誤り訂正符号による演算器の小面積耐故障設計法

吉田 卓矢[1,a)]　山森 一人[2,b)]　相川 勝[3,c)]

概要：宇宙環境下などでの稼働を想定した機器の設計は，重量などの制限を満たし，かつ耐故障機能を有することが不可欠である．本稿では，面積コストの小さい演算器の耐故障化を目的に，誤り訂正符号の一種であるブロック符号による誤り検出と誤り訂正を応用した耐故障設計を提案する．提案手法では，対象の演算器の演算結果と冗長符号生成器によって符号語を生成し，それを復号することで故障を検出する．また，復号結果から誤り訂正を行うことで故障を隠蔽する．提案手法は三重化設計より面積コストが小さく，またシミュレーションにより故障の検出と隠蔽が可能であることを示す．

キーワード：高信頼化技術，誤り訂正符号，耐故障設計，小面積設計

Small Area Fault-tolerant Design for Arithmetic Devices by Error-correction Code

Takuya Yoshida[1,a)]　Kunihito Yamamori[2,b)]　Masaru Aikawa[3,c)]

Abstract: Fault-tolerance is a key technology for equipments working in harsh environments. Combination of triple modular redundancy and fine-grained design shows high fault tolerance, but additional circuits occupy large area resources. In this paper, we propose a fault tolerant design for the arithmetic devices with small area resources. We focus on error detection and error-correction by error correction code. Our method directly generates error-correction code for target arithmetic device from input data. Device failure is detected by decryption of error-correction code, and errors by device failure are corrected by decrypted results. We evaluated our method by implementation and simulation.

Keywords: high-reliability, error correction code, fault-tolerant design, small area design

1. はじめに

近年の技術革新に伴い，宇宙探査機や災害救助用のレスキューロボットなど，劣悪な環境下での長期稼働を想定した機器が数多く設計・運用されている．こういった機器では，故障の修繕やメンテナンスを人手で行うことが非常に困難であるため，耐故障を考慮した設計が行われる．特に，宇宙環境下で稼働させる機器の設計では，消費電力や総重量など，地上での稼働を想定した機器よりも厳しい制限の下で設計を行う必要がある．また，放射線の影響による故障の発生確率の上昇も考慮する必要があるため，耐故障技術の向上は非常に重要かつ急務である [1]．

耐故障設計は，故障の検出や診断，復旧や隠蔽などの機能を内蔵することで，機器の機能を維持する設計である．故障の復旧は，故障の種類に応じて異なる対応をとる場合が多く，一時的な故障には内部状態のリフレッシュ，永久的な故障には予備部品との交換といった対応がとられる．また，故障検出手法としては，一定の領域を順次診断する Roving STAR [2] や，複数の同一回路の出力比較によって故障を推察する多重化などが用いられる．中でも多重化は，

1 宮崎大学 工学研究科
Graduate Engineering, University of Miyazaki, Japan
2 宮崎大学 工学教育研究部
Faculty of Engineering, University of Miyazaki, Japan
3 宮崎大学 工学部 教育研究支援技術センター
Technical Center, Faculty of Engineering, University of Miyazaki, Japan
a) yoshida@taurus.cs.miyazaki-u.ac.jp
b) yamamori@taurus.cs.miyazaki-u.ac.jp
c) aikawa@taurus.cs.miyazaki-u.ac.jp

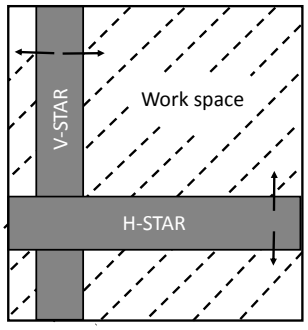


図 1　Roving STAR による故障診断.
Fig. 1　Fault diagnosis by Roving STAR.

単純な設計でありながら，故障検出機能と故障隠蔽機能を内蔵した手法であり，近年の研究では多重化の一つである三重化（TMR : triple modular redundant）[4] をベースとした研究が多く行われている．また，設計粒度を細かくすることで故障の影響範囲を小さくするとともに，予備回路を多く内蔵する細粒度設計と併せた細粒度 TMR [5] は高い信頼性を示す．しかし細粒度 TMR による設計では，設計粒度を細かくするほど故障判定回路を増設する必要があり，それに伴って面積規模が増大するという欠点がある．

本稿では，耐故障性と面積効率の両立を目的とし，誤り訂正符号を応用した故障検出および故障隠蔽機能による耐故障設計法を提案する．設計ツールを用いて TMR と提案手法による回路設計を行い，回路の面積規模を比較することで面積効率の定量的な評価を行うとともに，動作シミュレータによる稼働実験を行うことで耐故障機能の検証を行う．

