修士論文

メモリスタを用いたシステムに対する耐故障設計

石坂 守
奈良先端科学技術大学院大学
先端科学技術研究科
知能社会創成科学プログラム

主指導教員: 井上 美智子 教授
ディペンダブルシステム学研究室（情報科学領域）

令和 2 年 3 月 13 日提出

本論文は奈良先端科学技術大学院大学先端科学技術研究科に
修士 (工学) 授与の要件として提出した修士論文である。

石坂 守

審査委員：

井上 美智子 教授　（主指導教員）
太田 淳 教授　（副指導教員）
中島 康彦 教授　（副指導教員）
大下 福仁 准教授　（副指導教員）
新谷 道広 助教　（副指導教員）

メモリスタを用いたシステムに対する耐故障設計*

石坂 守

内容梗概

フォンノイマンボトルネックの先鋭化が進み，既存の計算機システムの性能向上が限界を迎えている．メモリスタという不揮発性メモリ素子で構成される抵抗変化型メモリ（Resistive Random Access Memory，ReRAM）やメモリスタニューラルネットワークは，そのような問題を解決するメモリシステム，アクセラレータとして注目されている．しかし，構成要素であるメモリスタは信頼性に課題を有し，特性ばらつきや短い書き込み寿命などの要因によって過渡故障や永久故障が生じる．メモリスタが故障すると，システムが正しく動作しなくなるため，各システムを耐故障設計することが不可欠となっている．

本論文では，ReRAM とメモリスタニューラルネットワークそれぞれに対して耐故障設計を提案する．ReRAM に対しては，メモリスタと CMOS を利用した誤り訂正符号（Error correcting code, ECC）回路を提案する．提案 ECC 回路では，書き込みが頻繁に発生する回路を CMOS で，その他の回路を高集積なメモリスタ回路を用いて設計し，高い信頼性と小さな面積オーバーヘッドを両立する．

メモリスタニューラルネットワークに対しては，逆方向伝播に対する誤り訂正機能と，既存手法よりも高い精度で故障の位置特定と修復を行うオンラインテスト機能を実現する耐故障設計手法を提案する．提案手法を利用することで，メモリスタニューラルネットワーク中に過渡故障や永久故障がある場合でも，既存手法より高い識別性能を実現できることを示す．

キーワード

メモリスタ，抵抗変化型メモリ（Resistive random access memory, ReRAM），メモリスタニューラルネットワーク，耐故障設計，誤り訂正符号，信頼性

*奈良先端科学技術大学院大学 先端科学技術研究科 修士論文, 令和 2 年 3 月 13 日.

Fault-tolerant Design for Memristor-based Systems*

Mamoru Ishizaka

Abstract

As the von Neumann bottleneck has become more severe, the performance of existing computer systems has reached its limits. Resistive Random Access Memory (ReRAM) and memristor neural network have attracted attention as a memory storage system and an accelerator that can solve such the problem. However, a memristor, which is component of the ReRAM and memristor neural network, has reliability problem, and transient faults or permanent faults occur due to severe process variation and lower write endurance. If a fault occurs in a memristor, the system will not operate properly. Hence, the fault-tolerant design is indispensable for the systems composed of memristors.

This thesis proposes the fault-tolerant design for ReRAM and memristor neural network. For ReRAM, this thesis proposes hybrid CMOS/memristor-based ECC circuit. In the proposed ECC circuit, the blocks with highly frequent write operations are implemented by CMOS and the other blocks implemented by highly-integratable memristor circuits to realize high reliability and low area overhead.

For memristor neural network, this thesis proposes the fault-tolerant design which can realize error correction for backpropagation and an online test which localizes and corrects the faults. Numerical experiments demonstrate the proposed method achieves higher classification accuracy compared with the conventional method.

Keywords:

memristor, resistive randome access memory (ReRAM), memristor neural network, fault-tolerant design, error correcting code, Reliability

*Master's Thesis, Graduate School of Science and Technology, Nara Institute of Science and Technology, March 13, 2020.

目 次

図 目 次

表 目 次

1. 序論

計算機システムにおけるCPUとメモリ間のデータ転送速度の制約であるフォンノイマンボトルネックを解消し，コンピュータの性能を飛躍的に向上できることからメモリスタと呼ばれる不揮発性メモリ素子を用いたシステムが注目されている [2–4]．メモリスタ [5] は，抵抗，キャパシタ，インダクタに次ぐ第4の受動素子と呼ばれる2端子素子で，素子内の電流履歴によって抵抗値が非線形的に変化する [6]．電荷を与えていない状態でも抵抗値は同じ値を保持し続けるため，メモリスタは不揮発であり，データを抵抗値として保持することで記憶素子としての動作が可能となる．メモリスタを用いた代表的なシステムとして，抵抗変化型メモリ（Resistive Random Access Memory, ReRAM）とメモリスタニューラルネットワークがある．ReRAMは，メモリスタで構成される不揮発性メモリで，既存の不揮発性メモリであるフラッシュメモリやSSDと比較して高速な読み書き動作ができる [2,4]．また，クロスバーアレイという3次元積層技術によって高い集積率が得られることから，ストレージクラスメモリ（Storage Class Memory, SCM）という不揮発性メモリとしての応用が期待されている [7]．メモリスタニューラルネットワークは，メモリスタをニューラルネットワークのシナプス（重み）として利用する脳型コンピュータで，メモリスタに重みの値を保持しながら，読み出し動作時に高速な積和演算を行える [8]．機械学習で頻繁に行われる積和演算と重みを保持するメモリ動作を単一のデバイスで実現できることから，メモリスタニューラルネットワークはエネルギー効率の良い機械学習用アクセラレータとして有望視されている [9]．

一方，メモリスタは製造技術が未成熟であることから，長期信頼性において課題を有する [1, 10–15]．メモリスタは，特性ばらつきや製造時の物理的欠陥，書き込み寿命などの要因によって，故障が生じる [1]．メモリスタに故障が生じると，抵抗値に理想的な値を書き込むことができなくなるため，故障を含むメモリスタによってシステムが構成されるとシステム全体の信頼性が低下する．そこで，メモリスタを用いたシステムの信頼性を向上させることが，重要な課題となっている．

メモリスタを用いたシステムの信頼性を向上させる手法に，耐故障設計があ

る [10, 16]．メモリスタを用いたシステムの耐故障設計とは，計算用のセルとは別に，検査用の冗長なセルをシステムに付加することで，計算用セルの故障の有無の検査・出力値の訂正を可能とする設計手法である．システム上の一部のメモリスタセルに故障が発生している場合でも，システムとして正しい動作を行うことができるため，耐故障設計されたシステムの信頼性は向上する．

ReRAM の耐故障設計として，誤り訂正符号（Error Correcting Code, ECC）回路がある [10, 17]．ECC 回路は，従来のメモリシステムでも用いられてきた耐故障設計で，文献 [10] では CMOS で設計された従来の ECC 回路を ReRAM に付加する手法を提案している．ECC 回路によって ReRAM の長寿命化を実現することができるが，CMOS を用いる ECC 回路は面積が大きい．CMOS を用いる ECC 回路をメモリスタによる論理回路 [18–20] によって実現することも可能だが，メモリスタによる論理回路は動作時に多くの書き込みを必要とするため，ECC 回路自体の信頼性の低下に繋がる．

メモリスタニューラルネットワークの耐故障設計としては，X-ABFT（eXtended-algorithm based fault tolerance）が提案されている [1, 16]．X-ABFT は，クロスバーアレイで構成されるメモリスタニューラルネットワークを複数のテストブロックに分割し，各テストブロックの列方向にチェックサムを追加することで実現される．X-ABFT を適用することで，(i) ニューラルネットワークの順方向伝播時の誤り訂正機能と (ii) 各テストブロックで，異なる行に発生した 2 つまでのメモリスタセルの故障の位置を特定して修復するオンラインテスト機能を実現することができる．しかし，X-ABFT は，ニューラルネットワークの学習フェーズで行われる誤差逆伝播動作に対する誤り訂正を考慮していない．誤差逆伝播の結果に誤りが生じた場合，重みの値が間違った値に更新される．また，X-ABFT のオンラインテストは，2 つの故障セルがテストブロック中の同一行で発生した場合に，故障箇所の特定と修復ができない．これらの問題は，メモリスタニューラルネットワークの識別性能の低下を招く．

そこで，本研究ではメモリスタで構成される ReRAM とメモリスタニューラルネットワークそれぞれに対して，既存の問題を解決する耐故障設計を提案する．ReRAM に対しては，メモリスタと CMOS の 2 つを利用して，低面積化を実現し

ながらCMOSによるECC回路と同等の信頼性を実現するECC回路を提案する．提案ECC回路では，メモリスタで設計した場合に多くの書き込みが必要となる回路ブロックをCMOS回路で実現し，その他の回路を集積率の高いメモリスタクロスバーアレイで実現する．これにより，高い信頼性を実現しながら，回路の低面積化を実現することができる．

メモリスタニューラルネットワークに対しては，列方向だけでなく行方向にもチェックサムを付加する，新しい耐故障設計法を提案する．行方向チェックサムの値を利用することで，誤差逆伝播の出力の誤りを順方向伝播と同様に訂正することができるようになる．また，オンラインテスト時に，列チェックサムの値と行チェックサムの値を組み合わせて故障解析を行うことで，同一行で発生しているメモリスタの故障位置を特定し，修復することができる．

本論文の構成は次のとおりである．2章では，本研究で対象としているメモリスタの動作原理，応用例であるReRAM，メモリスタニューラルネットワークの詳細，メモリスタの信頼性課題について述べる．3章では，ReRAMストレージの高信頼化とECC回路の高集積化を目的として，CMOSとメモリスタを用いたハイブリッドECC回路を提案する．4章では，誤差逆伝播を利用するメモリスタニューラルネットワークに対して高い信頼性を実現する耐故障設計手法を提案する．最後に，5章で本論文の結論を述べる．

2. メモリスタ

本章では，本研究で対象としているデバイスであるメモリスタの動作原理とクロスバーアレイという 3 次元積層技術について示す．また，メモリスタを用いた代表的なシステムとして ReRAM とメモリスタニューラルネットワークについて述べ，最後にメモリスタの信頼性課題について述べる．

2.1 メモリスタ

メモリスタは，電流履歴によって抵抗値が変化する 2 端子受動素子である [6]．1971 年に Leon Chua [5] らによってメモリスタの存在が指摘されたが，対応する物理現象を満たす材料や素子は長年発見されていなかった．2008 年に Strukov らによって初めて工業的な製造手法が確立され [21]，その後，メモリスタに関する多くの研究が行われている．

図 1 に，メモリスタの物理的構成と回路記号を示す．メモリスタは，2 つの金属電極の間に二酸化チタン（TiO_2）や酸化ハフニウム（HfO_2）といった金属酸化物を挟んだ形で構成される．金属酸化物はドーピングによって発生した酸素欠損を含む低抵抗の領域と，ドーピングされていない高抵抗の領域に分かれる．外部から電圧を印加して内部電流がデバイスに流れる際に，酸素欠損の量が増減し，その境界が移動するため抵抗値が変化すると言われている．メモリスタを回路記号で表現するときは，低抵抗領域側を黒太線で示す．動作例として，図 2 に正弦波交流電圧を与えた時のメモリスタの電流-電圧特性を示す．メモリスタに対してあるしきい値（V_{SET}）以上の電圧を印加すると，抵抗値が増加する．この動作を SET 動作と呼び，SET 動作によって抵抗値が最小になった時の状態を導通状態，もしくは，低抵抗状態（Low Resistance State）という．反対に，メモリスタに対してあるしきい値（V_{RESET}）以下の電圧が印加すると，抵抗値が減少する．この動作を RESET 動作と呼び，RESET 動作によって抵抗値が最大になった時の状態を非導通状態，もしくは，高抵抗状態（High Resistance State）という．メモリスタの現在の状態は，Read 電圧と呼ばれる比較的小さな電圧値を印加し，その値をセンスアンプを通して増幅することで得られる．抵抗値によって状態を保持する

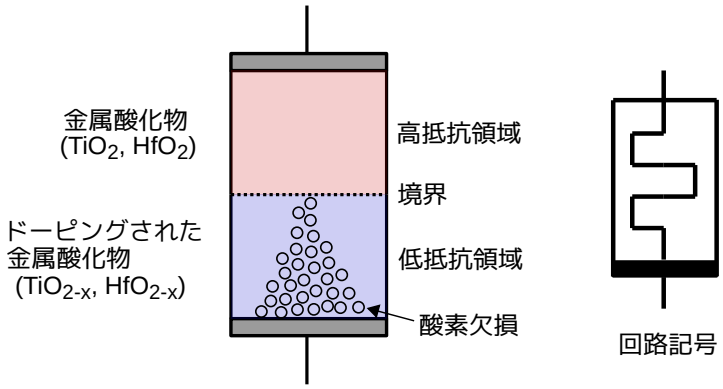


図 1 メモリスタの物理的構成と回路記号

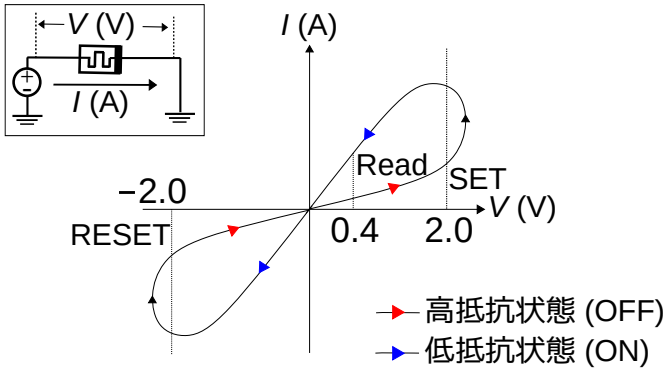


図 2 正弦波交流入力電圧に対するメモリスタの I-V 特性

ため不揮発であり，読み出しに必要な電圧が小さく，記憶保持のための電力が不要であるため低消費電力である．

また，メモリスタはクロスバーアレイという 3 次元積層技術を利用することで，高い集積率を得ることができる [22]．図 3 に M 行 N 列のワイヤによって構成されるクロスバーアレイを示す．クロスバーアレイ構造では，行方向と列方向のワイヤの接点にメモリスタが設置される．メモリスタに対して書き込みを行うときは，書き込み対象のメモリスタが設置されている行と列に対して SET（RESET）電圧を印加する．メモリスタの状態を読み出すときは，Read 電圧を対応する行方向のワイヤに印加する．このとき，列方向に流れる微小な電流をセンスアンプによって増幅することで，メモリスタの状態を読み出すことができる．

また，書き込みと読み出し動作に加えて，メモリスタはクロスバーアレイ構造を

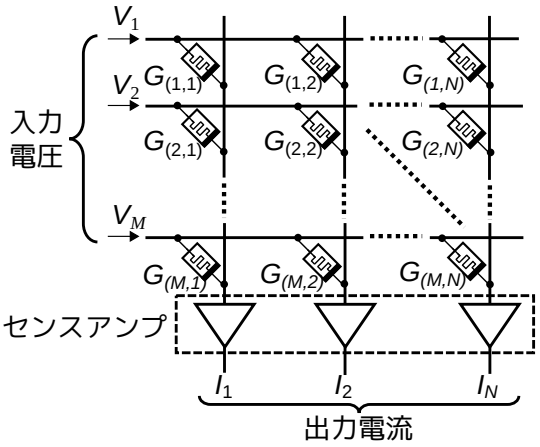


図 3 メモリスタのクロスバーアレイ構造

利用することで効率的な積和演算を行うことができる [22]．クロスバーアレイ中の i 行 j 列目のメモリスタのコンダクタンスを G_{ij} とする（$i = 0, ..., M, j = 0, ..., N$）．このとき，$j$ 列目のワイヤに流れる電流値 I_j は式 (1) のように表される．

$$I_j = \sum_{i=1}^{M} V_i \cdot G_{ij}, \tag{1}$$

ここで，V_i は i 行目のワイヤに印加される入力電圧である．従来の CMOS トランジスタによる積和演算回路と異なり重みが回路内に保存されるため，重みのデータ転送が不要となり，高速な積和演算が実現できる．

2.2 メモリスタの応用例

メモリスタは，その特性を利用して多くのアプリケーションに応用されているが [9]，特に代表的なものとして，ReRAM とメモリスタニューラルネットワークがある．

表 1 不揮発性メモリシステムの性能比較 [2]

	NAND フラッシュ	ReRAM	STT-RAM	FeRAM
Cell size (F^2)	4-6	**4-10**	6-50	6-40
Write Endurance	10^4 - 10^5	10^8 - 10^{11}	10^{12} - 10^{15}	10^{14} - 10^{15}
Read Latency	200-500μs	~10ns	2-35ns	20-80ns
Write Latency	200-500μs	~50ns	3-50ns	50-75ns
Leakage Power	Low	Low	Low	Low
Dynamic Energy (Read/Write)	Low	Low/High	Low/High	Low/High

2.2.1 ReRAM

ReRAM は抵抗値によってデータを保持するメモリスタで構成された不揮発性メモリである．表 1 に不揮発性メモリシステムの性能比較を示す．他の不揮発性メモリと比較したときの ReRAM の特筆すべき点は，セル面積である．ReRAM はクロスバーアレイ構造によって，スピン注入磁化反転メモリ（Spin transfer torque random access memory, STT-RAM）や強誘電体メモリ（Ferroelectric random access memory, FeRAM）といった不揮発性メモリと比較してセル面積を非常に小さくすることができ，NAND フラッシュと同等の集積率を実現することができる．また，読み出しと書き込みのレイテンシーも非常に小さく，特に，読み出しのレイテンシーに関しては 10 ns 程度と他の不揮発性メモリと比較しても非常に小さい．このような利点から，ReRAM は不揮発性メモリとしての利用が期待されている．一方で，書き込み寿命については STT-RAM や FeRAM などの他の不揮発生メモリシステムよりも小さい．揮発性メモリの代わりとして次世代不揮発性メモリを利用するアーキテクチャもあり [2]，不揮発性メモリに対してより多くの書き込

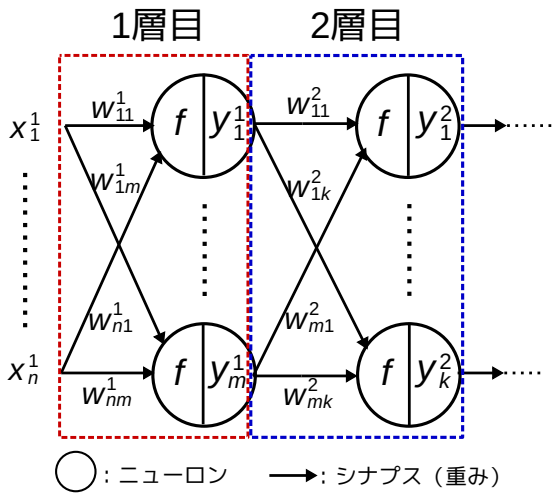


図 4 ニューラルネットワークの構成

みが行われることが考えられるため，ReRAM の書き込み寿命の向上は重要な課題となっている．

2.2.2 メモリスタニューラルネットワーク

メモリスタニューラルネットワークは，ニューラルネットワークの基本演算である積和演算を高速かつ低消費電力に行う，メモリスタで構成された機械学習アクセラレータである [8, 22]．

ニューラルネットワークは関数近似器の一種で，大量の訓練データからその特徴を学習することで任意の関数を近似できる能力を持つ．図 4 に示すように，ニューラルネットワークは複数のニューロンと呼ばれる演算器を階層的に結合したネットワーク構成となっている．ニューロン間を結ぶ線のことをシナプス（重み）といい，重みの大きさを学習によって調整し，所望の関数を近似する．

ニューラルネットワークは，順方向伝播と，順方向伝播の結果から重みを調整する学習という 2 つの動作を行う．例として，L 層ニューラルネットワークに対して順方向伝播と学習を行うことを考える．l 層目のニューロンへの入力ベクトルを $\boldsymbol{x}^l$，l 層目の重み行列を $\boldsymbol{W}^l$ とする（$l = 1, \cdots, L$）．このとき，ニューラルネットワークの順方向伝播によって得られる l 層目のニューロンの出力ベクトル

$\boldsymbol{y}^l$ は，式 (2) に示すように，l 層目の入力ベクトル $\boldsymbol{x}^l$ と l 層目の重み行列 $\boldsymbol{W}^l$ の積和の形で表すことができる．

$$\boldsymbol{y}^l = f(\boldsymbol{x}^l \boldsymbol{W}^l) \tag{2}$$

ここで，$f(\cdot)$ は活性化関数を表す．活性化関数には様々な種類があるが，本論文では，最終層以外 ($l \neq L$) では ReLU 関数，最終層 ($l = L$) では softmax 関数を用いる．入力を a とすると，ReLU 関数は式 (3) となる．

$$f(a) = \begin{cases} a & a > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

また，長さ n の入力ベクトルを $\boldsymbol{a}$ とすると，softmax 関数の j 番目の出力値は式 (4) のように表すことができる．

$$f(\boldsymbol{a})_j = \frac{e^{a_j}}{\sum_{i=1}^{n} e^{a_i}} \tag{4}$$

$l = L$ となるまで，$\boldsymbol{x}^{l+1} \leftarrow \boldsymbol{y}^l$ として順方向伝播を続ける．

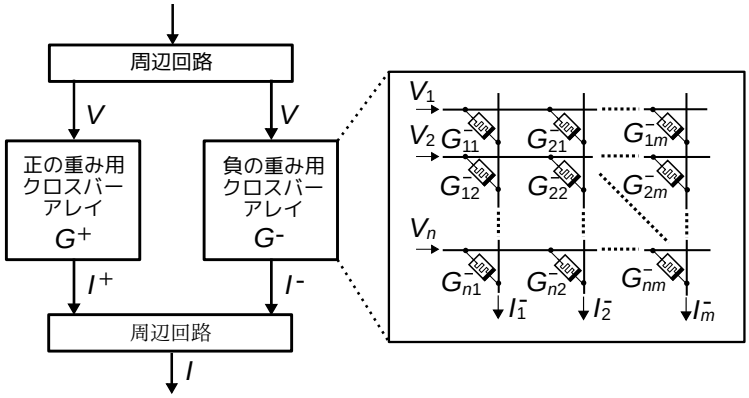
ニューラルネットワークの学習では，勾配降下法という手法が広く用いられている．勾配降下法では，誤差逆伝播によって得られる勾配 $\Delta \boldsymbol{W}$ を元の重み $\boldsymbol{W}$ に加算することで重みの値を更新する．誤差逆伝播は式 (5) から (7) のように表すことができる．

$$\boldsymbol{e}^L = \boldsymbol{t} - \boldsymbol{y}^L \tag{5}$$

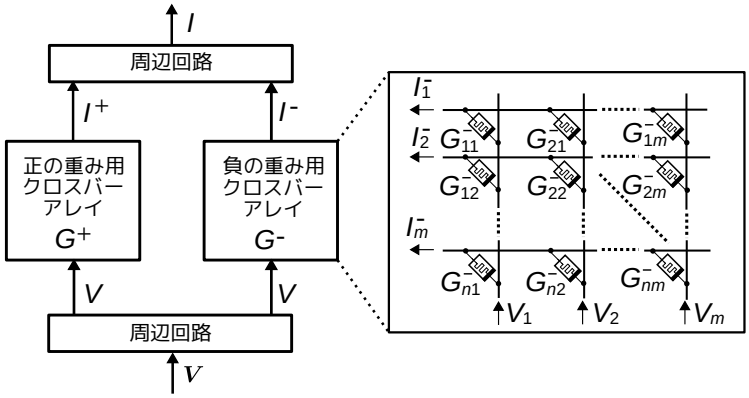
$$\boldsymbol{e}^l = (\boldsymbol{e}^{l+1} \cdot (\boldsymbol{W}^{l+1})^T) \cdot f'(\boldsymbol{x}^l \boldsymbol{W}^l) \tag{6}$$

$$\Delta \boldsymbol{W}^l = \eta \cdot (\boldsymbol{e}^l)^T \cdot \boldsymbol{x}^l \tag{7}$$

