方向を保存しておく．この情報は次のステップで使われる．

(4) 最初の Spike が Sink Unit に到達すると，Sink Unit は Spike の来た方向にあるデータ量子ビットの訂正のための信号を送信する．また，“*Syndrome*” という信号を Spike の来た方向に返す（図 3 (b)）．Syndrome 信号はステップ (3) で保存された方向の情報を用いて，Spike 信号の送信元まで伝播し，伝播の経路上にあるデータ量子ビットを訂正する（Algorithm 1 の *correctQubit*() および *sendCorrect*()）．

このアルゴリズムはある Sink Unit に対して最近傍にある Reg が 1 となる Unit を見つけることができるが，それにより得られた Reg が 1 である Unit 同士のペアの合計距離が最小となることは保証されない．そこで，上の 4 つのステップを実行する際に，Spike の伝播を Sink Unit が待つサイクル数を制限し，徐々に大きくしていくことで対応する．すなわち，1 回目の試行では Sink Unit が Spike 要求から 1 サイクル以内に Spike が返ってこなければ次に Token を渡すことで距離 1 のペアのみを探し，次の試行では距離 2 のペアを探す，という要領である．これにより，MWPM の貪欲法的な解法と同様の解が得られる．

あるエラー鎖が論理ビットの境界とつながっているように見える場合（図 3 (c)），論理ビット内の補助量子ビット同士の適切なペアは存在しない．このケースを扱うために，各境界に対応する “*Boundary Unit*” という新たな Unit を導入する．Boundary Unit は Token を受け取らず，常に *requestSpike*() に対して Spike 信号を送信する．通常の Unit 同士のマッチングとの優先度をつけるために，Boundary Unit の Spike を送信するタイミングは適切に調整する必要がある．

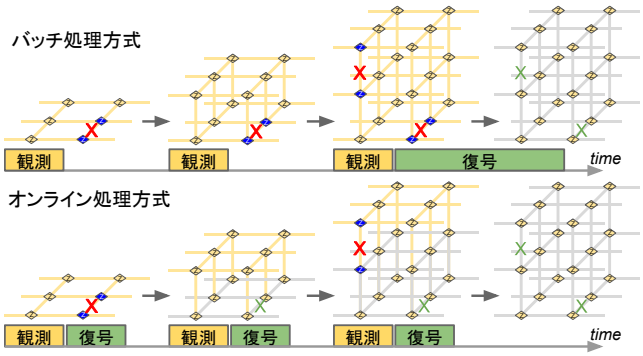


**図 4** バッチ処理方式とオンライン処理方式

3 次元構造上のマッチング問題への対応は，Unit を 3 次元に積層するのではなく，Reg のサイズを増やして対応する補助量子ビットの時系列で得られるシンドローム値を Reg に格納することで実現する．ステップ (1) で Token を受け取った Unit は，Reg のうち基準となる格子の高さに対応する $b$ 桁目の値に応じて Sink Unit となるかどうかを決定する (**RestartUnit** の 4 行目)．また，ステップ (2) において sink でない Unit は 1 サイクル毎に Reg の $b$ 桁目以降を順に参照し，値が 1 であれば Spike を送信する (**RestartUnit** の 20, 21 行目)．それ以前に他の Unit から Spike を受け取った場合は Reg の参照を途中で打ち切り，ステップ (3) へと移行して Spike の中継を行う．

### 4.2 3 次元構造上のオンライン QEC への拡張

図 4 は $d = 3$ の Surface code で補助量子ビットの測定にエラーが生じる可能性のある場合について，バッチ処理方式とオンライン処理方式の処理手順の違いを表している．バッチ処理の場合には 3 回の観測プロセスを行なった後に復号プロセスを行う一方，オンライン処理の場合には各観測プロセスが終了するたびに復号プロセスを行い，その時点で訂正可能なエラーを訂正する．これによりオンライン処理では即座にエラーを訂正可能であり，1 回の復号プロセスで扱う必要のあるエラーの数を減らし，復号プロセスの実行サイクル数を小さくできるというメリットもある．しかし，Surface code の格子の 1 層ごとに復号プロセスを行うのでは補助量子ビットの測定エラーに対応できなくなるため，Controller は適切な回数の観測プロセスが終了した後に適切な箇所のエラー訂正を行うように，復号プロセスを制御する必要がある．Algorithm 1 では，各 Unit の Reg の要素数，すなわち 3 次元格子のうちエラー訂正を行なっていない層の数が $th_v$ を超えている場合にのみ **RestartUnit** を行うことで実現できる (**Controller** の 9 行目)．しきい値である $th_v$ を大きくすると多くの観測値を Reg に保存したあとに復号プロセスを行うことで，補助量子ビットの連続した測定エラーに対応できるようになるが，処理の手順がバッチ処理方式に近づき実行サイクル数が大きくなる．一方で，$th_v$ を小さくすることで実行サイ

