不揮発メモリを対象とした
書き込みビット数削減と誤り訂正を実現する
メモリ符号化に関する研究

Bit-Write-Reducing and Error-Correcting Code Generation Methods for Non-Volatile Memories

2019年2月

古城 辰朗

Tatsuro KOJO

不揮発メモリを対象とした
書き込みビット数削減と誤り訂正を実現する
メモリ符号化に関する研究

Bit-Write-Reducing and Error-Correcting Code Generation Methods for Non-Volatile Memories

2019年2月

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻 情報システム設計研究

古城　辰朗
Tatsuro KOJO

目 次

第1章　序論

1.1　本論文の背景と意義

現在，半導体産業はあらゆる産業にとってなくてはならない存在となり，半導体技術がそれらの産業の発展を支える現状にある．今日の半導体技術は半導体集積回路の基本デバイスである CMOS FET (Complementary Metal-Oxide-Semiconductor Field-Effect Transistor) の微細化により実現されてきた．CMOS の微細化の進展により，高速化・省電力化・高集積化・低コスト化が実現できる．この微細化技術をベースに，LSI (Large-Scale Integration) 上のトランジスタ数はムーアの法則に従って増大してきた．ムーアの法則によると，LSI 上のトランジスタ数は 18ヶ月ごとに 2 倍になる [32]．[42] によると，1970 年には 1,000 個だった LSI 上のトランジスタの数が、1990 年には 100 万，2010 年には 10 億を超すまでに増えており，ムーアの法則が提唱された 1965 年から LSI の集積度はほぼ法則通りに進化を続けている．

集積度向上の過程で 1990 年代に，プロセッサ周辺の機能を統合して 1 チップに集積した SoC (System on a Chip) が出現した．機能ごとに複数のチップで実装する場合に比べ，SoC ではチップの小型化・配線の省略による高速化・部品削減による省電力化・製造コスト削減などのメリットが期待できる．SoC の回路モジュールである IP コアには，マイクロプロセッサ・信号処理回路・通信回路・バス・A/D コンバータなどがある．これらの IP コアと並んで SoC の中で重要な IP コアとしてメモリがある．SoC 上のオンチップメモリはワーキングメモリとして使われ，SRAM (Static Random-Access Memory) が一般的に用いられている．SRAM はフリップフロップを用いてデータを記憶する揮発メモリである．SRAM は読み書きにかかる消費電力が小さく，動作が高速であるという特徴がある一方，メモリセルが大きく大容量のメモリでは回路面積が非常に大きくなる欠点がある [40]．昨今の SoC におけ

表 1.1: メモリの性質 [29, 30].

	SRAM	DRAM	PCM	ReRAM	MRAM
セルサイズ	120–200F^2	6–10F^2	4–12F^2	4–10F^2	6–50F^2
読み込み時間	5ns	50ns	50ns	10ns	10ns
書き込み時間	5ns	50ns	500ns	50ns	50ns
書き込み耐性	10^{16}	10^{16}	10^8–10^9	10^{11}	10^{16}
リーク電力	大	中	0	0	0
読み込み電力	小	中	中	小	小
書き込み電力	小	中	大	大	大

る SRAM が占める面積は 50%以上と言われ，この割合は世代を経るごとに増加の傾向にある [13]．また，SoC の微細化に伴うトランジスタのリーク電流の増加により SRAM の消費電力は増大しており，SoC におけるオンチップメモリの消費電力が問題となっている [4]．つまり，SoC の今後の高集積化と省電力化を進めるためには，SoC の面積と消費電力の大部分を占めるワーキングメモリを高集積化・省電力化する必要がある．また，SoC の外部 (オフチップ) ではワーキングメモリとして DRAM (Dynamic Random-Access Memory) が用いられる．DRAM はキャパシタを用いてデータを記憶する揮発メモリである．DRAM は集積度が高く大容量のメモリを構成することに向いているが，微細化の限界を迎えている [25, 28]．さらに，データを保持し続けるにはリフレッシュ動作が必要で，リフレッシュのために常にエネルギーを消費し続けるという欠点がある [27].

また，近年スマートフォンやウェアラブル機器などの小型 IoT デバイスが爆発的に普及している．[58] によると，2014 年時点での小型 IoT デバイスの出荷数は約 94 億個であり，2020 年には約 190 億個にまで規模が増大する．小型 IoT デバイスでは物理的なサイズを小さくしつつ長時間のバッテリ駆動が求められるため，小型 IoT デバイスにおけるワーキングメモリの高集積化と省電力化は非常に重要な課題となっている [19, 49]．しかし，現状の SRAM/DRAM では高集積化と省電力化の劇的な改善は難しく，特に，SRAM/DRAM は電力を供給し続けないと記録したデータが消えてしまう揮発メモリであるため，データを保持し続けるためのエネルギーが無視できない．そこで，データを保持し続けるためのエネルギーの削減を実現するために，不揮発メモリが注目されている [39]．不揮発メモリは電力の

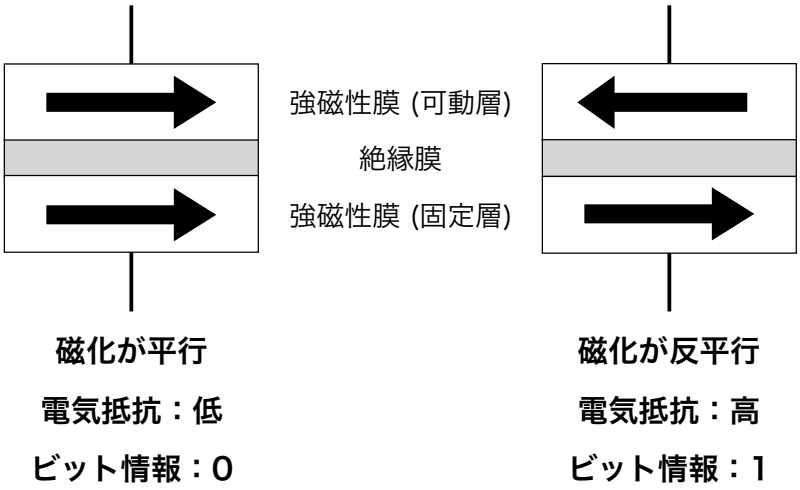


図 1.1: MRAM の構造.

供給がなくとも記録されたデータを保持し続けるメモリである．不揮発メモリでは，電力を供給しなくとも記憶したデータは失われないため，不要なときには電源を切り，必要なときだけ電源を入れて使用することで省電力化が期待できる．また，使用時のリフレッシュが不要なため，使用時の消費エネルギーが大きく削減できる．

代表的な不揮発メモリに PCM (Phase-Change Memory; 相変化メモリ) [36]，ReRAM (Resistive Random-Access Memory; 抵抗変化メモリ) [33]，MRAM (Magnetoresistive Random-Access Memory; 磁気抵抗メモリ) [52] がある．表 1.1 に各メモリの性質を示す [29,30]．PCM は電気抵抗の低い結晶相と電気抵抗の高いアモルファス相の 2 つの状態でデータを記録する不揮発メモリである．高集積が容易であるが，書き込み耐性が低く，書き込み時間が長いという欠点がある．ReRAM は電圧印加による電気抵抗の変化でデータを記録する不揮発メモリである．1 ビットあたりのサイズが小さく高集積に向いているが，書き込み耐性，速度，電力に課題がある．MRAM は磁性体の磁化の向きによってデータを記録する不揮発メモリである．MRAM は書き換え回数がほぼ無限大で読み書きが高速であるという利点がある．MRAM の構造を図 1.1 に示す．MRAM の記憶素子は MTJ (Magnetic Tunneling Junction; 磁気トンネル接合) 素子と呼ばれる 2 層構造の素子から構成される．2 層は，上部可動層と下部固定層の強磁性膜であり，間に絶縁膜を挟む構造をしている．固定層は，磁化 (磁気モーメントの方向) が特定の方向に固定されている層である．可動層は，磁化が固定層の磁化と

平行な方向，あるいは反平行な方向のどちらかの状態を取れる層である．絶縁膜層は，電子が磁気トンネル効果によって通過する層である．MRAM のメモリ素子からデータを読み出すときには，上部可動層の磁化方向による MTJ 素子の抵抗値の違いを読み取る．自由層の磁化が固定層とは逆 (反平行) ならば電気抵抗が高い状態，自由層の磁化が固定層と同じ (平行) ならば電気抵抗が低い状態となり，この電気抵抗の違いをそれぞれ 2 進情報の 1 と 0 に対応させる．MRAM にデータを書き込むときには，上部可動層の磁化方向を変化させることでデータを書き込む．電力供給を遮断した後でもデータは磁化の方向として残るためデータは保持される．

MRAM は磁化の向きを反転する原理によって 2 種類に分けられる．1 つ目は電流による磁界で磁化の方向を反転させる方式である．この方式は製造が容易である一方，磁界発生のために専用配線を必要とすることなどからメモリセル面積が大きくなる欠点がある．そこで 2 つ目の方式として電子のスピンによるトルクを利用して磁化を反転させる MRAM が提案された [7]．この方式では磁界発生用の配線が不要になるためメモリセルの面積を小さくできる．

以上に紹介した不揮発メモリの中でも MRAM は，不揮発性による省エネルギー、SRAM に匹敵する高速な読み書き性能、DRAM に匹敵する高集積性，そして特に無限回数に近い書き換え耐性の性質を持つため，新世代ユニバーサルメモリとして最も注目を集めている．実際の MRAM の製品としては，2016 年に東芝と SK Hynix が 4Gb の単体メモリを，同年に Samsung が 8Mb のオンチップメモリを開発している [9,41]．以上のように MRAM は現在積極的に開発が進められており，なおかつワーキングメモリとして使用するにあたって重要な性質である読み書き速度・書き換え耐性・高集積性を持つことから本論文は MRAM を対象とする．MRAM は上記の利点を持つ一方で，既存のワーキングメモリに比べ 1 ビットあたりの書き込みエネルギーが大きく，メモリに書き込むデータにソフトエラーが発生しやすいという欠点がある [16,17,30]．そこで，MRAM をワーキングメモリとして用いる場合，メモリにデータを書き込むときの書き込みエネルギー削減と，メモリに保存したデータに対しての信頼性向上が要求される．

メモリに保存したデータにソフトエラーが生じるパターンには 3 種類ある．書き込み時に

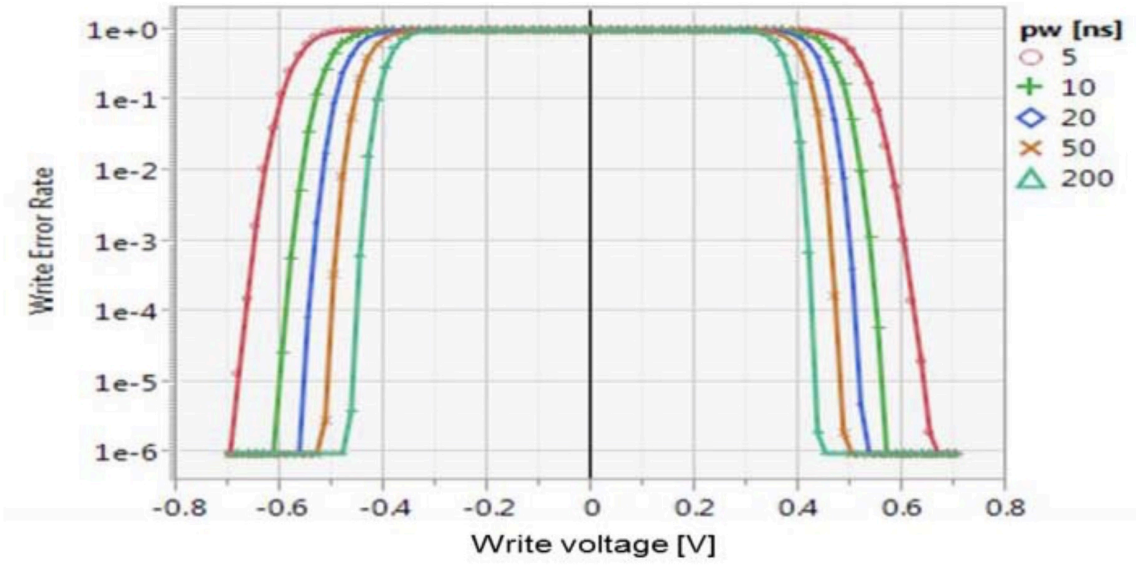


図 1.2: MRAM の書き込みエネルギーと書き込みエラー率の関係 [15].

誤る書き込みエラー，データが保存されている間に誤るリテンションエラー，読み込み時に誤る読み出しエラーである [46]．その中でも MRAM では書き込みエラーの割合が最も高い．図 1.2 に MRAM の書き込みエネルギーと書き込みエラー率の関係を示す．図 1.2 より，書き込みエネルギーと書き込みエラー率がトレードオフの関係にあることがわかる．一般的に用いられる MRAM の書き込みエラー率は 10^{-6} と言われている [15]．メモリに生じるソフトエラーに対して以下の手法が提案されている．書き込みエラーに対して，書き込み後に書き込みデータを確認して成功するまで書き込みを繰り返す手法 [3,50]，読み出しエラーに対して，読み込み後に再度正しいデータを書き込む手法 [37,43]，すべてのエラーに対して有効な，読み込み時に誤り訂正符号を利用して正しいデータを読み出す手法 [2,11,47] である．

誤り訂正符号を利用してデータを読み出す手法では対応できるエラー数に制約はあるものの，メモリに発生するソフトエラーのすべてのパターンに対応できるため，本論文ではこの手法を採用する．誤り訂正符号を用いてメモリを構成した場合には，保存したい値を符号語にエンコードして，その符号語をメモリに書き込むことで値を保存する．実際にメモリに用いられる代表的な誤り訂正符号として，ハミング符号 [14]，BCH 符号 [5] などがある．これらの誤り訂正符号は符号語同士のハミング距離を大きくすることで誤り訂正能力を得る．しかし符号語同士のハミング距離が大きいことから，誤り訂正符号を用いて不揮発メモリを構

成した場合にはメモリに符号語を書き込むときの書き込みビット数が増大する．メモリに書き込むビット数が多いほど，消費エネルギーも増えるため，誤り訂正符号を用いた不揮発メモリでは符号語の書き込みにかかるエネルギーが非常に大きくなるという欠点がある．加えて，前述の通り MRAM の書き込みエネルギーは既存のワーキングメモリよりも大きい．そのため誤り訂正符号を用いて不揮発メモリを構成した場合，書き込むエネルギーの削減が強く要求される．書き込みエネルギーは書き込むビット数に関係するため，メモリに書き込むデータを書き込みビット数が減るように変換することで書き込みエネルギーを削減できる．

不揮発メモリを対象とした書き込みビット数を削減する研究として [8, 44, 51, 59] がある．MRAM は同じビットを上書きする際にも書き込みエネルギーを必要とするため，Early Write Termination では保存したい値を表すビット列をメモリに書き込むとき，書き込むビットが書き込まれているビットと同じであればそのビットを書き込まないことで書き込みビット数を削減する [51]．Flip-N-Write はメモリに保存する値に 1 ビットの冗長部分を付加する [8]．付加ビットが 1 であれば値をビット反転し，付加ビットが 0 であれば値はビット反転しない．書き込むビット長の半数以上に書き込む場合に値をビット反転することで書き込みビット数を削減する．最大ハミング距離を制約した符号は，符号語間の最大ハミング距離を制約する符号を生成することで書き込み削減を行う [59]．書き込み削減符号は誤り訂正符号を元に符号語間の最大ハミング距離を制限する符号を生成することで書き込みビット数を削減する [44]．ところが著者の知る限り，書き込みビット数削減を考慮した誤り訂正符号は知られていない．

ここで，元の値のすべてのビットが反転する場合を最悪の場合と呼ぶことにする．書き込みビット数削減には平均的に書き込みビット数を削減する平均書き込みビット数削減と，最悪の場合に書き込みビット数を削減する最悪書き込みビット数削減がある．例えば，表 1.2 に示すハミング符号の最悪の場合とは，符号語 0000000 → 1111111 で 7 ビットの書き換えが必要となる場合である．つまり，最悪書き込みビット数は符号長に等しい 7 ビットとなる．ハミング符号を用いて値を符号化したとき，任意の値の遷移には平均的に 3.5 ビット，最悪の場合では 7 ビットの書き換えが必要である．

誤り訂正符号を用いて構成した不揮発メモリのメモリアーキテクチャを考える．メモリに値を保存するときにはエンコーダを用いて値を符号語にエンコードし，メモリから値を取り

表 1.2: ハミング符号の値と符号語の関係.

値	符号語	値	符号語
0000	0000000	1000	1000011
0001	0001111	1001	1001100
0010	0010110	1010	1010101
0011	0011001	1011	1011010
0100	0100101	1100	1100110
0101	0101010	1101	1101001
0110	0110011	1110	1110000
0111	0111100	1111	1111111

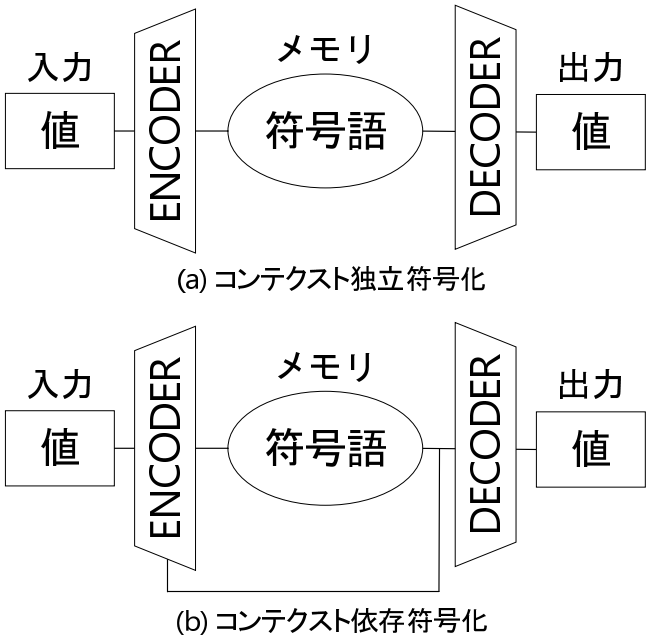


図 1.3: エンコーダ/デコーダを用いた不揮発メモリのメモリアーキテクチャ.

出すときにはデコーダを用いて符号語を値にデコードする．値をエンコードして保存するメモリのメモリアーキテクチャには 2 つの種類がある．メモリに書き込まれている符号語を考慮せずエンコードする方式（コンテクスト独立）と考慮してエンコードする方式（コンテクスト依存）である．図 1.3(a) にコンテクスト独立符号化のメモリアーキテクチャを，図 1.3(b) にコンテクスト依存符号化のメモリアーキテクチャをそれぞれ示す [31].

図 1.3(a) に示すメモリアーキテクチャでは，メモリに値を保存するとき，値を符号語にエ

ンコードし，その符号語をメモリに書き込む．メモリから値を取り出すときには，メモリに書き込まれている符号語を読み出し，読み出した符号語をデコードすることで元の値を得ることができる．このメモリアーキテクチャに対して，値と符号語が 1 対 1 に対応した符号を用いたメモリを構成することができる．図 1.3(b) に示すメモリアーキテクチャではメモリに値を保存するとき，まずメモリに書き込まれている符号語を読み出す．読み出した符号語を基に値を符号語にエンコードする．メモリから値を取り出すときには，メモリに書き込まれている符号語を読み出し，読み出した符号語をデコードすることで元の値を得ることができる．このメモリアーキテクチャに対して，値と符号語が 1 対多に対応した符号を用いたメモリを構成することができる．

以上の背景より，本論文では，ワーキングメモリとして誤り訂正符号を用いた不揮発メモリ (MRAM) を構成した場合の書き込みビット数削減手法を提案する．提案手法を用いて生成した符号は同じ誤り訂正能力を有する従来符号と比較して書き込みビット数を削減できる．

MRAM を SRAM/DRAM の代替としてワーキングメモリに用いる有効性は既に [34,45] 等で報告されているため，本論文では，提案手法を用いて構成したメモリと既存の誤り訂正符号を用いて構成したメモリとを比較評価する．

1.2 本論文の概要

本論文では，前節で述べた要求を満たすために，誤り訂正符号を用いて不揮発メモリを構成した場合の書き込みビット数削減手法を提案する．

本論文は 7 章から構成される．以下に本論文の構成を示す．

第 2 章**「研究動向」**では，書き込みビット数削減手法と誤り訂正手法を紹介する．

第 3 章**「ドーナツ符号」**では，最悪書き込みビット数削減と誤り訂正を同時に実現する符号としてドーナツ符号を提案する．また，符号拡張手法を提案し，ドーナツ符号に符号拡張手法を適用した拡張ドーナツ符号を提案する．コンテクスト独立メモリアーキテクチャに対して，拡張ドーナツ符号を用いて構成した不揮発メモリでは，メモリに生じた誤りを訂正でき，最悪書き込みビット数を削減できる．

第 4 章**「一対多符号」**では，最悪書き込みビット数削減と誤り訂正を同時に実現する符号

として一対多符号を提案する．コンテクスト依存メモリアーキテクチャに対して，一対多符号を用いて構成した不揮発メモリでは，メモリに生じた誤りを訂正でき，最悪書き込みビット数を大きく削減できる．

第 5 章**「REC 符号」**では，平均書き込みビット数削減と誤り訂正を同時に実現する符号として REC 符号を提案する．コンテクスト依存メモリアーキテクチャに対して，REC 符号を用いて構成した不揮発メモリでは，メモリに生じた誤りを訂正でき，平均書き込みビット数を削減できる．

第 6 章**「Relaxed-REC 符号」**では，REC 符号の元となる線形組織誤り訂正符号の制約を緩めた Relaxed-REC 符号を提案する．コンテクスト依存メモリアーキテクチャに対して，Relaxed-REC 符号を用いて構成した不揮発メモリでは，メモリに生じた誤りを訂正でき，平均書き込みビット数を大きく削減できる．

第 7 章**「結論」**では，本論文全体の内容を総括する．そして，提案手法の現状における課題点を踏まえ，今後の研究課題を検討する．

第2章　研究動向

2.1　本章の概要

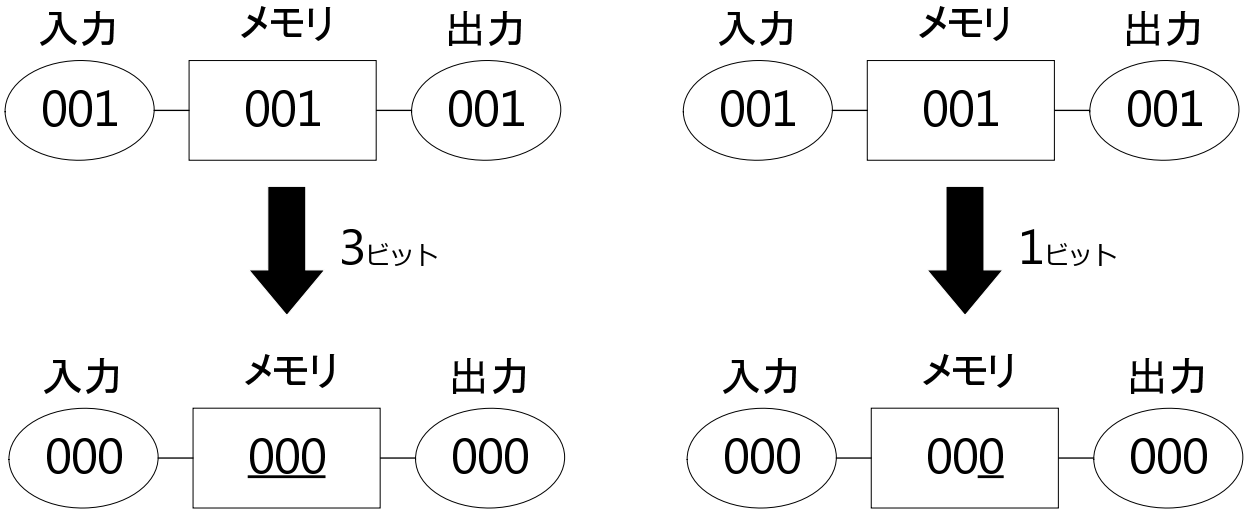
本章では，書き込みビット数削減と誤り訂正手法の既存研究を紹介する．不揮発メモリを対象とした書き込みビット数を削減する手法として Early Write Termination [51]，Flip-N-Write [8]，最大ハミング距離を制約した符号 [59]，書き込み削減符号 [44] がある．

不揮発メモリに保存されているデータに誤りが生じた際に誤り訂正する手法としてハミング符号 [14] がある．

しかし，既存の書き込みビット数削減手法はメモリに書き込まれているデータに誤りが生じた場合の誤り訂正を考慮しておらず，また，既存の誤り訂正手法はメモリにデータを書き込む際の書き込みビット数の削減を考慮していない．

2.2　書き込みビット数削減手法の既存研究

本節では，書き込みビット数削減手法の既存研究を紹介する．不揮発メモリを対象とした書き込みビット数を削減する手法として [8, 44, 51, 59] がある．Early Write Termination は，保存したい値を表すビット列をメモリに書き込むとき，書き込むビットが書き込まれているビットと同じであればそのビットを書き込まないことで書き込みビット数を削減する [51]．Flip-N-Write は，メモリに保存する値に 1 ビットの冗長部分を付加する [8]．付加ビットが 1 であれば値をビット反転し，付加ビットが 0 であれば値はビット反転しない．書き込むビット長の半数以上に書き込む場合に値をビット反転することで書き込みビット数を削減する．最大ハミング距離を制約した符号は，符号語間の最大ハミング距離を制約する符号を生成することで書き込み削減を行う [59]．書き込み削減符号は，誤り訂正符号を元に符号語間の最大ハミング距離を制約する符号を生成することで書き込みビット数を削減する [44]．



(a) 通常の不揮発メモリ.　(b) Early Write Termination を用いた不揮発メモリ.