ここで，$\boldsymbol{x}^l$，$\boldsymbol{W}^l$，$\boldsymbol{e}^l$ は，それぞれ l 層目のニューロンの入力ベクトル，重み行列，誤差ベクトルを表している．式 (5) において，$\boldsymbol{y}^L$ は L 層目（最終層）のニューロンの出力ベクトル，$\boldsymbol{t}$ は最終層の出力の目標値ベクトルを表しており，e^L は最終層の出力ベクトルと目標値ベクトルの誤差を表す．l 層目の誤差の値は，式 (6) のように，$l+1$ 層目の誤差と $l+1$ 層目の転置した重み行列を乗算した値と，l 層目



(a) 順方向伝搬



(b) 逆方向伝搬

図 5 メモリスタニューラルネットワークの構成と動作

のニューロンの出力ベクトル $\boldsymbol{x}^l\boldsymbol{W}^L$ を入力とした活性化関数の微分 $f'(\boldsymbol{x}^l\boldsymbol{W}^L)$ を乗算することで，算出する．そして，l 層目の重み行列の勾配 ΔW は，式 (7) のように，l 層目の誤差 $\boldsymbol{e}^l$ と l 層目の入力ベクトルの転置 $(\boldsymbol{x}^l)^T$ を乗算することで算出できる．重みの勾配を算出するために，誤差の値を出力側から入力側へ向かって順番に計算していくため，この一連の処理は誤差逆伝播と呼ばれる．

メモリスタニューラルネットワークでは，順方向伝播での式 (2) の積和演算 $(\boldsymbol{x}^l\boldsymbol{W}^l)$ と，誤差逆伝播の式 (6) の $(\boldsymbol{e}^{l+1}\cdot(W^{l+1})^T)$ の積和演算を高速に行う．図 5 に，メモリスタニューラルネットワークの構成と動作を示す．メモリスタニューラルネットワークでは，メモリスタのコンダクタンス G がニューラルネットワー

クの重み w に相当する．このとき，1 つのメモリスタでは正負の値を表現できないため，2 つのメモリスタクロスバーアレイを利用し，それぞれ正のコンダクタンス G^+，負のコンダクタンス G^- として利用することで，1 つの重み w を表現する．メモリスタニューラルネットワークの順方向伝播は，図 5(a) のように，行方向から入力電圧 V を印加することで，式 (8) のように実現される．

$$\boldsymbol{I} = \boldsymbol{V}\boldsymbol{G}^+ - \boldsymbol{V}\boldsymbol{G}^- \tag{8}$$

また，逆方向伝播は，図 5(b) のように，列方向から誤差の値を電圧値 $\boldsymbol{V}$ として印加することで，式 (9) のように実現できる．

$$\boldsymbol{I} = \boldsymbol{V} \cdot (\boldsymbol{G}^+)^T - \boldsymbol{V} \cdot (\boldsymbol{G}^-)^T \tag{9}$$

出力電流 $\boldsymbol{I}$ は列方向 (行方向) のワイヤから並列に取り出すことができ，また，印加する入力電圧 V は Read 電圧程度の小さな電圧の印加でよいため，この積和演算は高速かつ低消費電力に行える．また，高い集積率も実現できるため，IoT デバイス等への応用が期待されている．

2.3 メモリスタの信頼性課題

メモリスタを用いたシステムが多くの利点を有する一方で，構成要素であるメモリスタは信頼性に課題がある [1, 10–15]．メモリスタは製造技術が未成熟であるために，製造ばらつきや欠陥，書き込み寿命などの影響で故障が発生する．文献 [1] によると，メモリスタの故障は図 6 のように，その故障が永久か過渡か，静的か動的かによって 4 つに分類される．永久故障は，書き込み寿命 [11] や製造時の欠陥 [13] によって，メモリスタの抵抗値が高抵抗状態，もしくは，低抵抗状態に縮退する故障である．永久故障は，それが製造中に発生したものか実動作中に発生したものかによって，静的か動的かに分類される．製造中に発生した永久故障は，製造時の欠陥に起因するもので，静的な永久故障という．実動作中に発生した永久故障は，実動作中に書き込み寿命を超えることで発生し，動的な永久故障という．過渡故障は，製造ばらつきや実動作時の書き込み電圧のばらつきや，読み出し・書き込みディスターブによって発生する故障で，過渡故障も静的

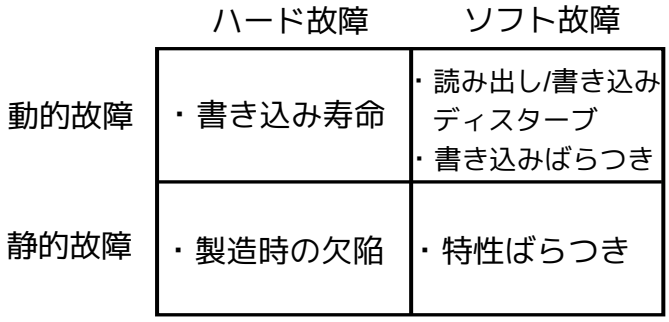


図 6 メモリスタの 4 種類の故障 [1]

か動的かに分類される．ここで，読み出し・書き込みディスターブとは，読み出し（書き込み）対象のメモリスタに電圧を印加する際に，読み出し（書き込み）対象の周辺のメモリスタセルにも少量の読み出し（書き込み）電圧が印加されることで，所望の値の読み出し（書き込み）に失敗するような故障のことである．過渡故障が発生すると，所望のコンダクタンスの値からずれた値が保持されるようになる．永久故障の場合は，故障したセルを正常なセルと入れ替えるリマッピングを行わない限り修復不可能であるが，過渡故障の場合は書き込みによって値を修復することが可能である．

メモリスタにこれらの故障が発生すると，メモリスタを用いて構成されるシステムが正常な動作ができなくなる．そのため，これらの故障に対する対策が必須となっている．

3. ReRAMの耐故障設計

3.1 はじめに

ReRAMは高い集積性能と高速な読み出し/書き込み動作が実現できることから，高性能な不揮発性メモリとして期待されている．一方で，構成要素であるメモリスタは信頼性に課題があり，特にReRAMは他の不揮発性メモリデバイスと比較して書き込み寿命が小さく，書き込み寿命を向上させる耐故障設計手法が不可欠となっている．

ReRAMの書き込み寿命を向上させる手法として，ECC回路がある．文献[10]では，既存のCMOSによるECC回路をReRAMストレージに適用している．しかし，CMOSによるECC回路はCMOSによってECC回路を実現するため，メモリスタの高い集積率を活かすことができない．これに対し，メモリスタの論理回路マッピング手法[18–20]を利用することで，メモリスタのみでECC回路を実現することも可能であるが，メモリスタの書き込み寿命が低いために，ECC回路自体の信頼性の低下を招く．

そこで，本章では，面積効率と高信頼化の両立を実現するメモリスタとCMOS回路を組み合わせた新しいECC回路を提案する．提案するECC回路では，メモリスタへの書き込みが頻繁に発生する回路ブロックはCMOS回路で実現し，その他の回路は文献[23, 24]のようにメモリスタで実現する．これにより，ECC回路の低面積化とReRAMの高信頼化を実現する．以下，本章は次のように構成する．3.2節にて，既存のECC回路で用いられてるハミング符号の動作原理と，既存のECC回路の問題点について紹介する．3.3節にて，低面積化と高信頼化を実現するCMOSとメモリスタのハイブリッドECC回路を提案する．3.4節にて，提案ECC回路の信頼性と面積，消費電力，レイテンシーを評価した結果について示し，3.5節で本章をまとめる．

3.2 CMOS による ECC 回路の動作原理と問題点

提案 ECC 回路は 1bit の誤り訂正を可能とするハミング符号 [17] がベースとなっているため，本節ではハミング符号の原理について説明する．なお，ハミング符号における各種演算はガロア体 $GF(2)$ 上の演算として扱う．そのため，加算は modulo 2 演算，もしくは XOR 演算を意味する．(n, k) ハミング符号による誤り訂正では，まず，$k$bit のオリジナルのデータから $\log_2 k + 1$bit の検査ビットを生成し，それらを組み合わせた $n(= k + \log_2 k + 1)$bit の符号語を生成するエンコーディングを行う．エンコーディングによって生成された n ビットの符号語は，ReRAM ストレージに保存される．エンコーディングは，生成行列を利用した積和演算によって行われる．例として，オリジナルのデータが 4bit (x_0, x_1, x_2, x_3)，検査ビットが 3bit (r_0, r_1, r_2) の $(7, 4)$ ハミング符号を考える．このとき，検査ビット (r_0, r_1, r_2) は式 (10) によって生成される．

$$
\begin{aligned}
&(r_0, r_1, r_2) \\
&= \quad (x_0, x_1, x_2, x_3) \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \pmod 2 \\
&= (x_1 \oplus x_2 \oplus x_3, x_0 \oplus x_2 \oplus x_3, x_0 \oplus x_1 \oplus x_3).
\end{aligned} \tag{10}
$$

式 (10) 中の行列が，生成行列である．

デコーディングでは，(i) シンドローム生成，(ii) 誤り訂正ビット生成，(iii) 誤り訂正という 3 つの処理が行われる．以下，エンコーディング動作と同様に，$(7, 4)$ ハミング符号の例を用いて説明する．シンドローム生成では，$n$bit の符号語 $(x_0, x_1, x_2, x_3, r_1, r_2, r_3)$ をストレージから読み出す時に，式 (11) の計算を行うことでシンドローム (s_1, s_2, s_3) を生成する．具体的には，オリジナルのデータから生成行列を利用して新たに生成した検査ビット (r'_0, r'_1, r'_2) と，ストレージ上に保持されていた検査ビット (r_0, r_1, r_2) との差を取ることで，シンドローム (s_1, s_2, s_3) の

表 2 シンドロームから訂正ビットの生成

Syndrome (3 bits)	Correction bits (4 bits)
011	1000
101	0100
110	0010
111	0001
other	0000

値を生成する．

$$
\begin{aligned}
&(r'_0, r'_1, r'_2) \\
&= \quad (x_0, x_1, x_2, x_3) \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \pmod 2. \qquad (11)
\end{aligned}
$$

$$
(s_0, s_1, s_2) = (r'_0, r'_1, r'_2) \oplus (r_0, r_1, r_2).
$$

次に，表 2 に従って，生成したシンドローム (s_0, s_1, s_2) の値から訂正用ビット (c_0, c_1, c_2, c_3) を生成する．表 2 のそれぞれの値は，生成行列の値から事前に計算して求められる．生成される訂正用ビット (c_0, c_1, c_2, c_3) において，"1"の値はデータビット (x_0, x_1, x_2, x_3) 中の誤り箇所を表している．そのため，式 12 に示すように，データビット (x_0, x_1, x_2, x_3) と訂正用ビット (c_0, c_1, c_2, c_3) の排他的論理和をとることで，データビット中の誤りを訂正することができ，最終的に誤りを訂正した値 (z_0, z_1, z_2, z_3) を出力することができる．

$$
(z_0, z_1, z_2, z_3) = (x_0, x_1, x_2, x_3) \oplus (c_0, c_1, c_2, c_3). \qquad (12)
$$

ハミング符号に基づく ECC 回路は，従来の CMOS による回路 [10] かメモリスタの論理回路マッピング手法 [18, 19] によって実現することができる．しかし，CMOS による ECC 回路はメモリスタの高い集積率を活かすことができないため，面積が大きくなる．また，メモリスタの論理回路マッピング手法を利用した ECC

回路は，XOR 演算によってメモリスタに対する書き込みが非常に多くなるため，ECC 回路自体の信頼性が低下する．

3.3 アナログ積和演算動作に基づく ECC 回路

本節では，長期信頼性の向上を実現しながら高い集積率を実現する，CMOS とメモリスタのハイブリッド ECC 回路を提案する．提案 ECC 回路は，メモリスタで実現すると書き込みが多く発生する論理演算を行う回路ブロックについては CMOS 回路で実現し，その他の回路ブロックはメモリスタで実現する．

本節では，CMOS とメモリスタのハイブリッド ECC 回路を実現するために，クロスバーアレイの積和演算動作に基づくハミング符号を提案する．ハミング符号の排他的論理和の演算はメモリスタの論理回路マッピング手法によって実現することが可能だが，メモリスタの論理回路マッピング手法は，メモリスタに対する多くの書き込みを必要とするため ECC 回路自体の信頼性が低下する．そのため，提案 ECC 回路では，エンコーディングとデコーディングにおいてメモリスタのクロスバーアレイ構造を利用したアナログ積和演算を利用する．アナログ積和演算は，書き込み電圧と比較して小さい読み出し電圧を印加するため，メモリスタの書き込み寿命には影響を与えない．そのため，書き込み寿命に影響を与えず，メモリスタの高い集積率を活かすことができる．アナログ積和演算に基づくハミング符号では，ハミング符号による誤り訂正と異なり，modulo 2 演算の変わりに通常の加算動作を用いるが，1bit の誤り訂正であれば，ハミング符号と同等の誤り訂正を実現することができる．

3.3.1 アナログ積和演算動作に基づくハミング符号

本節では，3.2 節と同様に (7,4) ハミング符号の例を用いてアナログ積和演算による誤り訂正の方法を説明するが，アナログ積和演算に基づくハミング符号では任意の (n, k) ハミング符号に適用することができる．アナログ積和演算によるハミング符号は，既存のハミング符号と同様にエンコーディングとデコーディング

という 2 つの操作からなる．そして，式 (10) に示すエンコーディングの計算と，式 (11) に示すシンドローム生成の計算を，アナログ積和演算によって実現する．

エンコーディングでは，式 (13) に示すように，2 値のデータビット (x_0, x_1, x_2, x_3) から検査ビット (r_0, r_1, r_2) を生成する．

$$
\begin{aligned}
&(r_0, r_1, r_2) \\
&= \quad (x_0, x_1, x_2, x_3) \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\
&= (x_1 + x_2 + x_3, x_0 + x_2 + x_3, x_0 + x_1 + x_3).
\end{aligned}
\tag{13}
$$

式 (13) は，式 (10) の modulo 2 演算と異なり，メモリスタのアナログ積和演算動作によって実現される．生成したデータビットと検査ビットの値は，1 ワードの値として ReRAM に書き込まれる．

ReRAM から値を読み出す時，既存のハミング符号と同様に (i) シンドローム生成，(ii) 誤り訂正ビット生成，(iii) 誤り訂正という 3 つの操作によってデコーディングを行う．シンドローム生成では，ReRAM に保持されている符号語 $(x'_0, x'_1, x'_2, x'_3, r_0, r_1, r_2)$ を読み出し，その値からシンドローム (s_0, s_1, s_2) を生成する．なお，(x'_0, x'_1, x'_2, x'_3) は ReRAM への読み出し動作によって得られたデータビット (x_0, x_1, x_2, x_3) の値を意味する．まず，式 (14) に示すように，テンポラリシンドローム (t_0, t_1, t_2) を生成する．

$$
\begin{aligned}
&(t_0, t_1, t_2) \\
&= \quad (x'_0, x'_1, x'_2, x'_3) \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} - (r_0, r_1, r_2) \\
&= (r'_0, r'_1, r'_2) - (r_0, r_1, r_2).
\end{aligned}
\tag{14}
$$

式 (14) において，(r'_0, r'_1, r'_2) は (x'_0, x'_1, x'_2, x'_3) と生成行列を乗算することで生成される．そして，(r_0, r_1, r_2) から (r'_0, r'_1, r'_2) の値を減算することでアナログ値のテンポ

ラリシンドローム (t_0, t_1, t_2) を生成する．ここで，もし，1bit の誤りがデータビットで発生しており，かつ，検査ビットに誤りが発生していない場合，テンポラリシンドロームの各ビットは，"-1"，"0"，"1"のいずれかの値となり，"-1"，もしくは"1"の値は，表 2 のシンドロームの"1"と同様のビットに表れる．また，検査ビットに 1bit の誤りが発生した場合，テンポラリシンドロームの値はいずれか 1bit のみが-1，もしくは 1 になり，データビットの 1bit の誤りによって生成されるシンドロームの値とは一致しない．そのため，式 (15) によってテンポラリシンドローム (t_0, t_1, t_2) の値をデジタル値に変換することで，シンドローム (s_0, s_1, s_2) を生成することができる．

$$s_i = \begin{cases} 1 & |t_i| \geq 1 \\ 0 & \text{otherwise} \end{cases}, \ 0 \leq i \leq 2. \tag{15}$$

最後に，シンドローム (s_0, s_1, s_2) を用いて 3.2 節と同様の誤り訂正ビット生成，および，誤り訂正の操作を行うことで，ハミング符号と同等誤り訂正を実現することができる．

実際の計算例として，ReRAM に書き込むデータビット (x_0, x_1, x_2, x_3) が $(1, 0, 1, 1)$ で，書き込み後，x_2 に 0 縮退故障が発生した場合，つまり，(x'_0, x'_1, x'_2, x'_3) が $(1, 0, 0, 1)$ の場合を考える．まず，式 (13) から検査ビット $(r_0, r_1, r_2) = (2, 3, 2)$ を生成する．次に，式 (14) より，ReRAM から読みだしたデータビット (x'_0, x'_1, x'_2, x'_3) から $(r'_0, r'_1, r'_2) = (1, 2, 2)$ を生成し，(r'_0, r'_1, r'_2) から (r_0, r_1, r_2) を減算することで，テンポラリシンドローム $(t_0, t_1, t_2) = (-1, -1, 0)$ を生成する．その後，式 (15) より，(t_0, t_1, t_2) からデジタル値のシンドローム $(s_0, s_1, s_2) = (1, 1, 0)$ を生成し，3.2 節の表 (2) によって誤り訂正ビット $(c_0, c_1, c_2, c_3) = (0, 0, 1, 0)$ を生成する．最後に，ReRAM から読み出したデータビット (x'_0, x'_1, x'_2, x'_3) と (c_0, c_1, c_2, c_3) との排他的論理和を計算することで，$(z_0, z_1, z_2, z_3) = (1, 0, 1, 1)$ が出力される．出力値 (z_0, z_1, z_2, z_3) は，誤りが発生する前のデータビット (x_0, x_1, x_2, x_3) の値に等しい．したがって，アナログ積和演算に基づくハミング符号は既存のハミング符号と同様に 1bit までの誤りを訂正することができる．

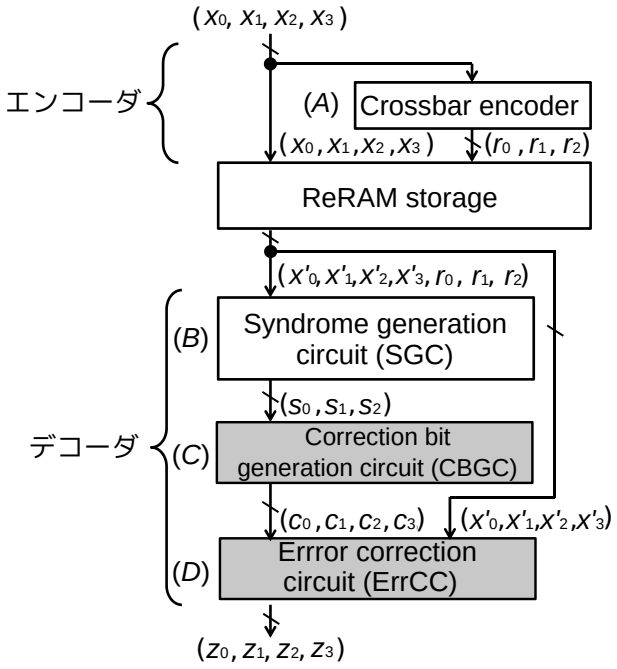


図 7 提案する ECC 回路のブロック図

3.3.2 回路構成

図 7 に提案する ECC 回路のブロック図を示す．図 7 において，グレーで示されるブロックは CMOS で実現される回路ブロックを表す．提案する ECC 回路は，（A）クロスバーエンコーダ，（B）シンドローム生成回路（Syndrome generaion circuit, SGC），（C）訂正ビット生成回路（Correction bit generation circuit, CBGC），（D）誤り訂正回路（Error correction circuit, ErrCC）の 4 つの回路ブロックによって構成される．クロスバーエンコーダが"エンコーダ"に相当し，その他の回路が"デコーダ"に相当する．

クロスバーエンコーダ

クロスバーエンコーダは，データビット (x_0, x_1, x_2, x_3) の値から，検査ビット (r_0, r_1, r_2) の値を生成する式 (13) の計算を行う．クロスバーエンコーダの回路図を図 8 に示す．式 (13) における生成行列は図 8 のクロスバーアレイ構造によって実現され，各メモリスタの値は生成行列の各要素の値に応じて事前に設定する．セ

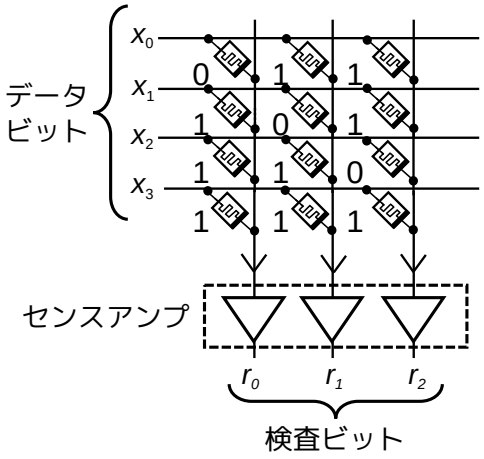


図 8 クロスバーエンコーダ

ンスアンプは，積和演算の結果として微小な電流で出力される (r_0, r_1, r_2) の値を増幅するために利用される．

シンドローム生成回路 (SGC)

シンドローム生成回路では，式 (14) と式 (15) の計算を行い，データビットと検査ビットの値からシンドロームの値を生成する．シンドローム生成回路のブロック図を図 9 に示す．

シンドローム生成回路では，まず，クロスバーエンコーダと同様の回路によってアナログ値の比較用検査ビット (r'_0, r'_1, r'_2) を生成する．その後，(r'_0, r'_1, r'_2) からアナログ値の検査ビット (r_0, r_1, r_2) を減算することで，テンポラリシンドローム (t_0, t_1, t_2) を生成する．減算動作は，図 9 の差動アンプによって実現する．次に，式 15 に示すように，アナログ値の (t_0, t_1, t_2) をデジタル値のシンドローム (s_0, s_1, s_2) に変換する．この動作は 2 つのセンスアンプから構成される絶対値変換回路によって実現される．1 段目のアンプでは (t_0, t_1, t_2) を絶対値に変換し，2 段目のアンプでは，"1"を超える値を"1"にする動作を行うことで，1 段目のアンプの出力を 2 値に変換する．この結果，(s_0, s_1, s_2) を得ることができる．

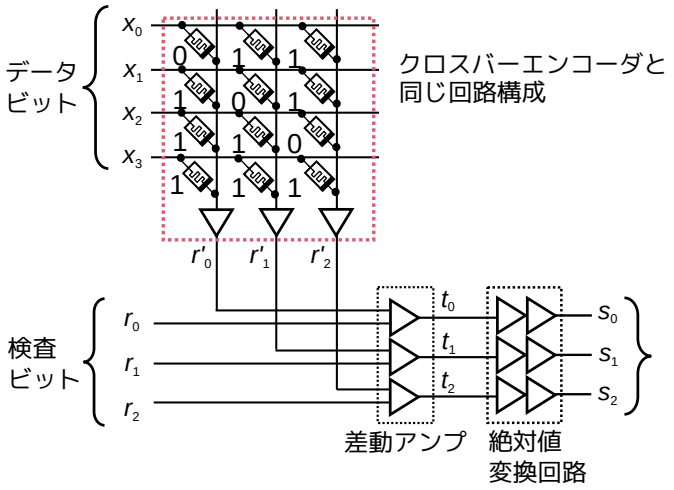


図 9 シンドローム生成回路

訂正ビット生成回路 (CBGC)