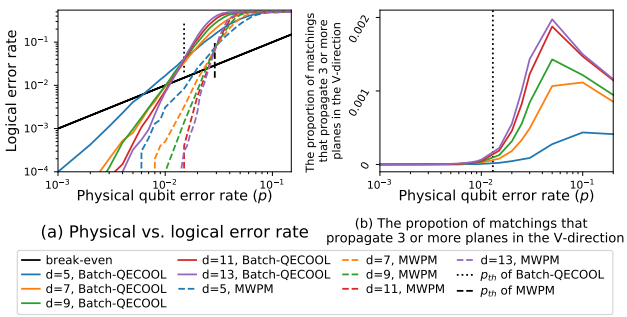


**図 5** (a) MWPM と提案手法（バッチ処理方式）のエラー訂正性能．提案手法のしきい値は 1.5%程度と確認できる．(b) 提案手法（バッチ処理方式）により得られたマッチングのうち格子の高さ方向に 3 層以上またがるものの割合．

クル数を削減できるが，エラー訂正性能は低くなる．次節で適切な $th_v$ を設定するための予備評価を行う．

### 4.3 オンライン QEC を行うハードウェア設計のための予備評価

本節では，QECOOL において $th_v$ の値がエラー訂正性能に与える影響を評価し，適切な $th_v$ の値を決定する．本節では QECOOL をバッチ処理方式，すなわち **MeasureEachUnit** を $d$ 回実行した後に **Controller** 処理を実行する場合の性能を評価する．以降，これを “バッチ QECOOL” と呼ぶ．

エラー訂正の性能評価はシミュレータを用いて数値実験により行う．ノイズモデルとしては，シンドローム測定を行うたびにデータ量子ビットとアンシラビットの両方に depolarizing noise が生じるノイズモデルを採用した [11]．$X, Y, Z$ エラーが全て独立に確率 $p/3$ で生じると仮定し，物理エラー率と論理エラー率の関係を示す．また，データ量子ビットと補助量子ビットのエラー発生確率は同じであるとする．同様の実験設定を 6.2 節でも用いる．

図 5(a) は物理エラー率と論理エラー率の対数スケールでのプロットを示している．黒い実線は物理エラー率 = 論理エラー率となる break-even を表している．その他の実線はバッチ QECOOL，破線は MWPM[4] の結果をそれぞれ表している．

エラー訂正性能の評価指標として，しきい値を用いる．しきい値 $p_{th}$ は復号器ごとに定義される値で，以下の条件を満たす物理エラー率を指す．すなわち，$p < p_{th}$ であれば符号距離 $d$ が大きくなるにつれて論理エラー率 $p_L$ が小さくなるような $p_{th}$ である．しきい値は，ある復号器について複数の $d$ で物理エラー率 $p$ と論理エラー率 $p_L$ をプロットした場合に，各プロットが 1 点で交差する点の $p$ として観察できる．この値が大きいほど，復号器のエラー訂正性能は高い．図 5(a) より，バッチ QECOOL と MWPM のしきい値はそれぞれ 0.015, 0.03 程度である．

また，図 5(b) はバッチ QECOOL により得られたマッチングのうち，格子の高さ方向に 3 以上の長さがあるものの割合を示している．$p$ が大きい場合はそのようなマッチングの割合は符号化距離 $d$ に応じて大きくなっているが，$p$ がバッチ QECOOL のしきい値よりも小さい場合には，その割合は $d$ に依らずに無視できるほど小さい．これは，オンライン QECOOL において復号プロセスを実行する際に Surface code の格子の高さが 3 以上あれば，バッチ QECOOL と同程度のエラー訂正性能を期待できるということである．以上の結果より，以降は $th_v = 3$ と想定する．