図 2.1: 通常の不揮発メモリ (a) と Early Write Termination を用いて構成した不揮発メモリ (b) の動作.

いずれの手法も書き込みビット数は削減できるが，メモリに書き込まれているデータに誤りが生じた場合の誤り訂正を考慮していない.

2.2.1 Early Write Termination

Early Write Termination は，保存したい値を表すビット列をメモリに書き込むとき，書き込むビットが書き込まれているビットと同じであればそのビットを書き込まないことで書き込みビット数を削減する [51]．通常の不揮発メモリの動作を図 2.1(a) に，Early Write Termination を用いて構成したメモリの動作を図 2.1(b) に示す．図 2.1(a) に示した通常の不揮発メモリでは，メモリに保存されている値と関係なく入力の全てのビット値をメモリを書き込むため，書き込みビット数は 3 ビットとなる．図 2.1(b) に示した Early Write Termination を用いて構成したメモリでは，メモリに同じビット値は書き込まず，異なるビット値のみ書き込むため，書き込みビット数は 1 ビットとなる．以降で記述する通常の不揮発メモリは Early Write Termination を用いて構成した不揮発メモリとする.

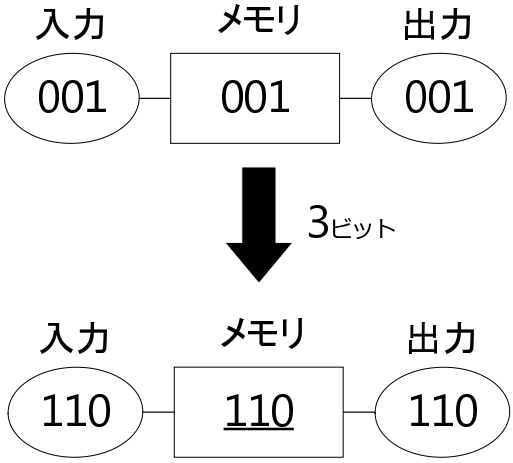


図 2.2: 通常の不揮発メモリの動作.

2.2.2 Flip-N-Write

Flip-N-Write は，コンテクスト依存符号化のメモリアーキテクチャを用いて書き込みビットの数を削減する [8]．メモリに保存する値に 1 ビットの冗長部分を付加する．メモリからデータを出力する際，付加ビットの値が 0 であればメモリに保存している符号語をそのまま出力し，付加ビットの値が 1 であればメモリに保存している符号語を全てビット反転して出力する．書き込むビット長の半数以上に書き込む場合に値をビット反転することで書き込みビット数を削減する．通常の不揮発メモリの動作を図 2.2 に示す．図 2.2 に示すように値 001 がメモリに保存されているとき，値 110 を書き込むと通常のメモリでは書き込みビット数は 3 となる．図 2.3 に示すように Flip-N-Write を用いて構成したメモリでは値 110 は付加ビット 0 を加えた符号語 1100 と付加ビット 1 を加えた符号語 0011 の 2 通りで表せる．このとき，図 2.4 に示すように Flip-N-Write を用いて構成したメモリでは，付加ビットの値を 1 とすることで，書き込みビット数は 1 ビットに削減できる．

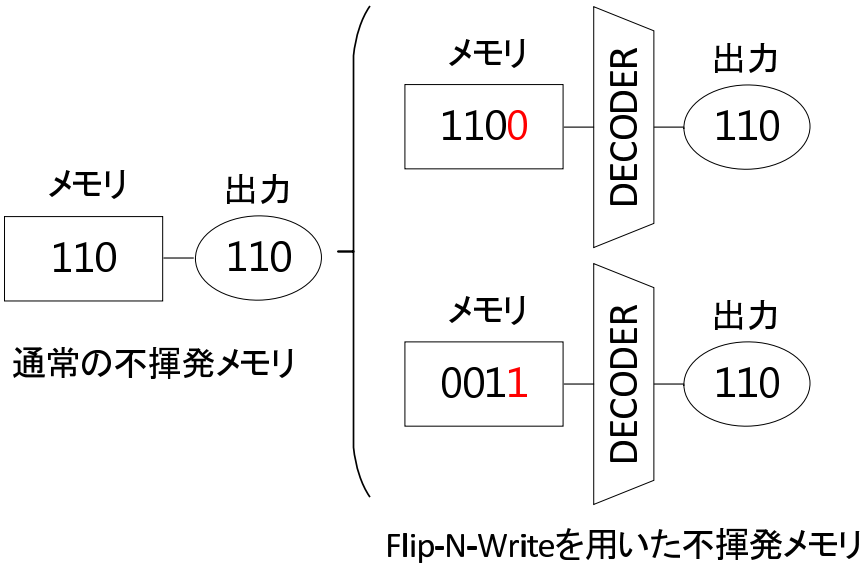


図 2.3: 通常の不揮発メモリと Flip-N-Write を用いて構成した不揮発メモリの関係.

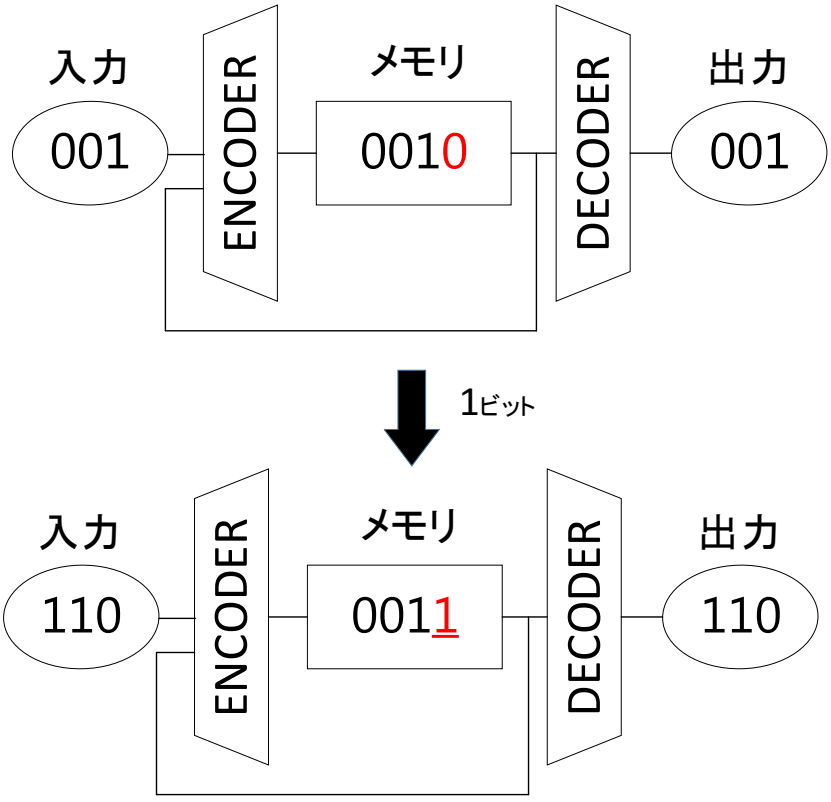


図 2.4: Flip-N-Write を用いて構成した不揮発メモリの動作.

表 2.1: 3 ビット値と最大ハミング距離を制約した 7 ビットの符号語の関係.

値	符号語
000	0000000
001	0000001
010	0000010
011	0000100
100	0001000
101	0010000
110	0100000
111	1000000

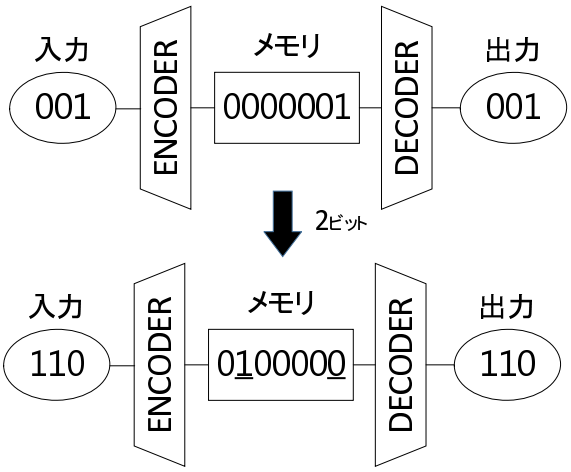


図 2.5: 最大ハミング距離を制約した符号を用いて構成した不揮発メモリの動作.

2.2.3 最大ハミング距離を制約した符号

最大ハミング距離を制約した符号は，符号語間の最大ハミング距離を制約する符号を生成することで書き込み削減を行う [59]．表 2.1 に 3 ビットの値とそれを 7 ビットにエンコードした符号語の関係を示す．符号語同士の最大ハミング距離を 2 以下に制約している．最大ハミング距離を制約した符号を用いて構成したメモリの動作を図 2.5 に示す．図 2.2 に示すように値 001 がメモリに保存されているとき，値 110 を書き込むと通常のメモリでは書き込みビット数は 3 となる．図 2.5 に示すように最大ハミング距離を制約した符号を用いて構成したメモリでは，符号語間の最大ハミング距離を 2 以下に制約した符号語を生成することで書き込みビット数はたかだか 2 ビットに削減できる．

2.2.4 書き込み削減符号

書き込み削減符号は，誤り訂正符号を元に符号語間の最大ハミング距離を制約する符号を生成することで書き込みビット数を削減する [44]．書き込み削減符号ではコンテクスト依存符号化のメモリアーキテクチャを用いて，メモリに保存したい値と現在メモリに書き込まれている符号語から実際にメモリに書き込む符号語を決定する．書き込み削減符号の生成法を以下に示す．

Step1: 書き込み削減符号の元となる誤り訂正符号を決定する．

Step2: 値と情報ベクトルを 1 対 1 に対応させる．情報ベクトルはビット長 n，ハミング重み t ビット以下の誤りベクトルである．

Step3: 情報ベクトルと書き込み削減符号の符号語を 1 対多に対応させる．書き込み削減符号の符号語は始点ベクトルと情報ベクトルの排他的論理和で，始点ベクトルは誤り訂正符号の符号語でそれぞれ表される．

以上より，符号長 n，情報量 k，誤り訂正能力 t のとき，$(n, k, 2t + 1)$ 誤り訂正符号から $(n, n - k, t)$ 書き込み削減符号が生成できる．t ビット誤り訂正可能な符号から書き込み削減符号を生成した場合，最大書き込みビット数をたかだか t ビットに制約できる．

図 2.6 に (7, 4, 3) ハミング符号から (7, 3, 1) 書き込み削減符号を生成する例を示す．(7, 4, 3) ハミング符号は 16 個の符号語があり，1 ビットの誤り訂正可能である．それぞれ符号語で 0 ビットから 1 ビットの誤りが発生するとき，誤りの位置は 8 種類ある．誤りの位置をそれぞれ情報ベクトルとし，値と情報ベクトルを 1 対 1 に対応させる．情報ベクトルは 8 個ある．(7, 4, 3) ハミング符号の各符号語を始点ベクトルとすると，16 個の始点ベクトルと 8 個の情報ベクトルの排他的論理和で書き込み削減符号の符号語が表される．書き込み削減符号の符号語数は 128 個となる．メモリに値を書き込む際には，どの符号語がメモリに書き込まれていても他の値を表す集合内の符号語にたかだか 1 ビットで遷移できる．

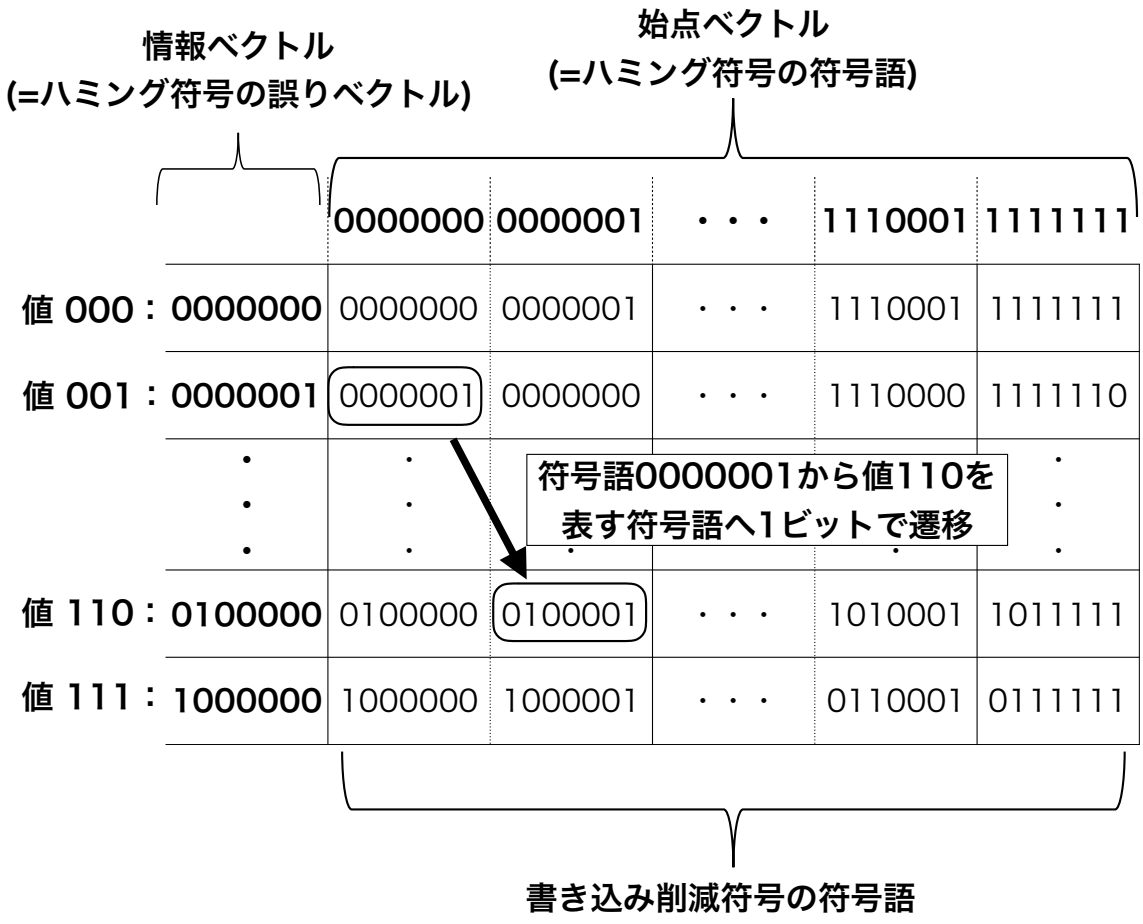


図 2.6: (7, 4, 3) ハミング符号から書き込み削減符号を生成する例.

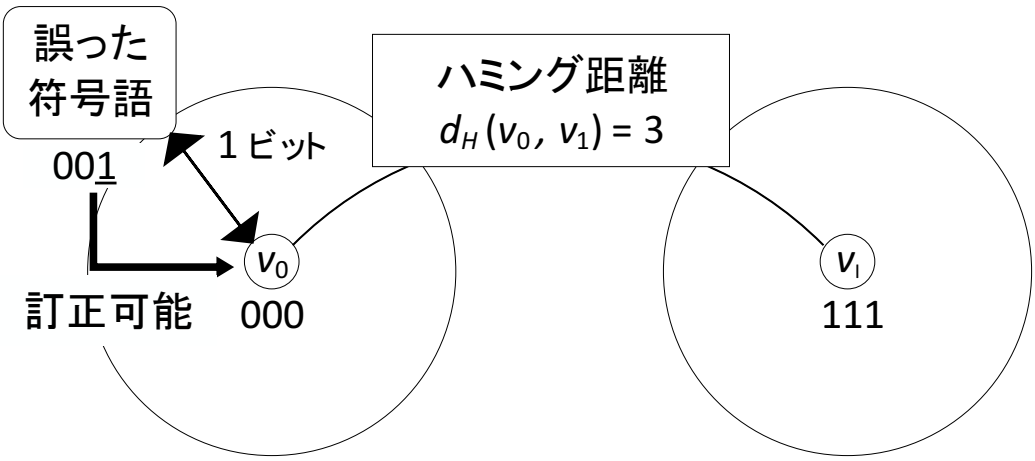


図 2.7: 誤り訂正の仕組み.

2.3 誤り訂正手法の既存研究

値に符号語が割り当てられているとき，符号語の集合を符号と呼ぶ．符号 V に属するある符号語を v_i, v_j と表す．v_i と v_j のハミング距離を $d_H(v_i, v_j)$ と表す．v_i のハミング重みを $w_H(v_i)$ と表す.

最大 (最小) ハミング距離とは，符号語間のハミング距離の最大値 (最小値) である．誤り訂正符号では，符号語間の最小ハミング距離を制約することにより誤り訂正が可能となる．一般に，最小ハミング距離が $2t+1$ のとき，t ビットの誤り訂正能力を持つ．符号 V の最大ハミング距離を $d_{max}(V)$，最小ハミング距離を $d_{min}(V)$ と表す.

図 2.7 に誤り訂正の仕組みを示す．符号語 $v_0 = \{000\}$ と符号語 $v_1 = \{111\}$ のハミング距離は 3 であるため，$d_H(v_0, v_1) = 3$ となる．符号語 v_0 の 3 ビット目に誤りが生じ，メモリに保存されているデータが $00\underline{1}$ となった場合でも，符号語 v_0 から 1 ビットだけ誤って得られる語と，符号語 v_1 から 1 ビットだけ誤って得られる語は一致することはないため，元の符号語に訂正可能である.

誤り訂正符号にはハミング符号などが知られている．ハミング符号は，符号語間の最小ハミング距離が 3 であり 1 ビットの誤り訂正が可能な完全符号である [14]．自然数 m を用いると，

符号長 [ビット]：$n = 2^m - 1$

情報量 [ビット]：$k = n - m$

が成立する．これを $(n, k, 3)$ ハミング符号と表す．

ハミング符号 [14] や BCH 符号 [5] などの誤り訂正符号は符号語間の最小ハミング距離が制約されていることによって誤り訂正能力は有するが，最大ハミング距離は制約されていない．例えば，(7, 4, 3) ハミング符号の符号語間の最大ハミング距離は 7 であり，符号長に等しい．つまり，ハミング符号では最悪の場合，メモリに書き込まれた符号長分の書き込みビット数が発生する可能性がある．(7, 4, 3) ハミング符号を用いて構成したメモリでは，1 回の書き込みでの平均書き込みビット数は 3.5 ビットである．

2.4 本章のまとめ

本章では，不揮発メモリを対象とする書き込みビット数削減手法と誤り訂正手法の既存研究を紹介した．不揮発メモリを対象とした書き込みビット数を削減する研究として Early Write Termination，Flip-N-Write，最大ハミング距離を制約した符号，書き込み削減符号を紹介した．本章で紹介した書き込みビット数削減手法はデータに誤りが生じた場合の誤り訂正を考慮していない．

次に，誤り訂正手法の既存研究としてハミング符号を紹介した．本章で紹介した誤り訂正手法はメモリにデータを書き込む際の書き込みビット数の削減を考慮していない．

著者の知る限り，書き込みビット数削減と誤り訂正を同時に実現する手法は知られていない．

第3章　ドーナツ符号

3.1　本章の概要

本章では，最大ハミング距離と最小ハミング距離を制約した符号としてドーナツ符号を提案する[1]．また，符号拡張手法を提案し，ドーナツ符号に符号拡張手法を適用した拡張ドーナツ符号を提案する．拡張ドーナツ符号を用いて構成した不揮発メモリは，メモリに生じた誤りを訂正でき，最悪書き込みビット数を削減できる．拡張ドーナツ符号を用いて構成した不揮発メモリはハミング符号を用いて構成した不揮発メモリと比較してエネルギーを最大35.7%削減した．

拡張ドーナツ符号ではメモリに書き込まれている符号語を考慮せずに値をエンコードするコンテクスト独立符号化のメモリアーキテクチャを用いる．

3.2　最大ハミング距離と最小ハミング距離を制約した符号

偶数パリティ符号について考える．パリティ符号とは，書き込む値にパリティビットと呼ばれる1ビットを付加した符号である．偶数パリティ符号では符号語のハミング重みが偶数になるようにパリティビットを付加する．ここで，ハミング重みとは，符号語における1の個数である．符号長が奇数となるように構成した偶数パリティ符号をドーナツ符号と呼ぶ．ドーナツ符号の性質を表3.1に示す．ドーナツ符号の符号長を決定するパラメータであるoを1以上の整数とする．符号長$2t+1$のドーナツ符号$DN(o)$は最大ハミング距離$d_{max}(DN(o)) \leq 2o$と最小ハミング距離$d_{min}(DN(o)) \geq 2$に制約された符号である．

定理 3.1. 符号長$2o+1$のドーナツ符号$DN(o)$は最大ハミング距離$d_{max}(DN(o)) \leq 2o$と最小ハミング距離$d_{min}(DN(o)) \geq 2$に制約された符号である．

[1]本章の内容は [20, 22, 53, 54] による．

表 3.1: ドーナツ符号の性質.

符号長	$2o+1$
情報量	$2o$
符号語数	2^{2o}
最大ハミング距離	$2o$
最小ハミング距離	2

表 3.2: ドーナツ符号 $DN(2)$.

値	符号語	値	符号語
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

証明 3.1. ドーナツ符号の符号語同士のハミング距離を考える．背理法で証明する．ドーナツ符号の符号語 v_i と v_j について，ハミング距離 $d_H(v_i, v_j)$ が奇数であると仮定する．偶数パリティ符号の性質により v_i のハミング重みは偶数であるため，ハミング重みを $w_H(v_i) = 2w$ とする．ハミング距離が奇数であるため，v_j は v_i の任意の $2s+1$ 個のビットが反転している．v_i 中の反転するビット 0 が m 個であり，ビット 1 が $2s+1-m$ 個である場合，v_j のハミング重みは $w_H(v_j) = 2w + m - (2s+1-m) = 2(w+m-s) - 1$ となる．ドーナツ符号の符号語のハミング重みは偶数であるが，$2(w+m-s)-1$ は奇数であるため，矛盾が生じる．よって，ドーナツ符号の符号語同士のハミング距離は偶数である．符号長 $2o+1$ のドーナツ符号の符号語同士の最大ハミング距離はたかだか $2o$，最小ハミング距離は少なくとも 2 となり，これは最大ハミング距離と最小ハミング距離が制約された符号である． □

例 3.1. ドーナツ符号 $DN(2)$ を表 3.2 に示す．性質を表 3.3 に示す.

表 3.3: ドーナツ符号 $DN(2)$ の性質.