訂正ビット生成回路では，表 2 にしたがい，論理演算を利用してシンドローム (s_0, s_1, s_2) を訂正ビット (c_0, c_1, c_2, c_3) に変換する．もし，訂正ビット生成回路メモリスタで実現すると，メモリスタの論理回路マッピング手法による実装が必要となり，多くの書き込みがメモリスタに行われるため，訂正ビット生成回路の信頼性の低下につながる．そのため，訂正ビット生成回路は CMOS 回路によって実現される．

誤り訂正回路 (ErrCC)

誤り訂正回路では，ReRAM から読み出したデータビット (x'_0, x'_1, x'_2, x'_3) と訂正ビット (c_0, c_1, c_2, c_3) との排他的論理和を計算する．もし，データビットに 1bit の誤りが生じた場合，訂正ビットの対応するビットが"1"となっているため，排他的論理和を計算することで誤りを訂正することができる．誤り訂正回路も訂正ビット生成回路と同様に論理演算を行う回路であるため，CMOS 回路で実現される．

3.4 シミュレーションによる評価実験

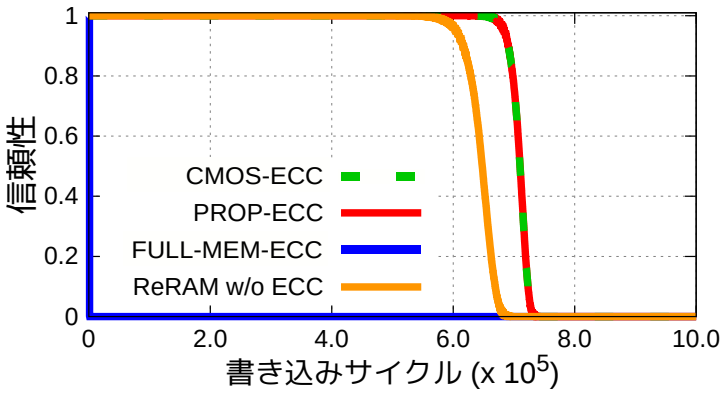
本節では，提案 ECC 回路の効果を定量的に評価する．本評価実験では，提案 ECC 回路（PROP-ECC），CMOS による ECC 回路（CMOS-ECC） [10]，全ての回路をメモリスタで実現した ECC 回路（FULL-MEM-ECC）の 3 つの回路の信頼性と面積，消費電力，レイテンシーを評価する．

FULL-MEM-ECC は IMPLY [18, 19] という，メモリスタの論理回路マッピング手法によって実現される．IMPLY では，$\bar{X} \wedge Y$ の演算を実現する IMPLY ゲートをメモリスタによって実現し，この論理ゲートを使って not，or，and などの基本的な論理演算の動作を実現する．1 つの IMPLY ゲートは 2 つのメモリスタと 1 つのセンスアンプと 1 つの抵抗器によって構成される．3bit 以上の入力から 1bit の値を出力する論理演算動作を IMPLY で実現するには，入力ビット数に対応するメモリスタがその分必要になる．論理演算を行う際には，複数のメモリスタに対して SET，RESET 動作を印加して，メモリスタの状態を更新する必要がある．したがって，入力数の大きい複雑な論理動作を実現するには，その分多くの書き込みが必要となるため，書き込み寿命の低下を引き起こす．

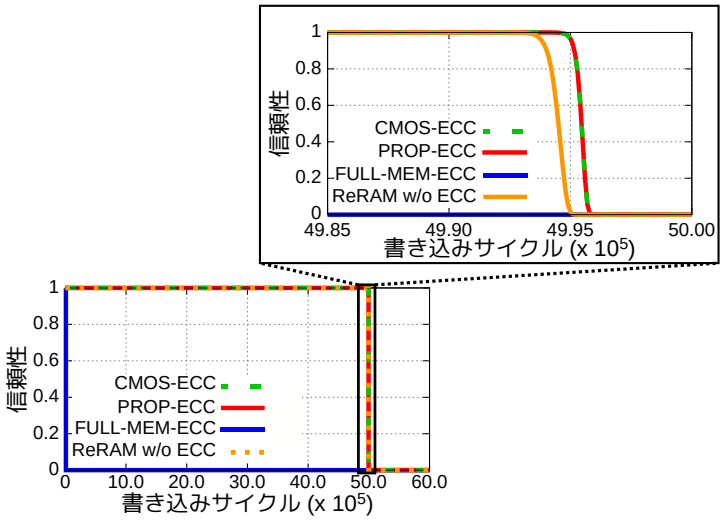
3.4.1 信頼性評価

本節では，PROP-ECC，FULL-MEM-ECC，CMOS-ECC，ECC 無しの ReRAM（ReRAM w/o ECC）の 4 つのシステム信頼性を比較する．本評価実験では，1 ワードあたりの符号長が 522bit で，ワード数が 256 の ReRAM ストレージを用いる．また，ECC 回路では (522, 512) ハミング符号を利用する．

もし，ReRAM に対して読み出し動作を行い，正しい値が出力されれば，そのシステムは正常であるとする．読み出し動作において，ECC 無しの ReRAM からは，誤りがない場合に 512bit のデータビットが正しく出力される．一方，ECC 付きの ReRAM からは，誤りが 1bit 以下である場合に，522bit の符号語が正しくデコードされ，誤りが訂正された 512bit のデータビットが正しく出力される．シミュレーションでは，各メモリシステムに対して，任意のワードを 1 つ選択し，そのワードに対して書き込みを実行する．その後，読み出し動作を行い，書き込みを



(a) $N(1.0\times10^6, 6.4\times10^9)$ [15]



(b) $N(5.0\times10^6, 1.4\times10^6)$ [12]

図 10 異なる寿命モデルごとの信頼性評価結果

行ったワードの値が正しく読み出せるかを確認する．この操作を各メモリシステムが故障するまで繰り返す．なお，本評価実験で用いるメモリスタクロスバーアレイは製造ばらつきに対してテストされており，デジタル値のデータビットとアナログ値の検査ビットのエラーレートはほとんど同じである．

メモリスタの書き込み寿命は，平均が μ，分散が σ^2 の正規分布 $N(\mu, \sigma^2)$ の正規分布に従うものとし，$N(1.0\times10^6, 6.4\times10^9)$ [15] と $N(5.0\times10^6, 1.4\times10^6)$ [12] という 2 つのメモリスタの書き込み寿命のモデルを評価実験に利用する．信頼性

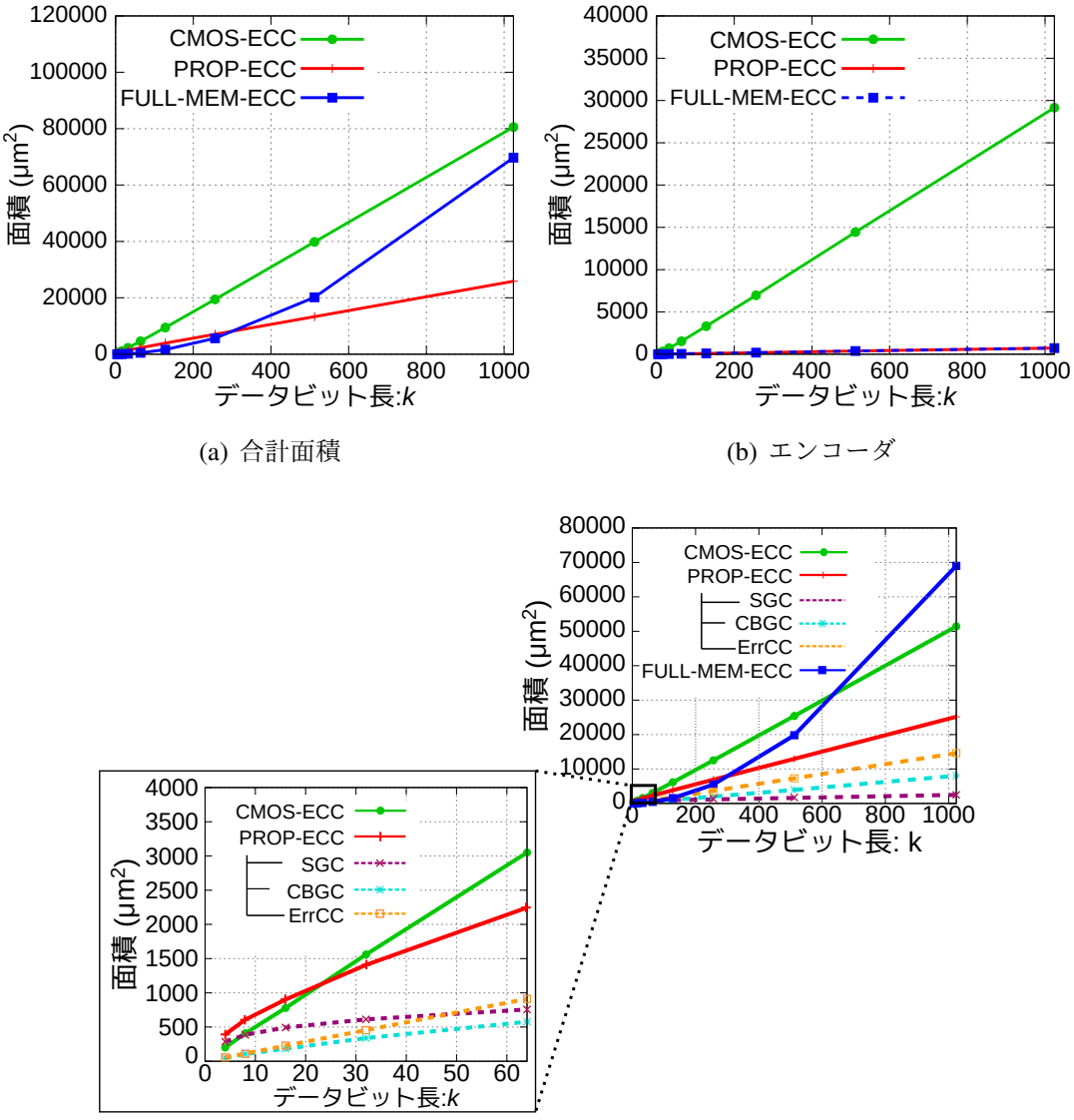
は，正しい値を出力する ReRAM ストレージシステムの割合として定義する．

PROP-ECC，CMOS-ECC，FULL-MEM-ECC，ReRAM w/o ECC の 4 つのシステムに対して複数回書き込みを行い，信頼性を評価した結果を図 10 に示す．図 10(a)，10(b) に示すように，FULL-MEM-ECC，ReRAM w/o ECC と比較して，PROP-ECC と CMOS-ECC は ECC 回路によってそれぞれ約 64,000 サイクル，約 2,000 サイクル程度書き込み寿命が向上した．FULL-MEM-ECC は，ReRAM w/o ECC と比較して，むしろ信頼性が低下した．これは，FULL-MEM-ECC の ECC 回路が IMPLY ゲートによって構成されたことで，エンコーディング，デコーディング動作のたびに同一のメモリスタに書き込みが行われようになり，その結果，ECC 回路のメモリスタが故障し，正しい読み出し動作が実現できなくなったためである．一方，PROP-ECC と CMOS-ECC は，書き込み動作は ReRAM ストレージ中のメモリスタに対してのみ行われる．また，PROP-ECC のエンコーダとデコーダは，積和演算に用いる読み出し動作のみが行われる．その結果，FULL-MEM-ECC が ECC 回路の故障によって正しい訂正を行うことができなくなった一方で，PROP-ECC と CMOS-ECC は，符号語に誤りが発生しても ECC 回路によって誤りを訂正し，正しい値を出力することができた．

FULL-MEM-ECC は信頼性に課題があるため，次節以降は主に PROP-ECC と CMOS-ECC に着目して評価を行う．

3.4.2 面積評価

図 11(a) に PROP-ECC，CMOS-ECC，FULL-MEM-ECC の面積評価を行った結果を示す．CMOS-ECC と，PROP-ECC の CMOS 回路部分は，市販論理合成ツール [25] と 90 nm プロセスのライブラリ [26] を用いて設計した．FULL-MEM-ECC は，IMPLY ゲートを用いて設計した．メモリスタを用いた回路の面積については，NVSim [27] という ReRAM を含む不揮発性メモリの面積やエネルギーを推定するツールを用いて測定を行った．NVSim では，1 つのメモリスタのセル面積を $4F^2$ として面積の測定を行う．ここで，F は加工寸法（feature size）を表し，F は CMOS 回路と同様に 90 nm とした [28]．センスアンプと差動アンプの面積は，それぞれ 20.09 μm^2 [29]，78.97 μm^2 [30] とした．絶対値変換回路の面積は，2 つの



(a) 合計面積

(b) エンコーダ

(c) デコーダ

図 11 3 つの ECC 回路の面積比較

センスアンプで構成されるため，センスアンプの 2 倍の面積として計算を行った．

図 11(a) に示すように，データビット長 k が 300 bit 以下のとき，FULL-MEM-ECC よりも PROP-ECC の方が面積が大きかった．しかし，k が 300 bit よりも大きくなると，PROP-ECC の面積が 3 つの ECC 回路の中で最も小さくなった．図 11(b),

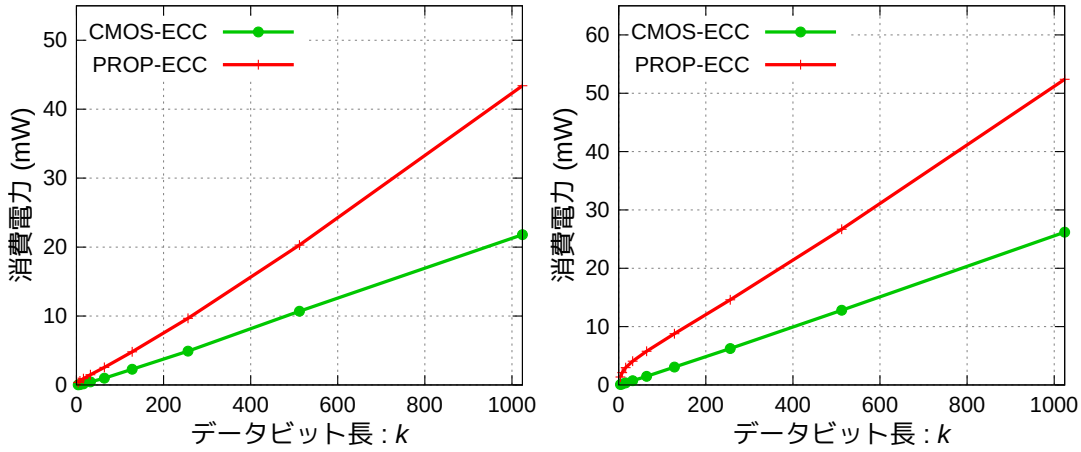
11(c) に各 ECC 回路のエンコーダとデコーダの面積を示す．図 11(b) に示すように，エンコーダの面積はそれぞれ線形的に増加し，PROP-ECC は CMOS-ECC と比較して傾きが小さかった．図 11(c) に示すように，FULL-MEM-ECC のデコーダは，k が増加するとともに非常に大きな面積となった．これは，FULL-MEM-ECC がデータビット長 k に依存した数のアンプ，つまり，$O(k)$ 個のアンプを必要とするためである．対照的に，PROP-ECC と CMOS-ECC のデコーダの面積は FULL-MEM-ECC と比較して小さく，特に，PROP-ECC の面積増加の傾きは CMOS-ECC よりも小さかった．

PROP-ECC のデコーダの面積の内訳を図 11(c) の点線に示す．シンドローム生成回路（SGC）では面積の大きい差動アンプが用いられるため，k が小さい時，PROP-ECC のうち差動アンプの占める面積の割合が大きい．しかし，差動アンプの数は検査ビット長（$= O(\log k)$）に依存するため，k の増加とともに差動アンプの占める面積の割合は小さくなっていく，そのため，k が増加すると，PROP-ECC は CMOS-ECC と比較して面積が小さくなった．このような結果から，k が大きい値のとき，PROP-ECC の面積が他の ECC 回路と比較して最も小さくなった．

3.4.3 消費電力・レイテンシー評価

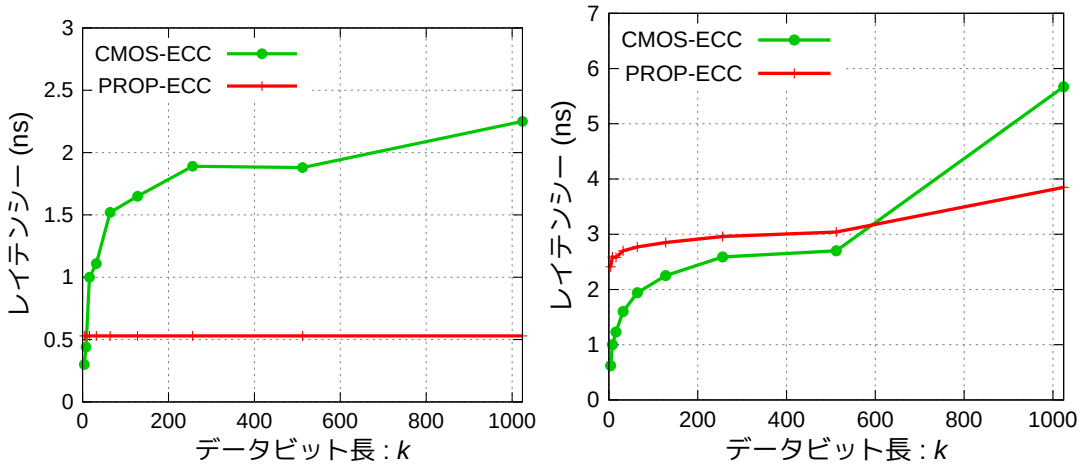
本節では，PROP-ECC と CMOS-ECC の消費電力とレイテンシーを評価する．PROP-ECC のメモリスタで構成された回路ブロックの消費電力とレイテンシーは NVSim d [27] というツールを利用して算出した．また，センスアンプと差動アンプの消費電力とレイテンシーについても，NVSim [27] によって算出した．CMOS-ECC で構成された回路ブロックの消費電力とレイテンシーについては，商用の論理合成ツール [25] を利用して算出した．

図 12 に CMOS-ECC と PROP-ECC の消費電力を評価した結果を示す．図 12(a) に示すように，CMOS-ECC の消費電力と PROP-ECC の消費電力は線形的に増加した．PROP-ECC の消費電力は，CMOS-ECC の約 2 倍ほど大きくなった．これは，PROP-ECC のエンコーダ・デコーダが，アナログ回路であるセンスアンプと差動アンプを用いて構成されるためだと考えられる．センスアンプの消費電力については，文献 [31] のように，複数のビットでセンスアンプを共有して利用する



(a) エンコーダ　　(b) デコーダ

図 12 CMOS-ECC と PROP-ECC の消費電力比較



(a) エンコーダ　　(b) デコーダ

図 13 CMOS-ECC の PROP-ECC レイテンシー比較

ことで，削減できる可能性がある．そのため，PROP-ECC の消費電力についてはまだ削減することができると考えられ，その点を考慮した設計については今後の課題だと考えられる．

対照的に，図 13(a) と 13(b) に示すように，データビット数が増加するとともに，PROP-ECC のエンコーダとデコーダのレイテンシーは，CMOS-ECC よりも

小さくなった．図 13(a) より，PROP-ECC のレイテンシーの変化は，CMOS-ECC よりも非常に小さいことがわかった．また，図 13(a) より，ファンアウト数が大きくなったために，データビット数 k が 512bit よりも大きくなると，CMOS-ECC のレイテンシーは劇的に増加した．

3.5 まとめ

本章では，ReRAM ストレージの高信頼化と ECC 回路の高集積化を目的として，CMOS とメモリスタを用いたハイブリッド ECC 回路を提案した．メモリスタを用いた回路は高い集積率を実現できるが，書き込み回数に強い制限が存在する．そのため，提案 ECC 回路は，高集積化と信頼性の向上を実現するために，メモリスタを用いると書き込みが頻繁に発生する回路については CMOS で設計し，その他の回路をメモリスタを用いて設計した．評価実験より，データビット長 k が 1,024 bit のとき，提案 ECC 回路は CMOS による ECC 回路と比較して 67.9%面積を削減した．将来利用が期待されているメモリスタベースのインメモリーコンピューティングアーキテクチャ [32, 33] は k が比較的大きいアプリケーションであり，提案 ECC 回路はそのようなアプリケーションに対してより効果的であると考えられる．一方で，PROP-ECC は消費電力について改善が必要であることが分かったが，レイテンシーについては CMOS-ECC よりも向上した．

4. メモリスタニューラルネットワークの耐故障設計

4.1 はじめに

メモリスタニューラルネットワークは，順方向・逆方向伝播をエネルギー効率良く実現できる，集積率の高い機械学習アクセラレータである．しかし，メモリスタは製造技術が未成熟であるために，書き込みばらつきや書き込み寿命などの要因によって故障が発生する．故障の存在は識別性能の低下を招くため，信頼性を向上させるメモリスタニューラルネットワーク用の耐故障設計が必要となっている．

メモリスタニューラルネットワーク用の耐故障設計手法として，X-ABFT がある [16]．X-ABFT は，メモリスタクロスバーアレイの列方向にチェックサムを追加することで，オンラインテストと順方向伝播の誤り訂正という 2 つの機能を実現する．X-ABFT のオンラインテストは，一定の書き込みサイクル毎にテストベクトルを印加し，その出力値を解析することでメモリスタニューラルネットワーク中の故障の値と故障箇所を特定する操作である．学習中に発生した故障箇所を特定することで，故障を修復することができるようになるため，識別性能を向上させることができる．また，順方向伝搬の誤り訂正は，メモリスタニューラルネットワーク中のセルに故障が発生し，順方向伝搬の出力値に誤りが生じた場合でも，チェックサムを利用して，出力値の 1 つの誤りをテスト操作無しで訂正できる機能である．少量の故障であれば解析操作無しで出力の誤りを訂正できるため，時間オーバーヘッドを減らすことができる．これらの操作を組み合わせることによって，短い時間オーバーヘッドで高い信頼性を担保する学習を行える動作を実現できる．しかし，X-ABFT には，(i) オンラインテスト動作において，故障箇所によっては故障の修復を正しく行うことができない，(ii) 逆方向伝搬時の誤り訂正が実現できないという 2 つの問題点がある．

そこで，本章では，列方向だけでなく行方向にもチェックサムを追加し，X-ABFT では位置特定できなかった故障の位置を正しく特定・訂正するより高品質なオンラインテスト手法と，逆方向伝搬の誤り訂正を可能とする耐故障設計手法を提案する．以下，本章は次のように構成する．4.2 節にて，X-ABFT の全体の動作フ

ローと順方向伝搬の誤り訂正，オンラインテスト動作の詳細について説明する．4.3 節にて，逆方向伝搬の誤り訂正機能と，X-ABFT より多くの故障を修復するオンラインテストを可能とする耐故障設計を提案する．4.4 節にて，提案手法のテスト機能と誤り訂正機能による識別性能の向上と面積・時間オーバーヘッドを実際のアプリケーションを用いて評価した結果について示し，4.5 節で本章をまとめる．

4.2 X-ABFT

X-ABFT は，少ない時間オーバーヘッドと高い信頼性を両立する学習動作を実現する．そのために，X-ABFT では，書き込みサイクル毎に発生する過渡故障に対しては時間オーバヘッドの小さい誤り訂正を利用し，書き込み寿命によって生じる永久故障に対しては高い信頼性を担保するオンラインテストを利用する．