## 5. ハードウェアによる実装

### 5.1 アーキテクチャ

図 6 に，符号化距離 $d$ の論理ビットの $X$ エラーの検出・訂正に用いるハードウェアの概要を示す．同様の構造のハードウェアが $Z$ エラーの検出・訂正にも用意される．図 6 の各 *Unit* は Algorithm 1 の “Unit” に対応しており，$d \times (d-1)$ 個の Unit が 2 次元格子状に並んでいる．各論理ビットに 1 つ Controller があり，各 Unit に対して $Push, Pop, Restart$ の各信号を送信できる．並列分散方式で構成されていることから，このアーキテクチャは Unit の数を増やすことで任意の $d$ に対応可能である．

各 Unit は対応する補助量子ビットのシンドローム値を保持するレジスタ（Reg）を持っており，それらはキューのように動作する．すなわち，観測プロセスが行われるたびに Controller は Push 信号を全 Unit に送信し，観測値（Meas-In）が Reg の末尾に保持される．全 Unit の Reg の最初のエントリが 0 になると，現在の層の復号プロセスが終了する．その場合 Controller は全 Unit に Pop 信号を送信し，Reg の値を 1 ビットシフトさせる．4.3 節の結果に基づき，$th_v = 3$ を想定するが，これは Reg に最低限必要なビット幅と一致する．本稿ではある程度のマージンを含め，Reg のビット幅を 7 とする．

“Row Master” は 2 次元格子状に並ぶ Unit の各行に配置され，各行の最初の Unit に Token を渡す役割を担う．Row Master は常に全 Unit の Reg の値を参照しており，もしその行の Unit の Reg が全て 1 を含んでいなければ，その行へ Token を渡さずに次の行の処理へとスキップする．つまり，その行に Sink Unit となりうる Unit がいない場合はその行へは Token を渡さず，不要なサイクルの削減を行う．また，その行の全 Unit に対して *CurrentRow* 信号を送る役割も持つ．

各論理ビットに対して 2 つの *BoundaryUnit* が Unit のアレイの両側に配置されており，それぞれ境界に置かれた $d$ 個の Unit と隣接している．Boundary Unit は Spike を $d$ 個の Unit に分配するが，そのうち有効なのは最短経路で Sink Unit に向かうものだけであり，エラー訂正性能への影響はない．

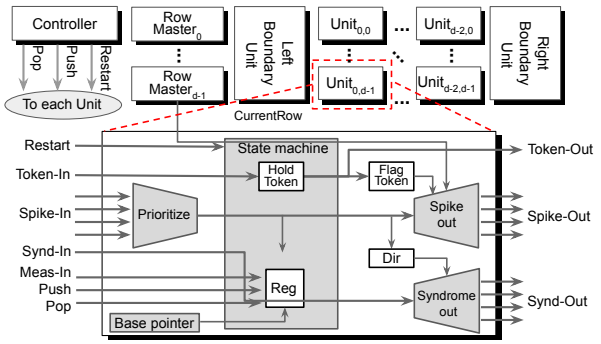


**図 6** QECOOL のアーキテクチャの概要及び Unit のマイクロアーキテクチャ

**表 1** Unit の設計に用いるロジックエレメント

| ロジックエレメント | JJ 数 | バイアス電流 (mA) | 面積 ($\mu$m$^2$) | 遅延 (ps) |
|---|---|---|---|---|
| splitter | 3 | 0.300 | 900 | 4.3 |
| merger | 7 | 0.880 | 900 | 8.2 |
| 1:2 switch | 33 | 3.464 | 8100 | 10.5 |
| destructive readout (DRO) | 6 | 0.720 | 900 | 5.1 |
| nondestructive readout (NDRO) | 11 | 1.112 | 1800 | 6.4 |
| resettable DRO (RD) | 11 | 0.900 | 1800 | 6.0 |
| dual-output DRO (D2) | 12 | 0.944 | 1800 | 6.8 |