符号長	5
情報量	4
最大ハミング距離	4
最小ハミング距離	2

3.3 符号拡張手法

符号長 $2o+1$ のドーナツ符号は，符号語同士の最大ハミング距離が $2o$，最小ハミング距離が 2 の性質を持つ．符号長 $2o+1$ に対して最大ハミング距離が $2o$ であるため効果的な書き込みビット数削減が実現できず，最小ハミング距離は 3 以上のときに誤り訂正能力を持つため，ドーナツ符号は誤り訂正能力を持たない．

そこで本節では，ドーナツ符号の最大ハミング距離と最小ハミング距離をより現実的な制約とするために，符号の性質を拡張する手法を提案する．これを符号拡張手法と呼び，符号拡張手法を適用した結果，構成される符号を拡張符号と呼ぶことにする．図 3.1 を基に符号拡張手法を述べる．

Step 1: 元となる符号語のリスト A を決定する．リスト A は n_A 個の符号語を持ち，それらを $A = \{A_1, A_2, A_3, \ldots, A_{n_A}\}$ とする．符号語 A_i をビット反転した符号語を $\overline{A_i}$ とするとき，リスト A から生成したリスト $\overline{A}$ は $\overline{A} = \{\overline{A_1}, \overline{A_2}, \overline{A_3}, \ldots, \overline{A_{n_A}}\}$ となる．リスト A に属する符号語の符号長を n とする．

Step 2: 拡張数 n_e を定める．拡張符号は，n_e 個のリストを連接して構成する．

Step 3: $p \in \{0, 1, 2, \ldots, \lfloor n_e/2 \rfloor\}$ とするとき，各要素 p に対して次のステップを実行する．

Step 4: p 個の $\overline{A}$ と $(n_e - p)$ 個の A を連接させてグループを作る．グループは符号長 $n_e \times n$ の新しい符号語をそれぞれ n_A 個持つ．

Step 5: $p < \lfloor n_e/2 \rfloor$ ならば，Step 3 を実行する．このステップで生成したグループ数は ${}_{n_e}C_p$ となる．

$p = \lfloor n_e/2 \rfloor$ ならば終了する．このステップで生成したグループ数は n_e が奇数のとき ${}_{n_e}C_p$, 偶数のとき ${}_{n_e}C_p/2$ となる．全体のグループ数は 2^{n_e-1} となる．

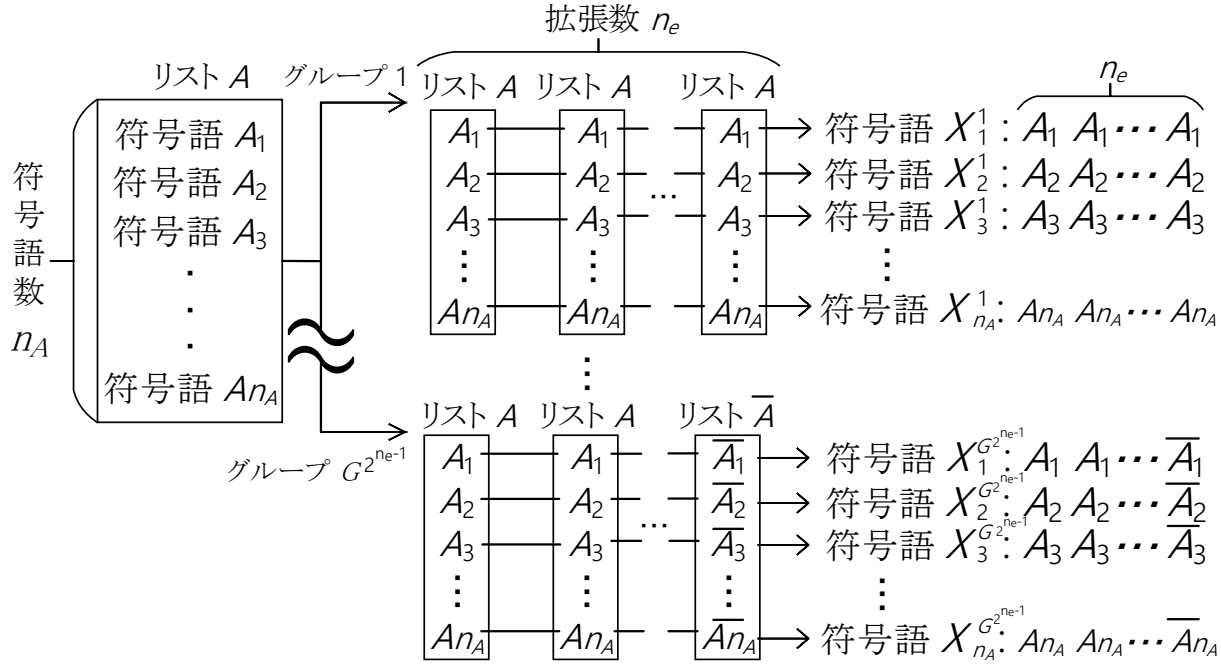


図 3.1: 符号拡張手法.

以上のように構成された $n_A \times 2^{n_e-1}$ 個の符号語を持つ符号が拡張符号である．元となる符号の符号長を n，情報量を k とすると，拡張符号の符号長は $n_e n$，情報量は $k + n_e - 1$ となる．

ここで，グループ G_j と G_j を構成する全てのリストが反転した $\overline{G_j}$ を考える．グループを構成する際に，リスト $\overline{A}$ はたかだか $p = \lfloor n_e/2 \rfloor$ しか用いていないため，$\overline{G_j}$ は拡張符号には含まない．グループ G_j に属する拡張符号のうち，G_j 内で k 番目の符号語を $X^j_k \in G_j$ と書く．

例 3.2. 符号長 3 ビットのドーナツ符号に拡張数 $n_e = 3$ の符号拡張手法を適用した拡張符号を図 3.2 に示す（詳細は 3.4.2 項を参照）.

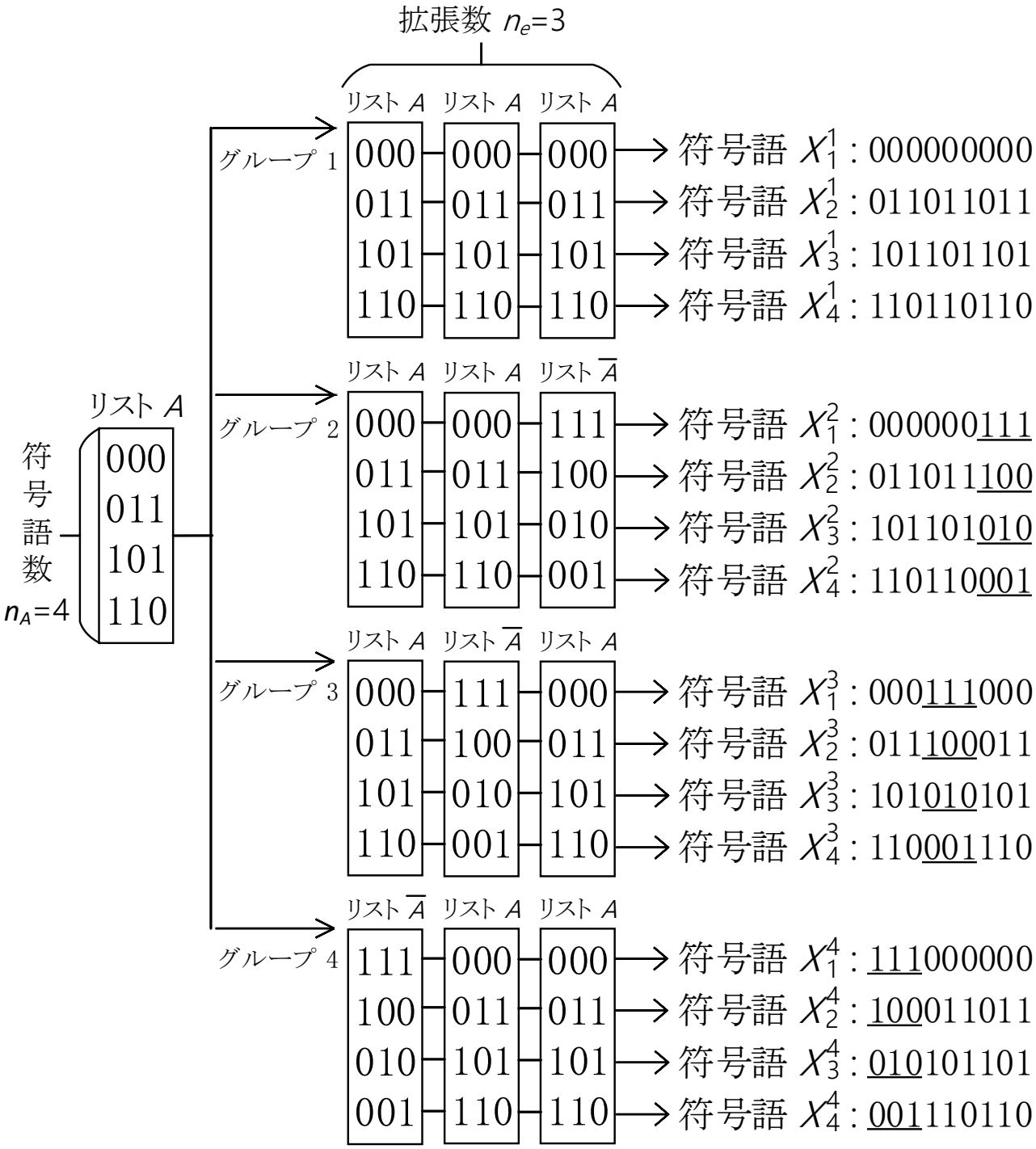


図 3.2: 符号拡張手法の例.

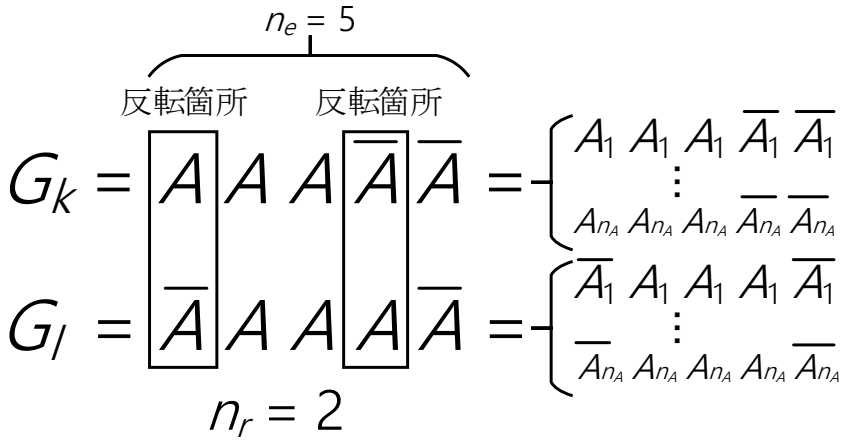


図 3.3: 反転箇所 n_r の例（$n_e = 5$）.

拡張符号の符号語間のハミング距離を考える.

定理 3.2. 元となる符号語のリスト A に属する符号語の符号長を n とし，符号語 A_i，A_j について，$d_H(A_i, A_j) = m$ とする．グループ G_k とグループ G_l の反転箇所の個数を n_r とする．図 3.3 に反転箇所 n_r の例を示す．G_k，G_l について，連接したリスト同士を比較したときに，一番左のリストおよび左から 4 番目のリストは反転の有無が異なっている．この場合の反転箇所の個数 n_r は 2 となる．符号拡張手法による拡張数を n_e，グループ G_k に属する符号語の 1 つを X_i^k，グループ G_l に属する符号語の 1 つを X_j^l とする．X_i^k と X_j^l のハミング距離は次式となる.

$$d_H(X_i^k, X_j^l) = n_r \times (n - 2m) + n_e m \tag{3.1}$$

証明 3.2. リスト A において，$d_H(A_i, A_j) = m$ であれば，符号語 A_i と反転した符号語 $\overline{A_j}$ について $d_H(A_i, \overline{A_j}) = n - m$ となる．拡張符号において，拡張数 n_e のうち，反転箇所の個数は n_r，反転されていない箇所の個数は $(n_e - n_r)$ である．反転された箇所だけに着目したときのハミング距離は $n_r \times d_H(A_i, \overline{A_j}) = n_r(n - m)$ となる．反転されていない箇所だけに着目したときのハミング距離は $(n_e - n_r) \times d_H(A_i, A_j) = m(n_e - n_r)$ となる．以上をまとめて，拡張符号の符号語同士のハミング距離は $d_H(X_i^k, X_j^l) = n_r(n - m) + m(n_e - n_r) = n_r(n - 2m) + n_e m$

となる．□

符号 V と拡張数 n_e から生成した拡張符号を $E(V, n_e)$ と表す．拡張符号 $E(V, n_e)$ について式 (3.1) を用いて最大ハミング距離と最小ハミング距離を考える．

3.3.1 最大ハミング距離

拡張符号 $E(V, n_e)$ の最大ハミング距離を比較する．符号 V の符号語同士の最大ハミング距離を $d_{max}(V)$ とする．ハミング距離を 3 パターンに分けて考える．

1. 拡張符号の 2 つの符号語 $X_i^k, X_j^k\,(i \neq j)$ を考える．X_i^k，X_j^k はともにグループ G_k に属すため，反転箇所の個数は $n_r = 0$ である．m が大きくなるほどハミング距離は大きくなるため，式 (3.1) に $m = d_{max}(V)$ を代入すると，$d_H(X_i^k, X_j^k) = n_r(n - 2d_{max}(V)) + n_e d_{max}(V) = n_e d_{max}(V)$ となる．
2. 拡張符号の 2 つの符号語 $X_i^k, X_i^l\,(k \neq l)$ を考える．X_i^k，X_i^l はともに同じインデックスを持ち，符号語 v_i あるいは $\overline{v_i}$ を連接したものである．$d_H(v_i, v_i) = 0$ であるため，式 (3.1) に $m = 0$ を代入すると，$d_H(X_i^k, X_i^l) = n_r(n - 2m) + n_e m = n_r n$ となる．この値が最も大きくなる場合は $n_r = \lfloor n_e/2 \rfloor$ のときで，ハミング距離は $n \times \lfloor n_e/2 \rfloor$ となる．
3. 拡張符号の 2 つの符号語 $X_i^k, X_j^l\,(i \neq j, k \neq l)$ を考える．式 (3.1) から，$d_H(X_i^k, X_j^l) = n_r(n - 2m) + n_e m$ となる．$n_e m \geq 2n_r m$ であるため，m が大きいほどハミング距離は大きくなる．$m = d_{max}(V)$ を代入すると，$n_r(n - 2d_{max}(V)) + n_e d_{max}(V)$ となる．

   $n > 2d_{max}(V)$ の場合，n_r が大きいほどハミング距離は大きくなる．$n_r = \lfloor n_e/2 \rfloor$ を代入すると，ハミング距離は $\lfloor n_e/2 \rfloor(n - 2d_{max}(V)) + n_e d_{max}(V)$ となる．ハミング距離は $n_e = 1$ のとき，$d_{max}(V)$ となり，$n_e > 1$ のとき，n 以上となる．

   $n < 2d_{max}(V)$ の場合，n_r が小さいほどハミング距離は大きくなる．$n_r = 1$ を代入すると，ハミング距離は $(n - 2d_{max}(V)) + n_e d_{max}(V) < n_e d_{max}(V)$ となる．

   $n = 2d_{max}(V)$ の場合，n_r に依存せず，ハミング距離は $n_e d_{max}(V)$ となる．

以上の議論から，拡張符号 $E(V, n_e)$ の符号語同士の最大ハミング距離は式 (3.2) で与えられる．

$$d_{max}(E(V, n_e)) = \max\{n_e d_{max}(V),\, n \times \lfloor n_e/2 \rfloor,\, n_r(n_e - 2d_{max}(V)) + n_e d_{max}(V)\} \quad (3.2)$$

3.3.2 最小ハミング距離

拡張符号 $E(V, n_e)$ の最小ハミング距離を比較する．符号 V の符号語同士の最小ハミング距離を $d_{min}(V)$ とする．ハミング距離を 3 パターンに分けて考える．

1. 拡張符号の 2 つの符号語 $X_i^k, X_j^k\,(i \neq j)$ を考える．X_i^k，X_j^k はともにグループ G_k に属すため，反転箇所の個数は $n_r = 0$ である．m が小さいほどハミング距離は小さくなるため，式 (3.1) に $m = d_{min}(V)$ を代入すると，$d_H(X_i^k, X_j^k) = n_r(n - 2d_{min}(V)) + n_e d_{min}(V) = n_e d_{min}(V)$ となる．

2. 拡張符号の 2 つの符号語 $X_i^k, X_i^l\,(k \neq l)$ を考える．X_i^k，X_i^l はともに同じインデックスを持ち，符号語 v_i あるいは $\overline{v_i}$ を連接したものである．$d_H(v_i, v_i) = 0$ であるため，式 (3.1) に $m = 0$ を代入すると，$d_H(X_i^k, X_i^l) = n_r(n - 2m) + n_e m = n_r n$ となる．G_k と G_l は違うグループであるため，n_r が最も小さくなる場合は $n_r = 1$ のときで，ハミング距離は $d_H(X_i^k, X_j^k) = n$ となる．

3. 拡張符号の 2 つの符号語 $X_i^k, X_j^l\,(i \neq j, k \neq l)$ を考える．式 (3.1) から，$d_H(X_i^k, X_j^l) = n_r(n - 2m) + n_e m$ となる．$n_e m \geq 2n_r m$ であるため，m が小さいほどハミング距離は小さくなる．$m = d_{min}(V)$ を代入すると，$n_r(n - 2d_{min}(V)) + n_e d_{min}(V)$ となる．

   $n > 2d_{min}(V)$ の場合，n_r が小さいほどハミング距離は小さくなる．$n_r = 1$ を代入すると，ハミング距離は $(n - 2d_{min}(V)) + n_e d_{min}(V) > n_e d_{min}(V)$ となる．

   $n < 2d_{min}(V)$ の場合，n_r が大きいほどハミング距離は小さくなる．$n_r = \lfloor n_e/2 \rfloor$ を代入すると，ハミング距離は $\lfloor n_e/2 \rfloor (n - 2d_{min}(V)) + n_e d_{min}(V)$ となる．ハミング距離は $n_e = 1$ のとき，$d_{min}(V)$ となり，$n_e > 1$ のとき，n 以上となる．

   $n = 2d_{min}(V)$ の場合，n_r に依存せず，ハミング距離は $n_e d_{min}(V)$ となる．

以上の議論から，拡張符号 $E(V, n_e)$ の符号語同士の最小ハミング距離は式 (3.3) で与えられる．

$$d_{min}(E(V, n_e)) = \min\{n_e d_{min}(V), n\} \tag{3.3}$$

3.4 ドーナツ符号を元にした符号拡張手法の適用

3.2 節で提案した符号長 $n = 2o + 1$ のドーナツ符号 $DN(o)$ の最大ハミング距離，最小ハミング距離はそれぞれ $d_{max}(DN(o)) = 2o$，$d_{min}(DN(o)) = 2$ となる．ドーナツ符号 $DN(o)$ を元に符号拡張手法を適用することで，最大ハミング距離と最小ハミング距離を現実的な制約とすることを考える．この符号を拡張ドーナツ符号 $E(DN(o), n_e)$ と呼ぶ．拡張ドーナツ符号の符号語同士の最大ハミング距離と最小ハミング距離を考える．

3.4.1 最大ハミング距離と最小ハミング距離

式 (3.2) に $n = 2o + 1$，最大ハミング距離 $d_{max}(V) = 2o$ を代入すると，拡張ドーナツ符号 $E(DN(o), n_e)$ の最大ハミング距離 $d_{max}(E(DN(o), n_e))$ を得る．

$$d_{max}(E(DN(o), n_e)) = \max\left\{2on_e, \left\lfloor \frac{n_e}{2} \right\rfloor (1 - 2o) + 2on_e\right\} \tag{3.4}$$

$1 - 2o < 0$ なので，$2on_e > \lfloor n_e/2 \rfloor (1 - 2o) + 2on_e$ である．

従って，最大ハミング距離は以下になる．

$$d_{max}(E(DN(o), n_e)) = 2on_e \tag{3.5}$$

式 (3.3) に $n = 2o + 1$，最小ハミング距離 $d_{min}(V) = 2$ を代入すると，拡張ドーナツ符号 $E(DN(o), n_e)$ の最小ハミング距離 $d_{min}(E(DN(o), n_e))$ を得る．

$$d_{min}(E(DN(o), n_e)) = \min\{2n_e, 2o + 1\} \tag{3.6}$$

以上から，拡張ドーナツ符号の性質を表 3.4 にまとめる．

表 3.4: 拡張ドーナツ符号の性質.

符号長	$n_e(2o+1)$
情報量	$2o+n_e-1$
符号語数	2^{2o+n_e-1}
最大ハミング距離	$2on_e$
最小ハミング距離	$\min\{2n_e, 2o+1\}$

3.4.2 最適なパラメータ

拡張ドーナツ符号 $E(DN(o), n_e)$ の符号語同士の最大ハミング距離と最小ハミング距離の組み合わせは次式となる.

$$\begin{cases} d_{max}(E(DN(o), n_e)) = 2on_e \\ d_{min}(E(DN(o), n_e)) = 2n_e \end{cases} \tag{3.7}$$

$$\begin{cases} d_{max}(E(DN(o), n_e)) = 2on_e \\ d_{min}(E(DN(o), n_e)) = 2o+1 \end{cases} \tag{3.8}$$

一般に最小ハミング距離が $2t+1$ のとき t ビットの誤りが訂正できるため，$d_{min}(E(DN(o), n_e)) = 2o+1$ のとき，拡張ドーナツ符号 $E(DN(o), n_e)$ は o ビットの誤りが訂正できる．しかし，式 (3.7) では $d_{min}(E(DN(o), n_e)) = 2n_e$ であるため，n_e-1 ビットしか誤りを訂正できない．最小ハミング距離は $2t+1$ とすると無駄がないため，$o < n_e$ のときに条件を満たす式 (3.8) が適当である.

例 3.3. 符号長 3 のドーナツ符号 $DN(1)$ を，拡張数 $n_e = 3$ によって符号拡張した拡張ドーナツ符号 $E(DN(1), 3)$ の，値と符号語の関係[2] を表 3.5 に，性質を表 3.6 に示す．符号長 9 ビットの中で最大ハミング距離が 6 ビットであるため，最大書き込みビット数が削減されており，最小ハミング距離が 3 であることから 1 ビットの誤り訂正能力を持つ．これにより，最大ハミング距離と最小ハミング距離を制約した符号が生成されていることが分かる．拡張ドーナツ符号を用いてメモリを冗長化した場合，メモリに生じた値の誤りは訂正可能であり，書き込みビット数は削減できる．また，もとの値のすべてのビットが反転したときに対応する符

[2] それぞれの値と符号語の対応は他のパターンも考えられる．ここでは，値に対して昇順でグループと符号語を対応付けている.

表 3.5: 拡張ドーナツ符号 $E(DN(1), 3)$.

値	符号語			値	符号語		
	$DN(1)$	$DN(1)$	$DN(1)$		$DN(1)$	$\overline{DN(1)}$	$DN(1)$
0000	000	000	000	1000	000	111	000
0001	011	011	011	1001	011	100	011
0010	101	101	101	1010	101	010	101
0011	110	110	110	1011	110	001	110
値	符号語			値	符号語		
	$DN(1)$	$DN(1)$	$\overline{DN(1)}$		$\overline{DN(1)}$	$DN(1)$	$DN(1)$
0100	000	000	111	1100	111	000	000
0101	011	011	100	1101	100	011	011
0110	101	101	010	1110	010	101	101
0111	110	110	001	1111	001	110	110

表 3.6: 拡張ドーナツ符号 $E(DN(1), 3)$ の性質.

符号長	9
情報量	4
最大ハミング距離	6
最小ハミング距離	3
最悪書き込みビット数	5

号語同士のハミング距離を表す最悪書き込みビット数は 5 ビットに制約できる.

3.5 エンコーダ/デコーダの設計

拡張ドーナツ符号は線形符号ではないため，演算によるエンコード/デコードができない. そこで，拡張ドーナツ符号のエンコーダ/デコーダはルックアップテーブルで実装する.

エンコーダでは 4 ビットの値を入力し，9 ビットの符号語を出力する．4 ビットの入力パターンは $2^4 = 16$ 個あるため，ルックアップテーブルのサイズは 9 ビット $\times 16 = 144$ ビットとなる.

デコーダは 9 ビットの符号語を入力し，4 ビットの値を出力する．デコーダでは誤り訂正を考慮する必要がある．線形符号で誤り訂正を実現する際にはシンドロームを計算し，エ

表 3.7: 実験に用いたハミング符号.

値	符号語	値	符号語
0000	0000000	1000	1000011
0001	0001111	1001	1001100
0010	0010110	1010	1010101
0011	0011001	1011	1011010
0100	0100101	1100	1100110
0101	0101010	1101	1101001
0110	0110011	1110	1110000
0111	0111100	1111	1111111

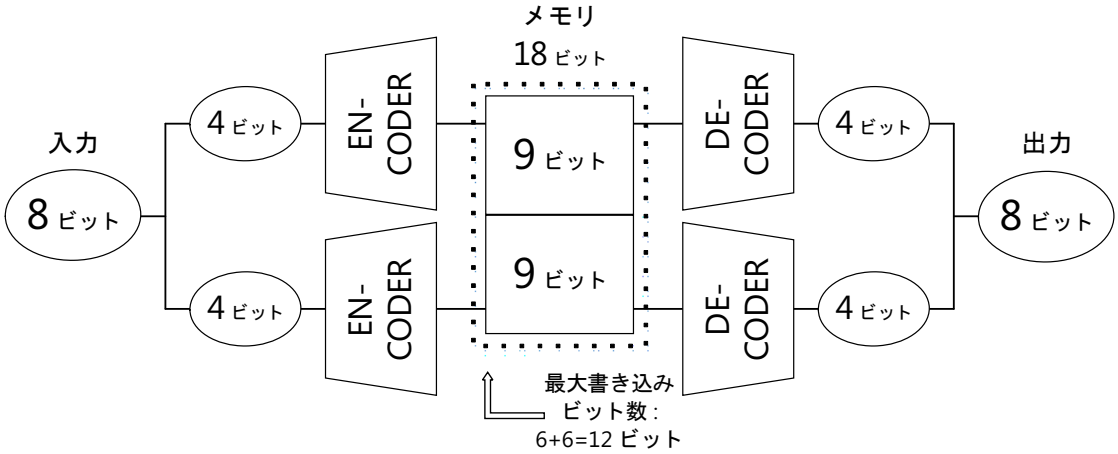
表 3.8: 実験に用いた符号の性質.