図 14 に，X-ABFT による学習動作の概要を示す．メモリスタニューラルネットワークの学習では，メモリスタに重みを書き込む際に，書き込みばらつきによる少量の過渡故障が発生する．過渡故障が発生すると，順方向伝播の出力値に誤りが生じる．そこで，X-ABFT では過渡故障によって生じる順方向伝播の出力ベクトル中の 1 つの誤りを，列チェックサムを利用して訂正する．列チェックサムの計算は積和演算と同時に行うことができるため，小さい時間オーバヘッドで出力の誤りを訂正できる．しかし，学習によってメモリスタに多くの書き込みが行われるようになると，書き込み寿命による永久故障が発生する．永久故障によって故障の数が増加し出力の 2 要素以上に誤りが発生すると，その誤りを訂正できなくなる．そこで，永久故障を修復するために，一定の書き込み回数ごとにメモリスタニューラルネットワーク中の故障の値と位置を特定するテスト動作を行う．具体的には，テスト用の入力ベクトルをメモリスタニューラルネットワークに印加し，その出力値を CPU 上で解析することで，故障の値と位置の特定を行う．メモリスタの永久故障を特定することで，永久故障を直接修復できるようになるため，高い信頼性を担保できる．また，テスト操作は一定回数の書き込み毎に行われるため，時間オーバーヘッドへの影響を抑えることができる．以上の 2 つの操作を組み合わせて利用することで，小さい時間オーバーヘッドで高信頼な学習を

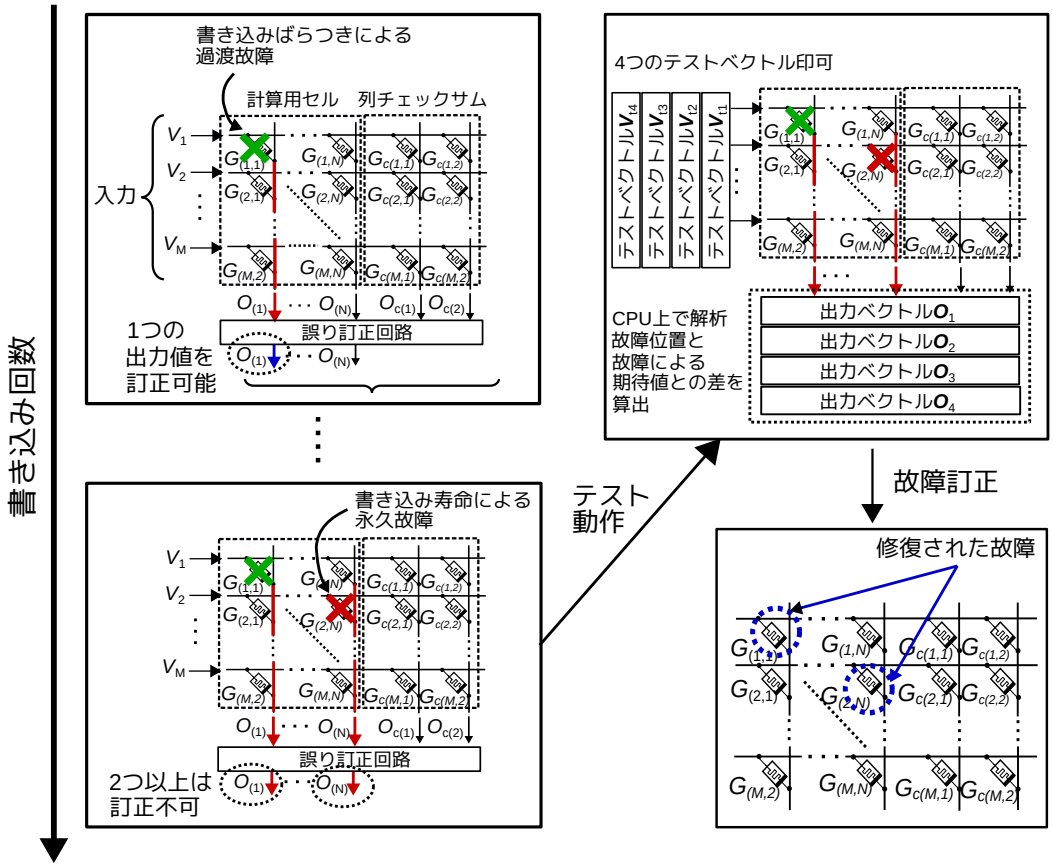


図 14　X-ABFT による学習動作の概要

行うことができる．

X-ABFT の全体の動作フローを図 15 に示す．X-ABFT では，まず，メモリスタニューラルネットワークを構成するクロスバーアレイを複数のブロックに分割する．具体的には，図 16 に示すように列数が N，行数が M のクロスバーアレイを列数が c_t，行数が r_t のサイズの複数のブロックに分割し，各テストブロックごとにチェックサム用の列を 2 列分確保する．そのため，チェックサム生成後のクロスバーアレイの合計列数は $N+2\times(N/c_t)$，行数は M となる．複数のテストブロックに分割し，ブロック毎に誤り訂正やオンラインテストを行うことで，訂正可能な出力数と修復可能な故障数を増やすことができ，誤り訂正やテストの品質を向上させることができる．

クロスバーアレイを複数のテストブロックに分割した後は，エンコーディング

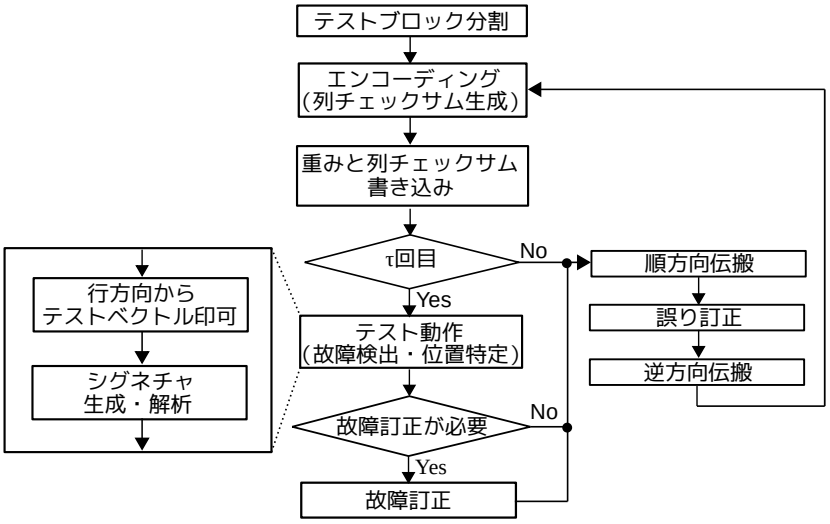


図 15 X-ABFT の全体の学習フロー

を行ってコンダクタンスの値から列チェックサムを生成する．列チェックサムは図 16 のように，各テストブロックごとに重み無し列チェックサムと重み有り列チェックサムという，2 列分のチェックサムを生成する．c_t をテストブロックの計算用メモリスタの列数，$G_{(i,j)}$ を i 行 j 列目の計算用メモリスタセルのコンダクタンスとすると，式 (16) に示すように，i 行目の重み無し列チェックサムのコンダクタンス $G_{c(i,1)}$ は i 行目の計算用メモリスタセルのコンダクタンスの総和となる．

$$G_{c(i,1)} = \sum_{j=1}^{c_t} G_{(i,j)} \tag{16}$$

また，式 (17) に示すように，i 行目の重み有り列チェックサムのコンダクタンス $G_{c(i,2)}$ は，i 行目の計算用メモリスタセルのコンダクタンスの値に列番号 j を乗算した値の総和となる．

$$G_{c(i,2)} = \sum_{j=1}^{c_t} j \cdot G_{(i,j)} \tag{17}$$

列チェックサムの値を用いることで，誤り訂正動作における故障列の特定と出力値の訂正，および，オンラインテスト動作における故障箇所の特定と訂正ができる．

列チェックサムを書き込んだ後は，一定の書き込み回数 (τ) 毎にオンラインテストを行う．オンラインテストでは，テストブロックの行ごとにテストベクトル

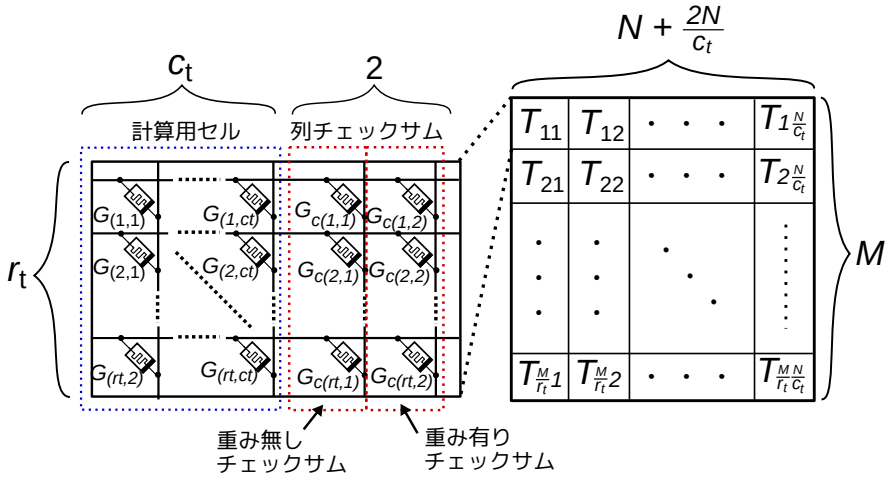


図 16 テストブロック分割

というテスト用の入力電圧を印加し，テストブロックごとに出力ベクトルを生成する．そして，出力ベクトルを CPU へ転送し，CPU 側で出力ベクトルの値からシグネチャを生成・解析し，テストブロックごとの同一行でない 2 つまでの故障セルの故障の値と位置を特定する．オンラインテストの詳細な動作については，4.2.2 節に示す．

テスト終了後，テストブロック中に故障が発生していれば，故障の修復を行う．故障の修復では，まず，テスト動作によって得られた故障発生前の正しいコンダクタンスの値を故障が発生しているメモリスタセルに書き込み，その後，書き込みを行ったメモリスタセルの値を読み出す．このとき，過渡故障であれば訂正された値が出力されるが，永久故障であれば書き込みによって故障を修復できないため，故障の値が出力される．永久故障については，例えば文献 [13, 34] で提案されているリマッピング手法を利用して修復を行う．

テスト終了後，もしくは，書き込み回数が τ に達していない場合は，メモリスタニューラルネットワークの標準動作である順方向伝搬を行う．このとき，各テストブロック列毎に，出力ベクトルの 1 要素の誤り訂正を行う．誤り訂正動作の詳細は，4.2.1 節に示す．その後，逆方向伝搬動作によって重みの更新値を求め，以後，重みの書き込みから順に上記の操作を繰り返す．この一連の操作によって，X-ABFT では高信頼な学習を行う．

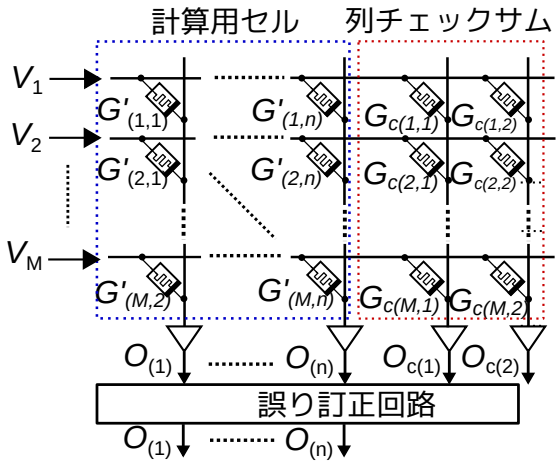


図 17 X-ABFT の順方向伝播に対する誤り訂正

4.2.1 順方向伝播の誤り訂正

順方向伝搬の誤り訂正では，図 17 に示すように，積和演算によって得られる出力ベクトルのうち，分割した列ごとに出力ベクトルの 1 要素の誤りを訂正する．誤り訂正処理は，出力結果に誤りが生じているかを検査するフェーズと，検査結果から誤りの値を訂正するフェーズからなる．クロスバーアレイの行数を M，テストブロックの列数を $c_t + 2$ とし，このうち，計算用メモリスタセルの列数を c_t，チェックサムの列数を 2 とする．今，行方向から長さ M の入力電圧ベクトル $\boldsymbol{V}$ を印加することを考える．i 番目の行に入力される電圧値を V_i とすると，j 列目から出力される積和演算の出力値 $O_{(j)}$ と列チェックサムの出力値 $O_{c(1)}, O_{c(2)}$ は，それぞれ，式 (18)，(19)，(20) のように表すことができる．

$$O_{(j)} = \sum_{i=1}^{M} G'_{(i,j)} \cdot V_i \tag{18}$$

$$O_{c(1)} = \sum_{i=1}^{M} G_{c(i,1)} \cdot V_i \tag{19}$$

$$O_{c(2)} = \sum_{i=1}^{M} G_{c(i,2)} \cdot V_i \tag{20}$$

ここで，$G'_{(i,j)}$ は列チェックサム生成後のコンダクタンスの値を表し，故障が発生していなければ $G'_{(i,j)} = G_{(i,j)}$ となる．積和演算の結果に誤りが発生しているかど

うかを確認する検査フェーズでは，式 (21)，(22) のように，式 (18) の総和と重み無し列チェックサムの出力値 (式 (19)) を減算した結果と，式 (18) の重み付き総和の値と重み有り列チェックサムの出力値 (式 20) を減算した結果を用いる．

$$O_{c(1)} - \sum_{j=1}^{c_t} O_{(j)} = \sum_{i=1}^{M} G_{c(i,1)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} G'_{(i,j)} \cdot V_i \tag{21}$$

$$O_{c(2)} - \sum_{j=1}^{c_t} j \cdot O_{(j)} = \sum_{i=1}^{M} G_{c(i,2)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G'_{(i,j)} \cdot V_i \tag{22}$$

このとき，M 行 $(ct + 2)$ 列のメモリスタセルとチェックサムに故障が発生していない場合，式 (21)，(22) の結果は 0 となる．そのため，複数の列に同時に故障が発生していなければ，出力に誤りが発生していないことが分かる．

次に，列チェックサムに故障が発生しておらず，列チェックサム生成後の x 行 y 列目 $(x = 1, \cdots, M, y = 1, \cdots, c_t)$ のメモリスタのコンダクタンス $G_{(x,y)}$ に，元の重みから $+d$ だけ値が変化する故障が 1 つ発生した場合を考える．このとき，x 行 y 列目のメモリスタのコンダクタンスが $G'_{(x,y)} = (G_{(x,y)} + d)$ となるため，式 (21)，(22) より，式 (23)，(24) に示す結果が得られる．

$$\begin{aligned}
O_{c(1)} - \sum_{j=1}^{c_t} O_{(j)} &= \sum_{i=1}^{M} G_{c(i,1)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} G'_{(i,j)} \cdot V_i \\
&= \sum_{i=1}^{M} \sum_{j=1}^{c_t} G_{(i,j)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} G'_{(i,j)} \cdot V_i \\
&= G_{(x,y)} \cdot V_x - G'_{(x,y)} \cdot V_x \\
&= G_{(x,y)} \cdot V_x - (G_{(x,y)} + d) \cdot V_x \\
&= -d \cdot V_x
\end{aligned} \tag{23}$$

$$
\begin{aligned}
O_{c(2)} - \sum_{j=1}^{c_t} j \cdot O_{(j)} &= \sum_{i=1}^{M} G_{c(i,2)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G'_{(i,j)} \cdot V_i \\
&= \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G_{(i,j)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G'_{(i,j)} \cdot V_i \\
&= y \cdot G_{(x,y)} \cdot V_x - y \cdot G'_{(x,y)} \cdot V_x \\
&= y \cdot G_{(x,y)} \cdot V_x - y \cdot (G_{(x,y)} + d) \cdot V_x \\
&= -y \cdot d \cdot V_x
\end{aligned}
\tag{24}
$$

式 (23)，(24) の結果を式 (25) のように割り算すると，誤りが発生している出力値のインデックス y を得ることができる．

$$
\frac{-y \cdot d \cdot V_x}{-d \cdot V_x} = y \tag{25}
$$

$O_{c(1)}$ の値から $O(y)$ 以外の出力ベクトルの総和の値を減算することで，誤りを訂正した値を得ることができる．なお，$V_x = 0$ である場合は出力に誤りが伝播されない．

また，故障の発生箇所が同一の列であれば，2 つ以上の故障によって生じた誤りも訂正することができる．列チェックサムに故障が発生しておらず，x_1 行 y 列目 $(x_1 = 1, \cdots, M, y = 1, \cdots, c_t)$ のメモリスタセルのコンダクタンス $G'_{(x_1,y)}$ に元の重みから $+d_1$ だけ値が変化する故障と，x_2 行 y 列目 $(x_2 = 1, \cdots, M)$ のメモリスタセルのコンダクタンス $G'_{(x_2,y)}$ に，元の重みから $+d_2$ だけ値が変化する故障が発生する場合を考える．このとき，x_1 行 y 列目のメモリスタのコンダクタンスが $G'_{(x_1,y)} = (G_{(x_1,y)} + d_1)$，$x_2$ 行 y 列目のメモリスタのコンダクタンスが $G'_{(x_2,y)} = (G_{(x_2,y)} + d_2)$ となるため，式 (21)，(22) より，式 (26)，(27) に示す結果が

得られる．

$$
\begin{aligned}
O_{c(1)} - \sum_{j=1}^{c_t} O_{(j)} &= \sum_{i=1}^{M} G_{c(i,1)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} G'_{(i,j)} \cdot V_i \\
&= \sum_{i=1}^{M} \sum_{j=1}^{c_t} G_{(i,j)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} G'_{(i,j)} \cdot V_i \\
&= (G_{(x_1,y)} \cdot V_{x_1} - G'_{(x_1,y)} \cdot V_{x_1}) + (G_{(x_2,y)} \cdot V_{x_2} - G'_{(x_2,y)} \cdot V_{x_2}) \\
&= (G_{(x_1,y)} \cdot V_{x_1} - (G_{(x_1,y)} + d_1) \cdot V_{x_1}) \\
&\quad + (G_{(x_2,y)} \cdot V_{x_2} - (G_{(x_2,y)} + d_2) \cdot V_{x_2}) \\
&= -d_1 \cdot V_{x_1} - d_2 \cdot V_{x_2}
\end{aligned}
\tag{26}
$$

$$
\begin{aligned}
O_{c(2)} - \sum_{j=1}^{c_t} j \cdot O_{(j)} &= \sum_{i=1}^{M} G_{c(i,2)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G'_{(i,j)} \cdot V_i \\
&= \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G_{(i,j)} \cdot V_i - \sum_{i=1}^{M} \sum_{j=1}^{c_t} j \cdot G'_{(i,j)} \cdot V_i \\
&= (y \cdot G_{(x_1,y)} \cdot V_{x_1} - y \cdot G'_{(x_1,y)} \cdot V_{x_1}) \\
&\quad + (y \cdot G_{(x_2,y)} \cdot V_{x_2} - y \cdot G'_{(x_2,y)} \cdot V_{x_2}) \\
&= (y \cdot G_{(x_1,y)} \cdot V_{x_1} - y \cdot (G_{(x_1,y)} + d_1) \cdot V_{x_1}) \\
&\quad + (y \cdot G_{(x_2,y)} \cdot V_{x_2} - y \cdot (G_{(x_2,y)} + d_2) \cdot V_{x_2}) \\
&= -y \cdot d_1 \cdot V_{x_1} - y \cdot d_2 \cdot V_{x_2} = -y(d_1 \cdot V_{x_1} + d_2 \cdot V_{x_2})
\end{aligned}
\tag{27}
$$

式 (26)，(27) の結果を式 (28) のように割り算すると，誤りが発生している出力値のインデックス y を得ることができる．

$$
\frac{-y(d_1 \cdot V_{x_1} + d_2 \cdot V_{x_2})}{-(d_1 \cdot V_{x_1} + d_2 \cdot V_{x_2})} = y \tag{28}
$$

$O_{c(1)}$ の値から $O(y)$ 以外の出力ベクトルの総和の値を減算することで，誤りを訂正した値を得ることができる．なお，故障が 1 つの場合と同様に，$V_x = 0$ である場合は出力に誤りが伝播されない．

重み無し列チェックサム，もしくは，重み付き列チェックサムのどちらかの列に 1 つ以上の誤りが発生した場合は，式 (21)，式 (22) に示す $O_{c(1)}$ か $O_{c(2)}$ のどち

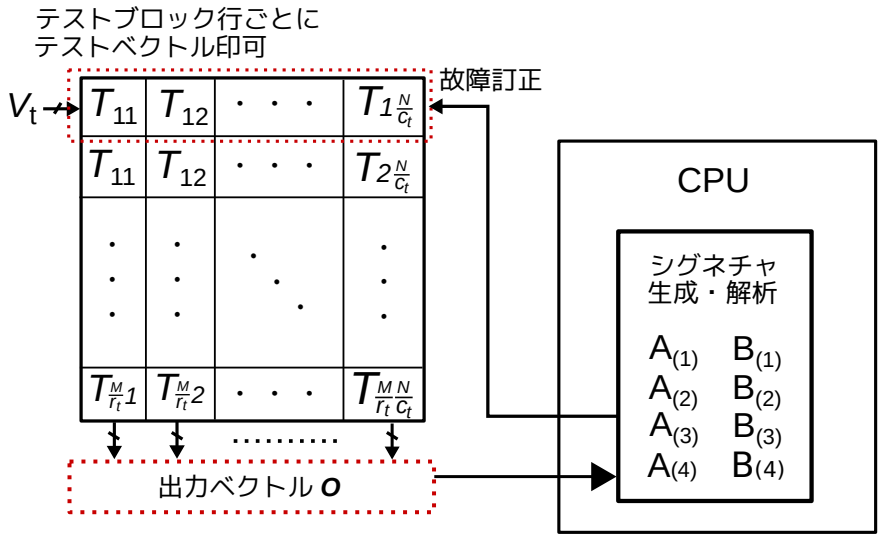


図 18 X-ABFT のオンラインテストの流れ

らか一方の出力値が 0 となり，もう一方の出力値が 0 以外の値となる．この出力パターンは列チェックサムの出力に誤りが生じた場合にのみ現れるため，計算用メモリスタセルによる積和演算の出力結果に誤りが生じていないことが分かる．

以上のように，行方向から入力電圧を印加し，列方向から出力される積和演算結果を，行列積和演算の結果と並列に出力されるチェックサムの値と比較することで，誤り訂正が実現できる．しかし，この誤り訂正は，逆方向伝播を考慮していない．逆方向伝搬では列方向から入力電圧を印加し，行方向から積和演算の結果を得るため，列チェックサムの値を用いた誤り訂正は実現できない．

4.2.2 オンラインテストによる故障の位置特定と訂正

X-ABFT のオンラインテストは，テストブロック中の同一行でない 2 つの故障の位置を特定し，故障を訂正する操作である．具体的には，図 18 に示すように，分割したテストブロックに行方向から 4 つのテストベクトルを印加し，その出力ベクトルを CPU 側で解析することで，故障の位置特定を行う．このとき，一度のテストベクトル印加で同一行のテストブロックの出力ベクトルを列方向から同時に取得することができ，それらを CPU 側で同時に解析する．

テストブロックの行数を r_t，列数を $c_t + 2$ とすると，テストブロック中のメモリスタのコンダクタンス $\boldsymbol{G}$ は式 (29) のように表すことができる．

$$\boldsymbol{G} = \begin{pmatrix} G'_{(1,1)} & \cdots & G'_{(1,ct)} & G_{c(1,1)} & G_{c(1,2)} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ G'_{(rt,1)} & \cdots & G'_{(rt,ct)} & G_{c(rt,1)} & G_{c(rt,2)} \end{pmatrix} \tag{29}$$