### 5.2 Unit の実装

各 Unit は以下の 5 つの構成要素から成る．

- **State machine**: Unit の動作を制御する．状態遷移は Restart，Token および Spike の入力によって起こる．また，HoldToken と Reg の 2 つのレジスタメモリを持っており，それらの値も状態遷移に影響を与える．
- **Prioritization module**: Unit は同時に複数の方向から Spike 信号を受け取る可能性があり，その場合にはそのうちの 1 つを選択する必要がある．このモジュールでは方向毎に予め優先度を決めておき，Spike が同時に到着した場合にはその優先度に基づき 1 つの方向を選択する．このモジュールは信号の到着するタイミングを積極的に利用して情報処理を行う race logic[12] のコンセプトを用いて設計している．Spike 信号の伝播に適切に遅延を入れることで，優先度の高い方角から来た Spike が先に Unit に到着するようにし，それにより Spike 到着方向を適切に選択する．
- **Spike out module**: CurrentRow と FlagToken の値に基づいて，適切な方向に Spike 信号を送信する．
- **Syndrome out module**: Spike が到着した方向と逆の方向を保持する *Dir* レジスタの値に基づき，適切な方向に Syndrome 信号を送信する．また，データ量子ビットのエラー訂正信号も同時に送信する．
- **Base pointer module**: 補助量子ビットの観測値を保持する Reg のうち，どの桁を参照するかを決定する．Reg から読み出した値に応じて Spike を送信するかどうかを決定する．

### 5.3 SFQ logic gates

QECOOL のハードウェア設計を行うにあたり，ニオブ

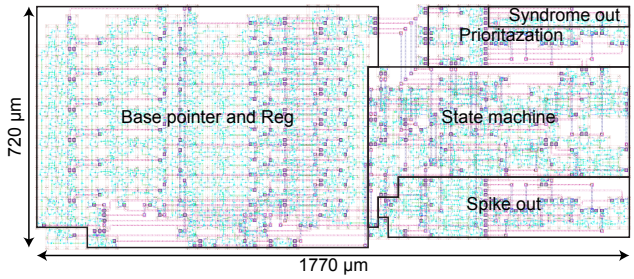


**図 7** QECOOL の Unit のレイアウト図

**表 2** Unit を構成するロジックエレメント，合計 JJ 数および回路面積（AIST 10-kA/cm$^2$ ADP セルライブラリ [15]）

| cell | State machine | Prio-ritiz-ation | Base pointer (7-bit) | Spike out | Synd-rome out | Other | Total (7-bit) |
|---|---|---|---|---|---|---|---|
| splitter | 17 | 4 | | 8 | | 2 | 31 |
| merger | 14 | 9 | 30 | 8 | 2 | 2 | 65 |
| 1:2 switch | 8 | | | 3 | | | 11 |
| DRO | | | 3 | | | | 3 |
| NDRO | | | 20 | | | | 20 |
| RD | 6 | | 30 | 4 | 4 | | 44 |
| D2 | | | 6 | | | | 6 |
| Wire | 196 | 82 | 1085 | 91 | | 18 | 1472 |
| JJ 数 | 675 | 157 | 1935 | 314 | 58 | 38 | 3177 |
| 面積 ($\mu$m$^2$) | 265500 | 82800 | 709200 | 129600 | 25200 | 62100 | 1274400 |
| バイアス電流 (mA) | 69.7 | 15.3 | 208.5 | 32.2 | 5.4 | 5.0 | 336 |
| 遅延 (ps) | 98.7 | 28.0 | 147 | 61.1 | 10.4 | | 215 |

9 層構造の 1.0 $\mu$m プロセス [13], [14] 向けに開発された SFQ セルライブラリ [15] を用いた．このセルライブラリは，4 K での動作を想定し、高速動作を優先してバイアス電圧は 2.5 mV で設計されている．本アプリケーションに対しては，このような熱雑音に対する大きなマージンや高速動作性は必要ない．ここでは，既存のセルライブラリによる詳細設計の結果を示し、消費電力低減手法を適用した場合の消費電力については 6.3 節で述べる．

7 ビットの Reg を持つ Unit を設計し，Josephson simulator (JSIM) [16] を用いて SPICE レベルのシミュレーションにより動作を検証した．表 2 は 1 つの Unit に必要な JJ 数，面積，バイアス電流，遅延をモジュール毎に示している．また，図 7 は Unit のレイアウト図である．1 つの Unit は 3177 JJ から成り，その回路面積は 1.274 mm$^2$ である．最大遅延は 215 ps で，最大動作周波数は 5 GHz 程度であり，後述するようにオンライン QEC に要求されるスピードを満たす．SFQ ロジックで実装した場合の Unit の消費電力は，336 mA $\times$ 2.5 mV = 840 $\mu$W である．