	拡張ドーナツ符号	ハミング符号
符号長	9	7
情報量	4	4
最大ハミング距離	6	7
最小ハミング距離	3	3
最悪書き込みビット数	5	7

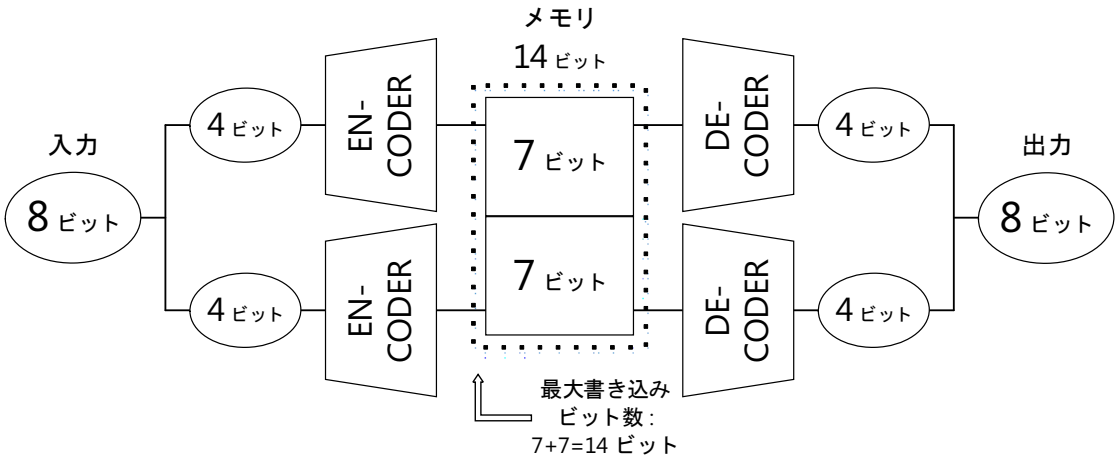
ラーベクトルを求める必要がある．しかし，拡張ドーナツ符号は線形符号ではないため，シンドロームの計算を行うことができない．そのため，訂正する必要のない符号語と訂正が可能な符号語全てのパターンをルックアップテーブルで実現する必要がある．符号語の個数は $2^4 = 16$ 個である．それぞれの符号語に対して誤りパターンを考える．符号語に 0 ビットの誤りが生じている (=誤りが生じていない) パターンは符号長が 9 ビットであるため ${}_9C_0 = 1$ パターンある．符号語に 1 ビットの誤りが生じているパターンは ${}_9C_1 = 9$ パターンある．符号語に 2 ビット以上の誤りが生じている場合は考慮しない．以上より，デコーダのルックアップテーブルのサイズは 4 ビット $\times (16 \times ({}_9C_0 +_9 C_1)) = 640$ ビットとなる．

3.6 評価実験

本節では，前節で提案した拡張ドーナツ符号を評価し，ハミング符号と比較する．表 3.5 に拡張ドーナツ符号 $E(DN(1), 3)$ の値と符号語の関係を示す．表 3.7 に (7, 4, 3) ハミング符



(a) 拡張ドーナツ符号



(b) ハミング符号

図 3.4: メモリアーキテクチャ.

号の値と符号語の関係を示す．2 つの符号の性質を表 3.8 に示す．実験に用いるメモリを図 3.4 に示す．メモリに値を書き込むとき，入力する値を符号語にエンコードしてから書き込む．メモリへの書き込みは 8 ビットワードごとに行われるため，実験に用いた符号は上位 4 ビットと下位 4 ビットに分けてエンコードする．このとき，拡張ドーナツ符号の場合は 4 ビットの値が 9 ビットの符号語にエンコードされ，ハミング符号の場合は 4 ビットの値が 7 ビットの符号語にエンコードされる．メモリは MRAM であると想定し，手法 [51] にならい，同

じビットの値は書き込まないものとする．メモリのすべてのビットの初期値は 0 とする．

3.6.1 項では，拡張ドーナツ符号を用いて構成した不揮発メモリを計算機上で実装し，書き込みビット数を評価する．3.6.2 項では，拡張ドーナツ符号を用いて構成した不揮発メモリの回路を実装し，エンコーダ/デコーダならびに不揮発メモリの面積とエネルギーを評価する．

3.6.1 書き込みビット数評価実験

1 ビットの誤り訂正可能な符号である拡張ドーナツ符号 $E(DN(1), 3)$ と (7, 4, 3) ハミング符号について，書き込みビット数の比較を行う計算機実験を行った．

2 つの符号について，以下の 11 パターンの 8 ビット入力データをエンコードしてメモリに 65,536 回書き込みを行った際の拡張ドーナツ符号とハミング符号の書き込みビット数を計測した．

Flip1: 8 ビットの入力データのうち毎回ランダムに 1 ビットが反転する入力データ

Flip2: 8 ビットの入力データのうち毎回ランダムに 2 ビットが反転する入力データ

Flip3: 8 ビットの入力データのうち毎回ランダムに 3 ビットが反転する入力データ

Flip4: 8 ビットの入力データのうち毎回ランダムに 4 ビットが反転する入力データ

Flip5: 8 ビットの入力データのうち毎回ランダムに 5 ビットが反転する入力データ

Flip6: 8 ビットの入力データのうち毎回ランダムに 6 ビットが反転する入力データ

Flip7: 8 ビットの入力データのうち毎回ランダムに 7 ビットが反転する入力データ

Flip8: 8 ビットの入力データのうち毎回 8 ビット（全てのビット）が反転する入力データ

Mixed_Flip1toFlip8: 8 ビットの入力データのうち毎回 1 ビットから 8 ビットがそれぞれ等確率で反転する入力データ

Mixed_Flip5toFlip8: 8 ビットの入力データのうち毎回 5 ビットから 8 ビットがそれぞれ等確率で反転する入力データ

Mixed_Flip7toFlip8: 8 ビットの入力データのうち毎回 7 ビットから 8 ビットがそれぞれ等確率で反転する入力データ

表 3.9: メモリ書き込みの遷移例.

		拡張ドーナツ符号		ハミング符号	
入力データ	Flip 数	符号語	書き込みビット数	符号語	書き込みビット数
0000 0000	-	000000000 000000000	-	0000000 0000000	-
0100 1000	Flip2	000000111 000111000	6	0100101 1000011	6
1011 0111	Flip8	110001110 110110001	8	1011010 0111100	14

Flip8 において書き込みビット数は初期の入力データに依存するため，8 ビットの全てのパターンを初期の入力データとして与え，その平均を使用した．なお，Flip1 から Flip8 は 2 つの符号の性質を評価するために用いた極端な入力データである．

以下にメモリ書き込みにおける Flip 数と書き込みビット数を例示し，それをまとめたものを表 3.9 に示す．拡張ドーナツ符号を用いて構成したメモリに初期値として入力データ 0000 0000 を与えると，符号語 000000000 000000000 にエンコードされメモリに書き込まれる．同様にハミング符号を用いて構成したメモリに初期値として入力データ 0000 0000 を与えると，符号語 0000000 0000000 にエンコードされメモリに書き込まれる．2 つのメモリに入力データ 0100 1000 を与えると，拡張ドーナツ符号を用いたメモリでは符号語 000000111 000111000，ハミング符号を用いたメモリでは符号語 0100101 1000011 にそれぞれエンコードされメモリに書き込まれる．入力データの反転ビット数 (Flip 数) は入力データの下線ビットの個数に対応し，2 ビットとなる．つまりこの入力データは Flip2 に相当する．実際にメモリに書き込まれた際に反転するビット数 (書き込みビット数) は符号語の下線ビットの個数に対応し，拡張ドーナツ符号・ハミング符号を用いたメモリではどちらも 6 ビットとなる．続いて入力データ 1011 0111 を与えると，拡張ドーナツ符号を用いたメモリでは符号語 110001110 110110001，ハミング符号を用いたメモリでは符号語 1011010 0111100 にそれぞれエンコードされメモリに書き込まれる．入力データの反転ビット数 (Flip 数) は入力データの下線ビットの個数に対応し，8 ビットとなる．つまりこの入力データは Flip8 に相当する．実際にメモリに書き込まれた際に反転するビット数 (書き込みビット数) は符号語の下線ビットの個数に対応し，拡張ドーナツ符号を用いたメモリでは 8 ビット，ハミング符号を用いたメモリでは 14 ビットとなる．

拡張ドーナツ符号を用いて構成したメモリと，ハミング符号を用いて構成したメモリの書き込みビット数の比較を表 3.10 に示す．反転ビット数が多くなった場合に，拡張ドーナツ符号は、

表 3.10: 書き込みビット数: B_w.

	拡張ドーナツ符号 [ビット]	ハミング符号 [ビット]
Flip1	330,584 (1.55)	213,768 (1.00)
Flip2	511,572 (1.48)	344,568 (1.00)
Flip3	590,096 (1.41)	416,564 (1.00)
Flip4	600,572 (1.30)	459,412 (1.00)
Flip5	610,800 (1.18)	515,616 (1.00)
Flip6	576,144 (0.98)	586,840 (1.00)
Flip7	588,432 (0.83)	706,240 (1.00)
Flip8	589,824 (0.64)	917,504 (1.00)
Mixed_Flip1toFlip8	514,288 (1.03)	498,752 (1.00)
Mixed_Flip5toFlip8	590,234 (0.87)	675,360 (1.00)
Mixed_Flip7toFlip8	589,824 (0.73)	806,432 (1.00)
adpcmE (MediaBench)	2,358,295 (1.15)	2,047,028 (1.00)
g721E (MediaBench)	2,481,184 (1.16)	2,129,981 (1.00)
RC4 (Stream Cipher)	590,080 (0.83)	709,888 (1.00)
Blowfish (Block Cipher)	564,096 (0.94)	597,632 (1.00)

ハミング符号よりも高い性能を示す．最悪の場合である Flip8 では，拡張ドーナツ符号はハミング符号に比べて 35.7%の書き込みビット数を削減した．Mixed_Flip1toFlip8 では，拡張ドーナツ符号はハミング符号とほぼ同じ性能を示す．Mixed_Flip5toFlip8，Mixed_Flip7toFlip8 では，拡張ドーナツ符号はハミング符号に比べてそれぞれ 12.6%，26.9%の書き込みビット数を削減した．拡張ドーナツ符号は入力データが Flip6–Flip8 に近似するアプリケーションに対して有効で，ハミング符号は入力データが Flip1–Flip5 に近似するアプリケーションに対して有効である．

さらに，拡張ドーナツ符号とハミング符号を用いて構成したメモリでマルチメディア系アプリケーションと暗号系アプリケーションを動作させた際の書き込みビット数を計測した．マルチメディア系の音声や画像を生成するベンチマークである MediaBench [26] を SimpleScalar [1] を用いてメモリへの読み書きのトレースを取得した．MediaBench に入力するデータは [26] と同じデータを用いた．暗号系アプリケーション [18, 38] は，平文を暗号化した暗号文を書き込む．平文はランダムに生成した 65,536 個のデータを使用した．対象となる誤り訂正符号を用いて構成したメモリをトレースに適用して書き込みビット数を計測した．

実験結果より，拡張ドーナツ符号はハミング符号と比較してマルチメディア系アプリケーションに対して書き込みビット数が 16.5%増加したが，暗号系アプリケーションに対して書き込みビット数を 16.9%削減した．

$B_w(EDN)$，$B_w(Hamming)$ をそれぞれ拡張ドーナツ符号とハミング符号を用いたメモリでアプリケーションを動作させたときの総書き込みビット数とする．各アプリケーションが Flip1–Flip8 のどの Flip 数に近似するかを検証するための指標 R_f を

$$R_f = B_w(EDN)/B_w(Hamming) \tag{3.9}$$

とする．R_f は表 3.10 の括弧内の数値が対応する．マルチメディア系アプリケーションの R_f は表 3.10 より，Flip5 に近似する．暗号系アプリケーションの R_f は表 3.10 より，Flip6–Flip7 に近似する．拡張ドーナツ符号を用いて構成したメモリにマルチメディア系アプリケーションを適用した場合，マルチメディア系アプリケーションは Flip 数が小さい入力データが多いため性能が悪く，暗号系アプリケーションを適用した場合，暗号系アプリケーションは Flip 数が大きい入力データが多いため性能が良くなると考えられる．

以上より，暗号アルゴリズムの中には高い反転ビット数の割合を持つアルゴリズムがあり，拡張ドーナツ符号はそのようなメモリの書き換えが頻繁に発生するアプリケーションに対して効果的な手法である．

3.6.2 メモリ評価実験

1 ビットの誤り訂正可能な符号である拡張ドーナツ符号 $E(DN(1), 3)$ と (7, 4, 3) ハミング符号を用いて構成した不揮発メモリの回路を実装し，エンコーダ/デコーダならびに不揮発メモリの面積とエネルギーを評価した．回路はハードウェア記述言語 Verilog HDL で記述し，Synopsys 社の Design Compiler のトポグラフィカルモードで論理合成した．拡張ドーナツ符号は前節で議論したエンコーダ/デコーダを用いる．エンコーダ/デコーダ内のルックアップテーブルは MRAM を仮定し，あらかじめデータが保存されているとする．MRAM の 1 ビットあたりの面積は $6.17 \times 10^{-1} um^2$ とする [10]．ルックアップテーブルのセル数を表 3.11 に示す．論理合成したエンコーダ本体とルックアップテーブル部分の面積を表 3.12 に示す．拡

表 3.11: ルックアップテーブルのセル数.

	エンコーダ	デコーダ
(7, 4, 3) ハミング符号	0	32
拡張ドーナツ符号 $E(DN(1), 3)$	144	640

表 3.12: 面積評価結果.

	エンコーダ $[um^2]$		デコーダ $[um^2]$		合計 $[um^2]$
	本体	LUT	本体	LUT	
(7, 4, 3) ハミング符号	27.5	-	2.1	19.7	49.4
拡張ドーナツ符号 $E(DN(1), 3)$	-	88.8	-	394.9	483.7

張ドーナツ符号を用いて構成したメモリはハミング符号を用いて構成したメモリと比較して面積は 9.8 倍となった.

次に回路のエネルギー評価を示す．メモリは動作周波数が 100MHz の MRAM を想定した．MRAM の 1 ビットあたりのリードエネルギー E_r は 1.06×10^{-15}J/bit，ライトエネルギー E_w は 2.60×10^{-12}J/bit とする [10]．N_w ，N_r，B_w をそれぞれメモリへの書き込みアクセス回数，読み込みアクセス回数，書き込みビット数とする．N_w，N_r は SimpleScalar [1] を用いて取得した．N_w，N_r を表 3.13 に示す．B_w は総書き込みビット数を表し，表 3.10 から得られる．L_c を拡張ドーナツ符号およびハミング符号の符号長とする.

エネルギー消費 E_t は式 (3.10) で与えられる.

$$E_t = \{N_r \times (L_c \times 2) \times E_r\} + (B_w \times E_w) + E_o \tag{3.10}$$

初項は，メモリからデータを読み出すエネルギーを示す．第二項は，メモリへの書き込みエネルギーを示す．E_o は，エンコーダ/デコーダのエネルギーオーバヘッドを示す．ドーナツ符号のオーバーヘッドは式 (3.11) で，ハミング符号のオーバーヘッドは式 (3.12) でそれぞれ表される.

$$E_o = 2 \times N_w \times L_c \times E_r + 2 \times N_r \times k \times E_r \tag{3.11}$$

表 3.13: メモリへの書き込みアクセス回数 N_w と読み込みアクセス回数 N_r.

	書き込みアクセス回数: N_w	読み込みアクセス回数: N_r
Flip1	65,536	0
Flip2	65,536	0
Flip3	65,536	0
Flip4	65,536	0
Flip5	65,536	0
Flip6	65,536	0
Flip7	65,536	0
Flip8	65,536	0
Mixed_Flip1toFlip8	65,536	0
Mixed_Flip5toFlip8	65,536	0
Mixed_Flip7toFlip8	65,536	0
adpcmE	450,692	147,756
g721E	482,019	162,925
RC4	65,536	65,536
Blowfish	65,536	65,536

$$
\begin{aligned}
E_o &= 2 \times N_w \times (d_E \times P_{dynamic,E} + P_{leakage,E}/100\text{MHz}) \\
&+ 2 \times N_r \times (d_D \times P_{dynamic,D} + P_{leakage,D}/100\text{MHz}) \\
&+ 2 \times N_r \times k \times E_r \qquad (3.12)
\end{aligned}
$$

式 (3.11) は，ドーナツ符号のエンコード/デコード時にルックアップテーブルからデータを読み出すエネルギーを表す．式 (3.12) での，d_E と d_D はエンコーダとデコーダのクリティカルパス遅延，$P_{dynamic,E}$ と $P_{dynamic,D}$ はエンコーダとデコーダのダイナミック電力，$P_{leakage,E}$ と $P_{leakage,D}$ はエンコーダとデコーダのリーク電力を表す．$d_E \times P_{dynamic,E}$，$d_D \times P_{dynamic,D}$ はそれぞれエンコーダ/デコーダのダイナミックエネルギー，$P_{leakage,E}$/100MHz，$P_{leakage,D}$/100MHz はそれぞれエンコーダ/デコーダのリークエネルギーを表す [6]．第三項はルックアップテーブルからエラーベクトルを読み出すためのエネルギーを表す．表 3.14 に各符号を用いて構成したメモリのエンコーダ/デコーダのクリティカルパス遅延，ダイナミック電力，リーク電力を示す．

拡張ドーナツ符号を用いて構成したメモリとハミング符号を用いて構成したメモリのエネ

表 3.14: 4 ビットの値に対するエンコーダおよびデコーダのエネルギー.

	拡張ドーナツ符号		ハミング符号	
	エンコーダ	デコーダ	エンコーダ	デコーダ
クリティカルパス遅延 [ns]	-	-	0.57	0.29
ダイナミック電力 [μW]	-	-	8.20	0.31
リーク電力 [μW]	-	-	0.23	0.018

表 3.15: エネルギー評価実験結果.

	拡張ドーナツ符号 [nJ]				ハミング符号 [nJ]			
	リード	ライト	オーバーヘッド	合計	リード	ライト	オーバーヘッド	合計
Flip1	0.0	859.5	1.3	860.8 (1.54)	0.0	555.8	0.9	556.7 (1.00)
Flip2	0.0	1330.1	1.3	1331.3 (1.48)	0.0	895.9	0.9	896.8 (1.00)
Flip3	0.0	1534.2	1.3	1535.5 (1.41)	0.0	1083.1	0.9	1084.0 (1.00)
Flip4	0.0	1561.5	1.3	1562.7 (1.30)	0.0	1194.5	0.9	1195.4 (1.00)
Flip5	0.0	1588.1	1.3	1589.3 (1.18)	0.0	1340.6	0.9	1341.5 (1.00)
Flip6	0.0	1498.0	1.3	1499.2 (0.98)	0.0	1525.8	0.9	1526.7 (1.00)
Flip7	0.0	1529.9	1.3	1531.2 (0.83)	0.0	1836.2	0.9	1837.1 (1.00)
Flip8	0.0	1533.5	1.3	1534.8 (0.64)	0.0	2385.5	0.9	2386.4 (1.00)
Mixed_Flip1toFlip8	0.0	1337.1	1.3	1338.4 (1.03)	0.0	1296.8	0.9	1297.7 (1.00)
Mixed_Flip5toFlip8	0.0	1534.6	1.3	1535.9 (0.87)	0.0	1755.9	0.9	1756.8 (1.00)
Mixed_Flip7toFlip8	0.0	1533.5	1.3	1534.8 (0.73)	0.0	2096.7	0.9	2097.6 (1.00)
adpcmE	1.4	6131.6	9.9	6142.8 (1.15)	1.1	5322.3	7.0	5330.3 (1.00)
g721E	1.6	6451.1	10.6	6463.2 (1.16)	1.2	5538.0	7.5	5546.6 (1.00)
RC4	0.6	1534.2	1.8	1536.6 (0.83)	0.5	1845.7	1.5	1847.7 (1.00)
Blowfish	0.6	1466.6	1.8	1469.1 (0.94)	0.5	1553.8	1.5	1555.8 (1.00)

ルギー評価実験の結果を表 3.15 に示す．Flip1–Flip8 より，反転ビット数が多くなるにつれて拡張ドーナツ符号はハミング符号よりも高い性能を示す．特に，最悪の場合である Flip8 のとき拡張ドーナツ符号はハミング符号と比較して 35.7%のエネルギーを削減した．MediaBench では，拡張ドーナツ符号はハミング符号よりエネルギーが増加するが，暗号のアプリケーションでは拡張ドーナツ符号はハミング符号よりも高い性能を示す．

実験結果より，不揮発メモリには入力データの反転ビット数に応じた最適なメモリエンコーディングを用いる必要があることがわかる．拡張ドーナツ符号は入力データの反転ビット数が大きい場合に効果的であり，ハミング符号は入力データの反転ビット数が小さい場合に効果的である．言い換えると，拡張ドーナツ符号は入力データが Flip6–Flip8 に近似するアプリケーションのときに効果的で，ハミング符号は入力データが Flip1–Flip5 に近似する

アプリケーションのときに効果的である．

3.7 本章のまとめ

本章では，不揮発メモリを対象とした最悪書き込みビット数削減と誤り訂正を同時に実現する手法について述べた．

まず，符号語間の最大ハミング距離と最小ハミング距離を制約した符号としてドーナツ符号を提案した．ドーナツ符号は十分な書き込みビット数削減と誤り訂正能力を有さないため，続いて符号拡張手法を提案した．ドーナツ符号を元に符号拡張手法を適用した拡張ドーナツ符号は最悪書き込みビット数を削減でき，メモリに生じた誤りを訂正できる．

拡張ドーナツ符号を用いた不揮発メモリを計算機上に実装し，メモリへの書き込みビット数を計測した．また，エンコーダ/デコーダ，並びに不揮発メモリの面積とエネルギーを評価した．拡張ドーナツ符号を用いて構成した不揮発メモリはハミング符号を用いて構成した不揮発メモリと比較して，エネルギーを最大 35.7%削減した．拡張ドーナツ符号は入力データの反転ビット数が大きい場合に有効であることを確認した．

第4章　一対多符号

4.1　本章の概要

本章では，値と符号語を一対多に割り当てた符号の構成手法を提案する[1]．提案手法を用いて生成した一対多符号は最悪書き込みビット数を最小化し，誤り訂正能力を維持したまま最大誤り訂正能力が増加する．一対多符号を用いてメモリを構成したとき，もとの誤り訂正符号と比較してエネルギーを最大56.8%削減した．

一対多符号ではメモリに書き込まれている符号語を考慮して値をエンコードするコンテクスト依存符号化のメモリアーキテクチャを用いる．

4.2　一対多符号生成手法

本節では，符号長7ビット，情報量4ビット，最小ハミング距離3ビットである (7, 4, 3) ハミング符号 [14] を取り上げて提案手法を説明する．(7, 4, 3) ハミング符号の情報量は4ビットであり，値と符号語は表4.1に示すように1対1に対応する．(7, 4, 3) ハミング符号は組織符号であるため，符号語の上位4ビットが値を表す [35]．また，符号語同士の最小ハミング距離が3ビットであるため，誤り訂正能力は1ビットである．図4.1に (7, 4, 3) ハミング符号の符号空間の概念図を示す．16個の各円はそれぞれ符号語を表し，円に隣接する長方形は符号語に対応する値を表す．(7, 4, 3) ハミング符号の符号語同士のハミング距離を考える．符号語0000000と他の符号語とのハミング距離は2円を接続する線分の傍の数字が表している．(7, 4, 3) ハミング符号では符号語0000000とハミング距離が0ビットの符号語が1つ（自身への遷移），符号語0000000とハミング距離が3ビットの符号語が6つ，4ビットの符号語が8つ，7ビットの符号語が1つある．ハミング符号を用いて値を符号化したとき，任意の

[1]本章の内容は [56,57] による．

表 4.1: (7, 4, 3) ハミング符号の値と符号語の関係.