また，印加するテストベクトル $\boldsymbol{V}_t$ は，式 (30) のように作成する．

$$\boldsymbol{V}_t = \begin{pmatrix} V_{t(1,1)} & \cdots & V_{t(1,rt)} \\ V_{t(2,1)} & \cdots & V_{t(2,rt)} \\ V_{t(3,1)} & \cdots & V_{t(3,rt)} \\ V_{t(4,1)} & \cdots & V_{t(4,rt)} \end{pmatrix} = \begin{pmatrix} 2^{0\times0} & 2^{0\times1} & \cdots & 2^{0\times(rt-1)} \\ 2^{1\times0} & 2^{1\times1} & \cdots & 2^{1\times(rt-1)} \\ 2^{2\times0} & 2^{2\times1} & \cdots & 2^{2\times(rt-1)} \\ 2^{3\times0} & 2^{3\times1} & \cdots & 2^{3\times(rt-1)} \end{pmatrix} \tag{30}$$

式 (30) に示す 4 つのテストベクトルを式 (29) に示すテストブロックに印加することで，式 (31)，(32)，(33)，(34) に示すような出力を得ることができる．

$$\boldsymbol{V}_t \cdot \boldsymbol{C} = \begin{pmatrix} O_{(1,1)} & \cdots & O_{(1,ct)} & O_{c(1,1)} & O_{c(1,2)} \\ O_{(2,1)} & \cdots & O_{(2,ct)} & O_{c(2,1)} & O_{c(2,2)} \\ O_{(3,1)} & \cdots & O_{(3,ct)} & O_{c(3,1)} & O_{c(3,2)} \\ O_{(4,1)} & \cdots & O_{(4,ct)} & O_{c(4,1)} & O_{c(4,2)} \end{pmatrix} \tag{31}$$

$$O_{(k,j)} = \sum_{i=1}^{rt} V_{t(k,i)} \cdot G'_{(i,j)} = \sum_{i=1}^{rt} 2^{(k-1)(i-1)} \cdot G'_{(i,j)} \tag{32}$$

$$O_{c(k,1)} = \sum_{i=1}^{rt} V_{t(k,i)} \cdot G_{c(i,1)} = \sum_{i=1}^{rt} 2^{(k-1)(i-1)} \cdot G_{c(i,1)} \tag{33}$$

$$O_{c(k,2)} = \sum_{i=1}^{rt} V_{t(k,i)} \cdot G_{c(i,2)} = \sum_{i=1}^{rt} 2^{(k-1)(i-1)} \cdot G_{c(i,2)} \tag{34}$$

出力ベクトルを生成したら CPU 側へ転送し，出力ベクトルの解析を行う．まず，式 (35)，(36) に従って，8 つのシグネチャ $A_{(1)}$，$A_{(2)}$，$A_{(3)}$，$A_{(4)}$，$B_{(1)}$，$B_{(2)}$，

$B_{(3)}$，$B_{(4)}$ を生成する．

$$
\begin{aligned}
A_{(k)} &= \sum_{j=1}^{ct} O_{t(k,j)} - O_{c(k,1)} = \sum_{j=1}^{ct}\sum_{i=1}^{rt} V_{t(k,i)} G'_{(i,j)} - \sum_{i=1}^{rt} V_{t(k,i)} G_{c(i,1)} \\
&= \sum_{j=1}^{ct}\sum_{i=1}^{rt} 2^{(k-1)(i-1)} G'_{(i,j)} - \sum_{i=1}^{rt} 2^{(k-1)(i-1)} G_{c(i,1)}
\end{aligned}
\tag{35}
$$

$$
\begin{aligned}
B_{(k)} &= \sum_{j=1}^{ct} j \cdot O_{t(k,j)} - O_{c(k,2)} = \sum_{j=1}^{ct}\sum_{i=1}^{rt} j \cdot V_{t(k,i)} G'_{(i,j)} - \sum_{i=1}^{rt} V_{t(k,i)} G_{c(i,2)} \\
&= \sum_{j=1}^{ct}\sum_{i=1}^{rt} j \cdot 2^{(k-1)(i-1)} G'_{(i,j)} - \sum_{i=1}^{rt} 2^{(k-1)(i-1)} G_{c(i,2)}
\end{aligned}
\tag{36}
$$

もし，いずれのメモリスタセルにも故障が発生していなければ，式 (35)，(36) より，$A_{(1)} = A_{(2)} = A_{(3)} = A_{(4)} = 0$，$B_{(1)} = B_{(2)} = B_{(3)} = B_{(4)} = 0$ となり，テストブロック中に 2 つ以下の故障が発生していないことが分かる．その他の 2 つ以下の故障の発生パターンは，(i)1 つの故障が計算用メモリスタセルで発生，(ii)1 つの故障が重み無し列チェックサムで発生，(iii)1 つの故障が重み付き列チェックサムで発生，(iv)2 つの故障が計算用メモリスタセルで発生，(v)2 つの故障が計算用メモリスタセルと重み無し列チェックサムで発生，(vi)2 つの故障が計算用メモリスタセルと重み有り列チェックサムで発生，(vii)2 つの故障が重み無し列チェックサムと重み有り列チェックサムで発生，(viii)2 つの故障が重み無し列チェックサムで発生，(ix)2 つの故障が重み有り列チェックサムで発生の合計 9 パターンある．次の方法でシグネチャの解析を行うことで，テストブロックがどの故障パターンであるかを判別し，故障箇所と期待値との差を求めることができる．

(i)1 つの計算用メモリスタセルに故障が発生

テストブロック中の x_1 行 y_1 列目 ($x_1 = 1, \cdots, r_t, y_1 = 1, \cdots, c_t$) の 1 つのメモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ に，元の重みから $+d_1$ だけ値が変化する故障

が発生した場合，8 つのシグネチャの値は式 (37) のようになる．

$$\begin{cases} A_{(1)} = d_1 & B_{(1)} = y_1 d_1 \\ A_{(2)} = 2^{(x_1-1)} d_1 & B_{(2)} = 2^{(x_1-1)} y_1 d_1 \\ A_{(3)} = 2^{2(x_1-1)} d_1 & B_{(3)} = 2^{2(x_1-1)} y_1 d_1 \\ A_{(4)} = 2^{3(x_1-1)} d_1 & B_{(4)} = 2^{3(x_1-1)} y_1 d_1 \end{cases} \tag{37}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} = \frac{A_{(3)}}{A_{(2)}} = \frac{A_{(4)}}{A_{(3)}}$ かつ $\frac{B_{(2)}}{B_{(1)}} = \frac{B_{(3)}}{B_{(2)}} = \frac{B_{(4)}}{B_{(3)}}$ かつ $B_{(1)} \mod A_{(1)} = 0$ であるかどうかを確かめることで特定できる．故障のパラメータ x_1, y_1, d_1 はそれぞれ，$x_1 = \log_2(\frac{A_{(2)}}{A_{(1)}}) + 1$，$y_1 = \frac{B_{(1)}}{A_{(1)}}$，$y_1 = \frac{B_{(1)}}{A_{(1)}}$，$d_1 = A_{(1)}$ となる．

(ii)1 つの重み無し列チェックサムに故障が発生

テストブロック中の x_1 行目 $(x_1 = 1, \cdots, r_t)$ の 1 つの重み無し列チェックサムのコンダクタンス $G_{c(x_1,1)}$ に，元の重みから $+d_1$ だけ値が変化する故障が発生した場合，8 つのシグネチャの値は式 (38) のようになる．

$$\begin{cases} A_{(1)} = d_1 & B_{(1)} = 0 \\ A_{(2)} = 2^{(x_1-1)} d_1 & B_{(2)} = 0 \\ A_{(3)} = 2^{2(x_1-1)} d_1 & B_{(3)} = 0 \\ A_{(4)} = 2^{3(x_1-1)} d_1 & B_{(4)} = 0 \end{cases} \tag{38}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} = \frac{A_{(3)}}{A_{(2)}} = \frac{A_{(4)}}{A_{(3)}}$ かつ $B_{(1)} = B_{(2)} = B_{(3)} = B_{(4)} = 0$ であるかどうかを確かめることで特定できる．故障のパラメータ x_1, d_1 はそれぞれ，$x_1 = \log_2(\frac{A_{(2)}}{A_{(1)}}) + 1$，$d_1 = A_{(1)}$ となる．

(iii)1 つの重み有り列チェックサムに故障が発生

テストブロック中の x_1 行目 $(x_1 = 1, \cdots, r_t)$ の 1 つの重み有り列チェックサムのコンダクタンス $G_{c(x_1,2)}$ に，元の重みから $+d_1$ だけ値が変化する故障が発生した

場合，8 つのシグネチャの値は式 (39) のようになる．

$$\begin{cases} A_{(1)} = 0 & B_{(1)} = d_1 \\ A_{(2)} = 0 & B_{(2)} = 2^{(x_1-1)}d_1 \\ A_{(3)} = 0 & B_{(3)} = 2^{2(x_1-1)}d_1 \\ A_{(4)} = 0 & B_{(4)} = 2^{3(x_1-1)}d_1 \end{cases} \tag{39}$$

この故障パターンは，$A_{(1)} = A_{(2)} = A_{(3)} = A_{(4)} = 0$ かつ $\frac{B_{(2)}}{B_{(1)}} = \frac{B_{(3)}}{B_{(2)}} = \frac{B_{(4)}}{B_{(3)}}$ であるかどうかを確かめることで特定できる．故障のパラメータ x_1, d_1 はそれぞれ，$x_1 = \log_2(\frac{B_{(2)}}{B_{(1)}}) + 1$，$d_1 = B_{(1)}$ となる．

(iv)2 つの計算用メモリスタセルに故障が発生

テストブロック中の x_1 行 y_1 列目 $(x_1 = 1, \cdots, r_t, y_1 = 1, \cdots, c_t)$ のメモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ に，元の重みから $+d_1$ だけ値が変化する故障と，x_2 行 y_2 列目 $(x_2 = 1, \cdots, r_t, y_2 = 1, \cdots, c_t)$ のメモリスタセルのコンダクタンス $G'_{(x_2,y_2)}$ に，元の重みから $+d_2$ だけ値が変化する故障が発生した場合を考える．このとき，$x_1 \neq x_2$ であれば，8 つのシグネチャの値は式 (40) のようになる．

$$\begin{cases} A_{(1)} = d_1 + d_2 & B_{(1)} = y_1 d_1 + y_2 d_2 \\ A_{(2)} = 2^{(x_1-1)}d_1 + 2^{(x_2-1)}d_2 & B_{(2)} = 2^{(x_1-1)}y_1 d_1 + 2^{(x_2-1)}y_2 d_2 \\ A_{(3)} = 2^{2(x_1-1)}d_1 + 2^{2(x_2-1)}d_2 & B_{(3)} = 2^{2(x_1-1)}y_1 d_1 + 2^{2(x_2-1)}y_2 d_2 \\ A_{(4)} = 2^{3(x_1-1)}d_1 + 2^{3(x_2-1)}d_2 & B_{(4)} = 2^{3(x_1-1)}y_1 d_1 + 2^{3(x_2-1)}y_2 d_2 \end{cases} \tag{40}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} \neq \frac{A_{(3)}}{A_{(2)}} \neq \frac{A_{(4)}}{A_{(3)}}$ かつ $\frac{B_{(2)}}{B_{(1)}} \neq \frac{B_{(3)}}{B_{(2)}} \neq \frac{B_{(4)}}{B_{(3)}}$ であるかどうかを確かめることで特定できる．式 40 には 6 個の未知変数が含まれるが，この式の解法は文献 [35] で提案されている．$\eta = \frac{(A_{(1)}A_{(4)} - A_{(2)}A_{(3)})}{(A_{(1)}A_{(3)} - A_{(2)}^2)} = 2^{x_1-1} + 2^{x_2-1}$ とする．一般性を失わないように，$x_1 > x_2, 2^{x_1-1} < \eta < 2^{x_1}$ と仮定すると，故障の位置 (x_1, y_1) と (x_2, y_2) と故障によって生じた元のコンダクタンスとの差 d_1，d_2 は，式 (41) を上

から順に解いていくことで各故障パラメータを算出することができる．

$$\begin{cases} x_1 = \lfloor \log_2(\eta) \rfloor + 1 \\ x_2 = \log_2(\eta - 2^{x_1-1}) + 1 \\ d_1 = \dfrac{-A_{(2)} + 2^{x_2-1}A_{(1)}}{2^{x_2-1} - 2^{x_1-1}} \\ d_2 = -A_{(1)} - d_1 \\ y_1 = \dfrac{-B_{(2)} + 2^{x_2-1}B_{(1)}}{d_1(2^{x_1-1} - 2^{x_2-1})} \\ y_2 = \dfrac{-B_{(1)} - y_1 d_1}{d_2} \end{cases} \tag{41}$$

ただし，同一行で故障が発生している場合（$x_1 \neq x_2$），式 (41) は $\eta = 0$ となり，また，d_1, d_2, y_1, y_2 の分母が 0 となるため解くことができない．同一行で故障が発生した場合のシグネチャの値は，x_2 を x_1 として書き直すことで，式 (42) のようになる．

$$\begin{cases} A_{(1)} = d_1 + d_2 & B_{(1)} = y_1 d_1 + y_2 d_2 \\ A_{(2)} = 2^{(x_1-1)}(d_1 + d_2) & B_{(2)} = 2^{(x_1-1)}(y_1 d_1 + y_2 d_2) \\ A_{(3)} = 2^{2(x_1-1)}(d_1 + d_2) & B_{(3)} = 2^{2(x_1-1)}(y_1 d_1 + y_2 d_2) \\ A_{(4)} = 2^{3(x_1-1)}(d_1 + d_2) & B_{(4)} = 2^{3(x_1-1)}(y_1 d_1 + y_2 d_2) \end{cases} \tag{42}$$

式 (42) に示すように，同一行で故障が発生したかどうかは，$\frac{A_{(2)}}{A_{(1)}} = \frac{A_{(3)}}{A_{(2)}} = \frac{A_{(4)}}{A_{(3)}}$ かつ $\frac{B_{(2)}}{B_{(1)}} = \frac{B_{(3)}}{B_{(2)}} = \frac{B_{(4)}}{B_{(3)}}$ かつ $\frac{B_{(1)}}{A_{(1)}} = \frac{B_{(2)}}{A_{(2)}} = \frac{B_{(3)}}{A_{(3)}} = \frac{B_{(4)}}{A_{(4)}}$ であるかどうかを確かめることで調べることができる．この場合，2 つの故障が発生している行 x_1 は，$x_1 = \log_2(\frac{A_{(2)}}{A_{(1)}}) + 1$ とすることで算出することができるが，y_1, y_2, d_1, d_2 は算出することができない．

(v)1 つの計算用メモリスタセルと 1 つの重み無し列チェックサムに故障が発生

テストブロック中の x_1 行 y_1 列目 ($x_1 = 1, \cdots, r_t, y_1 = 1, \cdots, c_t$) のメモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ に元の重みから $+d_1$ だけ値が変化する故障と，x_2 行目 ($x_2 = 1, \cdots, r_t, x_2 \neq x_1$) の重み無し列チェックサムのコンダクタンス $G_{c(x_2,1)}$

に，元の重みから $+d_2$ だけ値が変化する故障が発生した場合を考える．このとき，$x_1 \neq x_2$ であれば，8 つのシグネチャの値は式 (43) のようになる．

$$\begin{cases} A_{(1)} = d_1 + d_2 & B_{(1)} = y_1 d_1 \\ A_{(2)} = 2^{(x_1-1)} d_1 + 2^{(x_2-1)} d_2 & B_{(2)} = 2^{(x_1-1)} y_1 d_1 \\ A_{(3)} = 2^{2(x_1-1)} d_1 + 2^{2(x_2-1)} d_2 & B_{(3)} = 2^{2(x_1-1)} y_1 d_1 \\ A_{(4)} = 2^{3(x_1-1)} d_1 + 2^{3(x_2-1)} d_2 & B_{(4)} = 2^{3(x_1-1)} y_1 d_1 \end{cases} \tag{43}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} \neq \frac{A_{(3)}}{A_{(2)}} \neq \frac{A_{(4)}}{A_{(3)}}$ かつ $\frac{B_{(2)}}{B_{(1)}} = \frac{B_{(3)}}{B_{(2)}} = \frac{B_{(4)}}{B_{(3)}}$ であるかどうかを確かめることで特定できる．故障パラメータ x_1, x_2, d_1, d_2 については，式 (41) と同じ方法で算出することが可能で，y_1 については，$y_1 = B_{(1)}/d_1$ とすることで算出することができる．

ただし，故障パターン (iv) と同様に，故障の発生箇所が同一行であった場合（$x_1 = x_2$）は，y_1, d_1, d_2 を得ることができず，故障が発生している行の情報 x_1 の値しか得ることができない．

(vi)1 つの計算用メモリスタセルと 1 つの重み有りチェックサムに故障が発生

テストブロック中の x_1 行 y_1 列目 $(x_1 = 1, \cdots, r_t, y_1 = 1, \cdots, c_t)$ の計算用メモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ が元の値から $+d_1$ だけ値が変化する故障と，x_2 行目 $(x_2 = 1, \cdots, r_t, x_2 \neq x_1)$ の重み有り列チェックサムのコンダクタンス $G_{c(x_2,2)}$ が元の値から $+d_2$ だけ値が変化する故障が発生した場合を考える．このとき，$x_1 \neq x_2$ であれば，8 つのシグネチャの値は式 (44) のようになる．

$$\begin{cases} A_{(1)} = d_1 & B_{(1)} = y_1 d_1 + d_2 \\ A_{(2)} = 2^{(x_1-1)} d_1 & B_{(2)} = 2^{(x_1-1)} y_1 d_1 + 2^{(x_2-1)} d_2 \\ A_{(3)} = 2^{2(x_1-1)} d_1 & B_{(3)} = 2^{2(x_1-1)} y_1 d_1 + 2^{2(x_2-1)} d_2 \\ A_{(4)} = 2^{3(x_1-1)} d_1 & B_{(4)} = 2^{3(x_1-1)} y_1 d_1 + 2^{3(x_2-1)} d_2 \end{cases} \tag{44}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} = \frac{A_{(3)}}{A_{(2)}} = \frac{A_{(4)}}{A_{(3)}}$ かつ $\frac{B_{(2)}}{B_{(1)}} \neq \frac{B_{(3)}}{B_{(2)}} \neq \frac{B_{(4)}}{B_{(3)}}$ であるかどうかを確かめることで特定できる．故障パラメータ x_1, x_2, d_1, d_2, y_1 は，$\eta = \frac{(B_{(1)}B_{(4)} - B_{(2)}B_{(3)})}{(B_{(1)}B_{(3)} - B_{(2)}^2)} = 2^{x_1-1} + 2^{x_2-1}$ として，式 (41) を解くことで算出することができる．

ただし，故障パターン (iv)，(v) と同様に，故障の発生箇所が同一行であった場合（$x_1 = x_2$）は，y_1, d_1, d_2 を得ることができず，故障が発生している行の情報 x_1 の値しか得ることができない．

(vii)1 つの重み無し列チェックサムと 1 つの重み有り列チェックサムに故障が発生

テストブロック中の x_1 行目 ($x_1 = 1, \cdots, r_t$) の重み無し列チェックサムのコンダクタンス $G_{c(x_1,1)}$ が元の値から $+d_1$ だけ値が変化する故障と，x_2 行目 ($x_2 = 1, \cdots, r_t, x_2 \neq x_1$) の重み有りチェックサムのコンダクタンス $G_{c(x_2,2)}$ が元の値から $+d_2$ だけ値が変化する故障が発生した場合を考える．このとき，$x_1 \neq x_2$ であれば，8 つのシグネチャの値は式 (45) のようになる．

$$\begin{cases} A_{(1)} = d_1 & B_{(1)} = d_2 \\ A_{(2)} = 2^{(x_1-1)}d_1 & B_{(2)} = 2^{(x_2-1)}d_2 \\ A_{(3)} = 2^{2(x_1-1)}d_1 & B_{(3)} = 2^{2(x_2-1)}d_2 \\ A_{(4)} = 2^{3(x_1-1)}d_1 & B_{(4)} = 2^{3(x_2-1)}d_2 \end{cases} \tag{45}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} = \frac{A_{(3)}}{A_{(2)}} = \frac{A_{(4)}}{A_{(3)}}$ かつ $\frac{B_{(2)}}{B_{(1)}} = \frac{B_{(3)}}{B_{(2)}} = \frac{B_{(4)}}{B_{(3)}}$ かつ $\frac{B_{(1)}}{A_{(1)}} \neq \frac{B_{(2)}}{A_{(2)}} \neq \frac{B_{(3)}}{A_{(3)}} \neq \frac{B_{(4)}}{A_{(4)}}$ であるかどうかを確かめることで特定できる．故障パラメータ x_1, x_2, d_1, d_2 は，$d_1 = A_{(1)}$，$x_1 = \log_2(\frac{A_{(2)}}{d_1})$，$d_2 = B_{(1)}$，$x_2 = \log_2(\frac{B_{(2)}}{d_2})$ とすることで，算出することができる．

ただし，故障パターン (iv)，(v)，(vi) と同様に，故障の発生箇所が同一行であった場合（$x_1 = x_2$）は，d_1, d_2 を得ることができず，故障が発生している行の情報 x_1 の値しか得ることができない．

(viii)2 つの重み無し列チェックサムに故障が発生

テストブロック中の x_1 行目 ($x_1 = 1, \cdots, r_t$) の重み無しチェックサムのコンダクタンス $G_{c(x_1,1)}$ が元の値から $+d_1$ だけ値が変化する故障と，x_2 行目 ($x_2 = 1, \cdots, r_t, x_2 \neq x_1$) の重み無しチェックサムのコンダクタンス $G_{c(x_2,1)}$ が元の値から $+d_2$ だけ値が

変化する故障が発生した場合，8 つのシグネチャの値は式 (46) のようになる．

$$\begin{cases} A_{(1)} = d_1 + d_2 & B_{(1)} = 0 \\ A_{(2)} = 2^{(x_1-1)}d_1 + 2^{(x_2-1)}d_2 & B_{(2)} = 0 \\ A_{(3)} = 2^{2(x_1-1)}d_1 + 2^{2(x_2-1)}d_2 & B_{(3)} = 0 \\ A_{(4)} = 2^{3(x_1-1)}d_1 + 2^{2(x_2-1)}d_2 & B_{(4)} = 0 \end{cases} \tag{46}$$

この故障パターンは，$\frac{A_{(2)}}{A_{(1)}} \neq \frac{A_{(3)}}{A_{(2)}} \neq \frac{A_{(4)}}{A_{(3)}}$ かつ $B_{(1)} = B_{(2)} = B_{(3)} = B_{(4)}$ であるかどうかを確かめることで特定できる．故障パラメータ x_1, x_2, d_1, d_2 は，式 (41) を解くことで算出することができる．

(ix)2 つの重み有り列チェックサムに故障が発生

テストブロック中の x_1 行目 $(x_1 = 1, \cdots, r_t)$ の重み有りチェックサムのコンダクタンス $G_{c(x_1,2)}$ が元の値から $+d_1$ だけ値が変化する故障と，x_2 行目 $(x_2 = 1, \cdots, r_t, x_2 \neq x_1)$ の重み有り列チェックサムのコンダクタンス $G'_{c(x_2,1)}$ が元の値から $+d_2$ だけ値が変化する故障が発生した場合，8 つのシグネチャの値は式 (47) のようになる．

$$\begin{cases} A_{(1)} = 0 & B_{(1)} = d_1 + d_2 \\ A_{(2)} = 0 & B_{(2)} = 2^{(x_1-1)}d_1 + 2^{(x_2-1)}d_2 \\ A_{(3)} = 0 & B_{(3)} = 2^{2(x_1-1)}d_1 + 2^{2(x_2-1)}d_2 \\ A_{(4)} = 0 & B_{(4)} = 2^{3(x_1-1)}d_1 + 2^{3(x_2-1)}d_2 \end{cases} \tag{47}$$