2. 関連研究

2.1 Roving STAR

Roving STAR [2] は，FPGA（field-programmable gate array）などの動的再構成機能を持つデバイスを前提とした設計法であり，STAR（Self-Testing ARea）と呼ばれる故障診断対象の回路領域を順次移動させることで，全領域の故障検出を行う自己診断型の故障検出手法の一つである．

処理手順としては，まず，デバイスの未使用領域を最初の STAR として診断を行う．それが完了すると隣の領域の機能を現在の領域に複製し，隣の領域を次の STAR として診断を行う．これを繰り返し，STAR が全領域をくまなく巡回することで故障検出を行う．Roving STAR は，未使用領域の確保や機能の複製の際にデバイスの動的再構成機能を用いることで，全体の処理に影響を与えることなく全体の故障診断を行える手法である．

2.2 多重化設計

多重化設計 [3] は，複数の同一回路の処理結果を比較することで故障の検出と隠蔽を行う手法である．複数の同一回路による冗長設計を総じて多重化設計と称するが，二重化（DMR）と三重化（TMR）以上の設計では，機能的な面で大きな違いがある．

DMR は，同一回路 2 つによって構成されるため，比較する処理結果は 2 つである．両者の不一致判定により，故障検出は可能だが，故障回路の特定はできないため，故障の隠蔽機能を持たない設計である．一方，TMR は回路 3 つによって構成されるため，比較する処理結果は 3 つである．処理結果の多数決により，単一回路の故障に対する隠蔽機能を持つと同時に，不一致判定によって故障回路を把握できる．

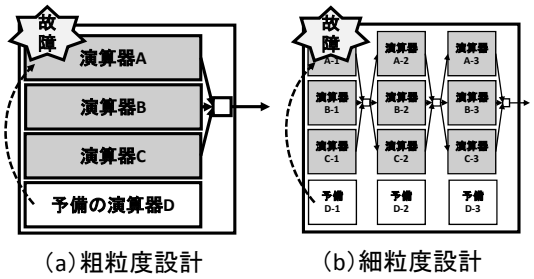


図 2　細粒度設計の例.
Fig. 2　An example of fine-grained design.

2.3 細粒度設計

細粒度設計 [6] は，図 2 に示すように，ある機能を実現する大きな回路を，複数の細かな回路に分割する設計手法である．耐故障手法ではないものの，故障による影響範囲を狭めることができるため，他の設計手法と併せて用いられることが多い．

永久故障の復旧では，予備回路と交換するという対応がとられるため，故障箇所を高い精度で特定する必要がある．TMR の場合，故障回路の特定は可能であるが故障箇所の特定はできないため，故障回路に対して別途故障診断を行う方法が取られる．細粒度設計の TMR では，細分化された回路単位で故障を検出できるため，故障診断時のオーバーヘッドを削減できる．

3. 誤り訂正符号による耐故障設計

3.1 誤り訂正符号

誤り訂正符号 [7], [8] は，ディジタル信号に誤りが発生した際のビット変化を検出し，正しく訂正する符号化技術の一つである．一般的な用途としては，記憶装置やディジタル通信等の信頼性を高めるために用いられる．基本的な原理は，元のディジタル情報（以降，元情報と称する）に基づいた冗長ビットを付加することで符号語を生成し，記憶装置への書き込みや通信を行う．この符号語に対して復号処理を行うことで，符号語に発生した誤りの検出や訂正を行うと共に，元情報を取得する．

一般的に，誤り訂正符号はブロック符号と畳み込み符号

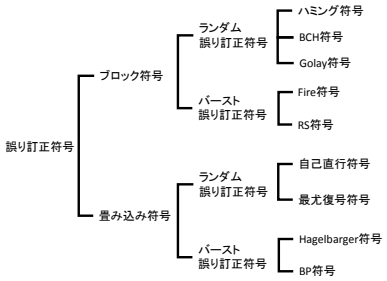


図 3　誤り訂正符号の分類.
Fig. 3　Classification of error correction codes.