値	符号語	値	符号語
0000	0000000	1000	1000011
0001	0001111	1001	1001100
0010	0010110	1010	1010101
0011	0011001	1011	1011010
0100	0100101	1100	1100110
0101	0101010	1101	1101001
0110	0110011	1110	1110000
0111	0111100	1111	1111111

値の遷移には平均的に 3.5 ビット，最悪の場合では 7 ビットの書き換えが必要である.

(7, 4, 3) ハミング符号を元に，値と符号語を一対多に対応させた書き込み削減・誤り訂正符号の生成手法を提案する．生成される符号は元の誤り訂正符号の符号長と最小ハミング距離を合わせた符号長となる．(7, 4, 3) ハミング符号を元にした場合，符号長は 10 ビットとなる．10 ビットのうち上位 7 ビットは (7, 4, 3) ハミング符号で構成されており，その 7 ビットを符号部と呼び，残りの 3 ビットを冗長部と呼ぶ.

Step 1: 符号部を元となる誤り訂正符号の符号語とする．つまり，(7, 4, 3) ハミング符号の 16 個の各符号語を符号部とする.

Step 2: 符号部に全て 0 のビット列と全て 1 のビット列の 2 種類の冗長部を付加する．そのため，1 つの符号部から 2 つの新たな符号語が生成される．冗長部のビット長は，元となる誤り訂正符号の最小ハミング距離と同じビット長とする．例えば (7, 4, 3) ハミング符号の符号語 0000000 を符号部とするとき，冗長部 000 と 111 を付加する．すると新たに符号語 0000000000 と 0000000111 が生成される．このようにして 16 個の符号語を符号部としたとき，新たに 32 個の符号語が生成できる.

Step 3: 新たに生成した 32 個の符号語にそれぞれ値を割り当てる．冗長部が 000 の符号語は符号部の上位 4 ビットを値とする．冗長部が 111 の符号語は符号部の上位 4 ビットを反転したビット列を値とする．例えば，値 0101 を表す符号語は 0101101000 と 1010010111 となる.

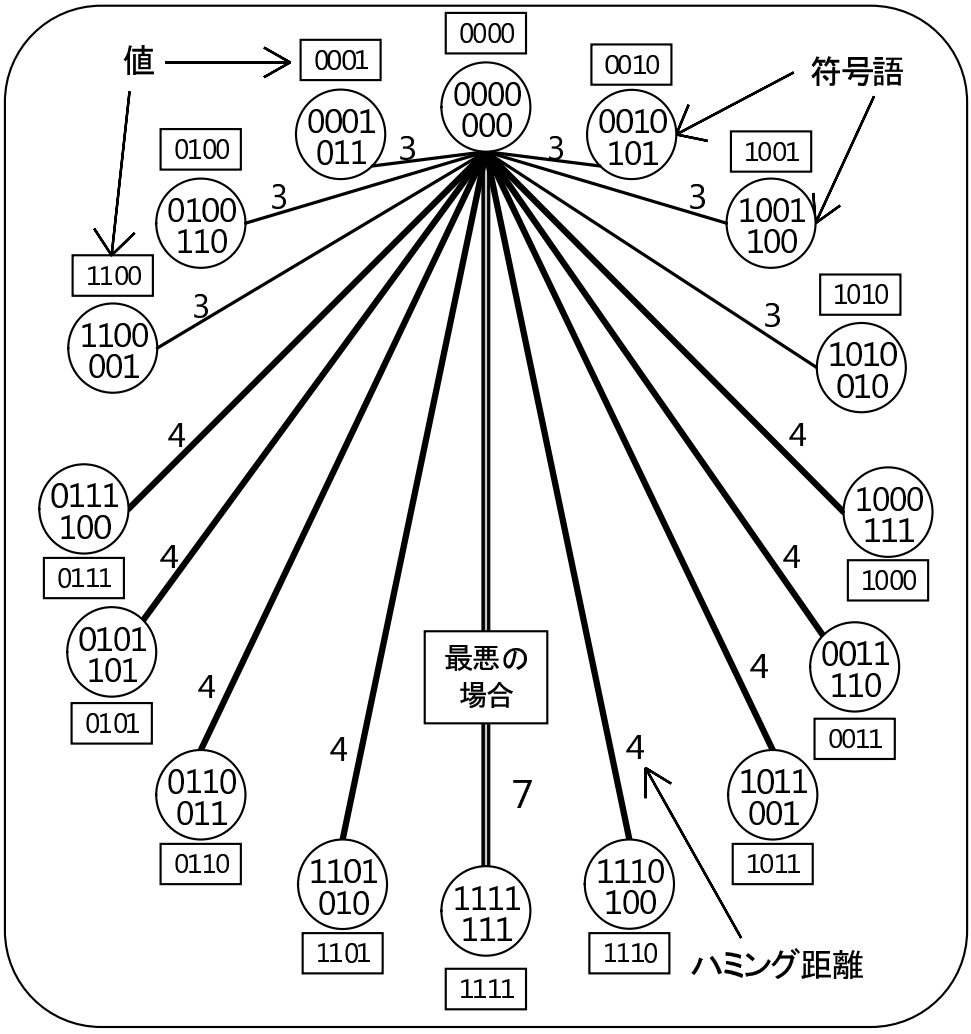


図 4.1: (7, 4, 3) ハミング符号の符号空間.

提案手法を用いて生成した符号を一対多符号と呼ぶ．(7, 4, 3) ハミング符号を元にして構成した一対多符号の値と符号語の関係を表 4.2 に示す．(7, 4, 3) ハミング符号以外の誤り訂正符号に対しても同様に一対多符号を構成することができる．

定理 4.1. t ビット誤り訂正符号に対して提案手法を用いて生成した一対多符号の最悪書き込みビット数は $2t+1$ ビットとなる．

証明 4.1. 値のすべてのビットが反転するような符号語の遷移の場合，メモリに保存された符号語と書き込む符号語の符号部は同一のビット列となり，ビット反転が発生するのは符号

表 4.2: (7, 4, 3) ハミング符号を元にして生成した一対多符号の値と符号語の関係.

値	符号語		値	符号語	
	符号部	冗長部		符号部	冗長部
0000	0000000	000	0000	1111111	111
0001	0001111	000	0001	1110000	111
0010	0010110	000	0010	1101001	111
0011	0011001	000	0011	1100110	111
0100	0100101	000	0100	1011010	111
0101	0101010	000	0101	1010101	111
0110	0110011	000	0110	1001100	111
0111	0111100	000	0111	1000011	111
1000	1000011	000	1000	0111100	111
1001	1001100	000	1001	0110011	111
1010	1010101	000	1010	0101010	111
1011	1011010	000	1011	0100101	111
1100	1100110	000	1100	0011001	111
1101	1101001	000	1101	0010110	111
1110	1110000	000	1110	0001111	111
1111	1111111	000	1111	0000000	111

語の冗長部のみである．冗長部のビット長は誤り訂正符号の最小ハミング距離と同じビット長であり，t ビット誤り訂正符号の最小ハミング距離は $2t+1$ であるため，冗長部のビット長は $2t+1$ ビットとなる．冗長部は全て 0 か全て 1 のビット列であるため，冗長部には冗長部のビット長分のビット反転が発生する．そのため，誤り訂正符号に対して提案手法を用いて生成した一対多符号の最悪書き込みビット数は $2t+1$ ビットとなる． □

例 4.1. 値が 0001 → 0010 → 1101 と遷移するときを考える．0001 を一対多符号の符号語にエンコードした 0001011000 がメモリに書き込まれているとする．値 0010 をメモリに保存する際に値 0010 は符号語 0010101000 と 110101010111 の 2 種類にエンコードできるが，メモリに書き込まれている符号語を考慮して値を 0010101000 にエンコードしてメモリに書き込む．次に，値 1101 をメモリに保存する際にメモリに書き込まれている符号語を考慮したとき，1101010000 にエンコードすると書き込みビット数は 7 ビット（下線部のビット）となるが，0010101111 にエンコードすることで書き込みビット数は 3 ビット（下線部のビット）に

表 4.3: 符号の性質.

	符号長	情報量	誤り訂正能力	最大書き込みビット数	最悪書き込みビット数	平均書き込みビット数
ハミング符号	7	4	1	7	7	3.50
拡張ドーナツ符号	9	4	1	6	5	4.50
一対多符号	10	4	1–2	4	3	3.25

制約できる.

情報量が 4 ビットの誤り訂正符号と (7, 4, 3) ハミング符号をもとに構成した一対多符号の性質の比較を表 4.3 に示す. 元の符号が t ビット誤り訂正符号である場合, 一対多符号では符号部と冗長部に独立して t ビットずつの誤りが生じても誤り訂正が可能である. つまり, 一対多符号の誤り訂正能力は誤りを持つビットの場所に依存するが, 最大 $2t$ ビットとなる. 最悪書き込みビット数はハミング符号では 7 ビット, 拡張ドーナツ符号では 5 ビットであるが, 一対多符号では最悪の場合の書き込みビット数は 3 ビットとなる[2] .

符号語の遷移には自身への遷移を除いて, 必ず最小ハミング距離 $2t+1$ ビット以上の書き込みビット数が発生するため, 書き込みビット数が最小ハミング距離と同じビット数となる場合, その書き込みビット数は最小値である.

定理 4.2. t ビット誤り訂正符号に提案手法を用いて生成した一対多符号は最悪書き込みビット数を最小化し, 元の誤り訂正符号の t ビット誤り訂正能力を維持したまま最大の誤り訂正能力は $2t$ ビットとなる.

証明 4.2. 上述より自明. □

(7, 4, 3) ハミング符号に対して提案手法を用いて構成した一対多符号の符号空間の概念図を図 4.2 に示す. 図 4.2 には上下 2 つの大きな円盤がある. 上の円盤は冗長部が 000 の一対多符号を表し, 下の円盤は冗長部が 111 の一対多符号を表す. 1 つの円盤にはある値を表す符号語が 1 つ存在する. もう 1 つの円盤にも同じ値を表す符号語が 1 つ存在するため, 各円盤

[2]表 4.3 に示すように, 最悪書き込みビット数が必ずしも書き込みビット数最大となるとは限らない. 例えば, 表 4.2 に示す提案符号において最悪書き込みビット数となる例として, 値 0000（符号語 0000000000）→ 値 1111（符号語 0000000111）の遷移があり, この場合の書き込みビット数は 3 ビットとなる. それに対して, 最大書き込みビット数となる例として, 値 0000（符号語 0000000000）→ 値 0001（符号語 0001111000）の遷移があり, この場合の書き込みビット数は 4 ビットとなる.

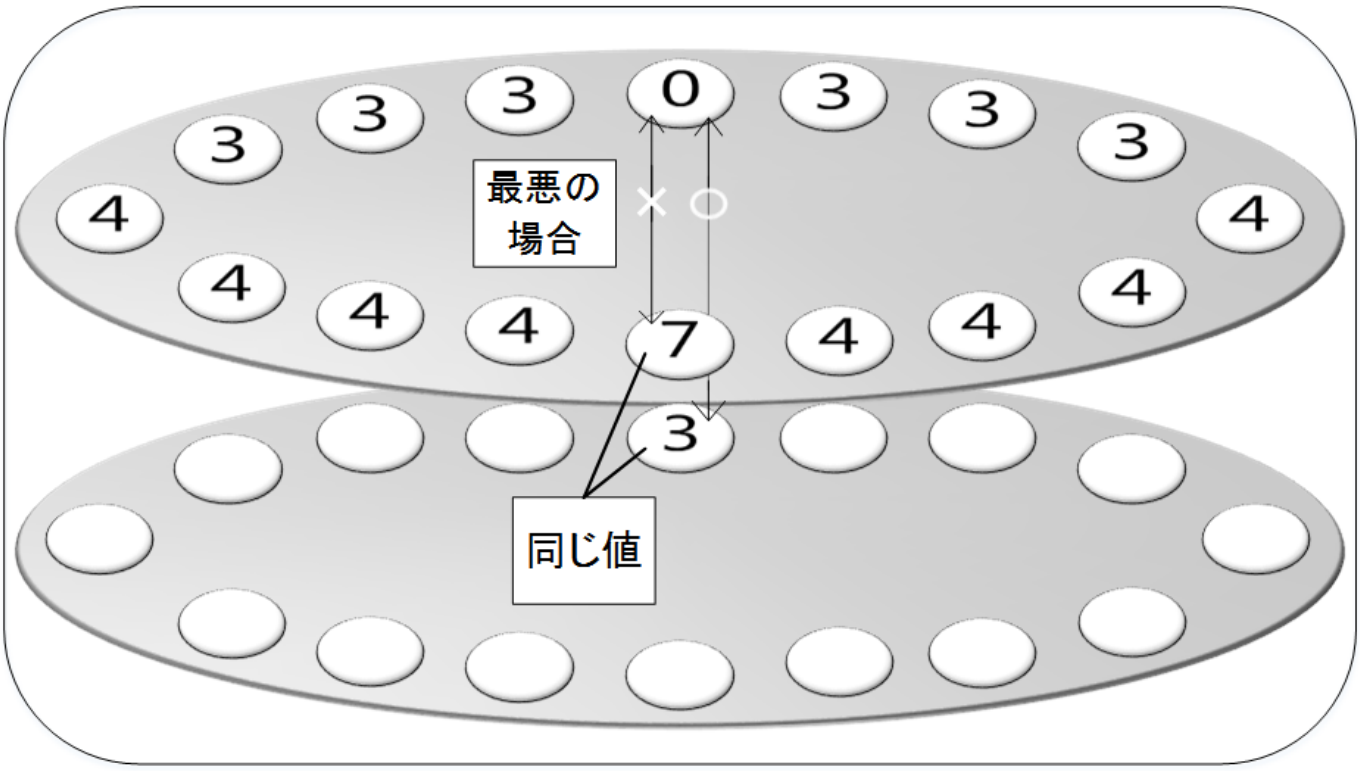


図 4.2: 一対多符号の符号空間.

には同じ値を表す符号語が 1 つずつ存在することとなる．つまり，この符号は値と符号語が 1 対 2 に対応した符号である．上の円盤のみでは最悪書き込みビット数は 7 ビットとなるが，下の円盤を用いると最悪の場合に上下の円盤を行き来するような書き込み遷移をすることで最悪書き込みビット数は 3 ビットに制約できる．最悪の場合以外は同じ円盤上で符号語の遷移を行い，最悪の場合は上下の円盤を行き来することで書き込みビット数が制約できる．

4.3 エンコーダ/デコーダの設計

本節では，一対多符号を用いて構成したメモリアーキテクチャのエンコーダとデコーダの設計方法を説明する．一対多符号を用いて構成したメモリでは，メモリに値 d を保存するとき，まずメモリに書き込まれている符号語 v_i を読み出す．読み出した符号語 v_i を基に値 d を符号語 v_o にエンコードする．メモリから値を取り出すときには，メモリに書き込まれてい

る符号語 v_i を読み出し，読み出した符号語をデコードすることで元の値 d を得ることができる．一対多符号の符号部は元となる誤り訂正符号，冗長部は反復符号 [12] と見なすことで，効率的にエンコーダ/デコーダ設計ができる．

4.3.1 エンコーダ

(7, 4, 3) ハミング符号を元にした一対多符号のエンコーダ設計を示す．エンコーダは入力としてメモリから読み出した符号語 v_i，保存したい値 d を与えると，書き込む符号語 v_o を出力する．エンコーダは以下のように動作する．

Step 1: 保存したい値 d をハミング符号のエンコーダを用いて符号語 v_H へエンコードする．

$$v_H = \mathrm{encode}(d) \tag{4.1}$$

Step 2: v_H に 3 ビットのビット列 000 を付加し，ビット列 v'_o とする．

$$v'_o = v_H[000] \tag{4.2}$$

Step 3: メモリから読み出した符号語 v_i の上位 4 ビットと 4 ビットの保存したい値 d の排他的論理和をとる．そのビット列が 1111 であれば v'_o を反転して出力し，1111 でなければ v'_o を出力する．符号長 10 ビットの符号語 v_i の上位 4 ビットを $v_i[9:6]$ と表す．

$$\text{if } v_i[9:6] \oplus d == 4\text{'b1111}:$$

$$v_o = \overline{v'_o} \tag{4.3}$$

$$\text{else}:$$

$$v_o = v'_o \tag{4.4}$$

以上の操作により 10 ビットの書き込む符号語 v_o が得られる．このエンコーダでは読み出す符号語の誤り訂正を考慮していないが，符号語に誤りが生じていてもエンコード可能である．

4.3.2 デコーダ

デコーダは入力として読みだした符号語を与えると，値 d を出力する．読み出した符号語には誤りが発生している可能性があるため，誤り訂正を考慮する．誤り箇所を表したビット列をエラーベクトル e，読み出した符号語を $v_i \oplus e$，出力する値を d とする．符号語に誤りが生じていない場合は $e = 0$ となる．

Step 1: 誤り訂正を行うためにエラーベクトル v_e を求める．符号語 v_i を構成する符号部 (ハミング符号) を v_H，冗長部 (反復符号) を v_R とする．エラーベクトル e を構成する符号部 (ハミング符号) を e_H，冗長部 (反復符号) を e_R とする．

Step (1-1): 符号部 (ハミング符号) の誤り訂正をする．

Step (1-1-1): 一般的な線形組織符号と同様の手順でハミング符号の生成行列 G から検査行列 H を求める．

Step (1-1-2): 読み出した符号語 $v_H \oplus e_H$ と検査行列 H からシンドローム s を求める．

$$s = (v_H \oplus e_H)H \tag{4.5}$$

ここで，符号語 v_H に誤りが生じていなければシンドローム s は

$$s = (v_H \oplus e_H)H = v_H H = 0 \tag{4.6}$$

となるため，

$$s = (v_H \oplus e_H)H = v_H H \oplus e_H H = e_H H \tag{4.7}$$

と表せる．つまり，シンドローム s はエラーベクトル e_H のみに依存する．s と e_H を 1 対 1 に対応させることで一意に誤りを訂正できる．

Step (1-1-3): シンドローム s よりエラーベクトル e_H をルックアップテーブルから読み出し，読み出した符号語 $v_H \oplus e_H$ との排他的論理和をとることで正しい符号語 v_H が得られる．

$$(v_H \oplus e_H) \oplus e_H = v_H \tag{4.8}$$

Step (1-2): ハミング符号と同様の手順で冗長部 (反復符号) の誤り訂正をする．

Step 2: ハミング符号，反復符号のデコーダを用いて符号部 v_H，冗長部 v_R をデコードし，それぞれ d_H，d_R とする．d_R が 0 であれば d_H を出力し，d_R が 1 であれば d_H を反復して出力する．

$$\begin{aligned} &\text{if } d_R == 1\text{'b0} : \\ &\qquad d = d_H \qquad (4.9) \\ &\text{else} : \\ &\qquad d = \overline{d_H} \qquad (4.10) \end{aligned}$$

以上の操作により，t ビット誤り訂正したデコードができる．デコーダ内部ではシンドロームをエラーベクトルに変換するためのルックアップテーブルが必要となる．ルックアップテーブルのサイズは符号部が 4 ビット $\times$ 8 行，冗長部が 1 ビット $\times$ 4 行 であるため，32+4 = 36 ビットとなる．

4.4 評価実験

本節では，一対多符号を評価し，ハミング符号と比較する．表 4.2 に一対多符号の値と符号語の関係を示す．実験に用いる符号の性質を表 4.3 に示す．メモリに値を書き込むとき，入力する値を符号語にエンコードしてから書き込む．メモリへの書き込みは 8 ビットワードごとに行われるため，実験に用いた符号は上位 4 ビットと下位 4 ビットに分けてエンコードする．このとき，一対多符号の場合は 4 ビットの値が 10 ビットの符号語にエンコードされ，ハミング符号の場合は 4 ビットの値が 7 ビットの符号語にエンコードされる．メモリは MRAM であると想定し，手法 [51] にならい，同じビットの値は書き込まないものとする．メモリのすべてのビットの初期値は 0 とする．一対多符号は，(7, 4, 3) ハミング符号をもとに前節の議論の通り構成したものである．

4.4.1 項では，一対多符号を用いて構成した不揮発メモリを計算機上で実装し，書き込み

表 4.4: 書き込みビット数評価実験結果.

	adpcmE	adpcmD	epicE	epicD	g721E	g721D	FlipAll
ハミング符号	2,047,028 (1.00)	735,281 (1.00)	935,223 (1.00)	520,951 (1.00)	2,129,981 (1.00)	829,899 (1.00)	917,504 (1.00)
拡張ドーナツ符号	2,358,295 (1.15)	1,003,629 (1.36)	1,26,627 (1.35)	717,682 (1.38)	2,481,184 (1.16)	1,099,970 (1.33)	589,824 (0.64)
一対多符号	1,767,860 (0.86)	697,469 (0.94)	874,975 (0.93)	488,503 (0.93)	1,856,933 (0.87)	758,959 (0.91)	393,216 (0.43)

ビット数を評価する．4.4.2 項では，一対多符号を用いて構成した不揮発メモリの回路を実装し，エンコーダ/デコーダならびに不揮発メモリの面積とエネルギーを評価する．

4.4.1 書き込みビット数評価実験

マルチメディア系の音声や画像を生成するベンチマークである MediaBench [26] を SimpleScalar [1] を用いてメモリトレースを取得した．また，用いるベンチマークである FlipAll はメモリに保存されている値に対してすべてのビットが反転するように値を毎回書き込むアプリケーションである．対象となる誤り訂正符号を用いて構成したメモリをトレースに適用して書き込みビット数を計測した．

不揮発メモリへの書き込みビット数を計測した結果を表 4.4 に示す．実験結果より，一対多符号は (7, 4, 3) ハミング符号と比較し最大 57.1%の書き込みビット数を削減した．

4.4.2 メモリ評価実験

メモリのエンコーダ/デコーダをハードウェア記述言語 Verilog HDL で記述し，Synopsys 社の Design Compiler のトポグラフィカルモードを用いて論理合成した．セルライブラリには STARC90nm を用いた．エンコーダ/デコーダで用いるルックアップテーブルの面積とエネルギーは MRAM を仮定し，あらかじめデータが保存されているとする．実装したメモリの面積と MediaBench を動作させたときのエネルギーを評価した．

エンコーダ/デコーダ内のルックアップテーブルは MRAM を仮定し，あらかじめデータが保存されているとする．MRAM の 1 ビットあたりの面積は $6.17 \times 10^{-1} um^2$ とする [10]．ルックアップテーブルのセル数を表 4.5 に示す．論理合成したエンコーダ/デコーダ本体とルックアップテーブル部分の面積を表 4.6 に示す．一対多符号を用いて構成したメモリの面積はハ

表 4.5: ルックアップテーブルのセル数.

	エンコーダ	デコーダ
ハミング符号	0	32
拡張ドーナツ符号	144	640
一対多符号	0	36

表 4.6: 面積評価結果.

	エンコーダ $[um^2]$		デコーダ $[um^2]$		合計 $[um^2]$
	本体	LUT	本体	LUT	
ハミング符号	27.5	-	2.1	19.7	49.4
拡張ドーナツ符号	-	88.8	-	394.9	483.7
一対多符号	211.0	-	36.7	22.2	269.9

ミング符号を用いて構成したメモリと比較して 5.4 倍となった.

次に回路のエネルギー評価を示す．メモリへの書き込みは同じ値は書き込まないものとする手法 [51] を想定する．N_r，N_w，B_w をアプリケーション動作時の総読み込みアクセス回数，総書き込みアクセス回数，総書き込みビット数とする．メモリは動作周波数が 100MHz の MRAM を想定した．MRAM の 1 ビットあたりのリードエネルギー E_r は 1.06×10^{-15}J/bit，ライトエネルギー E_w は 2.60×10^{-12}J/bit とする [10]．表 4.7 にメディアベンチアプリケーションの N_r，N_w を示す．B_w は表 4.4 で与えられる．L_c を符号長，k を情報量とする．

一対多符号を用いて構成したメモリのエネルギー E_t は式 (4.11) で与えられる.

$$E_t = \{(N_r + N_w) \times (L_c \times 2) \times E_r\} + (B_w \times E_w) + E_o \tag{4.11}$$

初項はメモリから符号語を読み出すエネルギー，第二項はメモリへ符号語を書き込むエネルギー，第三項はエンコーダ/デコーダのオーバーヘッドである．一対多符号では，書き込む符号語をメモリに書き込まれている符号語から決定するためにメモリから符号語の読み出しが必要である.

表 4.7: MediaBench の N_r と N_w.

	adpcmE	adpcmD	epicE	epicD	g721E	g721D	FlipAll
N_r	147,756	368,985	150,462	128,829	162,925	384,238	0
N_w	450,692	229,316	357,576	276,827	482,019	260,804	65,536

表 4.8: エンコーダ/デコーダのクリティカルパス遅延/ダイナミック電力/リーク電力.