## 6. 評価

### 6.1 実行サイクル数

表 3 はいくつかの符号距離 $d$ および物理エラー率 $p$ の組み合わせについて，QECOOL の 1 層あたりの復号プロセスにかかる実行サイクル数を示している．実行サイクル数は $d$ および $p$ に強く依存していることがわかる．

1 回の測定プロセスには 1$\mu$s 程度の時間がかかると報告されている [17]．つまり，1 回の復号プロセスが 1$\mu$s 以内に終了すれば十分であるといえる．

**表 3** 1 層あたりの QECOOL の実行サイクル数

| | $p = 0.001$ | | | $p = 0.005$ | | | $p = 0.01$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $d$ | Max | Avg | $\sigma$ | Max | Avg | $\sigma$ | Max | Avg | $\sigma$ |
| 5 | 104 | 6.10 | 4.99 | 144 | 10.4 | 11.2 | 166 | 15.6 | 15.8 |
| 7 | 303 | 11.8 | 14.5 | 515 | 28.7 | 30.1 | 557 | 47.4 | 43.9 |
| 9 | 800 | 22.7 | 30.6 | 1018 | 64.2 | 57.7 | 1308 | 107 | 89.7 |
| 11 | 996 | 41.6 | 53.6 | 1779 | 120 | 95.3 | 2435 | 201 | 161 |
| 13 | 1890 | 71.3 | 82.9 | 3289 | 199 | 147 | 4072 | 337 | 266 |

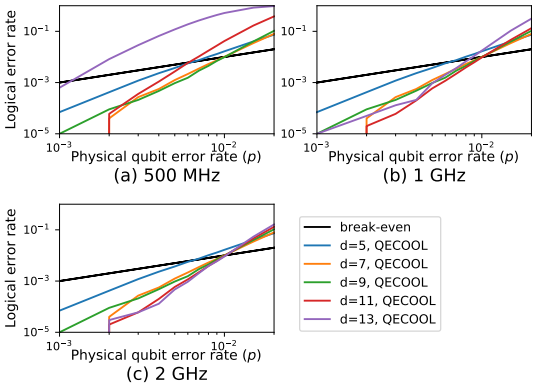


**図 8** オンライン QECOOL のエラー訂正性能

### 6.2 エラー訂正性能

オンライン QECOOL のエラー訂正性能の評価は 4.3 節と同様にして行う．測定プロセスは 1 $\mu$s に 1 度実行されると仮定する．各 Unit は 7 ビットの Reg を持っており，Algorithm 1 中の $th_v$ は 3 とする．測定プロセスに対して復号プロセスが遅く，Reg がオーバーフローした場合にはエラー訂正は失敗したとみなす．

図 8(a) と (b) から，動作周波数が小さいと復号プロセスが遅くなり，符号化距離 $d$ が大きい場合には Reg のオーバーフローによりエラー訂正性能を悪化させていることが読み取れる．図 8(c) でのみオンライン QECOOL のしきい値を $p_{th} = 0.01$ 程度と確認でき，その値はバッチ QECOOL よりも若干小さい．

### 6.3 ERSFQ で実装した場合の消費電力

より多くの復号器を希釈冷凍機の中に配置し，極低温環境で保護できる論理ビットの数を増やすためには，復号器はより低消費電力である必要がある．そこで，より消費電力を低減した SFQ 回路である ERSFQ ロジック [18] を用いて復号器を実装することで，復号器の消費電力を削減する．通常の SFQ 回路では，バイアス電流の制御に抵抗が用いられており，これが大きなスタティック電力消費の原因となっている．ERSFQ ロジックでは，抵抗の代わりに JJ を使って電流を制御する．JJ のスイッチに起因するダイナミックな消費電力は通常の SFQ の 2 倍程度になるが，抵抗による電力消費がゼロになるため，QECOOL のように要求される動作周波数がそれほど高くない場合は、大幅な電力削減が可能となる。

前章の SFQ 回路による設計および ERSFQ の消費電力モデル [19] に基づき，ERSFQ 回路で QECOOL の Unit を設計した場合の消費電力を見積もる．ERSFQ の消費電力 $P$ は以下の式で見積もられる．