の二種類に大別される．また，それぞれにランダム誤りに強い符号とバースト誤りに強い符号があるため，図 3 のように分類できる．

ブロック符号と畳み込み符号について説明する．ブロック符号は，一定のビット長（ブロック）を基準とした符号化手法であり，ブロック符号で用いる符号語の符号長は，元情報のビット長 k に検査符号 m ビットを加えた n ビットとなる．また，ブロック単位で復号処理を行うため短時間での処理が可能である．符号長 n，元情報のビット長 k，誤り訂正可能ビット数 t のブロック符号を (n, k)，あるいは (n, k, t) のブロック符号といい，ブロック符号の多くは，元情報に検査符号を付加した図 4 のような構成になっている．

一方の畳み込み符号は，元情報を連続したビット列として考え，現在の入力ビットと過去に入力された数ビットから符号語を生成する．畳み込み符合の符号化処理では，図 5(a) のような符号化器を用い，現在の入力ビットと符号化器の内部状態 x，y によって符号語 ν，v が決定される．元情報のビット長 k は任意であり，k に応じて符号長 n も決定されるため，符号化や復号処理に長い時間を要する場合もある．畳み込み符合の符号化処理は，現在の入力ビットと符号化器の内部状態によって決定されるため，図 5(b) のような状態遷移図やトレリス線図で表される．

ランダム誤りとバースト誤りについて説明する．ランダム誤りは，誤りの出現頻度に極端な偏りがなく，散発的，あるいは単独的に発生するような性質を持っている．一方のバースト誤りはランダム誤りとは対照的に，誤りの出現頻度に極端な偏りがあり，集中的に発生する箇所とまったく発生しない箇所がはっきり分かれているような性質を持っている．

誤り訂正符号による誤り検出・訂正の例として，ハミング符号 [9] での処理を以下で説明する．

ハミング符号は，1 ブロック当たり 1 ビットの誤り検出と訂正を行うことのできる誤り訂正符号である．ハミング符号の元情報のビット長 k と符号長 n は，式 (1) によって決定される．

$$k = n - m, \quad n = 2^m - 1. \tag{1}$$

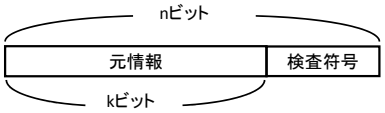
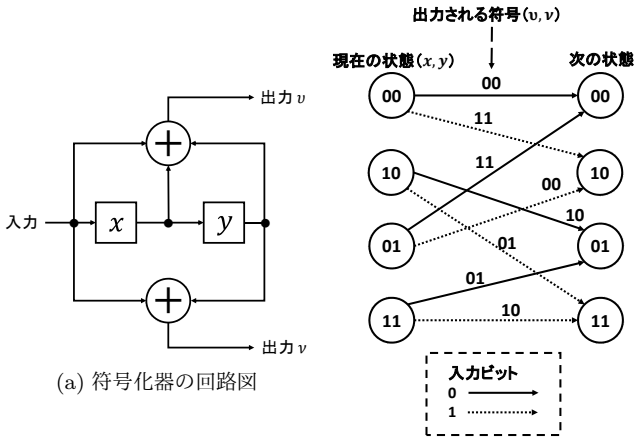


図 4　(n, k) のブロック符号の構成.
Fig. 4　Configuration of (n, k) block code.



(a) 符号化器の回路図
(b) 符号処理の状態遷移図

図 5　畳み込み符号の符号化処理例.
Fig. 5　An examples of convolutional code generation.

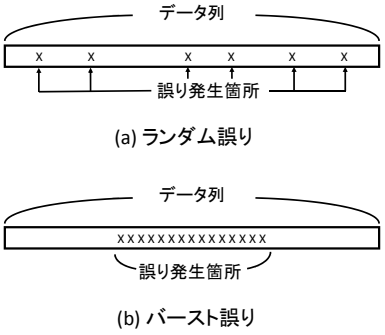


図 6　ランダム誤りとバースト誤りの例.
Fig. 6　An examples of random error and burst error.

ここで，m は任意の整数を表す．

ハミング符号の符号化および復号化処理には，0 か 1 を要素に持つ検査行列 H と生成行列 G という 2 種類の行列を用いる．検査行列は m 行 n 列の行列であり，すべての要素が 0 の行や列がなく，かつ各行および各列が相違であるという条件がある．上記の条件を満たす行列であれば，任意の行列を用いることが可能であるが，生成行列の決定を容易にするため，一般的には式 (2) に示す組織行列と呼ばれる行列を用いる場合が多い．式 (2) の I_m は $m \times m$ の単位行列であり，A は $m \times k$ の任意の行列である．

$$H = [A \; I_m]. \tag{2}$$

一方の生成行列 G は，式 (3) の条件を満たす非零の行列である．

$$H \; G^\top = G \; H^\top = 0. \tag{3}$$

ここで，$G^\top$ は生成行列 G の転置を表す．

検査行列と同様に，式 (3) の条件を満たす行列であれば任意の行列を用いることが可能であるが，検査行列 H が組織行列の場合，生成行列 G は式 (4) に示す行列になる．

$$G = [I_k \ A^\top]. \tag{4}$$

ハミング符号における符号語は，元情報と生成行列 G との乗算によって生成される．誤り検出は，符号語と検査行列の転置行列 $H^\top$ の乗算による復号処理の結果を用いて行う．なお，各行列計算で行われる加算はすべて排他的論理和を用いる．