	エンコーダ			デコーダ		
	d_E [ns]	$P_{dynamic,E}$ [μW]	$P_{leakage,E}$ [μW]	d_D [ns]	$P_{dynamic,D}$ [μW]	$P_{leakage,D}$ [μW]
ハミング符号	0.57	8.20	0.23	0.29	0.31	0.018
拡張ドーナツ符号	-	-	-	-	-	-
一対多符号	0.57	72.9	2.1	0.33	14.2	0.36

エンコーダ/デコーダのオーバーヘッドは式 (4.12) で求める.

$$\begin{aligned} E_o &= 2 \times N_w \times (d_E \times P_{dynamic,E} + P_{leakage,E}/100\text{MHz}) \\ &+ 2 \times N_r \times (d_D \times P_{dynamic,D} + P_{leakage,D}/100\text{MHz}) \\ &+ 2 \times N_r \times k \times E_r \end{aligned} \tag{4.12}$$

d_E と d_D はエンコーダとデコーダのクリティカルパス遅延，$P_{dynamic,E}$ と $P_{dynamic,D}$ はエンコーダとデコーダのダイナミック電力，$P_{leakage,E}$ と $P_{leakage,D}$ はエンコーダとデコーダのリーク電力を表す．$d_E \times P_{dynamic,E}$，$d_D \times P_{dynamic,D}$ はそれぞれエンコーダ/デコーダのダイナミックエネルギー，$P_{leakage,E}/100\text{MHz}$，$P_{leakage,D}/100\text{MHz}$ はそれぞれエンコーダ/デコーダのリークエネルギーを表す [6]．第三項はルックアップテーブルからエラーベクトルを読み出すためのエネルギーを表す．表 4.8 に各符号を用いて構成したメモリのエンコーダ/デコーダのクリティカルパス遅延，ダイナミック電力，リーク電力を示す.

表 4.9 にエネルギー評価実験結果を示す．一対多符号を用いて構成したメモリはハミング符号を用いて構成したメモリと比較して最大 56.8%のエネルギーを削減した.

表 4.9: エネルギー評価実験結果 (単位：nJ).

		adpcmE	adpcmD	epicE	epicD	g721E	g721D	FlipAll
ハミング符号	リード	1.1	2.7	1.1	1.0	1.2	2.9	0.0
	ライト	5,322.3	1,911.7	2,431.6	1,354.5	5,538.0	2,157.7	2,385.5
	オーバーヘッド	7.0	5.0	5.7	4.5	7.5	5.5	0.9
	合計	5,330.3 (1.00)	1,919.4 (1.00)	2,438.4 (1.00)	1,360.0 (1.00)	5,546.6 (1.00)	2,166.0 (1.00)	2,386.4 (1.00)
拡張ドーナツ符号	リード	1.4	3.5	1.4	1.2	1.6	3.7	0.0
	ライト	6,131.6	2,609.4	3,282.8	1,866.0	6,451.1	2,859.9	1,533.5
	オーバーヘッド	9.9	7.5	8.1	6.4	10.6	8.2	1.3
	合計	6,142.8 (1.15)	2,620.5 (1.36)	3,292.4 (1.35)	1,873.6 (1.38)	6,463.2 (1.16)	2,871.8 (1.32)	1,534.8 (0.64)
一対多符号	リード	6.3	3.9	1.6	1.4	1.7	4.1	0.0
	ライト	4,596.4	1,813.4	2,274.9	1,270.1	4,828.0	1,973.3	1,022.4
	オーバーヘッド	60.1	37.9	48.5	37.9	64.4	42.3	8.2
	合計	4,662.9 (0.87)	1,855.3 (0.97)	2,325.1 (0.95)	1,309.3 (0.96)	4,894.2 (0.88)	2,019.6 (0.93)	1,030.6 (0.43)

4.5 本章のまとめ

本章では，値と符号語を一対多に割り当てる書き込み削減・誤り訂正符号を構成する手法を提案した．t ビット誤り訂正符号を元にして提案手法を用いて生成した一対多符号は最悪書き込みビット数を最小化し，誤り訂正能力を保持したまま最大 $2t$ ビットまでの誤り訂正が可能となる．実験結果より，一対多符号を用いてメモリを構成したとき，ハミング符号と比較してエネルギーを最大 56.8%削減した．

第5章　REC符号

5.1　本章の概要

本章では，書き込みビット数削減と誤り訂正を同時に実現する手法として，誤り訂正符号の符号語をクラスタリングし，各クラスタに値を割り当てることで新たな符号を生成するアルゴリズムを提案する[1]．提案アルゴリズムにより生成したREC符号を用いて構成したメモリをアプリケーションに適用した結果，ハミング符号を用いて構成したメモリと比較して最大23.5%，BCH符号と反復符号を連接した符号を用いて構成したメモリと比較して最大27.5%のエネルギーをそれぞれ削減した．

REC符号ではメモリに書き込まれている符号語を考慮して値をエンコードするコンテクスト依存符号化のメモリアーキテクチャを用いる．

5.2　符号語のクラスタリング

本節では，いくつかの条件を満たすように誤り訂正符号の符号語をクラスタリングして，各クラスタに値を割り当てることで，書き込みビット数削減と誤り訂正を実現する符号が生成できることを証明する．

tビットの誤り訂正能力を持つ誤り訂正符号を考える．誤り訂正符号の情報量をk_pとするとき，符号語を$v_0, \ldots, v_{2^{k_p}-1}$とする．符号語$v_i$の符号長は$n$ビットである．すなわち，$k_p$ビットの値をエンコードすることで$n$ビットの符号語を得る．符号語同士のハミング距離は$2t+1$以上であり，誤り訂正能力は$t$ビットである．この誤り訂正符号を$(n, k_p, 2t+1)$誤り訂正符号と表す．

定数$S \geq 2t+1$を与えたとき，$(n, k_p, 2t+1)$誤り訂正符号から以下のようなグラフを生成

[1]本章の内容は [21, 23, 55] による．

できる．

1. 誤り訂正符号の符号語 $v_0, \ldots, v_{2^{k_p}-1}$ をノードとする．
2. 符号語 v_i, v_j のハミング距離が S 以下であれば，エッジ $e_{ij} = (v_i, v_j)$ を結ぶ．

ノードの集合を $V = \{v_0, \ldots, v_{2^{k_p}-1}\}$，エッジの集合を E とするとき，このグラフを S-バウンドグラフ $G = (V, E)$ と呼ぶ．図 5.1(a) に S-バウンドグラフの例を示す．詳細は例 5.1 で述べる．

r を冗長量とし，$k_r = k_p - r$ とする．冗長量とは，1 つのクラスタに属するノード数を定める定数である．ノードの集合 V から以下の条件を満たすクラスタ c_i $(i = 0, \ldots, 2^{k_r} - 1)$ を生成する．

(C1) $c_0 \cup \cdots \cup c_{2^{k_r}-1} = V$

(C2) $c_i \cap c_j = \emptyset$ $(0 \leq (i, j) \leq 2^{k_r} - 1, i \neq j)$

(C3) $|c_i| = 2^r$ $(0 \leq i \leq 2^{k_r} - 1)$

以上の条件 (C1)–(C3) をクラスタリング条件と呼ぶ．$C = \{c_0, \ldots, c_{2^{k_r}-1}\}$ をクラスタの集合とする．クラスタ c_i, c_j $(i \neq j)$ をクラスタの集合 C に属するクラスタとする．c_i, c_j が以下の条件 (F1), (F2) を満たすとき，c_i, c_j をエッジ $e'_{ij} = (c_i, c_j)$ で結ぶ．

(F1) クラスタ c_i に属する任意のノード v_k から，クラスタ c_j に属するあるノード v_l に対してエッジ e_{kl} が存在する．

(F2) クラスタ c_j に属する任意のノード v_l から，クラスタ c_i に属するあるノード v_k に対してエッジ e_{lk} が存在する．

以上の条件 (F1), (F2) を S ビット反転条件と呼ぶ．S-バウンドグラフをもとに，クラスタリング条件からクラスタの集合 C を生成し，S ビット反転条件からクラスタ間のエッジの集合 E' を生成する．クラスタの集合 C とクラスタ間のエッジの集合 E' からクラスタグラフ $G' = (C, E')$ を得る．図 5.1(b) にクラスタグラフの例を示す．詳細は例 5.1 で述べる．

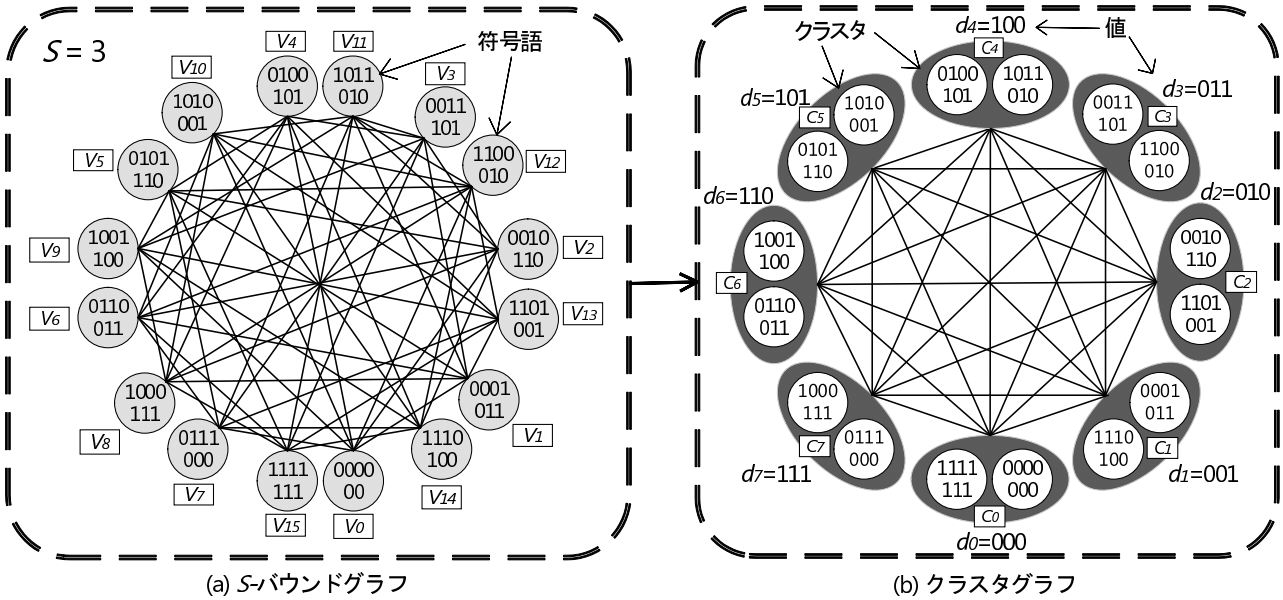


図 5.1: S-バウンドグラフ $G = (V, E)$(a) とクラスタグラフ $G' = (C, E')$(b).

$D = \{d_0, \ldots, d_{2^{k_r}-1}\}$ を k_r ビットの値の集合とする．各値は $d_0 = 00\cdots00, d_1 = 00\cdots01, \ldots,$ $d_{2^{k_r}-1} = 11\cdots11$ となる．値 $d_i \in D$ をクラスタ $c_i \in C$ に割り当てる．つまり，クラスタ $c_i \in C$ に属する全ての符号語は値 $d_i \in D$ を表すことにする．

定理 5.1. 値 d_i, d_j $(i \neq j)$ をクラスタ c_i, c_j にそれぞれ割り当てる．クラスタ c_i に属する任意のノードを v_k とする．クラスタグラフ $G' = (C, E')$ が完全グラフとなるとき，クラスタ c_j に属するあるノード v_l に対して，ハミング距離 $d_H(v_k, v_l) \leq S$ が成立する．

証明 5.1. クラスタ c_i, c_j 間には S ビット反転条件を満足するエッジが必ず存在するため，上記定理が成立する． □

系 5.1. 値 d_i はエンコードして v_k に，値 d_j はエンコードして v_l になるとする．クラスタグラフ $G' = (C, E')$ が完全グラフとなるとき，値 d_i を表す符号語 v_k から値 d_j を表す符号語 v_l へ書き込みビット数 S ビット以下で遷移できる．

定理 5.2. クラスタ c_i に属する符号語 v_k は t ビットの誤り訂正能力を持つ．

証明 5.2. 符号語 v_k は $(n, k_p, 2t+1)$ 誤り訂正符号の符号語であるため，上記定理が成立する．□

以上の定理より，クラスタグラフが完全グラフとなるような符号を用いたメモリでは，クラスタ c_i に属する符号語は全て値 d_i を表し，誤り訂正能力 t ビットを持ち，値 d_i を表す符号語 v_k から値 d_j を表す符号語 v_l へは書き込みビット数 S ビット以下で遷移できる．

例 5.1. (7, 4, 3) ハミング符号を考える．(7, 4, 3) ハミング符号は符号語 $\{v_0, \ldots, v_{15}\}$ から構成される．最小ハミング距離は 3 であるため，$t = 1$ ビットの誤り訂正能力を持つ．

定数 $S = 3$ を与え，S-バウンドグラフを生成する．生成した S-バウンドグラフを図 5.1(a) に示す．図 5.1(a) での各ノードは (7, 4, 3) ハミング符号の符号語を表す．$H(v_0, v_1) = 3 \leq 3$ であるため，ノード v_0, v_1 間にはエッジが存在する．$H(v_0, v_{15}) = 7 \geq 3$ であるため，ノード v_0, v_{15} 間にはエッジが存在しない．

S-バウンドグラフを元にクラスタグラフを生成する．生成したクラスタグラフを図 5.1(b) に示す．図 5.1(b) では 8 つのクラスタ $c_0, \ldots, c_7$ があり，各クラスタは 2 つのノードを持つ．例えば，クラスタ c_0 は $\{v_0, v_{15}\}$ を持ち，クラスタ c_1 は $\{v_1, v_{14}\}$ を持つ．S-バウンドグラフでは v_0, v_1 と v_{14}, v_{15} はそれぞれエッジで結ばれているため，S ビット反転条件を満足する．そのため，エッジ $e' = (c_0, c_1)$ を結ぶ．図 5.1(b) のクラスタグラフは完全グラフとなる．

次に，3 ビットの 8 つの値 $d_0, \ldots, d_7$ を考える．各値は $d_0 = 000, d_1 = 001, \ldots, d_7 = 111$ となる．値 d_i を c_i $(0 \leq i \leq 7)$ に割り当てる．例えば，クラスタ c_0 の符号語 $v_0 = 0000000, v_{15} = 1111111$ はどちらも値 $d_0 = 000$ を表す．また，クラスタ c_1 の符号語 $v_1 = 0001011, v_{14} = 1110100$ はどちらも値 $d_1 = 001$ を表す．

値 $d_0 = 000$ を表す符号語 $v_0 = 0000000$ がメモリに書き込まれているとする．値 $d_1 = 001$ をメモリに保存するとき，$v_1 = 0001011$ と $v_{14} = 1110100$ を書き込む 2 通りの書き込み方が存在する．この場合，$v_1 = 0001011$ を書き込むと値 d_0 を表す符号語 v_0 から値 d_1 を表す符号語 v_1 の遷移に 3 ビットの書き込みで済む．

以上より，生成した符号は 3 ビットの値を 7 ビットの符号語にエンコードする．この符号は 1 ビットの誤り訂正能力を持ち，全ての値の遷移は適切に符号語を選択することで 3 ビットで書き換えができる．

5.3 提案手法

5.2 節で全てのクラスタの組み合わせが S ビット反転条件を満足することで誤り訂正能力 t を維持しつつ，書き込みビット数を S ビット以下に制約できる符号を生成できることを示した．本節では，クラスタリング条件を満足し，全てのクラスタの組み合わせが S ビット反転条件を満足する誤り訂正符号の符号語のクラスタリング手法を示す．

$(n, k_p, 2t+1)$ 線形組織誤り訂正符号の符号語をクラスタリングすることを考える [35]．線形符号とはその符号の 2 つの符号語を XOR して得られるビット列が符号語となる符号である．組織符号とは符号語がメッセージと冗長なビット列で構成された符号である．$(n, k_p, 2t+1)$ 線形組織誤り訂正符号の符号語を $\{v_0, \ldots, v_{2^{k_p}-1}\}$ とする．符号には n ビットの符号語 $11\cdots11$ が含まれていることとする．この符号では，k_p ビットのメッセージ m_i $(0 \leq i \leq 2^{k_p}-1)$ がエンコードされて n ビットの符号語 v_i となる．誤り訂正能力は t ビットである．

提案アルゴリズムでは，始点ベクトル a と情報ベクトル x を用いてクラスタリングする．a, x はどちらも $(n, k_p, 2t+1)$ 誤り訂正符号の符号語である．a を始点ベクトル，x を情報ベクトルと呼ぶ．始点ベクトルの集合を $A = \{a_0, \ldots, a_{2^r-1}\}$，情報ベクトルの集合を $X = \{x_0, \ldots, x_{2^{k_r}-1}\}$ とする．ここで，r を冗長量としたとき，$k_r = k_p - r$ とする，A，X をうまく設定することで全ての符号語は $v = a \oplus x$ として表せる．

5.3.1 始点ベクトルの生成法

冗長量を $r \geq 1$，情報量を $k_r = k_p - r$ と表す．また，k_r, r は $k_r \equiv 0 \pmod{r}$ とする．n ビットの始点ベクトルの集合を $A = \{a_0, \ldots, a_{2^r-1}\}$ と表し，k_p ビットの始点メッセージの集合を $M^a = \{m_0^a, \ldots, m_{2^r-1}^a\}$ とする．$m_i^a(j)$ を m_i^a の j 番目のビットとする．つまり，$m_i^a(0)$ は m_i^a の LSB，$m_i^a(k_p-1)$ は m_i^a の MSB を表す．始点ベクトル a_i は始点メッセージ m_i^a をエンコードして得られる．

始点ベクトルの集合 A は以下の操作により得られる．

Step 1: $0 \leq i < 2^{r-1}$ に対して，m_i^a の上位 r ビットは i を 2 進数表記したものとする．例えば $k_p = 6, r = 3$ のとき，$m_0^a = \underline{000}***$, $m_1^a = \underline{001}***$, $m_2^a = \underline{010}***$, $m_3^a = \underline{011}***$ とな

る．ただし，*は未定義である．

Step 2: $0 \leq i < 2^{r-1}$ に対して，Step (2-1), Step (2-2) を実行する．

Step (2-1): $W_H(i)$ が偶数ならば，メッセージ m_i^a の下位 k_r の各 j ビット目は下式となる．

$$m_i^a(j) = m_i^a(k_r + \lfloor j \times r/k_r \rfloor) \tag{5.1}$$

Step (2-2): $W_H(i)$ が奇数ならば，メッセージ m_i^a の下位 k_r の各 j ビット目は下式となる．

$$m_i^a(j) = \overline{m_i^a(k_r + \lfloor j \times r/k_r \rfloor)} \tag{5.2}$$

Step 3: $2^{r-1} \leq i \leq 2^r - 1$ に対して，メッセージ m_i^a は以下のようになる．

$$m_i^a = \overline{m_{2^r-1-i}^a} \tag{5.3}$$

以上の操作で得られた k_p ビットのメッセージ m_i^a をエンコードすることで n ビットの始点ベクトル a_i が得られる．

例 5.2. $k_p = 6$，$r = 2$ とする．$k_r = k_p - r = 4$ となる．6 ビットのメッセージ m_i^a を上に示した手法により生成する．

Step 1 より，m_0^a, m_1^a の上位 2 ビットは以下のようになる．

$$\begin{aligned} m_0^a &= \underline{00}**** \\ m_1^a &= \underline{01}**** \end{aligned} \tag{5.4}$$

ただし，*は未定義である．

Step 2 より，m_0^a, m_1^a の下位 4 ビットは以下のようになる．

$$\begin{aligned} m_0^a &= 00\underline{0000} \\ m_1^a &= 01\underline{1100} \end{aligned} \tag{5.5}$$

Step 3 より，m_2^a, m_3^a は m_0^a, m_1^a を元に以下のようになる．

$$\begin{aligned} m_2^a &= \underline{100011} \\ m_3^a &= \underline{111111} \end{aligned} \tag{5.6}$$

以上の操作で得られた 6 ビットのメッセージ m_i^a をエンコードすることで n ビットの始点ベクトルが得られる．

5.3.2 情報ベクトルの生成法

n ビットの情報ベクトルの集合を $X = \{x_0, \ldots, x_{2^{k_r}-1}\}$ と表し，k_p ビットの情報メッセージの集合を $M^x = \{m_0^x, \ldots, m_{2^{k_r}-1}^x\}$ とする．$m_i^x(j)$ を m_i^x の j 番目のビットとする．つまり，$m_i^x(0)$ は m_i^x の LSB，$m_i^x(k_p-1)$ は m_i^x の MSB を表す．情報ベクトル x_i は情報メッセージ m_i^x をエンコードして得られる．

情報ベクトルの集合 X は以下の操作により得られる．

Step 1: m_i^x の上位 r ビットは 0 を 2 進数表記したものとする．

Step 2: m_i^x の下位 k_r ビットは i を 2 進数表記したものとする．

以上の操作で得られた k_p ビットのメッセージ m_i^x をエンコードすることで n ビットの情報ベクトル x_i が得られる．

例 5.3. $k_p = 6$，$r = 2$ とする．$k_r = k_p - r = 4$ となる．6 ビットのメッセージ m_i^x を上に示した手法により生成する．

Step 1 より，$m_0^x, \ldots, m_{15}^x$ の上位 2 ビットは以下のようになる．

$$\begin{aligned} m_0^x &= \underline{00}**** \\ m_1^x &= \underline{00}**** \\ m_2^x &= \underline{00}**** \\ &\vdots \\ m_{15}^x &= \underline{00}**** \end{aligned} \tag{5.7}$$

ただし，*は未定義である．

Step 2 より，$m_0^x, \ldots, m_{15}^x$ の下位 4 ビットは以下のようになる．

$$\begin{aligned} m_0^x &= 00\underline{0000} \\ m_1^x &= 00\underline{0001} \\ m_2^x &= 00\underline{0010} \\ &\vdots \\ m_{15}^x &= 00\underline{1111} \end{aligned} \tag{5.8}$$

以上の操作で得られた 6 ビットのメッセージ m_i^x をエンコードすることで n ビットの情報ベクトル x_i が得られる．

5.3.3 始点ベクトルと情報ベクトルの性質

本項では始点ベクトルと情報ベクトルの性質を示す．

定理 5.3. 前項で示した手法によって始点ベクトルの集合 A と情報ベクトルの集合 X を生成する．全ての符号語 $v_i \in \{v_0, \ldots, v_{2^{k_p}-1}\}$ は $v_i = a \oplus x$ $(a \in A, x \in X)$ によって一意に表せる．

証明 5.3. $a \oplus x$ で全ての符号語が表され，$a \oplus x$ は $2^r \times 2^{k_r} = 2^{r+k_r} = 2^{k_p}$ 個の組み合わせを持ち，誤り訂正の符号語数も 2^{k_p} 個であるため，上記定理が成立する． □

定数 $S = \lfloor n/2 \rfloor$ を与え，S-バウンドグラフ $G = (V, E)$ を生成する．r を冗長量として $k_r = k_p - r$ とする．ノードの集合 V からクラスタ c_i $(i = 0, \ldots, (2^{k_r} - 1))$ を以下のように生成する．

$$c_i = \{a_0 \oplus x_i, \ldots, a_{2^r-1} \oplus x_i\} \tag{5.9}$$

これらのクラスタはクラスタ条件 (C1)–(C3) を満足する．

クラスタの集合を $C = \{c_0, \ldots, c_{2^{k_r}-1}\}$ とする．c_i, c_j $(i \neq j)$ が S ビット反転条件 (F1), (F2) を満たすとき，c_i, c_j をエッジ $e' = (c_i, c_j)$ で結ぶ．このようにしてクラスタグラフ $G' = (C, E')$ を新たに生成する．

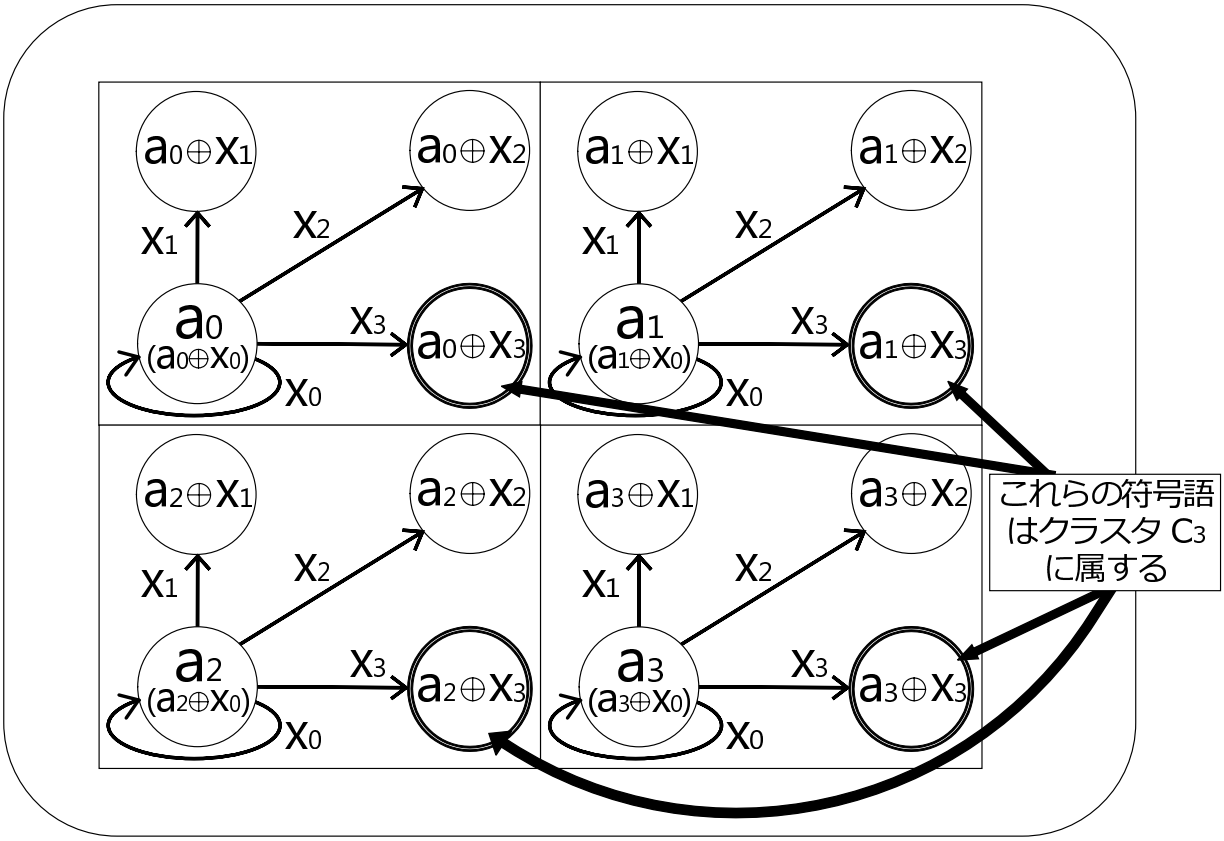


図 5.2: $r = 2$ のときの符号空間.