この故障パターンは，$A_{(1)} = A_{(2)} = A_{(4)} = A_{(3)}$ かつ $\frac{B_{(2)}}{B_{(1)}} \neq \frac{B_{(3)}}{B_{(2)}} \neq \frac{B_{(4)}}{B_{(3)}}$ であるかどうかを確かめることで特定できる．故障パラメータ x_1, x_2, d_1, d_2 は，$\eta = \frac{(B_{(1)}B_{(4)} - B_{(2)}B_{(3)})}{(B_{(1)}B_{(3)} - B_{(2)}^2)} = 2^{x_1-1} + 2^{x_2-1}$ として，式 (41) を解くことで算出することができる．

以上の方法で，同一行でない 2 つ以下の故障を訂正することができる．2 つの故障が同一行で発生している場合（$x_1 = x_2$) は，式 (41) を解くことができなくなり，故障位置の特定と訂正が実現できない．

4.3 提案する耐故障設計

本章では，X-ABFT の機能に加えて，逆方向伝搬時の誤り訂正と，同一行で発生した故障の位置を特定・訂正できるオンラインテストを可能とする耐故障設計を提案する．提案する耐故障設計では，列方向だけでなく，行方向にもチェックサムを付加する．行チェックサムの値を利用することで，順方向伝播と同様に，逆方向伝搬時にもチェックサムの値を並列に取り出すことができるようになるため，誤りを訂正できるようになる．また，オンラインテスト時に行チェックサムの値を利用し，列方向からもテストベクトルを印加してテストすることで，同一行で発生した故障の位置特定・修復を行うオンラインテストを実現できるようになる．

提案する耐故障設計手法の動作フローを図 19 に示す．提案手法では，X-ABFT と同様に，まず，図 20 のように，1 つのクロスバーアレイを複数のブロックに分割する．このとき，付加する行チェックサムを考慮して，1 つのテストブロックのサイズが列数が r_t+2，行数が c_t+2 となるように分割を行う．そのため，チェックサム生成後のクロスバーアレイの合計列数は $N+2\times(N/c_t)$，行数は $M+2\times(M/r_t)$ となる．このとき，各テストブロックごとに，行チェックサムと列チェックサムが交差する部分に 2×2 の余分なメモリスタセルが発生するが，このメモリスタセルは誤り訂正とテストでは利用しないため，0 に設定しておく．

テストブロック分割後は，エンコーディングを行い，各テストブロックごとに行・列チェックサムを生成する．列チェックサムは，X-ABFT と同様の方法で生成する．行チェックサムは図 20 のように，各テストブロックごとに重み無し行チェックサムと重み有り行チェックサムという，2 行分の行チェックサムを生成する．c_t，r_t をテストブロックの計算用メモリスタの列数と行数，$G_{(i,j)}$ を i 行 j 列目の計算用メモリスタセルのコンダクタンスとすると，式 (16) に示すように，重み無し行チェックサムのコンダクタンス $G_{r(1,j)}$ は j 列目の計算用メモリスタセルのコンダクタンスの総和となる．

$$G_{r(1,j)} = \sum_{i=1}^{rt} G_{(i,j)} \tag{48}$$

また，式 (17) に示すように，j 列目の重み有り行チェックサムのコンダクタンス $G_{r(2,j)}$ は，j 列目の計算用メモリスタセルのコンダクタンスの値に行番号 i を乗算

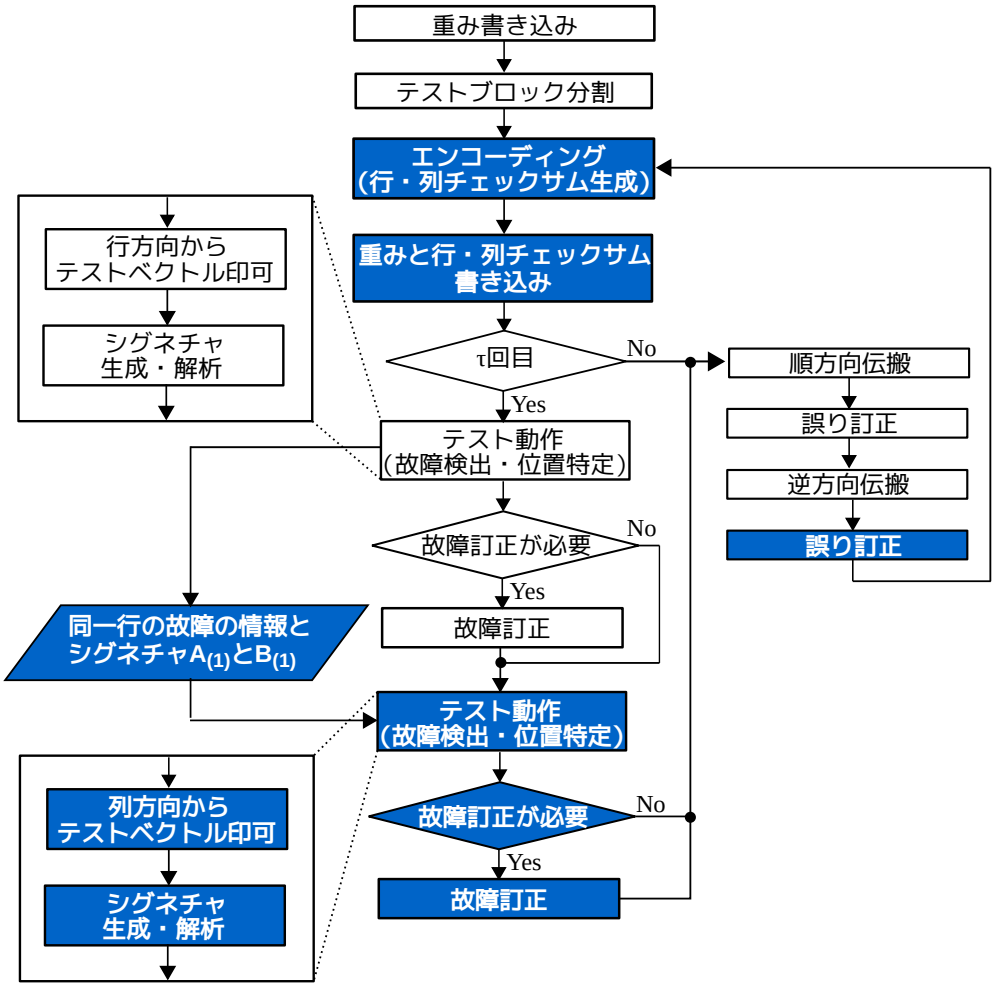


図 19 提案手法のフローチャート

した値の総和となる．

$$G_{r(2,j)} = \sum_{i=1}^{rt} i \cdot G_{(i,j)} \tag{49}$$

行チェックサム生成後は，τ 回の書き込み毎にテストを行う．提案手法では，X-ABFT と同様に，まず，行方向からテストブロックにテストベクトルを印加し，列チェックサムの値を利用してテストを行う．このとき，各テストブロックで，同一行の故障が発生しているかどうかの情報と，もし同一行の故障が発生していた場合，シグネチャ $A_{(1)}$，$B_{(1)}$ の値を保持しておく．そして，行方向からテストベク

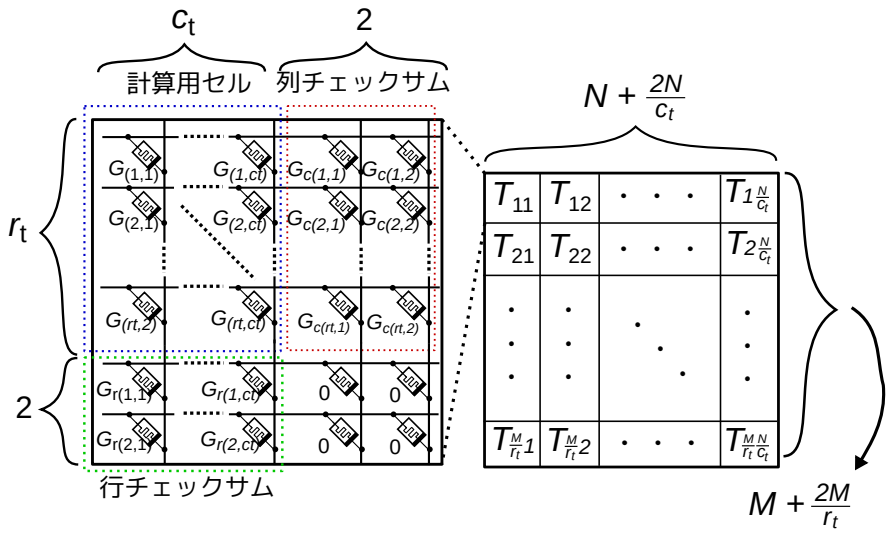


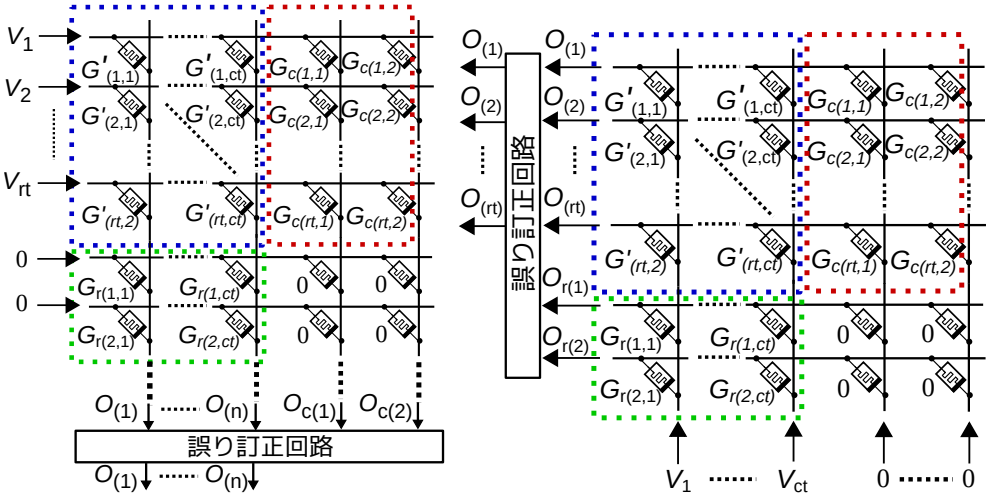
図 20 提案手法のテストブロックの構成

トル印加してテストを行い故障を修復した後は，列方向からテストベクトルを印加し，行チェックサムの値を利用してテストを行う．列方向からテストベクトルを印加し，それによって得られるシグネチャと行方向テストベクトル印加によるテストで得られたシグネチャ$A_{(1)}$，$B_{(1)}$ を解析することで，同一行で発生している故障に関しても故障位置を特定し，修復することができるようになる．列方向からテストベクトルを印加して行うオンラインテストの詳細については，4.3.2 節に示す．

テスト終了後，もしくは，書き込み回数がτ に達していない場合，順方向伝搬と逆方向伝搬を行うが，提案手法では順方向伝搬だけでなく逆方向伝搬時にも誤り訂正を行う．提案手法の誤り訂正動作の詳細は，4.3.1 節に示す．

4.3.1 提案手法による順方向・逆方向伝搬の誤り訂正

提案手法による順方向・逆方向伝播時の電圧印加方法を図 21 に示す．図 21(a) に示すように，提案手法での順方向伝播では，各テストブロックごとの行チェックサムに対して電圧を印加しないようにする．これにより，4.2.1 節と同様の方法で順方向伝播の出力の 1 つの誤りを訂正できる．また，図 21(b) に示すように，提



(a) 順方向伝播　　(b) 逆方向伝播

図 21 提案手法による順方向・逆方向伝播時の電圧印加方法

案手法での逆方向伝播では，各テストブロックごとの列チェックサムに対して電圧を印加しないようにする．

逆方向伝搬の誤り訂正では，図 22 に示すように，分割した行毎に 1bit までの出力ベクトルの誤りを訂正することができる．各行ごとのテストブロックの計算用メモリスタセルの合計列数を N，行数を r_t+2 とし，このうち，計算用メモリスタセルの行数を r_t，行チェックサムの行数を 2 とし，長さ N の入力電圧ベクトル $\boldsymbol{V}$ を列方向から印加することを考える．j 列目に入力される電圧を V_j とすると，i 行目の出力値 $O_{(i)}$ と行チェックサムの出力値 $O_{r(1)}, O_{r(2)}$ は，それぞれ，式 (50), (51), (52) のように表すことができる．

$$O_{(i)} = \sum_{j=1}^{N} G'_{(i,j)} \cdot V_j \tag{50}$$

$$O_{r(1)} = \sum_{j=1}^{N} G_{r(1,j)} \cdot V_j \tag{51}$$

$$O_{r(2)} = \sum_{j=1}^{N} G_{r(2,j)} \cdot V_j \tag{52}$$

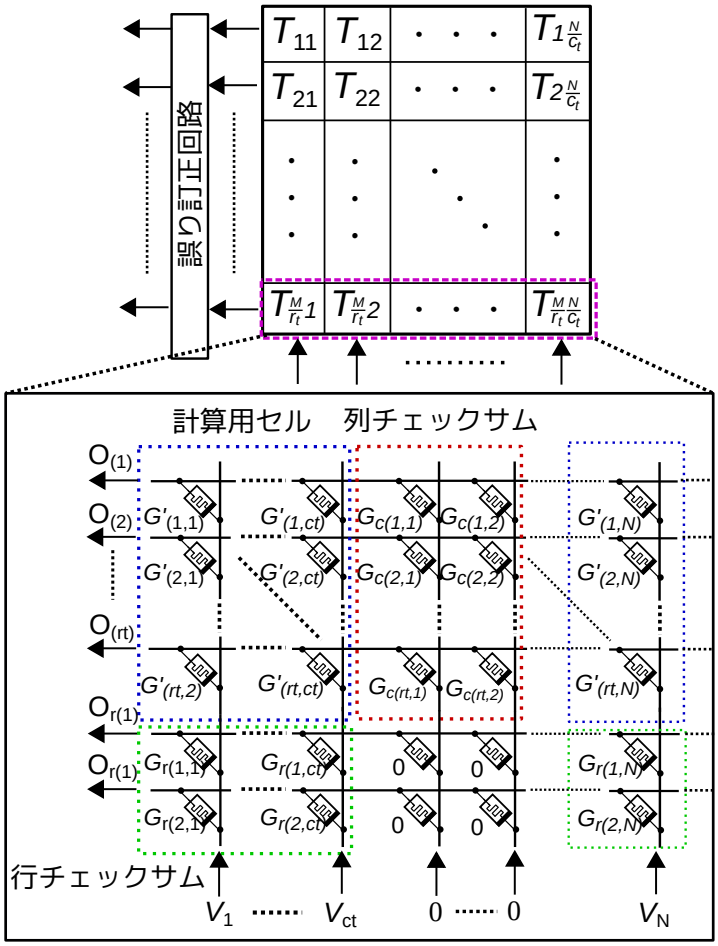


図 22 逆方向伝播の誤り訂正

ここで，$G'_{(i,j)}$ はチェックサム生成後の i 行 j 列目のメモリスタのコンダクタンスを表し，$G_{r(1,j)}, G_{r(2,j)}$ は重み無し・重み付き行チェックサムのコンダクタンスを表す．

逆方向伝播の結果の誤りの有無を確認する検査フェーズでは，式 (53)，(54) のように，式 (50) の総和と重み無し列チェックサムの出力値 (式 (51)) を減算した結果と，式 (50) の重み付き総和の値と重み有り列チェックサムの出力値 (式 52) を減算した結果を用いる．

$$O_{r(1)} - \sum_{i=1}^{r_t} O_{(i)} = \sum_{j=1}^{N} G_{r(1,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} G'_{(i,j)} \cdot V_j \tag{53}$$

$$O_{r(2)} - \sum_{i=1}^{r_t} i \cdot O_{(i)} = \sum_{j=1}^{N} G_{(2,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G'_{(i,j)} \cdot V_j \tag{54}$$

このとき，$(r_t + 2)$ 行 N 列のいずれのメモリスタセルにも故障が発生していない場合，式 (53)，(54) の結果はいずれも 0 となり，誤りが発生していないことが分かる．

次に，行チェックサムに故障が発生しておらず，行･列チェックサム生成後の x 行 y 列目 $(x = 1, \cdots, r_t, y = 1, \cdots, N)$ のメモリスタセルのコンダクタンス $G'_{(x,y)}$ に元の値から $+d$ だけ値が変化する故障が 1 つ発生した場合を考える．このとき，x 行 y 列目のメモリスタのコンダクタンスが $G'_{(x,y)} = (G_{(x,y)} + d)$ となるため，式 (53)，(54) より，式 (55)，(56) に示す結果が得られる．

$$\begin{aligned}
O_{r(1)} - \sum_{i=1}^{r_t} O_{(i)} &= \sum_{j=1}^{N} G_{r(1,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} G'_{(i,j)} \cdot V_j \\
&= \sum_{i=1}^{r_t} \sum_{j=1}^{N} G_{(i,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} G'_{(i,j)} \cdot V_j \\
&= G_{(x,y)} \cdot V_y - G'_{(x,y)} \cdot V_y \\
&= G_{(x,y)} \cdot V_y - (G_{(x,y)} + d) \cdot V_y \\
&= -d \cdot V_y
\end{aligned} \tag{55}$$

$$\begin{aligned}
O_{r(2)} - \sum_{i=1}^{r_t} i \cdot O_{(i)} &= \sum_{j=1}^{N} G_{r(2,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G'_{(i,j)} \cdot V_j \\
&= \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G_{(i,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G'_{(i,j)} \cdot V_j \\
&= x \cdot G_{(x,y)} \cdot V_y - x \cdot G'_{(x,y)} \cdot V_y \\
&= x \cdot G_{(x,y)} \cdot V_y - x \cdot (G_{(x,y)} + d) \cdot V_y \\
&= -x \cdot d \cdot V_y
\end{aligned} \tag{56}$$

式 (55)，(56) の結果を式 (57) のように割り算すると，誤りが発生している出力値のインデックス x を得ることができる．

$$\frac{-x \cdot d \cdot V_y}{-d \cdot V_y} = x \tag{57}$$

$O_{r(1)}$ の値から $O(x)$ 以外の出力の総和の値を減算することで，誤りを訂正した値を得ることができる．

また，同一の行で発生していれば，2 つ以上の故障によって生じた誤りも訂正することができる．x 行 y_1 列目 $(x = 1, \cdots, r_t, y_1 = 1, \cdots, N)$ のメモリスタセルのコンダクタンス $G'_{(x,y_1)}$ に $+d_1$ だけ値が変化する故障と，x 行 y_2 列目 $(x = 1, \cdots, r_t, y_2 = 1, \cdots, N)$ のメモリスタセルのコンダクタンス $G_{(x,y_2)}$ に $+d_2$ だけ値が変化する故障が発生する場合を考える．このとき，x 行 y_1 列目のメモリスタのコンダクタンスが $G'_{(x,y_1)} = (G_{(x,y_1)} + d_1)$，$x$ 行 y_2 列目のメモリスタのコンダクタンスが $G'_{(x,y_2)} = (G_{(x,y_2)} + d_2)$ となるため，式 (53)，(54) より，式 (58)，(59) に示す結果が得られる．

$$
\begin{aligned}
O_{r(1)} - \sum_{i=1}^{r_t} O_{(i)} &= \sum_{j=1}^{N} G_{r(1,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} G'_{(i,j)} \cdot V_j \\
&= \sum_{i=1}^{r_t} \sum_{j=1}^{N} G_{(i,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} G'_{(i,j)} \cdot V_j \\
&= (G_{(x,y_1)} \cdot V_{y_1} - G'_{(x,y_1)} \cdot V_{y_1}) + (G_{(x,y_2)} \cdot V_{y_2} - G'_{(x,y_2)} \cdot V_{y_2}) \\
&= (G_{(x,y_1)} \cdot V_{y_1} - (G_{(x,y_1)} + d_1) \cdot V_{y_1}) \\
&\quad + (G_{(x,y_2)} \cdot V_{y_2} - (G_{(x,y_2)} + d_2) \cdot V_{y_2}) \\
&= -d_1 \cdot V_{y_1} - d_2 \cdot V_{y_2}
\end{aligned}
\tag{58}
$$

$$
\begin{aligned}
O_{r(2)} - \sum_{i=1}^{r_t} i \cdot O_{(i)} &= \sum_{j=1}^{N} G_{r(2,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G'_{(i,j)} \cdot V_j \\
&= \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G_{(i,j)} \cdot V_j - \sum_{i=1}^{r_t} \sum_{j=1}^{N} i \cdot G'_{(i,j)} \cdot V_j \\
&= (x \cdot G_{(x,y_1)} \cdot V_{y_1} - x \cdot G'_{(x,y_1)} \cdot V_{y_1}) \\
&\quad + (x \cdot G_{(x,y_2)} \cdot V_{y_2} - x \cdot G'_{(x,y_2)} \cdot V_{y_2}) \\
&= (x \cdot G_{(x,y_1)} \cdot V_{y_1} - x \cdot (G_{(x,y_1)} + d_1) \cdot V_{y_1}) \\
&\quad + (x \cdot G_{(x,y_2)} \cdot V_{y_2} - x \cdot (G_{(x,y_2)} + d_2) \cdot V_{y_2}) \\
&= -x \cdot d_1 \cdot V_{y_1} - x \cdot d_2 \cdot V_{y_2} = -y(d_1 \cdot V_{y_1} + d_2 \cdot V_{y_2})
\end{aligned}
\tag{59}
$$

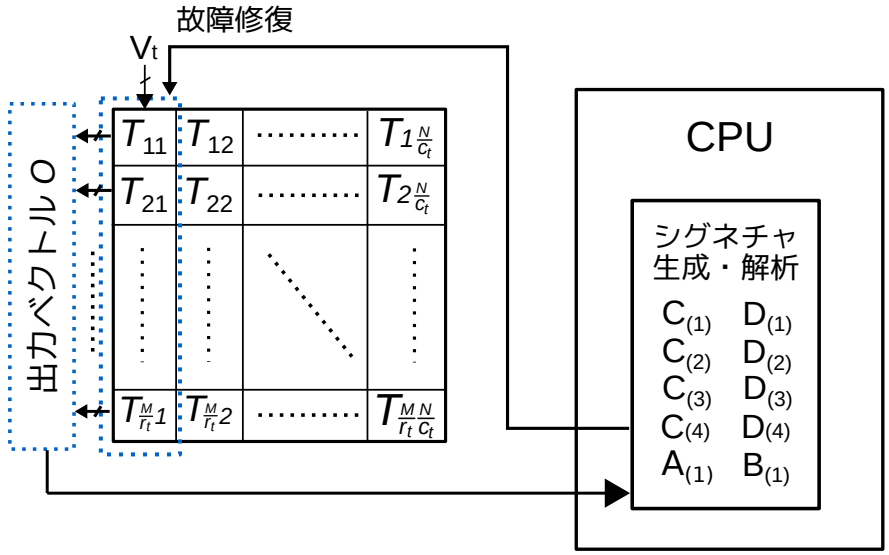


図 23 列方向テストベクトル印加によるオンラインテストの流れ

式 (58)，(59) の結果を式 (60) のように割り算すると，誤りが発生している出力値のインデックス x を得ることができる．

$$\frac{-x(d_1 \cdot V_{y_1} + d_2 \cdot V_{y_2})}{-(d_1 \cdot V_{y_1} + d_2 \cdot V_{y_2})} = x \tag{60}$$

誤り訂正は，$O_{r(1)}$ の値から $O(x)$ 以外の出力ベクトルの総和の値を減算することで，誤りを訂正した値を得ることができる．

重み無し行チェックサム，もしくは，重み付き行チェックサムのどちらかの列に 1 つ以上の誤りが発生した場合は，式 (53)，式 (54) に示す $O_{r(1)}$ か $O_{r(2)}$ のどちらか一方の出力値が 0 となり，もう一方の出力値が 0 以外の値となる．この出力パターンはチェックサムの出力に誤りが生じた場合にのみ現れるため，計算用メモリスタセルによる積和演算の出力結果に誤りが生じていないことが分かる．このような計算方法で，逆方向伝搬の誤りを訂正できる．