符号化前の元情報を X とし，符号化処理によって生成される符号語を Y とした場合，Y は式 (5) で求めることができる．

$$Y = X \ G. \tag{5}$$

復号処理は，検査行列の転置行列との乗算を行うことと，式 (3) の条件から式 (6) となる．

$$Y \ H^\top = X \ G \ H^\top = 0. \tag{6}$$

Y に 1 ビットの誤りが発生した場合の符号語を Y' とした場合，Y' は式 (7) で求めることができる．

$$Y' = X \ G \oplus e_i. \tag{7}$$

ここで，e_i は i ビット目のみ 1 である単位ベクトルを表す．Y' に対し復号処理を行った結果は式 (8) となる．復号結果が 0 ではないため，誤りが発生していることが分かる．

$$Y' \ H^\top = (X \ G \oplus e_i) \ H^\top = e_i \ H^\top = e_i \ H^\top \neq 0. \tag{8}$$

H は各行各列が相違であることから，復号結果は発生した誤りの箇所によって異なることとなるため，ハミング符号は 1 ビットの誤り検出および訂正が可能な符号と言える．また，ハミング符号は誤り検出だけであれば 2 ビットまでの誤り検出が可能な符号でもある．

Y に 2 ビットの誤りが発生した場合の符号語 Y'' は式 (9) で表すことができ，その復号結果は式 (10) となる．

$$Y'' = X \ G \oplus e_i \oplus e_j. \tag{9}$$

$$Y'' \ H^\top = e_i \ H^\top \oplus e_j \ H^\top. \tag{10}$$

この時，$e_i \neq e_j$ であるため $H^\top$ の任意の 2 つの行の排他的論理和をとることになるが，$H^\top$ の各行は相違であるため結果が零ベクトルになることはなく，誤り検出が可能であることが分かる．ただし，$H^\top$ の各行は 0 ベクトル以外のすべての要素を含むため，2 ビットの誤り時の復号結果は式 (11) となる．

$$e_i \ H^\top \oplus e_j \ H^\top = e_k \ H^\top. \tag{11}$$

式 (11) は見かけ上式 (8) と区別することができず，2 ビット誤りを 1 ビット誤りと誤認する結果となるため，2 ビットの誤り検出と 1 ビットの誤り訂正を両立させることは出来ない．

拡張ハミング符号は，ハミング符号の符号語に偶数パリティビットを加えることで，2 ビットの誤り検出と 1 ビットの誤り訂正を同時に行うことのできる誤り訂正符号である．誤りが発生していない場合，復号後のパリティビットは 0 になるが，誤りが発生した場合は発生した誤りの数だけ値が反転する．そのため，誤り検出時にパリティビットが 1 の場合は，1 ビットの誤りとしてハミング符号の復号結果に従って誤り訂正を行うことができる．また，パリティビットが 0 の場合は 2 ビット誤りであるため，復号結果による訂正を行うことは出来ないと判断できる．誤りが検出されていないにも関わらずパリティビットが 1 である場合は，パリティビットに対する 1 ビットの誤りが発生したと判断できるため，パリティビット自身の誤り検出および訂正も可能である．

4. 誤り訂正符号による故障検出と誤り訂正

演算器故障による演算結果の誤りを訂正する場合，符号語生成時には演算結果を用いることが考えられる．また，訂正可能な誤りは式 (8) のように符号語に発生したものだけである．そのため，演算結果を式 (5) の元情報 X として生成された符号語では，演算器の故障を検出することはできない．

この問題を解決するにあたって，ブロック符号の構造に着目する．ブロック符号の符号語 Y の構造は，元情報 X に検査符号 $\mathbf{z}$ を付加した，式 (12) の形で表すことができる．

$$Y = [\ X \ \mathbf{z}\ ]. \tag{12}$$

故障のない状態で得られるはずの演算結果に対応する検査符号 $\mathbf{z}$ をあらかじめ用意しておくことで，演算器の故障検出を行うことのできる符号語の生成が可能となる．