定理 5.4. 上述のようにして生成したクラスタグラフ $G' = (C, E')$ は完全グラフとなる.

証明 5.4. $0 \leq i \leq 2^{r-1}$ のとき，始点ベクトル a_i は始点ベクトル a_{2^r-1-i} を反転した符号語である．この性質を用いると全てのクラスタ間で S ビット反転条件を満足するエッジが存在するため，上記定理が成立する． □

5.4 REC 符号

k_r ビットの値の集合を $D = \{d_0, \ldots, d_{2^{k_r}-1}\}$ とする．クラスタ c_i に値 d_i を割り当てる．つまり，クラスタ $c_i \in C$ に属する全ての符号語は値 $d_i \in D$ を表すことにする.

定理 5.1, 5.2, 5.4 より，クラスタ c_i に属する全ての符号語は誤り訂正能力 t を持ち，値 d_i を表す．値 d_i を表す符号語 v_k から値 d_j を表す符号語 v_l への遷移は $\lfloor n/2 \rfloor$ 以下の書き込みビット数で済む．新たに生成した符号を $(n, k_r, 2t+1, r)$-REC 符号と呼ぶ．n は符号長，k_r は情報量，$2t+1$ は最小ハミング距離，r は冗長量を表す．

図 5.2 に $(n, 2, 2t+1, 2)$-REC 符号の符号空間を示す．4 つの始点ベクトル a_0, a_1, a_2, a_3 があり，4 つの長方形の中の左下の円が対応する．4 つの情報ベクトル x_0, x_1, x_2, x_3 があり，4 つの長方形の中の 4 本の矢印が対応する．全ての符号語は $a_i \oplus x_j$ $(i = 0, 1, 2, 3)$, $(j = 0, 1, 2, 3)$ で表せる．クラスタ c_j $(j = 0, 1, 2, 3)$ には符号語 $c_j = \{a_0 \oplus x_j, a_1 \oplus x_j, a_2 \oplus x_j, a_3 \oplus x_j\}$ が属し，全ての c_j に属する符号語 $a_i \oplus x_j \in c_j$ は値 d_j を表す．

5.5 エンコーダ/デコーダの設計

本節では，REC 符号を用いて構成したメモリアーキテクチャのエンコーダとデコーダの設計方法を説明する．REC 符号を用いて構成したメモリでは，メモリに値 d_k を保存するとき，まずメモリに書き込まれている符号語 $a_i \oplus x_j$ を読み出す．読み出した符号語 $a_i \oplus x_j$ を基に値 d_k を符号語 $a_z \oplus x_k$ にエンコードする．メモリから値を取り出すときには，メモリに書き込まれている符号語 $a_i \oplus x_j$ を読み出し，読み出した符号語をデコードすることで元の値を得ることができる．

5.5.1 エンコーダ

エンコーダは入力として読み出した符号語 $a_i \oplus x_j$，保存したい値 d_k を与えると，書き込む符号語 $a_z \oplus x_k$ を出力する．エンコーダは以下のように動作する．

Step 1: 読み出した符号語の上位 k_p ビットを抽出する．

$$m_i^a \oplus m_j^x = a_i \oplus x_j[n : n - k_p] \tag{5.10}$$

また，保存したい値を k_p ビットの情報メッセージになるよう上位 r ビットを 0 でパディ

ングする．

$$m_k^x = \overbrace{00\cdots00}^{r\ \text{ビット}} d_k \tag{5.11}$$

Step 2: $m_i^a \oplus m_j^x \oplus m_k^x$ より a_z の上位 r ビット (r_{a_z}) をテーブルから読み出す．

Step 3: r_{a_z} から始点ベクトルの生成法によって k_p ビットの始点メッセージ m_a^z を生成する．

Step 4: $m_a^z \oplus m_k^x$ と生成行列から書き込む符号語 $a_z \oplus x_k$ を得る．

符号語を書き込んだときのメモリの反転ビットを W とする．メモリに書き込みたい符号語 $a_z \oplus x_k$ は，メモリに書き込まれている符号語 $a_i \oplus x_j$ と反転ビット W を用いて

$$\begin{aligned} a_i \oplus x_j \oplus W &= a_z \oplus x_k \\ \therefore a_i \oplus x_j \oplus x_k &= a_z \oplus W \end{aligned} \tag{5.12}$$

と表せる．式 (5.12) の左辺は既知である．線形符号の性質により，$W = a_s \oplus x_t$ で表せるため，右辺は a_s，a_z のみ未知である．a_z は計算により一意に求まるため，Step 2 ではルックアップテーブルから値を参照することで a_z を得る．Step 2 では $a_i \oplus x_j \oplus x_k$ と $m_i^a \oplus m_j^x \oplus m_k^x$ は 1 対 1 に対応しており，a_z と始点ベクトルの上位 r ビット (r_{a_z}) も 1 対 1 に対応していることを利用している．また，Step 2 では保存したい値を表す符号語とメモリに書き込まれている語とのハミング距離が最も小さくなるような符号語の始点ベクトルを返すため，符号語に誤りが生じていてもエンコード可能である．Step 3 では，始点メッセージを生成する際に，r_{a_z} のハミング重みの偶奇の情報が必要である．通常ハードウェアでハミング重みを算出する際，計算量が大きくなるためあらかじめハミング重みを調べたいビット列とそのビット列のハミング重みをテーブルに保存しておく．今回は，Step 2 で r_{a_z} を得る際に $r+1$ ビットをルックアップテーブルから読み出し，追加した 1 ビットでハミング重みの偶奇を表すこととした．必要となるルックアップテーブルのサイズは $r+1$ ビット $\times 2^{k_p}$ 行である．

5.5.2 デコーダ

デコーダは入力として読み出した符号語を与えると，値を出力する．読み出した符号語には誤りが発生している可能性があるため，誤り訂正を考慮する．誤り箇所を表したビット列をエラーベクトル e とし，読み出した符号語は $a_i \oplus x_j \oplus e$ とする．符号語に誤りが生じていない場合は $e = 0$ となる．

Step 1: 誤り訂正を行うためにエラーベクトル e を求める．

Step (1-1): 一般的な線形組織符号と同様の手順で REC 符号の生成行列 G から検査行列 H を求める．

Step (1-2): 符号語 $a_i \oplus x_j \oplus e$ と検査行列 H からシンドローム s を求める．

$$s = (a_i \oplus x_j \oplus e)H \tag{5.13}$$

ここで，符号語 $a_i \oplus x_j \oplus e$ に誤りが生じていなければシンドローム s は

$$s = (a_i \oplus x_j)H = 0 \tag{5.14}$$

となるため，

$$s = (a_i \oplus x_j \oplus e)H = (a_i \oplus x_j)H \oplus eH = eH \tag{5.15}$$

と表せる．つまり，シンドロームはエラーベクトルのみに依存する．s と e を 1 対 1 に対応させることで一意に誤りを訂正できる．

Step (1-3): シンドロームよりエラーベクトルをテーブルから読み出し，符号語に XOR する．

$$(a_i \oplus x_j \oplus e) \oplus e = a_i \oplus x_j \tag{5.16}$$

Step 2: $a_i \oplus x_j$ の上位 r ビットから a_i が特定できるため，$(a_i \oplus x_j) \oplus a_i$ により，x_j が一意に求まる．情報ベクトル x_j と値 d_j は 1 対 1 に対応しているため，値は $d_j = x_j[n-r : n-k_p]$ により求まる．

以上の操作により，t ビット誤り訂正したデコーディングが行える．デコーダ内部では $a_i \oplus x_j$

の上位 r ビットから a_i を特定するためのハミング重みを格納したテーブルと，シンドロームをエラーベクトルに変換するためのテーブルの 2 種類が必要となる．それぞれのルックアップテーブルのサイズは 1 ビット $\times 2^r$ 行，k_p ビット $\times 2^{n-k_p}$ 行となる．

5.6 評価実験

本節では，REC 符号を用いて構成した不揮発メモリとそのエンコーダ/デコーダの性能を評価する．REC 符号と，一般に用いられる誤り訂正符号としてハミング符号と BCH 符号の符号の性質の比較，それらの符号を用いて構成したメモリにアプリケーションを適用した結果を示す．

5.6.1 符号の性質

本項では REC 符号の性質を示す．表 7.1 上部に $(7, 4, 3)$ ハミング符号，$(9, 4, 3, 1)$-REC，$(10, 4, 3, 2)$-REC，$(12, 4, 3, 4)$-REC の符号の性質の比較を示す．次に，$(15, 7, 5)$-BCH 符号 [5] と (5, 1, 5) 反復符号 [12] を連結した符号を考える．この符号では 8 ビットの情報を 7 ビット+1 ビットに分けてそれぞれエンコードを行い，15 ビットと 5 ビットの符号語を連結した 20 ビットがメモリに格納されるものとする．表 7.1 下部に $(15, 7, 5)$-BCH 符号+$(5, 1, 5)$ 反復符号，$(23, 8, 5, 1)$-REC，$(25, 8, 5, 2)$-REC，$(29, 8, 5, 4)$-REC，$(37, 8, 5, 8)$-REC の符号の性質の比較を示す．

いずれの結果も最大書き込みビット数，平均書き込みビット数ともに $\lfloor n/2 \rfloor$ 以下となっている．REC 符号において，各値はそれぞれ 2^r 個の符号語で表される．例えば，$(9, 4, 3, 1)$-REC 符号では冗長量は $r = 1$ であるため，各値はそれぞれ $2^r = 2$ 個の符号語で表される．冗長量 r を大きくするほど符号長と各値を表す符号語数は大きくなるが，最大書き込みビット数と平均書き込みビット数は小さくなる．

表 5.1: 符号の性質の比較.

	符号長	誤り訂正能力	最大書き込みビット数	最小書き込みビット数	平均書き込みビット数
(7, 4, 3) ハミング符号	7	1	7	3	3.50 (1.00)
(9, 4, 3, 1)-REC	9	1	4	3	3.25 (0.93)
(10, 4, 3, 2)-REC	10	1	4	3	3.12 (0.89)
(12, 4, 3, 4)-REC	12	1	4	3	2.93 (0.83)
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	20	2	20	5	10.0 (1.00)
(23, 8, 5, 1)-REC	23	2	11	5	8.92 (0.89)
(25, 8, 5, 2)-REC	25	2	12	5	8.67 (0.87)
(29, 8, 5, 4)-REC	29	2	12	5	7.80 (0.78)
(37, 8, 5, 8)-REC	37	2	10	5	7.18 (0.72)

表 5.2: 書き込みビット数評価実験結果.

	adpcmE	adpcmD	epicE	epicD	g721E	g721D
(7, 4, 3) ハミング符号	2,047,028 (1.00)	735,281 (1.00)	935,223 (1.00)	520,951 (1.00)	2,129,981 (1.00)	829,899 (1.00)
(9, 4, 3, 1)-REC	1,696,602 (0.83)	697,731 (0.95)	879,946 (0.94)	492,978 (0.95)	1,785,396 (0.84)	759,439 (0.92)
(10, 4, 3, 2)-REC	1,628,371 (0.80)	666,442 (0.91)	836,704 (0.90)	466,870 (0.90)	1,712,191 (0.80)	716,672 (0.86)
(12, 4, 3, 4)-REC	1,557,902 (0.76)	638,242 (0.87)	787,874 (0.84)	441,599 (0.85)	1,637,188 (0.77)	679,526 (0.82)
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	2,902,865 (1.00)	1,022,642 (1.00)	1,213,777 (1.00)	670,655 (1.00)	3,030,135 (1.00)	1,140,475 (1.00)
(23, 8, 5, 1)-REC	2,492,757 (0.86)	974,215 (0.95)	1,240,812 (1.02)	695,669 (1.04)	2,625,590 (0.87)	1,073,272 (0.94)
(25, 8, 5, 2)-REC	2,303,056 (0.79)	929,465 (0.91)	1,168,265 (0.96)	649,003 (0.97)	2,426,246 (0.80)	1,012,208 (0.89)
(29, 8, 5, 4)-REC	2,146,736 (0.74)	873,537 (0.85)	1,111,116 (0.92)	629,212 (0.94)	2,264,785 (0.75)	929,645 (0.82)
(37, 8, 5, 8)-REC	2,085,224 (0.72)	811,245 (0.79)	1,038,296 (0.86)	592,397 (0.88)	2,199,998 (0.73)	888,672 (0.78)

5.6.2 書き込みビット数評価実験

マルチメディア系の音声や画像を生成するベンチマークである MediaBench [26] を SimpleScalar [1] を用いてトレースを取得した．対象の誤り訂正符号を用いて構成したメモリをトレースに適用して書き込みビット数を計測した.

メモリへの書き込みは同じビットは書き込まないものとする手法 [51] を想定する．表 5.2 上部にハミング符号を用いて構成したメモリと REC 符号を用いて構成したメモリでアプリケーションを動作させたときの結果を示す．表 5.2 下部に (15, 7, 5)-BCH 符号+(5, 1, 5) 反復符号を用いて構成したメモリと REC 符号を用いて構成したメモリでアプリケーションを動作させたときの結果を示す．(12, 4, 3, 4)-REC を用いて構成したメモリはハミング符号を用いて構成したメモリと比較して最大 23.9%の書き込みビット数を削減した．(37, 8, 5, 8)-REC

表 5.3: ルックアップテーブルのセル数.

	エンコーダ	デコーダ
(7, 4, 3) ハミング符号	0	32
(9, 4, 3, 1)-REC	32	80
(10, 4, 3, 2)-REC	192	100
(12, 4, 3, 4)-REC	1,280	144
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	0	1,808
(23, 8, 5, 1)-REC	512	147,456
(25, 8, 5, 2)-REC	3,072	327,684
(29, 8, 5, 4)-REC	20,480	1,572,880
(37, 8, 5, 8)-REC	589,824	33,554,688

を用いて構成したメモリは (15, 7, 5)-BCH 符号+(5, 1, 5) 反復符号を用いて構成したメモリと比較して最大 28.2%の書き込みビット数を削減した.

5.6.3 メモリ評価実験

メモリのエンコーダ/デコーダをハードウェア記述言語 Verilog HDL で記述し，Synopsys 社の Design Compiler のトポグラフィカルモードを用いて論理合成した．セルライブラリは STARC90nm を用いた．エンコーダ/デコーダで用いるルックアップテーブルの面積とエネルギーは MRAM を仮定し，あらかじめデータが保存されているとする．実装したメモリの面積と MediaBench を動作させたときのエネルギーを評価した.

ルックアップテーブルの 1 ビットあたりの面積は $6.17 \times 10^{-1} um^2$ とする [10]．ルックアップテーブルのセル数を表 5.3 に示す．論理合成したエンコーダ本体とルックアップテーブル部分の面積を表 5.4 に示す．REC 符号を用いて構成したメモリの面積は既存の誤り訂正符号を用いて構成したメモリと比較して最大で 17,000 倍となった.

次に回路のエネルギー評価を示す．メモリへの書き込みは同じ値は書き込まないものとする手法 [51] を想定する．N_r，N_w，B_w をアプリケーション動作時の総読み込みアクセス回数，総書き込みアクセス回数，総書き込みビット数とする．メモリは動作周波数が 100MHz の MRAM を想定した．MRAM の 1 ビットあたりのリードエネルギー E_r は 1.06×10^{-15}J/bit,

表 5.4: 面積評価結果.

	エンコーダ $[um^2]$		デコーダ $[um^2]$		合計 $[um^2]$
	本体	LUT	本体	LUT	
(7, 4, 3) ハミング符号	27.5	-	2.1	19.7	49.4
(9, 4, 3, 1)-REC	74.1	19.7	43.0	49.4	186.2
(10, 4, 3, 2)-REC	36.0	118.5	36.7	61.7	252.8
(12, 4, 3, 4)-REC	57.9	789.8	36.7	88.8	973.2
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	108.0	-	2.1	1,115	1,225
(23, 8, 5, 1)-REC	174.3	316	45.2	90,980	91,515
(25, 8, 5, 2)-REC	203.9	1,895	79.7	202,181	204,360
(29, 8, 5, 4)-REC	136.9	12,636	86.1	970,467	983,326
(37, 8, 5, 8)-REC	129.8	363,921	86.1	20,703,242	21,067,379

ライトエネルギー E_w は 2.60×10^{-12}J/bit とする [10]．表 5.5 にメディアベンチアプリケーションの N_r，N_w を示す．B_w は表 5.2 で与えられる．L_c を符号長とする．

BCH 符号+反復符号を用いて構成したメモリのエネルギー E_t は式 (5.17) で与えられる．また，REC 符号を用いて構成したメモリのエネルギー E_t は情報量が 4 ビットのときは式 (5.18) で，8 ビットのときは式 (5.19) でそれぞれ与えられる．

$$E_t = (N_r \times L_c \times E_r) + (B_w \times E_w) + E_o \tag{5.17}$$

$$E_t = \{(N_r + N_w) \times (L_c \times 2) \times E_r\} + (B_w \times E_w) + E_o \tag{5.18}$$

$$E_t = \{(N_r + N_w) \times L_c \times E_r\} + (B_w \times E_w) + E_o \tag{5.19}$$

初項はメモリから符号語を読み出すエネルギー，第二項はメモリへ符号語を書き込むエネルギー，第三項はエンコーダ/デコーダのオーバーヘッドである．REC 符号では，書き込む符号語をメモリに書き込まれている符号語から決定するためにメモリから符号語の読み出しが必要である．エンコーダ/デコーダのオーバーヘッドは符号の情報量が 4 ビットのときは式 (5.20) で，8 ビットのときは式 (5.21) でそれぞれ求める．

表 5.5: MediaBench の N_r と N_w.

	adpcmE	adpcmD	epicE	epicD	g721E	g721D
N_r	147,756	368,985	150,462	128,829	162,925	384,238
N_w	450,692	229,316	357,576	276,827	482,019	260,804

$$\begin{aligned} E_o &= 2 \times N_w \times (d_E \times P_{dynamic,E} + P_{leakage,E}/100\text{MHz}) \\ &+ 2 \times N_w \times \{(r+1) \times E_r\} \\ &+ 2 \times N_r \times (d_D \times P_{dynamic,D} + P_{leakage,D}/100\text{MHz}) \\ &+ 2 \times N_r \times \{(k_p+1) \times E_r\} \end{aligned} \tag{5.20}$$

$$\begin{aligned} E_o &= N_w \times (d_E \times P_{dynamic,E} + P_{leakage,E}/100\text{MHz}) \\ &+ N_w \times \{(r+1) \times E_r\} \\ &+ N_r \times (d_D \times P_{dynamic,D} + P_{leakage,D}/100\text{MHz}) \\ &+ N_r \times \{(k_p+1) \times E_r\} \end{aligned} \tag{5.21}$$

d_E と d_D はエンコーダとデコーダのクリティカルパス遅延，$P_{dynamic,E}$ と $P_{dynamic,D}$ はエンコーダとデコーダのダイナミック電力，$P_{leakage,E}$ と $P_{leakage,D}$ はエンコーダとデコーダのリーク電力を表す．$d_E \times P_{dynamic,E}$，$d_D \times P_{dynamic,D}$ はそれぞれエンコーダ/デコーダのダイナミックエネルギー，$P_{leakage,E}$/100MHz，$P_{leakage,D}$/100MHz はそれぞれエンコーダ/デコーダのリークエネルギーを表す [6]．表 5.6 に各符号を用いて構成したメモリのエンコーダ/デコーダのクリティカルパス遅延，ダイナミック電力，リーク電力を示す．第二項と第四項は，エンコーダ/デコーダで用いているルックアップテーブルから値を読み出すエネルギーを表す．

表 5.7 にエネルギー評価実験結果を示す．(12, 4, 3, 4)-REC を用いて構成したメモリはハミング符号を用いて構成したメモリと比較して最大 23.5%，(37, 8, 5, 8)-REC を用いて構成したメモリは BCH 符号+反復符号を用いて構成したメモリと比較して最大 27.5%のエネルギーをそれぞれ削減した．

表 5.6: エンコーダ/デコーダのクリティカルパス遅延/ダイナミック電力/リーク電力.