4.3.2 列方向テストベクトル印加によるオンラインテスト

列方向テストベクトル印加によるオンラインテストは，行方向テストベクトル印加によってテストを行った後に行われる．行方向テストベクトル印加によって

テストを行ったとき，もし，そのテストブロックで同一行の故障が発生していれば，そのテストブロックに対するシグネチャ $A_{(1)}$，$B_{(1)}$ を保持しておくようにする．そして，図 23 に示すように，列方向からテストベクトルを印加し，その出力ベクトルを用いて故障位置の特定を行う際に，$A_{(1)}$，$B_{(1)}$ の値を利用しながらシグネチャの解析を行う．

テストブロックの行数を $r_t + 2$，列数を $c_t + 2$ とすると，提案手法のテストブロックは式 (61) のように表すことができる．

$$\boldsymbol{G} = \begin{pmatrix} G'_{(1,1)} & \cdots & G'_{(1,c_t)} & G_{c(1,1)} & G_{c(1,2)} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ G'_{(r_t,1)} & \cdots & G'_{(r_t,c_t)} & G_{c(r_t,1)} & G_{c(r_t,2)} \\ G_{r(1,1)} & \cdots & G_{r(1,c_t)} & 0 & 0 \\ G_{r(2,1)} & \cdots & G_{r(2,c_t)} & 0 & 0 \end{pmatrix} \tag{61}$$

ここで，$G'_{(i,j)}$ は行・列チェックサム生成後の重みの値を表す．列方向からテストベクトルを印加してテストを行うとき，4 つのテストベクトルは式 (62) のように作成する．このとき，列チェックサムのコンダクタンス $\boldsymbol{G}_c$ は利用しないため，テストベクトルの各行の末尾 2 つの値は 0 に設定する．

$$\begin{aligned} \boldsymbol{V}_t &= \begin{pmatrix} V_{t(1,1)} & \cdots & V_{t(1,c_t)} & 0 & 0 \\ V_{t(2,1)} & \cdots & V_{t(2,c_t)} & 0 & 0 \\ V_{t(3,1)} & \cdots & V_{t(3,c_t)} & 0 & 0 \\ V_{t(4,1)} & \cdots & V_{t(4,c_t)} & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 2^{0\times0} & 2^{0\times1} & \cdots & 2^{0\times(c_t-1)} & 0 & 0 \\ 2^{1\times0} & 2^{1\times1} & \cdots & 2^{1\times(c_t-1)} & 0 & 0 \\ 2^{2\times0} & 2^{2\times1} & \cdots & 2^{2\times(c_t-1)} & 0 & 0 \\ 2^{3\times0} & 2^{3\times1} & \cdots & 2^{3\times(c_t-1)} & 0 & 0 \end{pmatrix} \end{aligned} \tag{62}$$

式 (62) に示す 4 つのテストベクトルを式 (61) に示すテストブロックに印加することで，式 (63)，(64)，(65)，(66) に示すような出力を得ることができる．このとき，テストブロックに対して列方向にテストベクトルを印加するため，テストブ

ロックのコンダクタンス $\boldsymbol{G}$ は転値した形で表される．

$$\boldsymbol{V}_t \cdot \boldsymbol{G}^T = \begin{pmatrix} O_{(1,1)} & \cdots & O_{(1,r_t)} & O_{r(1,1)} & O_{r(1,2)} \\ O_{(2,1)} & \cdots & O_{(2,r_t)} & O_{r(2,1)} & O_{r(2,2)} \\ O_{(3,1)} & \cdots & O_{(3,r_t)} & O_{r(3,1)} & O_{r(3,2)} \\ O_{(4,1)} & \cdots & O_{(4,r_t)} & O_{r(4,1)} & O_{r(4,2)} \end{pmatrix} \tag{63}$$

$$O_{(k,i)} = \sum_{j=1}^{c_t} V_{t(k,j)} \cdot G'^T_{(j,i)} = \sum_{j=1}^{c_t} V_{t(k,j)} \cdot G'_{(i,j)} = \sum_{j=1}^{c_t} 2^{(k-1)(j-1)} \cdot G'_{(i,j)} \tag{64}$$

$$O_{r(k,1)} = \sum_{j=1}^{c_t} V_{t(k,j)} \cdot G_{r(1,j)} = \sum_{j=1}^{c_t} 2^{(k-1)(j-1)} \cdot G_{r(1,j)} \tag{65}$$

$$O_{r(k,2)} = \sum_{j=1}^{c_t} V_{t(k,j)} \cdot G_{r(2,j)} = \sum_{j=1}^{c_t} 2^{(k-1)(j-1)} \cdot G_{r(2,j)} \tag{66}$$

これらの出力ベクトルを CPU 側へ転送し，式 (67)，(68) に従って 8 つのシグネチャ $C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ を生成する．

$$\begin{aligned} C_{(k)} &= \sum_{i=1}^{r_t} O_{(k,i)} - O_{r(k,1)} = \sum_{j=1}^{c_t} \sum_{i=1}^{r_t} V_{t(k,j)} G'_{(i,j)} - \sum_{j=1}^{c_t} V_{t(k,j)} G_{r(1,j)} \\ &= \sum_{j=1}^{c_t} \sum_{i=1}^{r_t} 2^{(k-1)(j-1)} G'_{(i,j)} - \sum_{j=1}^{c_t} 2^{(k-1)(j-1)} G_{r(1,j)} \end{aligned} \tag{67}$$

$$\begin{aligned} D_{(k)} &= \sum_{i=1}^{r_t} i \cdot O_{t(k,i)} - O_{r(k,2)} = \sum_{j=1}^{c_t} \sum_{i=1}^{r_t} i \cdot V_{t(k,j)} G'_{(i,j)} - \sum_{j=1}^{c_t} V_{t(k,j)} G_{r(1,j)} \\ &= \sum_{j=1}^{c_t} \sum_{i=1}^{r_t} i \cdot 2^{(k-1)(j-1)} G'_{(i,j)} - \sum_{j=1}^{c_t} 2^{(k-1)(j-1)} G_{r(2,j)} \end{aligned} \tag{68}$$

シグネチャの値を利用することで，同一行で発生している故障箇所の特定と故障メモリスタセルの期待値との差の算出を行う．行方向テストベクトルによるテスト終了後のテストブロックの状態としては，(i) 同一行の 2 つの計算用メモリスタセルに故障が発生，(ii) 同一行の 1 つの計算用メモリスタセルと 1 つの重み無し列チェックサムに故障が発生，(iii) 同一行の 1 つの計算用メモリスタセルと 1 つの重み有り列チェックサムに故障が発生，(iv) 同一行の 1 つの重み無し列チェック

サムと 1 つの重み有り列チェックサムに故障が発生，(v) 計算用メモリスタセルと列チェックサムに故障無しの 5 パターンが考えられる．次の方法でシグネチャの解析を行うことで，テストブロックがどの故障パターンであるかを判別し，同一行で発生した故障箇所と期待値との差を求めることができる．

(i) 同一行の 2 つの計算用メモリスタセルに故障が発生

テストブロック中の x_1 行 y_1 列目 $(x_1 = 1, \cdots, r_t, y_1 = 1, \cdots, c_t)$ のメモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ に $+d_1$ だけ元の値から変化する故障と，x_2 行 y_2 列目 $(x_2 = 1, \cdots, r_t, y_2 = 1, \cdots, c_t)$ のメモリスタセルのコンダクタンス $G'_{(x_2,y_2)}$ に $+d_2$ だけ元の値から変化する故障が発生した場合を考える．式 (67)，(68) より，8 つのシグネチャの値 $C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ は式 (69) のようになる．

$$\begin{cases} C_{(1)} = d_1 + d_2 & D_{(1)} = x_1 \cdot d_1 + x_2 \cdot d_2 \\ C_{(2)} = 2^{(y_1-1)} d_1 + 2^{(y_2-1)} d_2 & D_{(2)} = 2^{(y_1-1)} x_1 \cdot d_1 + 2^{(y_2-1)} x_2 \cdot d_2 \\ C_{(3)} = 2^{2(y_1-1)} d_1 + 2^{2(y_2-1)} d_2 & D_{(3)} = 2^{2(y_1-1)} x_1 \cdot d_1 + 2^{2(y_2-1)} x_2 \cdot d_2 \\ C_{(4)} = 2^{3(y_1-1)} d_1 + 2^{3(y_2-1)} d_2 & D_{(4)} = 2^{3(y_1-1)} x_1 \cdot d_1 + 2^{3(y_2-1)} x_2 \cdot d_2 \end{cases} \tag{69}$$

この故障パターンは，$\frac{C_{(2)}}{C_{(1)}} \neq \frac{C_{(3)}}{C_{(2)}} \neq \frac{C_{(4)}}{C_{(3)}}$ かつ $\frac{D_{(2)}}{D_{(1)}} \neq \frac{D_{(3)}}{D_{(2)}} \neq \frac{D_{(4)}}{D_{(3)}}$ であるかどうかを確かめることで特定できる．この 8 つのシグネチャの値は，式 (40) の x_1 と y_1，x_2 と y_2 の関係を入れ替えたものになっていることが分かる．したがって，$\eta = \frac{C_{(1)}C_{(4)} - C_{(2)}C_{(3)}}{C_{(1)}C_{(3)} - C_{(2)}^2} = 2^{y_1-1} + 2^{y_2-1}$ とすると，各故障パラメータは，式 (70) を上か

ら順に解いていくことで算出することができる．

$$
\begin{cases}
y_1 = \lfloor \log_2(\eta) \rfloor + 1 \\
y_2 = \log_2(\eta - 2^{y_1-1}) + 1 \\
d_1 = \dfrac{-C_{(2)} + 2^{y_2-1} C_{(1)}}{2^{y_2-1} - 2^{y_1-1}} \\
d_2 = -C_{(1)} - d_1 \\
x_1 = \dfrac{-D_{(2)} + 2^{y_2-1} D_{(1)}}{d_1(2^{y_1-1} - 2^{y_2-1})} \\
x_2 = \dfrac{-D_{(1)} - x_1 d_1}{d_2}
\end{cases}
\tag{70}
$$

(ii) 同一行の 1 つの計算用メモリスタセルと 1 つの重み無し列チェックサムに故障が発生

テストブロック中の x_1 行 y_1 列目 $(x_1 = 1, \cdots, r_t, y_1 = 1, \cdots, c_t)$ のメモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ に $+d_1$ だけ元の値から変化する故障と，x_2 行目 $(x_2 = 1, \cdots, r_t)$ の重み無し列チェックサムのコンダクタンス $G_{c(x_1,1)}$ に $+d_2$ だけ元の値から変化する故障が発生した場合を考える．式 (62) に示すように，列方向テストベクトルは列チェックサムに対して印加されないため，シグネチャ$C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ は，列チェックサムの故障による影響を受けない．したがって，各シグネチャの値は式 (71) のようになる．

$$
\begin{cases}
C_{(1)} = d_1 & D_{(1)} = x_1 \cdot d_1 \\
C_{(2)} = 2^{(y_1-1)} d_1 & D_{(2)} = 2^{(y_1-1)} x_1 \cdot d_1 \\
C_{(3)} = 2^{2(y_1-1)} d_1 & D_{(3)} = 2^{2(y_1-1)} x_1 \cdot d_1 \\
C_{(4)} = 2^{3(y_1-1)} d_1 & D_{(4)} = 2^{3(y_1-1)} x_1 \cdot d_1
\end{cases}
\tag{71}
$$

この故障パターンは，$\frac{C_{(2)}}{C_{(1)}} = \frac{C_{(3)}}{C_{(2)}} = \frac{C_{(4)}}{C_{(3)}}$ かつ $\frac{D_{(2)}}{D_{(1)}} = \frac{D_{(3)}}{D_{(2)}} = \frac{D_{(4)}}{D_{(3)}}$ かつ $D_{(1)} \mod C_{(1)} = 0$ であるかどうかを確かめることで特定できる．計算用メモリスタセルの故障のパラメータ y_1, d_1 はそれぞれ，$y_1 = \log_2(\frac{C_{(2)}}{C_{(1)}}) + 1$，$d_1 = A_{(1)}$ となる．y_1，d_1 を取得した後は，事前に保持しておいた $A_{(1)}$ のシグネチャの値を利用して，重み無し列

チェックサムの期待値との差 d_2 を計算する．同一行の 1 つの計算用メモリスタセルと重み無し列チェックサムに故障が発生している場合，式 (43) より，$d_2 = A_{(1)} - d_1$ となる．

(iii) 同一行の 1 つの計算用メモリスタセルと 1 つの重み有り列チェックサムに故障が発生

テストブロック中の x_1 行 y_1 列目 ($x = 1, \cdots, r_t, y_1 = 1, \cdots, c_t$) のメモリスタセルのコンダクタンス $G'_{(x_1,y_1)}$ に $+d_1$ だけ元の値から変化する故障と，x_2 行目 ($x_2 = 1, \cdots, r_t$) の重み無し列チェックサムのコンダクタンス $G_{r(x,y_2)}$ に $+d_2$ だけ元の値から変化する故障が発生した場合を考える．式 (62) に示すように，列方向テストベクトルは列チェックサムに対して印加されないため，シグネチャ$C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ は列チェックサムの故障による影響を受けず，各シグネチャの値は式 (72) のようになる．

$$\begin{cases} C_{(1)} = d_1 & D_{(1)} = x_1 \cdot d_1 \\ C_{(2)} = 2^{(y_1-1)} d_1 & D_{(2)} = 2^{(y_1-1)} x_1 \cdot d_1 \\ C_{(3)} = 2^{2(y_1-1)} d_1 & D_{(3)} = 2^{2(y_1-1)} x_1 \cdot d_1 \\ C_{(4)} = 2^{3(y_1-1)} d_1 & D_{(4)} = 2^{3(y_1-1)} x_1 \cdot d_1 \end{cases} \tag{72}$$

この故障パターンは，$\frac{C_{(2)}}{C_{(1)}} = \frac{C_{(3)}}{C_{(2)}} = \frac{C_{(4)}}{C_{(3)}}$ かつ $\frac{D_{(2)}}{D_{(1)}} = \frac{D_{(3)}}{D_{(2)}} = \frac{D_{(4)}}{D_{(3)}}$ かつ $D_{(1)} mod C_{(1)} = 0$ であるかどうかを確かめることで特定できる．計算用メモリスタセルの故障のパラメータ y_1, d_1 はそれぞれ，$y_1 = \log_2(\frac{C_{(2)}}{C_{(1)}}) + 1$，$d_1 = A_{(1)}$ となる．y_1，d_1 を取得した後は，事前に保持しておいた $B_{(1)}$ のシグネチャの値を利用して，重み有り列チェックサムの期待値との差 d_2 を計算する．同一行の 1 つの計算用メモリスタセルと重み有り列チェックサムに故障が発生している場合，式 (44) より，$d_2 = B_{(1)} - y_1 * d_1$ となる．

(iv) 同一行の 1 つの重み無し列チェックサムと 1 つの重み有り列チェックサムに故障が発生

テストブロック中の x_1 行目 $(x = 1, \cdots, r_t)$ の重み無し列チェックサムのコンダクタンス $G'_{c(x_1,y_1)}$ に $+d_1$ だけ元の値から変化する故障と，x_2 行目 $(x_2 = 1, \cdots, r_t)$ の重み有り列チェックサムのコンダクタンス $G_{c(x,y_2)}$ に $+d_2$ だけ元の値から変化する故障が発生した場合を考える．この場合，シグネチャ $C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ は式 (73) のようになる．

$$\begin{cases} C_{(1)} = 0 \quad D_{(1)} = 0 \\ C_{(2)} = 0 \quad D_{(2)} = 0 \\ C_{(3)} = 0 \quad D_{(3)} = 0 \\ C_{(4)} = 0 \quad D_{(4)} = 0 \end{cases} \tag{73}$$

式 (73) より，計算用メモリスタセルには故障がないことが分かるので，2 つの同一行の故障は，重み無し列チェックサムと重み有り列チェックサムで発生していることが分かる．また，式 (44) より，d_1, d_2 はそれぞれ $d_1 = A_{(1)}$，$d_2 = B_{(1)}$ となる．

(v) 計算用メモリスタセルと列チェックサムに故障無しの場合

計算用メモリスタセルと列チェックサムに故障が発生していない場合は，行チェックサムに対して誤りが無いかどうかのテストを行う．列方向テストベクトル印加によるテストでは，転置したクロスバーアレイに対してテストを行っていることと同じである．そのため，行チェックサムに 2 つ以下の故障が発生した場合，シグネチャ $C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ を解析すると，4.2.2 節に示した X-ABFT の列チェックサムに対するテストの (ii), (iii), (viii), (ix) において，x_1 と y_1，x_2 と y_2 の関係を入れ替えた形の解析結果が得られる．例として，テストブロック中の y_1 列目 $(y_1 = 1, \cdots, c_t)$ の 1 つの重み無し行チェックサム $G_{r(1,y_1)}$ に，元の重みから $+d_1$ だけ値が変化する故障が発生した場合を考えると，8 つのシグ

ネチャの値 $C_{(1)}$，$C_{(2)}$，$C_{(3)}$，$C_{(4)}$，$D_{(1)}$，$D_{(2)}$，$D_{(3)}$，$D_{(4)}$ は式 (74) のようになる．

$$\begin{cases} C_{(1)} = d_1 & D_{(1)} = 0 \\ C_{(2)} = 2^{(y_1-1)}d_1 & D_{(2)} = 0 \\ C_{(3)} = 2^{2(y_1-1)}d_1 & D_{(3)} = 0 \\ C_{(4)} = 2^{3(y_1-1)}d_1 & D_{(4)} = 0 \end{cases} \tag{74}$$

この解析結果は，式 (38) の解析結果の x_1 の値を y_1 に変更したものと同じになる．他の行チェックサムの故障パターンにおいても x と y の値を入れ替えた解析結果が得られるため，列チェックサムと同様に，行チェックサムで 2 つ以下の故障が発生した場合，それらの故障の位置と故障による期待値との差を得ることができる．

4.4 シミュレーションによる評価実験

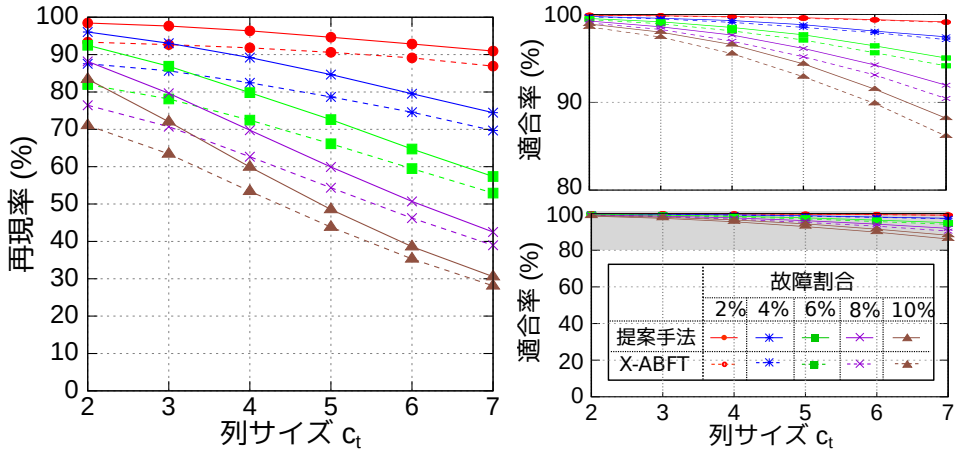
本評価実験では，提案手法と X-ABFT のテスト性能，ホップフィールドネットワークと 3 層ニューラルネットワークの識別率，面積・時間オーバーヘッドの評価を行う．

4.4.1 提案手法のテスト性能評価

まず，提案手法および X-ABFT を適用したメモリスタクロスバーアレイに対して，一定割合の故障を注入し，オンラインテストを行った際の故障箇所特定の再現率と適合率を比較した．再現率と適合率は，式 (75)，(76) に示すように，それぞれ，全故障数に対する位置を正しく特定できた故障数の割合，位置特定した故障数に対する位置を正しく特定できた故障数の割合とした．

$$\text{再現率} = \frac{\text{位置を正しく特定できた故障数}}{\text{全故障数}} \tag{75}$$

$$\text{適合率} = \frac{\text{位置を正しく特定できた故障数}}{\text{位置特定した故障数}} \tag{76}$$



(a) 再現率　　(b) 適合率

図 24　テスト性能の評価

クロスバーアレイのサイズは 512×512 とし [36]，故障はメモリスタクロスバーアレイ上に一様分布で生じると仮定した．故障割合はクロスバーアレイ中の全メモリスタのうち 2% から 10% まで，2% ずつ変化させている [11]．1 つのテストブロックのサイズは $(r_t + 2) \times (c_t + 2)$ であり，実験では，計算用メモリスタセルの行数 $r_t = 2$ に固定し，列数 c_t を 2 から 7 まで変化させて再現率と適合率を評価した．

図 24 にテスト性能を評価した結果を示す．図 24(a), 24(b) に示すように，提案手法の適合率と再現率の値は，テストブロックの列サイズ c_t の大きさに関わらず，X-ABFT よりも常に大きい値となり，提案手法は高い精度で多くの故障箇所を特定できることが分かった．特に，図 24(a) に示すように，故障割合が 10% の場合でも，テストブロックの行サイズ c_t を 2 に設定することで提案手法の再現率は 83.4%となり，X-ABFT の再現率と比較して約 12.4%向上した．

4.4.2 識別性能の評価

続いて，ホップフィールドネットワーク [37, 38] と 3 層ニューラルネットワークを用いて，提案手法と X-ABFT の耐故障性能を比較した．

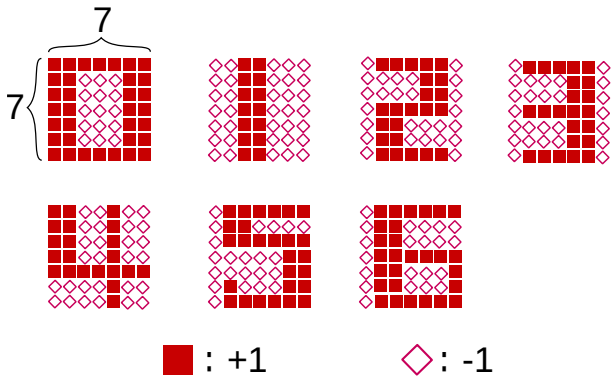


図 25 ホップフィールドネットワークの評価実験で利用する学習データ

ホップフィールドネットワーク

本実験では，図 25 に示す 7 × 7 のサイズの 0 から 6 までの 7 つの 2 値数字画像（訓練データ）[38] をホップフィールドネットワークに学習させ，ノイズの加えた 0〜6 までの画像（テストデータ）を入力した際の提案手法，X-ABFT，テストを行わない場合（baseline），故障を印加しなかった場合（故障無し）の 4 つの識別率を比較する．実験では，49 × 49 のクロスバーアレイを利用し，提案手法，X-ABFT，baseline は全メモリスタセル中の 10% のメモリスタセルが故障しているものとする．学習は，ヘブ則 [39] に則って行われ，記憶させるデータを基にメモリスタへの書き込みが 1 回だけ行われる．提案手法と X-ABFT では重み書き込み後に 1 度テストを行い，故障箇所の特定・修復を行う．提案手法と X-ABFT のテストブロックのサイズは $(r_t + 2) \times (c_t + 2)$ とし，$r_t = c_t = 2$ とした．テストデータには，訓練データに $\sigma = 0.2$ のガウシアンノイズを加えた画像を各 5,000 枚ずつ，合計 35,000 枚を用いた．識別率は，ピクセルの誤りが 1 つ以下のものは正しく識別できたものとし，それ以外は識別に失敗したものとして評価した．