演算器に故障が発生していないとき，入力 $\mathbf{x}$ に対する演算 f の結果 $\mathbf{y}$ は式 (13) として表すことができる．

$$\mathbf{y} = f(\mathbf{x}). \tag{13}$$

また，この演算結果 $\mathbf{y}$ に対応した検査符号 $\mathbf{z}$ は生成行列による乗算 $\mathbf{g}$ で表すと式 (14) で表すことができる．

$$\mathbf{z} = g(\mathbf{y}). \tag{14}$$

式 (13) と式 (14) から，検査符号 $\mathbf{z}$ は式 (15) として表すことができ，入力から検査符号を直接生成できる．

$$\mathbf{z} = g(f(\mathbf{x})). \tag{15}$$

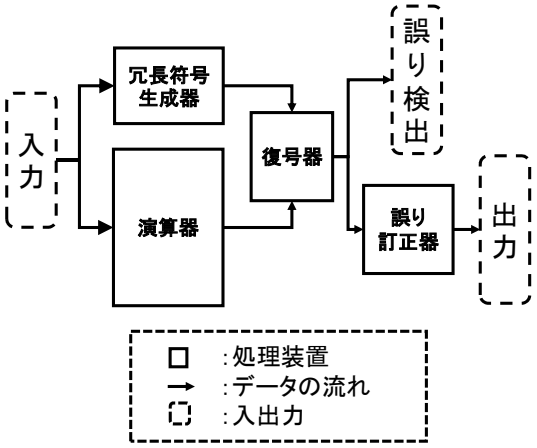


図 7　提案手法の回路構成図.
Fig. 7　Block diagram of proposed method.

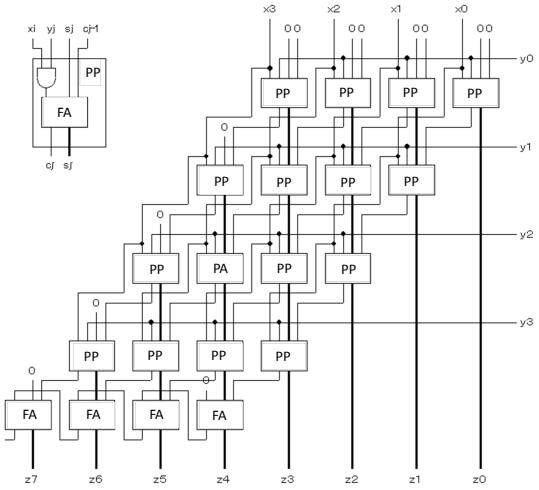


図 8　乗算器の内部構造の例.
Fig. 8　An example of a multiplier.

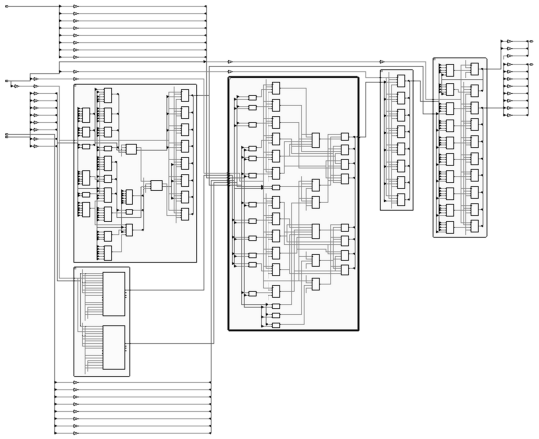


図 9　検証実験用の回路図例.
Fig. 9　An example circuit diagram for evaluation.

本手法では，入力と検査符号の対応を事前に把握し，入力から検査符号を直接生成する検査符号生成器を設計する．対象の演算器に対応した検査符号生成器を併設し，演算器による演算結果と検査符号を並行して算出することで，演算器における故障に対応可能な符号語を生成する．また，符号語を復号することで演算器の故障検出を行うと共に，誤り訂正による故障隠蔽を行う．本手法での回路構成を図 7 に示す．本手法により設計される回路では，次のような処理が行われる．

- 演算器と冗長符号生成器に入力データを送り，演算結果と冗長符号を復号器に送る．
- 復号器で演算結果と冗長符号を連結して符号語を作成し，これを復号することで誤り検出を行う．誤りが検出された場合，故障があると判断できる．
- 復号結果を誤り訂正器に送り，演算結果の訂正を行うことで正しい値を出力する．なお，誤りがない場合は特になにも行わない．

5. 評価実験

5.1 実験内容

提案手法の定量的な評価を行うため，設計ツールを用いて回路の設計および耐故障シミュレートを行う．評価する項目は，設計回路の面積規模の比較，故障の検出・隠蔽機能の確認の 2 点である．

設計回路の面積規模の評価は，対象となる各演算装置に対し，提案手法および TMR により設計した回路を比較することで行う．比較に用いる演算装置には，乗算器，デマルチプレクサ，ALU の 3 種を用い，設計した回路に使用される論理セル数を比較する．なお，乗算器には図 8 に示す 4 ビット乗算器を用い，デマルチプレクサには入力 1 ビット，制御信号 3 ビット，出力 8 ビットのものを用いる．また，ALU には STMicroelectronics 社製の 4 ビット ALU である 74HC181 [10] を用いる．

故障の検出・隠蔽機能の確認では，提案手法により設計された回路の演算装置と復号器の間に疑似故障生成器を追加する．疑似故障生成回路によって演算結果に誤りを付加することで，演算器の疑似的な故障を発生させる．この誤りを提案手法により設計された回路が正しく検出・隠蔽ができるかを確認する．なお，疑似故障の発生は宇宙環境下を想定した故障発生率を元に行う [11] [12] [13].