$P =$ (バイアス電流) $\times$ (動作周波数) $\times \Phi_0 \times 2$

磁束量子 $\Phi_0$ は $2.068 \times 10^{-15}$ Wb とする．また，式中の $\times 2$ は ERSFQ のダイナミックな消費電力が通常の SFQ のそれの 2 倍程度となることを表している．

表 2 より，1 つの Unit のバイアス電流は合計で 336 mA である．動作周波数は 2 GHz を仮定すると，4 K 環境における Unit1 つあたりの消費電力は，以下のようになる．

$$336\ \text{mA} \times 2\ \text{GHz} \times (2.068 \times 10^{-15})\ \text{Wb} \times 2 = 2.78\ \mu\text{W}. \quad (1)$$

**表 4** いくつかの Surface code 復号器の簡潔な比較

| 復号アルゴリズム | しきい値 (2-D / 3-D) | 遅延 | 動作環境 |
|---|---|---|---|
| MWPM[4] | 10.3% / 2.9% | High | Software |
| UF[8] | 9.9% / 2.6% | Medium | FPGA [9] |
| AQEC[5] | 5% / - | Very low | SFQ |
| QECOOL | 6.0% / 1.0% | Low | SFQ |

**表 5** 提案手法 QECOOL と AQEC[5] の性能比較

| | $p_{th}$ (2-D/3-D) | 1 層あたりの実行時間 (Max/Avg.) ($ns$) | ユニットあたりの消費電力 ($\mu W$) | 論理ビットあたりのユニット数 | 保護できる論理ビット数 |
|---|---|---|---|---|---|
| AQEC | 5.0% / - | 19.8 / 3.93 | 13.44 | $(2d-1)^2$ | 37 |
| QECOOL (7-bit Reg) | 6.0% / 1.0% | 400 / 20.8 | 2.78 | $2d(d-1)$ | 2498 |

### 6.4 既存の復号器との比較

MWPM および効率的なハードウェア実装を目指した復号器と提案手法の簡潔な比較を表 4 に示す．MWPM および Union-Find (UF) は比較的高いしきい値を持つが，ソフトウェアや FPGA などで実装することが必要であり動作環境は室温環境である．

AQEC[5] と我々の提案した QECOOL はともに極低温環境での動作を目指しており，他の手法に比べてしきい値が低くても高いスケーラビリティを持つと考えられる．表 5 に AQEC と QECOOL の詳細な比較についてまとめる．ここでは，符号化距離 $d$ は AQEC の論文で評価されているもののうち最大の 9，物理エラー率 $p$ はオンライン QECOOL のしきい値の 0.1 倍である 0.001 を想定する．希釈冷凍機の 4 K 層で許容される消費電力は 1 W とし [20]，保護できる論理ビットの数を許容消費電力に対する復号器の消費電力の観点から見積もる．AQEC が解けるのは 2 次元平面上のマッチング問題のみであるので，3 次元格子上のマッチング問題へと応用するには 4.3 節と同様の議論を適用して 7 倍のリソースが必要であると仮定する．また，AQEC が 3 次元格子上のエラー訂正を行う場合の 1 層あたりの実行時間については，2 次元平面上のエラー訂正を行うのに必要な実行時間と同等とするが，これは非常に楽観的な見積もりである．表は実行時間以外の項目で QECOOL のほうが AQEC よりも優れていると示しているが，どちらも観測プロセスの間隔である 1 $\mu$s 以内に 1 層の復号プロ

セスが終了しているので，十分高速であると言える．また，AQEC は QECOOL と同程度の実行時間になるように動作周波数を小さくすることで消費電力を小さくするアプローチを取ることもできるが，それを考慮しても QECOOL のほうが低消費電力である．

## 7. まとめと今後の展望

本稿では，Surface code のオンライン復号アルゴリズムである QECOOL を提案した．また，QECOOL を実行する復号器を SFQ 回路を用いて設計し，消費電力やエラー訂正性能を評価した．その結果，我々の復号器は 4K 環境で符号距離 9 の論理ビットを 2500 個程度保護できるほど低消費電力で，かつ十分な速さで動作することが分かった．