図 26 にホップフィールドネットワークの識別率を評価した結果を示す．故障無しの場合，識別率が 92.61%となったが，10%の故障が注入されると，baseline に示すように，識別率が 49.76%まで下がった．しかし，提案手法によるテストを行った場合，識別率は 75.57%となり，baseline よりも 25.81%識別率が向上した．また，X-ABFT によるテストを行った場合と比較して識別率は 5.25%向上した．

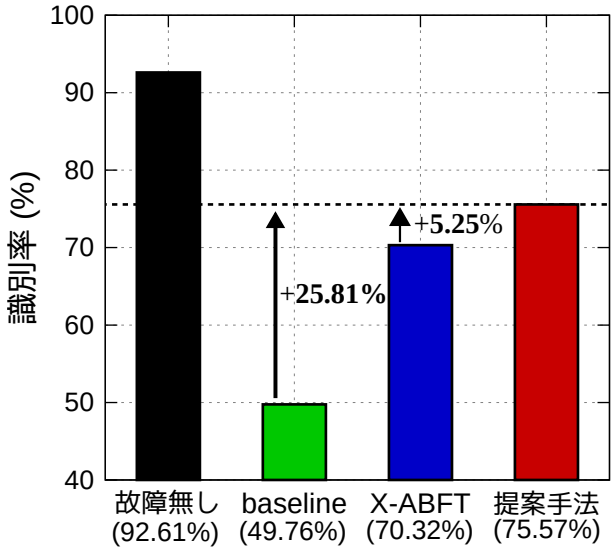


図 26 ホップフィールドネットワークの識別率評価

3 層ニューラルネットワーク

メモリスタニューラルネットワークでは，1 度の書き込み毎に書き込みばらつきによる過渡故障が発生し寿命を超えると書き込み寿命による永久故障が発生する．本実験では，全メモリスタセルのうち 2%のセルに書き込みばらつきが発生するものとした．メモリスタの書き込み寿命は，正規分布 $\mathcal{N}(2.5\times10^5, 6.4\times10^9)$ の正規分布に従うものとした [15]．3 層ニューラルネットワークの構造は，入力層 (784)-中間層 (100)-出力層 (10) とした．学習データには，60,000 個の訓練データと 10,000 個のテストデータからなる手書き数字画像 MNIST [40] を用いた．学習では，バッチサイズを 300 としたミニバッチ学習を行い，ミニバッチ学習により 200 回重みを更新したとき，1epoch としている．勾配の計算には確率的勾配降下法を利用し，学習率 $\eta = 0.1$ とした．提案手法と X-ABFT の 1 つのテストブロックのサイズは，$(r_t+2)\times(c_t+2)$ とし，$r_t = c_t = 2$ とした．また，テストを行うときの書き込み回数 $\tau = 5,000$ とし，25epoch 毎にオンラインテストを行うものとした．活性化関数は，入力層と中間層では ReLU 関数を利用し，出力層では SoftMax 関数を用いた．

図 27 に 3 層ニューラルネットワークの識別率を評価した結果を示す．識別率は epoch の増加とともに上昇していったが，提案手法と X-ABFT は過渡故障と永

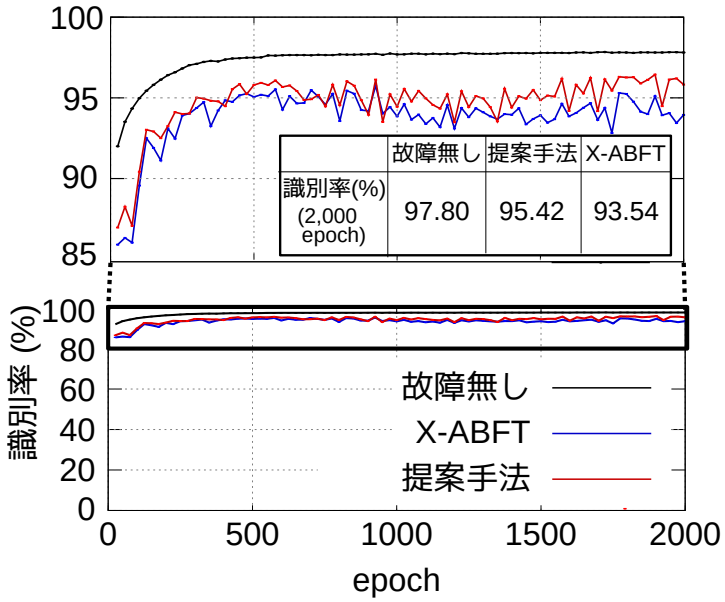


図 27 3 層ニューラルネットワークの識別率評価

久故障の影響を受け，故障がない場合と比較して，識別率は小さくなった．しかし，逆方向伝播による誤り訂正と改善したオンラインテストによって，提案手法は X-ABFT よりも識別率が向上し，2,000epoch での識別率は提案手法が 95.42%，X-ABFT が 93.54%となり，X-ABFT と比較して 1.88%識別率が向上した．

4.4.3 面積と時間オーバーヘッドの評価

最後に，提案手法の面積・時間オーバーヘッドを評価する．面積オーバーヘッドは，提案手法，および，X-ABFT によって追加される余分なチェックサムの割合によって評価を行い，時間オーバーヘッドは，提案手法，および，X-ABFT によって追加される冗長なテストベクトルの割合によって評価を行う．ニューラルネットワークの層数を L，l 層目のクロスバーアレイの行サイズを M_l，列サイズを N_l とし，1 つのテストブロックの行サイズを r_t，列サイズを c_t とし，τ をテストを行うまでの学習回数とする．X-ABFT の面積オーバーヘッド A_{XABFT} と時間

オーバーヘッド T_{XABFT} は，それぞれ次式のように表すことができる．

$$A_{\mathrm{XABFT}} = 1 + \frac{\sum_{l=1}^{L-1} 2\lceil N_l/c_t \rceil \cdot M_l}{\sum_{l=1}^{L-1} N_l \cdot M_l} \tag{77}$$

$$T_{\mathrm{XABFT}} = 1 + \frac{\sum_{l=1}^{L-1} 4\lceil M_l/r_t \rceil}{\tau} \tag{78}$$

また，提案手法の面積オーバーヘッド A_{PROP} と時間オーバーヘッド T_{PROP} は，それぞれ，

$$A_{\mathrm{PROP}} = 1 + \frac{\sum_{l=1}^{L-1} 2\lceil N_l/c_t \rceil \cdot M_l}{\sum_{l=1}^{L-1} N_l \cdot M_l} + \frac{\sum_{l=1}^{L-1} 2\lceil M_l/r_t \rceil \times (N_l + \lceil N_l/c_t \rceil)}{\sum_{l=1}^{L-1} N_l \cdot M_l} \tag{79}$$

$$T_{\mathrm{PROP}} = 1 + \frac{\sum_{l=1}^{L-1} 4\lceil M_l/r_t \rceil}{\tau} + \frac{\sum_{l=1}^{L-1} 4\lceil N_l/c_t \rceil}{\tau} \tag{80}$$

と書ける．

式 (77)，(79) から，提案手法では，X-ABFT と比べて，余分なメモリスタセル数が $\frac{\sum_{l=1}^{L-1} 2\lceil M_l/r_t \rceil \times (N_l + \lceil N_l/c_t \rceil)}{\sum_{l=1}^{L-1} N_l \times M_l}$ だけ増えている．例えば，4.4.2 節の 3 層ニューラルネットワークでの実験条件では，式 (77)，(79) より，X-ABFT のメモリスタセル数はチェックサム無しと比較して 2.00 倍，提案手法は 3.50 倍となり，X-ABFT と比較して，提案手法は各クロスバーアレイ毎にメモリスタセル数が 1.75 倍増加した．

一方，式 (78)，(80) に示すように，提案手法では，X-ABFT と比較して，余分なテストベクトル数が $\sum_{l=1}^{L-1} \frac{4\lceil N_l/c_t \rceil}{\tau}$ だけ増加した．面積オーバーヘッドと同様に，4.4.2 節の 3 層ニューラルネットワークの実験条件を考えると，式 (78)，(80) より，X-ABFT で追加される余分な入力ベクトル数は元のメモリスタニューラルネットワークの 1.35 倍，提案手法は 1.40 倍となり，提案手法の時間オーバーヘッドは既存手法と比較して僅かな増加となった．

4.5 まとめ

本節では，メモリスタニューラルネットワークの高信頼な学習動作を実現する耐故障設計を提案した．既存の耐故障設計である X-ABFT では，メモリスタクロ

スバーアレイの列方向にチェックサムを付加することで，順方向伝播の誤り訂正とオンラインテストにより高信頼化を図っていたが，逆方向伝播の誤り訂正ができない，テスト時に同一行で発生した 2 つの故障位置を正しく特定・修復することができない課題があった．そこで，提案手法では，X-ABFT 手法に対し，行方向にチェックサムを付加し，さらに列方向からテストベクトルを印加するテストを追加することで，逆方向伝播の誤り訂正と同一行の 2 つの故障位置を特定・修復するオンラインテストを可能にした．3 層ニューラルネットワークを用いた数値実験では，書き込みばらつきによる過渡故障や書き込み寿命による永久故障が存在する場合を想定し，提案手法を適用することで識別率は 95.42%となり X-ABFT と比較して 1.88%識別率が向上した．

5. 結論

メモリスタによって実現される 2 つのシステムである ReRAM とメモリスタニューラルネットワークは，フォンノイマンボトルネックを解消し高い性能を実現できる新しい計算機システムとして注目されている．一方で，システムの構成要素であるメモリスタは信頼性に課題を有し, 特性ばらつきや製造時の欠陥, 短い書き込み寿命などの要因によって故障が生じるため，システムの信頼性を担保する耐故障設計が不可欠である．そこで，本論文では，ReRAM とメモリスタニューラルネットワークの信頼性向上を目的として，それぞれのシステムに対して高い信頼性を実現する耐故障設計手法を提案した．

第 3 章では，CMOS とメモリスタを利用したハイブリッド ECC 回路を提案した. ReRAM では，特に長期信頼性が重要な課題となっており，長期信頼性を担保するための手法として CMOS による ECC 回路が提案されているが，ECC 回路を CMOS で実現するため，メモリスタの高い集積性能を活かせない．メモリスタの論理回路マッピング手法を用いて ECC 回路を構成することも可能だが，その場合，メモリスタに対する書き込みが非常に多くなるため，ECC 回路自体の信頼性が低下する．そこで，本章ではメモリスタで実現すると書き込みが多く発生する論理演算を行う回路ブロックについては CMOS 回路で実現し, その他の回路ブロックをメモリスタクロスバーアレイによる積和演算回路によって実現する，CMOS とメモリスタのハイブリッド ECC 回路を提案した．評価実験において，データビット長 k が 1,024bit のとき，提案 ECC 回路は CMOS による ECC 回路と比較して 67.9% 面積を削減できることを示した．

第 4 章では，メモリスタニューラルネットワークに対して，新しい耐故障設計手法を提案した．メモリスタニューラルネットワークに対する耐故障設計として，順方向伝播時の誤り訂正機能と，オンラインテストを実現する X-ABFT という手法が提案されている．しかし，X-ABFT は誤差逆伝播の誤り訂正を考慮しておらず，また，オンラインテスト時に特定の故障パターンの訂正ができないという問題点があった. 本章では，X-ABFT で実現されていた順方向伝播の誤り訂正を実現しながら誤差逆伝播の誤り訂正を可能とし，また，X-ABFT で位置特定・訂正することのできない故障を訂正するオンラインテストを可能とする新しい耐故障設

計を提案した．評価実験では，書き込みばらつきによる過渡故障や書き込み寿命による永久故障が存在する状況においても，提案手法を適用することで 95.42%の識別率を達成し，X-ABFT と比較して 1.88%識別率が向上した．

以上のように，本論文では，ReRAM，メモリスタニューラルネットワークそれぞれに対して高い信頼性を実現する耐故障設計手法を実現することができた．今後の課題として，提案する誤り訂正符号回路に対しては，消費電力の改善が挙げられる．また，提案するメモリスタニューラルネットワークの耐故障設計に対しては，面積オーバーヘッドを考慮したテストブロック分割手法の実現が挙げられる．

謝辞

本研究に関して的確なご指導，ご鞭撻を頂きました主指導教員の井上美智子教授に心より感謝いたします．入学前から研究指導を行っていただき，研究に関する様々な助言をいただきました．充実した研究生活を送ることができたのは井上先生のおかげだと思っております．心からお礼申し上げます．本論文をまとめるにあたって貴重な助言を賜りました，太田淳教授，中島康彦教授に心より感謝申し上げます．本研究を進めるにあたり，適切な助言を賜りました大下福仁准教授に心より感謝申し上げます．また，丁寧かつ熱心な指導を絶えず行ってくださいました，新谷道広助教に深く感謝申し上げます．研究の"け"の字もわからず彷徨っていた僕に，研究者としての道標を指し示して頂きました．熱心かつ厳しい指導に何度か心が挫けましたが，その度に温かい叱咤激励をいただき，僕を奮い立たせてくださいました．また，社会人としてのいろはについてもご指導いただき，社会人の基礎について学ばせていただきました．先生からご指導頂いたことは，僕にとってかけがえのない貴重な経験になりました．深く感謝申し上げます．

研究生活を送る上で多大なご支援を頂いた，同期の安見嘉人君，三野智貴君，長濵将太君，後輩の上田葵君，廣瀬慈恩君に深く感謝致します．特に安見君は，奈良高専の頃から約 5 年間同じ研究室で過ごし，今まで様々なことについて相談にのっていただきました．本当にありがとうございます．様々な研究室の行事においてお世話になりました土田将司さん，Riaz-UI-Haque Mian さん，Foisal Ahmed さんに感謝致します．また，奈良高専時代から NAIST に至るまで生活面でお世話になりました，三上徹郎君，中村優太君に深く感謝致します．研究生活が充実したのは，ひとえに皆様のご支援のおかげです．深く感謝致します．

また，画面の中からいつも励ましてくれた，アイドルマスターシンデレラガールズの皆さんに深く感謝致します．特に，いつも画面から声をかけて励ましてくれた三船美優さん，白雪千夜さん，黒埼ちとせさんと，各アイドルに命を吹き込んでくださる声優の原田彩楓様，関口理咲様，佐倉薫様に深く感謝致します．

最後に，いつも支えてくれた家族に深く感謝致します．

参考文献

[1] Mengyun Liu, Lixue Xia, Yu Wang, and Krishnendu Chakrabarty. Fault tolerance in neuromorphic computing systems. In *Proceedings of IEEE/ACM Asia and South Pacific Design Automation Conference*, pages 216–223, 2019.

[2] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging NVM: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems*, 23(2):1–32, 2017.

[3] Sparsh Mittal. A survey of reram-based architectures for processing-in-memory and neural networks. *Machine learning and knowledge extraction*, 1(1):75–114, 2019.

[4] Kosuke Suzuki and Steven Swanson. A survey of trends in non-volatile memory technologies: 2000-2014. In *Proceedings of IEEE International Memory Workshop*, pages 1–4, 2015.

[5] Leon Chua. Memristor—the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, 1971.

[6] Leon Chua. Resistance switching memories are memristors. *Applied Physics A*, 102:765–783, 2011.

[7] Richard F Freitas and Winfried W Wilcke. Storage-class memory: The next storage system technology. *IBM Journal of Research and Development*, 52(4/5):439, 2008.

[8] Satoshi Yamamori, Masayuki Hiromoto, and Takashi Sato. Efficient mini-batch training on memristor neural network integrating gradient calculation and weight update. *IEICE Transaction on Fundamentals*, 101(7):1092–1100, 2018.

[9] Sparsh Mittal. A survey of reram-based architectures for processing-in-memory and neural networks. *Machine learning and knowledge extraction*, 1(1):75–114, 2019.

[10] Dimin Niu, Yang Xiao, and Yuan Xie. Low power memristor-based ReRAM design with error correcting code. In *Proceedings of IEEE/ACM Asia and South Pacific Design Automation Conference*, pages 79–84, 2012.

[11] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu, and Frederick T. Chen. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Transactions on Computers*, 64(3):180–190, 2015.

[12] Lixue Xia, Mengyun Liu, Xuefei Ning, Krishnendu Chakrabarty, and Yu Wang. Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. In *Proceedings of IEEE/ACM Design Automation Conference*, pages 33:1–33:6, 2017.

[13] Lixue Xia, Wenqin Huangfu, Tianqi Tang, Xiling Yin, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, and Huazhong Yang. Stuck-at fault tolerance in RRAM computing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(1):102–115, 2018.

[14] Yi Cai, Yujun Lin, Lixue Xia, Xiaoming Chen, Song Han, Yu Wang, and Huazhong Yang. Long live TIME: improving lifetime for training-in-memory engines by structured gradient sparsification. In *Proceedings of IEEE/ACM Design Automation Conference*, pages 1–6, 2018.

[15] Peyman Pouyan, Esteve Amat, and Antonio Rubio. Memristive crossbar memory lifetime evaluation and reconfiguration strategies. *IEEE Transactions on Emerging Topics in Computing*, 6(2):207–218, 2018.

[16] Mengyun Liu, Lixue Xia, Yu Wang, and Krishnendu Chakrabarty. Fault tolerance for rram-based matrix operations. In *Proceedings of IEEE International Test Conference*, pages 1–10, 2019.

[17] U. K. Kumar and B. S. Umashankar. Improved hamming code for error detection

and correction. In *Proceedings of IEEE International Symposium on Wireless Pervasive Computing*, pages 498–500, 2007.

[18] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. Memristor-based material implication (IMPLY) logic: Design principles and methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2054–2066, 2014.

[19] Anika Raghuvanshi and Marek Perkowski. Logic synthesis and a generalized notation for memristor-realized material implication gates. *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 470–477, 2014.

[20] John Reuben, Rotem Ben-Hur, Nimrod Wald, Nishil Talati, Ameer Haj Ali, Pierre-Emmanuel Gaillardon, and Shahar Kvatinsky. Memristive logic: A framework for evaluation and comparison. In *Proceedings of International Symposium on Power And Timing Modeling, Optimization and Simulation*, pages 1–8, 2017.

[21] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. The missing memristor found. *Nature*, 453:80–83, 2008.

[22] J. Joshua Yang, Dmitri B. Strukov, and Duncan R. Stewart. Memristive devices for computing. *Nature Nanotechnology*, 8:13–24, 2013.

[23] Isha Gupta, Alexantrou Serb, Ali Khiat, Ralf Zeitler, Stefano Vassanelli, and Themistoklis Prodromakis. Real-time encoding and compression of neuronal spikes by metal-oxide memristors. *Nature Communications*, 7(12805):1–9, 2016.

[24] Alexander Serb, Christos Papavassiliou, and Themistoklis Prodromakis. A memristor-CMOS hybrid architecture concept for on-line template matching. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 1–4, 2017.

[25] Synopsys, Inc. *Design Compiler User Guide Version I-2013.06*, 2013.

[26] R. Goldman, K. Bartleson, T. Wood, K. Kranen, C. Cao, V. Melikyan, and G. Markosyan. Synopsys' open educational design kit: Capabilities, deployment and future. In *Proceedings of IEEE International Conference on Microelectronic Systems Education*, pages 20–24, 2009.

[27] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. NVSim: a circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[28] Shyh-Shyuan Sheu, Meng-Fan Chang, Ku-Feng Lin, Che-Wei Wu, Yu-Sheng Chen, Pi-Feng Chiu, Chia-Chen Kuo, Yih-Shan Yang, Pei-Chia Chiang, Wen-Pin Lin, et al. A 4mb embedded slc resistive-ram macro with 7.2 ns read-write random-access time and 160ns mlc-access capability. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 200–0202, 2011.

[29] Masood Qazi, Kevin Stawiasz, Leland Chang, and Anantha P Chandrakasan. A 512kb 8T SRAM macro operating down to 0.57V with an AC-coupled sense amplifier and embedded data-retention-voltage sensor in 45nm SOI CMOS. *IEEE Journal of Solid-State Circuits*, 46(1):85–96, 2011.

[30] Payali Das, Suraj Kumar Saw, and Preetisudha Meher. Design of differential amplifier using current mirror load in 90 nm cmos technology. In *Information Systems Design and Intelligent Applications*, pages 421–429. Springer, 2019.

[31] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S. Rose, and Richard W. Linderman. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE Transactions on Neural Networks and Learning Systems*, 25(10):1864–1878, 2014.

[32] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike,

and Jan van Lunteren. Memristor based computation-in-memory architecture for data-intensive applications. In *Proceedings of IEEE/ACM Design, Automation and Test in Europe Conference and Exhibition*, pages 1718–1725, 2015.

[33] Jintao Yu, Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, and Said Hamdioui. Memristive devices for computation-in-memory. In *Proceedings of IEEE/ACM Design, Automation and Test in Europe Conference and Exhibition*, pages 1646–1651, 2018.

[34] Wenqin Huangfu, Lixue Xia, Ming Cheng, Xiling Yin, Tianqi Tang, Boxun Li, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, and Huazhong Yang. Computation-oriented fault-tolerance schemes for rram computing systems. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 794–799, 2017.

[35] Cynthia J. Anfinson and Franklin T. Luk. A linear algebraic model of algorithm-based fault tolerance. *IEEE Transactions on Computers*, 37(12):1599–1604, 1988.

[36] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *Proceedings of IEEE/ACM Design Automation Conference*, page 19, 2016.

[37] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[38] Shukai Duan, Zhekang Dong, Xiaofang Hu, Lidan Wang, and Hai Li. Small-world hopfield neural networks with weight salience priority and memristor synapses for digit recognition. *Neural Computing and Applications*, 27(4):837–844, 2016.

[39] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[40] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits. Available: http://yann.lecun.com/exdb/mnist/.

発表リスト

[1] 石坂守, 新谷道広, 井上美智子:“メモリスタ論理による誤り訂正符号回路の設計と評価”, 電子情報通信学会技術研究報告 (ディペンダブルコンピューティング研究会), CPSY2017-146, DC2017-102, pp.257-262, 2018 年 3 月.

[2] 石坂守, 新谷道広, 井上美智子:“メモリスタニューラルネットワークにおける縮退故障による識別性能への影響”, 電子情報通信学会総合大会, D-10-4, P.58, 2018 年 3 月.

[3] Mamoru Ishizaka, Michihiro Shintani, and Michiko Inoue:“Area-Efficient Memristor-Crossbar-Based Error Correcting Code Circuit,”In *Proceedings of Workshop on Security, Reliability, Test, Privacy, Safety and Trust of Future Devices (SURREALIST)*, May 2018.

[4] Mamoru Ishizaka, Michihiro Shintani, and Michiko Inoue:“Area-efficient and Reliable Hybrid CMOS/Memristor ECC Circuit for ReRAM Storage,”In *Proceedings of IEEE Asian Test Symposium (ATS)*, pp.167-172, Oct. 2018. (DOI: 10.1109/ATS.2018.00040)

[5] 石坂守, 新谷道広, 井上美智子:“重み推定によるメモリスタニューラルネットワークの信頼性向上の試み”, 電子情報通信学会技術研究報告, VLD2018-50, DC2018-36, pp.83-88, 2018 年 12 月.

[6] 石坂守, 山口賢一, 岩田大志:“レースフリーでかつ O(n) な非同期式スキャンパステスト法”, 電子情報通信学会論文誌 A, Vol.J102-A, No.6, pp.172-181, 2019 年 6 月.

[7] 石坂守, 新谷道広, 井上美智子:“チェックサムとオンラインテストによるメモリスタニューラルネットワークの耐故障設計”電子情報通信学会技術研究報告 (VLSI 設計技術研究会), VLD2019-112, HWS2019-85, pp.107-112, 2020 年 3 月.