故障生成回路を追加した例として，4 ビット乗算器を対象とした場合の回路構成図を図 9 に示す．図 9 中の太線で囲まれた回路ブロックが疑似故障生成器であり，その左上部が冗長符号生成器，左下部が 4 ビット乗算器である．また，故障生成回路の右隣りは復号器であり，復号器の右隣りが誤り訂正器である．

5.2 実験環境

提案手法および TMR による設計と耐故障シミュレーションは，Xilinx 社の FPGA 設計開発ツールである Vivado Design Suite を用いて行う．また，設計言語には Verilog HDL を使用し，実装対象となる FPGA には Virtex-7 を想

表 1　実装環境の諸元.
Table 1　Experiment environments.

設計ツール	Vivado Design Suite 2014.3
対象 FPGA	Virtex-7 （XC7V585T FFG1157 – 2L）
設計言語	Verilog HDL
対象回路	4 ビット乗算器 8 ビット出力デマルチプレクサ 4 ビット ALU
誤り訂正符号	(8, 4) ハミング符号

定する．実験環境の諸元を表 1 に示す．

5.3　回路規模の比較

各対象回路に対し，提案手法によって設計された 4 ビット乗算器，8 ビット出力デマルチプレクサ，4 ビット ALU の回路図を図 10，図 11，図 12 にそれぞれ示すと共に，TMR と各設計での回路規模の比較を図 13 に示す．

図 10 に示した 4 ビット乗算器は，大きく 4 つの回路ブロックから構成されており，左から乗算器，冗長符号生成器，復号器，誤り訂正器となっている．また，図 11 で示した 8 ビット出力デマルチプレクサや図 12 で示した 4 ビット ALU も同様の構成となっている．なお，回路図によって配線や回路ブロック内部の構成などの差異が存在するが，これは Vivado Design Suite による設計回路の最適化を行った結果であり，耐故障機能や面積規模への影響はなかった．

図 13 から，4 ビット乗算器および 4 ビット ALU の場合，提案手法による設計は TMR による設計の約 83% の面積規模となっていることが確認できる．しかし，8 ビット出力デマルチプレクサを対象とした場合，提案手法による設計が TMR による設計の約 129% の面積規模となり，回路規模が増大している．

この原因として，提案手法によって設計された冗長符号生成器や復号器と対象演算器との相対的な面積規模の差が挙げられる．誤り訂正符号における検査符号は，対象となる演算器の出力ビット数と誤り訂正能力に比例して増加する傾向にある．今回の実験では，各演算器の出力ビット数や用いた符号の誤り訂正能力は同じであるため，対象演算器の異なる実験であっても，検査符号生成器の面積規模は同程度になっている．しかし，対象演算器自体の面積規模は各々であり，特にデマルチプレクサは比較的小規模な演算器である．そのため，相対的に検査符号生成器などは大規模な回路となり，面積規模の増加につながったと考えられる．

5.4　耐故障機能の検証

耐故障機能の検証に関しては，耐故障シミュレーションの結果である波形図を元に説明する．波形図では，横軸に

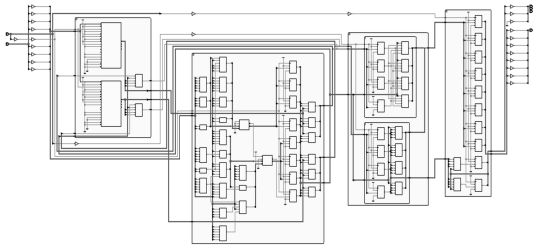


図 10　提案手法による 4 ビット乗算器の回路図.
Fig. 10　Circuit diagram of multiplier by proposed method.

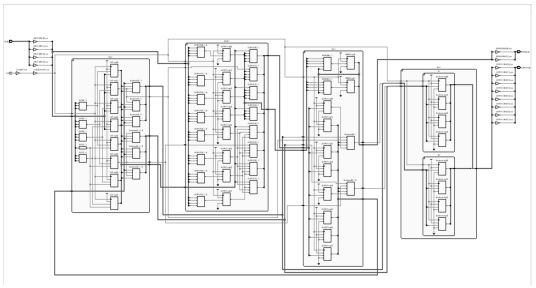


図 11　提案手法による 8 ビットデマルチプレクサの回路図.
Fig. 11　Circuit diagram of demultiplexer by proposed method.