本稿の手法も含め，これまでに提案されてきた復号器の多くは単一論理ビットの保護のみを目的としていたが，論理ビット同士の演算ができなければ量子計算機で意味のある計算を行うことができない．今後の課題として，Lattice Surgery[21] などを用いた論理ビット同士の演算をサポートする復号器の開発が挙げられる．

## 謝辞

本研究は，JST 未来社会創造事業 課題番号 JPMJMI18E1，JST CREST 課題番号 JPMJCR18K1，JST PRESTO 課題番号 JPMJPR1916, JST ERATO 課題番号 JPMJER1601, MEXT Q-LEAP 課題番号 JPMXS0120319794 および JPMXS0118068682 によるものである．

### 参考文献

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. GSL. Brandao, D. A. Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[2] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[3] A. G. Fowler, M. Mariantoni, J. Martinis, and A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, 2012.

[4] A. G. Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average O(1) parallel time. *Quantum Info. Comput.*, 15(1–2), 2015.

[5] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong. NISQ+: Boosting quantum computing power by approximating quantum error correction. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 556–569, 2020.

[6] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 679–691, 2017.

[7] S. S. Tannu, D. M. Carmean, and M. K. Qureshi. Cryogenic-DRAM based memory system for scalable quantum computers: A feasibility study. In *Proceedings of the International Symposium on Memory Systems*, page 189–195, 2017.

[8] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv preprint arXiv:1709.06218*, 2017.

[9] P. Das, C. A. Pattison, S. Manne, D. Carmean, K. Svore, M. Qureshi, and N. Delfosse. A scalable decoder micro-architecture for fault-tolerant quantum computing. *arXiv preprint arXiv:2001.06598*, 2020.

[10] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13(4):475–493, 1983.

[11] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[12] A. Madhavan, T. Sherwood, and D. Strukov. Race logic: A hardware acceleration for dynamic programming algorithms. In *2014 ACM/IEEE 41st Annual International Symposium on Computer Architecture (ISCA)*, pages 517–528, 2014.

[13] S. Nagasawa, K. Hinode, T. Satoh, M. Hidaka, H. Akaike, A. Fujimaki, N. Yoshikawa, K. Takagi, and N. Takagi. Nb 9-layer fabrication process for superconducting large-scale SFQ circuits and its process evaluation. *IEICE Transactions on Electronics*, E97.C(3):132–140, 2014.

[14] A. Fujimaki, M. Tanaka, R. Kasagi, K. Takagi, M. Okada, Y. Hayakawa, K. Takata, H. Akaike, N. Yoshikawa, S. Nagasawa, K. Takagi, and N. Takagi. Large-scale integrated circuit design based on a Nb nine-layer structure for reconfigurable data-path processors. *IEICE Transactions on Electronics*, E97.C(3):157–165, 2014.

[15] Y. Yamanashi, T. Kainuma, N. Yoshikawa, I. Kataeva, H. Akaike, A. Fujimaki, M. Tanaka, N. Takagi, S. Nagasawa, and M. Hidaka. 100 GHz demonstrations based on the single-flux-quantum cell library for the 10 kA/cm$^2$ nb multi-layer process. *IEICE Transactions on Electronics*, E93.C(4):440–444, 2010.

[16] E. S. Fang and T. Van Duzer. A josephson integrated circuit simulator (JSIM) for superconductive electronics application. In *Extended Abstracts of 1989 International Superconductivity Electronics Conference (ISEC'89)*, pages 407–410, 1989.

[17] C. Gidney and M. Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *arXiv preprint arXiv:1905.09749*, 2019.

[18] D. E. Kirichenko, S. Sarwana, and A. F. Kirichenko. Zero static power dissipation biasing of RSFQ circuits. *IEEE Transactions on Applied Superconductivity*, 21(3):776–779, 2011.

[19] O. A. Mukhanov. Energy-efficient single flux quantum technology. *IEEE Transactions on Applied Superconductivity*, 21(3):760–769, 2011.

[20] J. M. Hornibrook, J. I. Colless, I. D. Conway Lamb, S. J. Pauka, H. Lu, A. C. Gossard, J. D. Watson, G. C. Gardner, S. Fallahi, M. J. Manfra, and D. J. Reilly. Cryogenic control architecture for large-scale quantum computing. *Phys. Rev. Applied*, 3:024010, 2015.

[21] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.