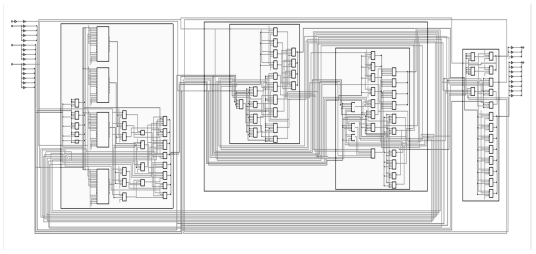


図 12　提案手法による 4 ビット ALU の回路図.
Fig. 12　Circuit diagram of ALU by proposed method.

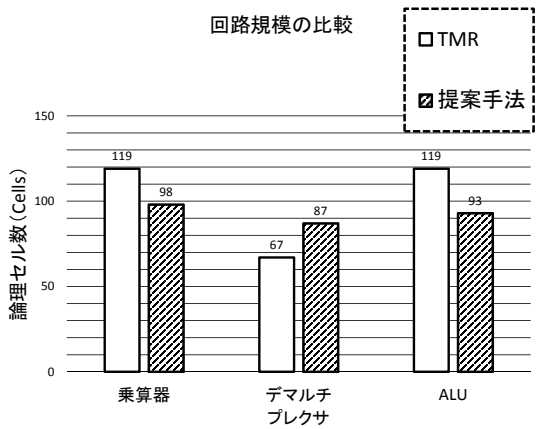


図 13　回路規模の比較.
Fig. 13　Circuit size of each design.

時間をとり，左端の列 name に示す各変数の値を表している．各変数はそれぞれ表 2 に示す役割を持っている．

耐故障機能の確認については，一例として乗算器による実験結果を図 14 と図 15 に示す．図 14 では，疑似故障を表す ERROR と故障検出を表す ERROR_SIGNAL に注目する．

表 2　波形図における各変数の説明.
Table 2　Variables in waveform diagram.

変数名	説明
CLKs	演算器や冗長符号生成器などの制御信号群.
INPUT	演算器や冗長符号生成器への入力. 演算器毎にビット数や各ビットの役割は異なる.
ERROR	疑似故障による誤り箇所.
SEED TMP	疑似故障生成用の乱数.
ADD_ERROR	故障による誤りを含む演算結果. ADD_ERROR = TLUE_RESULT ⊕ ERROR
ENCODE_RESULT	生成された符号語.
DECODE_RESULT	符号語の復号結果.
ERROR_CORRECTED	復号後の誤り訂正処理の結果.
ERROR_SIGNAL	故障の検出結果.
TRUE_RESULT	故障による誤りがない場合の演算結果.

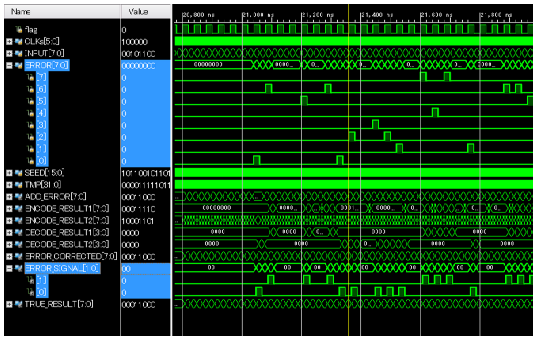


図 14　故障検出機能の確認.
Fig. 14　Simuration for fault detection.

図 14 の縦線は，ERROR[2] が 0 から 1 に更新される瞬間であり，この直後に ERROR_SIGNAL[0] が 0 から 1 に更新される．このように，故障検出機能は ERROR と ERROR_SIGNAL の対応を見ることで確認できる．

図 15 では，誤り訂正による故障隠蔽機能に関わる各変数に注目する．図 15 の縦線は，INPUT が 00111000 から 00111001 に更新される瞬間であり，同時に ERROR が 00100000 に更新されている．INPUT が 00111001 に更新されるため，正しい出力は 00011011 に更新されるはずであるが，縦線の段階では各処理が行われていないため ERROR_CORRECTED は更新前の出力である 00011000 を示している．しかし，次の更新後には 00011011 を示しているため，正しい出力であることが確認できる．故障隠蔽機能に関しては，ERROR_SIGNAL が故障を示している際の TRUE_RESULT と ERROR_CORRECTED が一致しているかを見ることで確認できる．

各図中では，更新のタイミングが 1 サイクル遅れている変数も確認できる．これは，各回路の処理結果を次のサイクルで取得する同期設計を行った影響であり，実際の処理は CLKs の更新に応じて段階的に行われる．

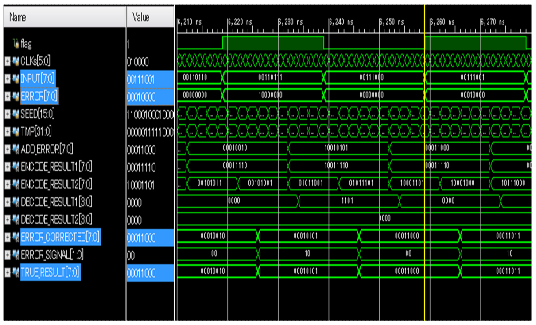


図 15　故障隠蔽機能の確認.
Fig. 15　Simuration for error correction.

6. おわりに

劣悪な環境下での長期稼働を想定した機器において，耐故障設計は必須である．特に宇宙環境下で稼働する機器の設計においては，宇宙放射線による高い故障確率に対する耐故障設計と，総重量や消費電力量を考慮した設計とを両立させる必要がある．細粒度設計と併せた TMR は，耐故障設計として最も優秀とされる手法であるが，粒度を細かくするほど故障判定回路を増設する必要があり，細粒度化に伴って相対的に回路が大規模化するという問題点がある．

本稿では，省面積設計と耐故障設計の両立を目的とし，誤り訂正符号の一種であるブロック符号に着目することで，対象の演算器に合わせた検査符号生成器を併設する耐故障設計法を提案した．また，回路設計ツールである Vivado Design Suite を用いて提案手法と TMR によって設計される回路を Verilog HDL を用いてそれぞれ実装することで回路規模の比較を行い，耐故障シミュレートを行うことで耐故障機能の検証を行った．

回路規模の比較実験の結果から，4 ビット乗算器および 4 ビット ALU を対象とした設計では，TMR に比べ回路規模を約 83% に抑えることができた．また，耐故障機能の検証実験により入力，符号語，出力の各対応を確認し，すべての設計回路において故障検出および故障隠蔽による耐故障機能を有していることを示した．しかし，8 ビット出力デマルチプレクサを対象とした設計では回路規模が TMR の約 129% となり，回路規模が増大する結果となった．これは，提案手法による設計の回路規模が訂正可能ビット数と対象演算器の出力ビット数に依存する関係から，対象演算器に対して提案手法により設計される回路が相対的に大きくなり，TMR に比べ大規模化したと考えられる．

今後の課題としては，他の対象演算器の設計における回路規模の検証，符号語の見直しによる小規模化や耐故障性能の向上などが挙げられる．

参考文献

[1]　独立行政法人 新エネルギー・産業技術総合開発機構 ：機械システム技術開発部．宇宙等極限環境における電子部

品の利用に関する研究開発，2014.
[2] Stroud-C. Emmert, J. and Abramovici-M. Skaggs, B. Dynamic fault tolerance in fpgas via partial reconfiguration. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*.
[3] Jie Han, J. Gao, Pieter Jonker, Yan Qi, and J.A.B. Fortes. Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *Design Test of Computers, IEEE*, Vol. 22, No. 4, pp. 328–339, July 2005.
[4] Vanderkulk-W. Lyons, R. E. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, Vol. 6, No. 2, pp.200–209，1962.
[5] Pettit-D. E. Patterson D. W. Nielsen K. E. Xiaoyin Yao Holbert K. E. Clark L. T. Hindman, N. D. High speed redundant self-correcting circuits for radiation hardened by design logic. In *Radiation and Its Effects on Components and Systems (RADECS), 2009 European Conference on*, pp. 465–472, Sept 2009.
[6] A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *Design Test of Computers, IEEE*, Vol. 22, No. 4, pp. 306–315, July 2005.
[7] 今井秀樹，岩垂好祐，宮川洋共著．符号理論．コンピュータ基礎講座/駒宮安男 [他] 編，No. 18．昭晃堂，1973.
[8] 憲一宮．衛星通信工学．ラテイス，丸善， 新版， 1972.
[9] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, Vol. 29, No. 2, pp. 147–160，1950.
[10] Philips semiconductors. data sheet 74hc/hct181. 入手先 ⟨http://pdf. datasheetcaralog. com/datasheet/74HC181N3. pdf⟩ (2015.02.10).
[11] 道家忠義．放射線工学．電気学会大学講座 / 電気学会編．電気学会， 1971.
[12] 奥田治之．スペースアストロノミー．現代天文学講座，第 12 巻． 宇宙の観測 ; 2．恒星社厚生閣， 1982.
[13] 晃関口．放射線計測概論．原子力工学シリーズ，No. 9．東京大学出版会， 1979.