	エンコーダ			デコーダ		
	d_E [ns]	$P_{dynamic,E}$ [μW]	$P_{leakage,E}$ [μW]	d_D [ns]	$P_{dynamic,D}$ [μW]	$P_{leakage,D}$ [μW]
(7, 4, 3) ハミング符号	0.57	8.20	0.23	0.29	0.31	0.018
(9, 4, 3, 1)-REC	0.40	29.8	0.76	0.33	17.1	0.43
(10, 4, 3, 2)-REC	0.40	13.2	0.32	0.33	14.5	0.36
(12, 4, 3, 4)-REC	0.40	24.8	0.62	0.33	14.4	0.36
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	0.64	35.3	0.96	0.29	0.30	0.018
(23, 8, 5, 1)-REC	0.52	78.9	1.73	0.38	12.7	0.26
(25, 8, 5, 2)-REC	0.51	82.2	2.03	0.33	31.5	0.79
(29, 8, 5, 4)-REC	0.60	55.2	1.31	0.33	34.3	0.87
(37, 8, 5, 8)-REC	0.58	54.0	1.26	0.33	34.2	0.87

5.7 本章のまとめ

本章では，書き込みビット数削減と誤り訂正を同時に実現する手法として，誤り訂正符号の符号語をクラスタリングし，各クラスタに値を割り当てることで新たな符号を生成するアルゴリズムを提案した．提案手法を用いて符号を生成すると，符号長が n ビットのとき，メモリに符号語を書き込むときの最大書き込みビット数は $\lfloor n/2 \rfloor$ に制約でき，平均書き込みビット数を削減できる．(12, 4, 3, 4)-REC を用いて構成したメモリはハミング符号を用いて構成したメモリと比較して最大 23.5%，(37, 8, 5, 8)-REC を用いて構成したメモリは BCH 符号+反復符号を用いて構成したメモリと比較して最大 27.5%のエネルギーをそれぞれ削減した．

表 5.7: エネルギー評価実験結果 (単位：nJ).

		adpcmE	adpcmD	epicE	epicD	g721E	g721D
(7, 4, 3) ハミング符号	リード	1.1	2.7	1.1	1.0	1.2	2.9
	ライト	5,322.3	1,911.7	2,431.6	1,354.5	5,538.0	2,157.7
	オーバーヘッド	7.0	5.0	5.7	4.5	7.5	5.5
	合計	5,330.3 (1.00)	1,919.4 (1.00)	2,438.4 (1.00)	1,360.0 (1.00)	5,546.6 (1.00)	2,166.0 (1.00)
(9, 4, 3, 1)-REC	リード	5.7	3.5	1.4	1.2	1.6	3.7
	ライト	4,411.2	1,814.1	2,287.9	1,281.7	4,642.0	1,974.5
	オーバーヘッド	23.8	20.4	19.8	15.7	25.5	22.2
	合計	4,440.6 (0.83)	1,838.1 (0.95)	2,309.1 (0.95)	1,298.6 (0.95)	4,669.1 (0.84)	2,000.4 (0.92)
(10, 4, 3, 2)-REC	リード	6.3	3.9	1.6	1.4	1.7	4.1
	ライト	4,233.8	1,732.7	2,175.4	1,213.9	4,451.7	1,863.3
	オーバーヘッド	15.2	15.1	12.9	10.3	16.3	16.3
	合計	4,255.3 (0.80)	1,751.8 (0.91)	2,189.9 (0.90)	1,225.5 (0.90)	4,469.8 (0.81)	1,883.7 (0.87)
(12, 4, 3, 4)-REC	リード	7.6	4.7	1.9	1.6	2.1	4.9
	ライト	4,050.5	1,659.4	2,048.5	1,148.2	4,256.7	1,766.8
	オーバーヘッド	23.9	19.6	19.8	15.7	25.7	21.4
	合計	4,082.1 (0.76)	1,683.7 (0.88)	2,070.2 (0.85)	1,165.5 (0.85)	4,284.5 (0.77)	1,793.0 (0.83)
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	リード	3.1	7.8	3.2	2.7	3.5	8.1
	ライト	7,547.4	2,658.9	3,155.8	1,743.7	7,878.4	2,965.2
	オーバーヘッド	15.8	10.6	12.8	10.1	17.0	11.8
	合計	7,566.4 (1.00)	2,677.3 (1.00)	3,171.9 (1.00)	1,756.5 (1.00)	7,898.8 (1.00)	2,985.1 (1.00)
(23, 8, 5, 1)-REC	リード	14.6	9.0	3.7	3.1	4.0	9.4
	ライト	6,481.2	2,533.0	3,226.1	1,808.7	6,826.5	2,790.5
	オーバーヘッド	29.6	19.7	24.0	18.8	31.7	21.9
	合計	6,525.3 (0.86)	2,561.7 (0.96)	3,253.8 (1.03)	1,830.7 (1.04)	6,862.2 (0.87)	2,821.8 (0.95)
(25, 8, 5, 2)-REC	リード	15.9	9.8	4.0	3.4	4.3	10.2
	ライト	5,987.9	2,416.6	3,037.5	1,687.4	6,308.2	2,631.7
	オーバーヘッド	33.9	25.1	27.8	21.9	36.4	27.6
	合計	6,037.7 (0.80)	2,451.5 (0.92)	3,069.3 (0.97)	1,712.7 (0.98)	6,349.0 (0.80)	2,669.5 (0.89)
(29, 8, 5, 4)-REC	リード	18.4	11.3	4.6	4.0	5.0	11.8
	ライト	5,581.5	2,271.2	2,888.9	1,636.0	5,888.4	2,417.1
	オーバーヘッド	27.9	22.6	23.1	18.2	30.0	24.7
	合計	5,627.8 (0.74)	2,305.1 (0.86)	2,916.6 (0.92)	1,658.1 (0.94)	5,923.4 (0.75)	2,453.5 (0.82)
(37, 8, 5, 8)-REC	リード	23.5	14.5	5.9	5.1	6.4	15.1
	ライト	5,421.6	2,109.2	2,699.6	1,540.2	5,720.0	2,310.5
	オーバーヘッド	28.8	23.0	23.8	18.8	30.9	25.1
	合計	5,473.8 (0.72)	2,146.7 (0.80)	2,729.2 (0.86)	1,564.0 (0.89)	5,757.3 (0.73)	2,350.8 (0.79)

第6章　Relaxed-REC符号

6.1　本章の概要

REC符号は不揮発メモリの書き込みビット数削減と誤り訂正を同時に実現する．REC符号は線形組織誤り訂正符号を元にして生成するが，この線形組織誤り訂正符号はすべてのビットが1の符号語$11\cdots1$を含む必要がある．REC符号ではこの制約により符号長が増加するという欠点がある．

本章では，すべてのビットが1の符号語$11\cdots1$を含む必要のないRelaxed-線形組織誤り訂正符号を用いたRelaxed-REC符号の構成手法を提案する[1]．Relaxed-REC符号を用いて構成したメモリをアプリケーションに適用した結果，既存のREC符号を用いて構成したメモリに比べ最大20.0%，BCH符号と反復符号を連接した符号を用いて構成したメモリに比べ最大38.3%のエネルギーを削減した．

Relaxed-REC符号ではメモリに書き込まれている符号語を考慮して値をエンコードするコンテクスト依存符号化のメモリアーキテクチャを用いる．

6.2　REC符号とRelaxed-REC符号

REC符号は書き込みビット数削減と誤り訂正を同時に実現する符号である [23]．REC符号の生成にはまず，tビットの誤り訂正能力を持つ誤り訂正符号を用意する．その符号語をクラスタリングし，クラスタをノードとするグラフを生成する．各クラスタに値を割り当てることで，クラスタに属する符号語はクラスタに割り当てた値を表すことにする．このようにして値と符号語が1対多に対応した符号が生成される．生成したグラフのエッジが条件を満足するとき，符号はtビットの誤り訂正能力を維持しつつ最大書き込みビット数を$\lfloor n/2 \rfloor$

[1]本章の内容は [24] による．

に制約できる．

REC 符号の元となる線形組織誤り訂正符号はすべてのビットが 1 の符号語 $11\cdots1$ を含む必要がある．しかし，この制約を満たす線形組織誤り訂正符号の符号長は増大し，同時に REC 符号の符号長も増大してしまう．そこで，Relaxed-線形組織誤り訂正符号を元に生成する Relaxed-REC 符号を提案する．Relaxed-線形組織誤り訂正符号はすべてのビットが 1 の符号語 $11\cdots1$ を含む必要のない線形組織誤り訂正符号である．REC 符号，Relaxed-REC 符号はどちらも元となる線形組織誤り訂正符号をクラスタリングすることで生成できる．Relaxed-REC 符号は REC 符号に比べ，符号長が小さく，書き込みビット数もより大きく削減できる．符号長を n，メッセージのビット長を k_p としたとき Relaxed-REC 符号の最大書き込みビット数は $n - \lceil k_p/2 \rceil$ に制約できる[2]．

6.3 Relaxed-LSECC

符号長 n，情報量 k_p，誤り訂正能力 t の線形組織誤り訂正符号を考える．線形符号とはその符号の 2 つの符号語を XOR して得られるビット列が符号語となる符号である．組織符号とは符号語がメッセージと冗長なビット列で構成された符号である．線形組織誤り訂正符号を $LSECC$ と表す．誤り訂正能力を得るためには誤り訂正符号のシンドロームの個数 N_s がエラーパターンの個数 N_e よりも大きい必要がある．N_s，N_e はそれぞれ以下で与えられる．

$$N_s = 2^{n-k_p} \tag{6.1}$$

$$N_e = {}_nC_0 + \cdots + {}_nC_t \tag{6.2}$$

$LSECC$ において，$N_e \leq N_s$ は必ず成立する．N_s が N_e に漸近することは，$LSECC$ の符号空間をより効率的に利用していることを意味する．

以下の例に示すとおり，すべてのビットが 1 の符号語 $11\cdots1$ を持つ $LSECC$ は，そうでない $LSECC$ に比べて符号長が増大する可能性がある．

例 6.1. 情報量 $k_p = 2$，誤り訂正能力 $t = 1$ の $LSECC$ を例に説明する．まず，すべてのビッ

[2]Relaxed-線形組織誤り訂正符号自体は一般的な線形組織誤り訂正符号であるため提案には含めない．

トが 1 の符号語を含む $LSECC$ を考える．$k_p = 2$ であるため，メッセージは $\{00, 01, 10, 11\}$ となる．条件を満たす $LSECC$ として

$$LSECC_1 = \{000000,\ 010011,\ 101100,\ 111111\} \tag{6.3}$$

が考えられる．このときの符号長は 6 ビットとなる．メッセージ 00, 01, 10, 11 はそれぞれ符号語 000000, 010011, 101100, 111111 にエンコードされ，特に符号語 111111 はすべてのビットが 1 の符号語である．$LSECC_1$ の符号長はこれ以上削減することはできず，この条件での最小の符号長となる．$LSECC_1$ のエラーパターンは $N_e =_6 C_0 +_6 C_1 = 7$，シンドロームの個数は $N_s = 2^{6-2} = 16$ となり，$N_e \leq N_s$ を満たす．

次に，すべてのビットが 1 の符号語を含む必要のない Relaxed-$LSECC$ を考える．条件を満たす $LSECC$ として

$$LSECC_2 = \{00000,\ 01011,\ 10110,\ 11101\} \tag{6.4}$$

が考えられる．このときの符号長は 5 ビットとなる．メッセージ 00, 01, 10, 11 はそれぞれ符号語 00000, 01011, 10110, 11101 にエンコードされ，その中に符号語 11111 は含まない．$LSECC_2$ のエラーパターンは $N_e =_5 C_0 +_5 C_1 = 6$，シンドロームの個数は $N_s = 2^{5-2} = 8$ となり，$N_e \leq N_s$ を満たす．

以上の 2 つの符号は同じ情報量と同じ誤り訂正能力を有するが，$LSECC_2$ の符号長は $LSECC_2$ よりも小さく，シンドロームの個数もエラーパターンにより漸近している．つまり，$LSECC_2$ は $LSECC_1$ よりも効率的な符号だと言える．

6.4 Relaxed-LSECC を用いた REC 符号

符号長 n，情報量 k_p，誤り訂正能力 t の線形組織誤り訂正符号の符号語 $\{v_0, \ldots, v_{2^{k_p}-1}\}$ を考える．k_p ビットのメッセージ m_i $(0 \geq i \geq 2^{k_p} - 1)$ は n ビットの符号語 v_i にエンコードされる．すべてのビットが 1 の符号語 $11\cdots1$ を含む LSECC を $(n, k_p, 2t+1)$-LSECC と表す．また，同様にすべてのビットが 1 の符号語 $11\cdots1$ を含まない LSECC を $(n, k_p, 2t+1)$-relaxed-LSECC と表す．前節より，k_p と t を与えたとき，$(n, k_p, 2t+1)$-relaxed-LSECC の符

号長は $(n, k_p, 2t+1)$-LSECC の符号長よりも小さくなる．

提案アルゴリズムでは，始点ベクトル a と情報ベクトル x を用いてクラスタリングする．a, x はどちらも $(n, k_p, 2t+1)$ 誤り訂正符号の符号語である．始点ベクトルの集合を $A = \{a_0, \ldots, a_{2^r-1}\}$，情報ベクトルの集合を $X = \{x_0, \ldots, x_{2^{k_r}-1}\}$ とする．ここで，r を冗長量とし，$k_r = k_p - r$ とする，A，X をうまく設定することで全ての符号語は $v = a \oplus x$ として表せる．

始点ベクトルの集合 A と情報ベクトルの集合 X の生成法を提案する．以降は，$(n, k_p, 2t+1)$-relaxed-LSECC を LSECC と表す．

6.4.1 始点ベクトルの生成法

冗長量を $r \geq 1$，情報量を $k_r = k_p - r$ と表す．また， k_r，r は $k_r \equiv 0 \pmod{r}$ とする．n ビットの始点ベクトルの集合を $A = \{a_0, \ldots, a_{2^r-1}\}$ と表し，k_p ビットの始点メッセージの集合を $M^a = \{m_0^a, \ldots, m_{2^r-1}^a\}$ とする．$m_i^a(j)$ を m_i^a の j 番目のビットとする．つまり，$m_i^a(0)$ は m_i^a の LSB，$m_i^a(k_p - 1)$ は m_i^a の MSB を表す．始点ベクトル a_i は始点メッセージ m_i^a をエンコードして得られる．

始点ベクトルの集合 A の生成法を提案する．

Step 1: $0 \leq i < 2^{r-1}$ に対して，m_i^a の上位 r ビットは i を 2 進数表記したものとする．例えば $k_p = 6$, $r = 3$ のとき，$m_0^a = \underline{000}$***, $m_1^a = \underline{001}$***, $m_2^a = \underline{010}$***, $m_3^a = \underline{011}$*** となる．ただし，*は未定義である．

Step 2: $0 \leq i < 2^{r-1}$ に対して，メッセージ m_i^a の下位 k_r の各 j ビット目は下式となる．

$$m_i^a(j) = m_i^a(k_r + \lfloor j \times r / k_r \rfloor) \tag{6.5}$$

Step 3: $2^{r-1} \leq i \leq 2^r - 1$ に対して，メッセージ m_i^a は以下のようになる．

$$m_i^a = \overline{m_{2^r-1-i}^a} \tag{6.6}$$

以上の操作で得られた k_p ビットのメッセージ m_i^a をエンコードすることで n ビットの始点ベ

クトル a_i が得られる．

例 6.2. $k_p = 6$，$r = 2$ とすると，$k_r = k_p - r = 4$ となる．6 ビットのメッセージ m_i^a を上に示した手法により生成する．

Step 1 より，m_0^a, m_1^a の上位 2 ビットは以下のようになる．

$$\begin{aligned} m_0^a &= \underline{00}**** \\ m_1^a &= \underline{01}**** \end{aligned} \tag{6.7}$$

ただし，*は未定義である．

Step 2 より，m_0^a, m_1^a の下位 4 ビットは以下のようになる．

$$\begin{aligned} m_0^a &= 00\underline{0000} \\ m_1^a &= 01\underline{0011} \end{aligned} \tag{6.8}$$

Step 3 より，m_2^a, m_3^a は m_0^a, m_1^a を元に以下のようになる．

$$\begin{aligned} m_2^a &= \underline{101100} \\ m_3^a &= \underline{111111} \end{aligned} \tag{6.9}$$

以上の操作で得られた 6 ビットのメッセージ m_i^a をエンコードすることで n ビットの始点ベクトルが得られる．

6.4.2 情報ベクトルの生成法

n ビットの情報ベクトルの集合を $X = \{x_0, \ldots, x_{2^{k_r}-1}\}$ と表し，k_p ビットの情報メッセージの集合を $M^x = \{m_0^x, \ldots, m_{2^{k_r}-1}^x\}$ とする．$m_i^x(j)$ を m_i^x の j 番目のビットとする．つまり，$m_i^x(0)$ は m_i^x の LSB，$m_i^x(k_p - 1)$ は m_i^x の MSB を表す．情報ベクトル x_i は情報メッセージ m_i^x をエンコードして得られる．

情報ベクトルの集合 X の生成法を提案する．

Step 1: m_i^x の上位 r ビットは 0 を 2 進数表記したものとする．

Step 2: m_i^x の下位 k_r ビットは i を 2 進数表記したものとする．

以上の操作で得られた k_p ビットのメッセージ m_i^x をエンコードすることで n ビットの情報ベクトル x_i が得られる．

例 6.3. $k_p = 6$，$r = 2$ とすると，$k_r = k_p - r = 4$ となる．6 ビットのメッセージ m_i^x を上に示した手法により生成する．

Step 1 より，$m_0^x, \ldots, m_{15}^x$ の上位 2 ビットは以下のようになる．

$$\begin{aligned} m_0^x &= \underline{00}**** \\ m_1^x &= \underline{00}**** \\ m_2^x &= \underline{00}**** \\ &\vdots \\ m_{15}^x &= \underline{00}**** \end{aligned} \tag{6.10}$$

ただし，*は未定義である．

Step 2 より，$m_0^x, \ldots, m_{15}^x$ の下位 4 ビットは以下のようになる．

$$\begin{aligned} m_0^x &= 00\underline{0000} \\ m_1^x &= 00\underline{0001} \\ m_2^x &= 00\underline{0010} \\ &\vdots \\ m_{15}^x &= 00\underline{1111} \end{aligned} \tag{6.11}$$

以上の操作で得られた 6 ビットのメッセージ m_i^x をエンコードすることで n ビットの情報ベクトル x_i が得られる．

6.4.3 始点ベクトルと情報ベクトルの性質

本項では提案手法で生成した始点ベクトルと情報ベクトルの性質を示す．

定理 6.1. 提案手法によって始点ベクトルの集合 A と情報ベクトルの集合 X を生成する．全ての符号語 $v_i \in \{v_0, \ldots, v_{2^{k_p}-1}\}$ は $v_i = a \oplus x$ $(a \in A, x \in X)$ によって一意に表せる．

証明 6.1. 線形符号では符号語同士の排他的論理和が符号語となる．始点ベクトル a と情報ベクトル x は LSECC に属す符号語であるため，$a \oplus x$ も同様に LSECC に属す符号語となる．

提案手法において，始点ベクトル a_i にエンコードされる始点メッセージ m_i^a の上位 r ビットは始点メッセージごとに異なるため，組織符号の性質により始点ベクトル a_i の上位 r ビットも始点ベクトルごとに異なる．同様に，情報ベクトル x_i の上位 r ビットはすべて 0，次の k_r ビットは情報メッセージごとに異なる．

以上より，始点ベクトル a と情報ベクトル x には重複がないため，符号語 $a \oplus x$ $(a \in A,$ $x \in X)$ も重複がない．符号語 $a \oplus x$ の個数は始点ベクトル a の個数 2^r と情報ベクトルの個数 2^{k_r} の組み合わせとなるため，$2^r \times 2^{k_r} = 2^{r+k_r} = 2^{k_p}$ となる．$|LSECC| = 2^{k_p}$ であるため，LSECC の符号語 v_i は $a \oplus x$ $(a \in A,$ $x \in X)$ で一意に表せる． □

ただし，LSECC はすべてのビットが 1 の符号語を含む必要がないため，k_p ビットのメッセージ $m = 11\cdots1$ は符号語 $11\cdots1$ にエンコードされるとは限らない．

定理 6.2. M^a を始点メッセージの集合，m_i^a $(0 \leq i < 2^{r-1})$ を集合の半分に属する始点メッセージとすると，始点メッセージ m_i^a を反転した始点メッセージ $m_{2^r-1-i}^a$ は存在する．つまり，$m_i^a = \overline{m_{2^r-1-i}^a}$ が成立する．

証明 6.2. m_i^a は M^a の半分に属する始点メッセージであるため，m_i^a は 6.4.1 節の Step1，Step2 より生成できる．Step3 のように，$j = 2^r - 1 - i$ であるため，$m_i^a = \overline{m_j^a}$ となる．以上より，$m_i^a = \overline{m_{2^r-1-i}^a}$ が成立する． □

5.2 節の議論と同様に，定数 $S = n - \lceil k_p/2 \rceil$ を与え，S-バウンドグラフ $G = (V, E)$ を生成する．r を冗長量として $k_r = k_p - r$ とする．ノードの集合 V からクラスタ c_i $(i = 0, \ldots, (2^{k_r} - 1))$ を以下のように生成する．

$$c_i = \{a_0 \oplus x_i, \ldots, a_{2^r-1} \oplus x_i\} \tag{6.12}$$

これらのクラスタは 5.2 節で示したクラスタ条件 (C1)–(C3) を満足する．

クラスタの集合を $C = \{c_0, \ldots, c_{2^{k_r}-1}\}$ とする．c_i, c_j $(i \neq j)$ が S ビット反転条件 (F1), (F2) を満たすとき，c_i, c_j を新たにエッジ $e' = (c_i, c_j)$ で結ぶ．クラスタを結ぶエッジ e' の集合を E' とする．このようにしてクラスタグラフ $G' = (C, E')$ を新たに生成する．

定理 6.3. 上述のようにして生成したクラスタグラフ $G' = (C, E')$ は完全グラフとなる．

証明 6.3. 定理 6.2 に示したように，$m_i^a = \overline{m_{2^r-1-i}^a}$ が成立する．始点ベクトル a_i は始点メッセージ m_i^a をエンコードすることで得られる．組織符号の性質により a_i の上位 k_p ビットはメッセージ m_i^a となる．

c_j, c_l をクラスタグラフの 2 つのクラスタとする．$a_i \oplus x_j$，$a_k \oplus x_l$ をそれぞれクラスタ c_j, c_l に属する任意の符号語とする．ここで，$d_H(a_i \oplus x_j, a_k \oplus x_l) \leq n - \lceil k_p/2 \rceil$ であれば，S バウンドグラフ G においてエッジ $e = (a_i \oplus x_j, a_k \oplus x_l)$ が結べる．一方，$d_H(a_i \oplus x_j, a_k \oplus x_l) > n - \lceil k_p/2 \rceil$ であれば，符号語 $a_{2^r-1-k} \oplus x_l$ がクラスタ c_l に含まれている．符号語 $a_{2^r-1-k} \oplus x_l$ はメッセージ m_k^a を反転したメッセージ $m_{2^r-1-k}^a$ をエンコードした符号語である．符号語 $\overline{a_k} \oplus x_l$ の上位 k_p ビットは符号語 $a_{2^r-1-k} \oplus x_l$ の上位 k_p ビットが反転したビット列である．仮に，$a_{2^r-1-k} \oplus x_l$ の下位 $(n - k_p)$ ビットに対して $a_i \oplus x_j$ の下位 $(n - k_p)$ ビットのすべてのビットが反転していたとしても，上位 k_p ビットのハミング距離は $\lfloor k_p/2 \rfloor$ 以下となるため，$d_H(a_i \oplus x_j, a_{2^r-1-k} \oplus x_l) > n - \lceil k_p/2 \rceil$ が成立する．そのためこの場合でも，S バウンドグラフ G においてエッジ $e = (a_i \oplus x_j, a_{2^r-1-k} \oplus x_l)$ が結べる．

この議論はクラスタ c_i と c_l を入れ替えても成立するため，全てのクラスタ間で S-ビット反転条件を満足するエッジが存在し，上記定理が成立する． □

6.5 Relaxed-REC 符号

k_r ビットの値の集合を $D = \{d_0, \ldots, d_{2^{k_r}-1}\}$，クラスタグラフを $G' = (C, E')$ とする．$(n, k_p, 2t+1)$-relaxed-LSECC を $G' = (C, E')$ 生成のための元となる誤り訂正符号とする．クラスタ $c_i \in C$ に値 $d_i \in D$ を割り当てる．つまり，クラスタ $c_i \in C$ に属する全ての符号語は値 $d_i \in D$ を表すことにする．

定理 5.1, 5.2, 6.3 より，クラスタ c_i に属する全ての符号語は誤り訂正能力 t を持ち，値 d_i を表す．値 d_i を表す符号語 v_k から値 d_j を表す符号語 v_l への遷移は $n - \lceil k_p/2 \rceil$ 以下の書き込みビット数で済む．n を符号長，k_r を情報量，$2t+1$ を最小ハミング距離，r を冗長量とする．新たに生成した符号を $(n, k_r, 2t+1, r)$-relaxed-REC 符号と呼ぶ．

$(n, k_r, 2t+1, r)$-relaxed-REC 符号では，k_r ビットの値 d が n ビットの符号語にエンコードされ，t ビットの誤り訂正能力を持つ．異なる k_r ビットの値 d' をメモリに書き込むとき，たかだか $n - \lceil k_p/2 \rceil$ の書き込みビット数で済む．このとき，値 d' を表す複数の符号語は n ビットの符号空間に一様に分布する．複数の符号語の中から適切に書き込む符号語を選ぶことで書き込みビット数を最小化できる．6.6 節より，冗長量 r が大きいほど書き込みビット数は小さくなる．以上の議論をまとめると Relaced-REC 符号は以下の性質を持つ．

1. n を符号長，k_r を元となる LSECC の情報量とすると，Relaxed-REC 符号の最大書き込みビット数は $(n - \lceil k_p/2 \rceil)$ となる (定理 6.3，定理 5.1)．
2. Relaxed-REC 符号の誤り訂正能力は t ビットとなる (定理 5.2)．
3. Relaxed-REC 符号の平均書き込みビット数は REC 符号よりも小さくなる (6.6 節 表 6.1)．

上記性質は Relaxed-LSECC を本章で提案した手法を用いて Relaxed-REC 符号に効果的に適用できていることを示す．REC 符号はすべてのビットが 1 の符号語を含む LSECC を元となる符号として使用し，最大書き込みビット数を $\lfloor n/2 \rfloor$ まで制約した [23]．本章で提案した Relaxed-REC 符号の理論的な性質は REC 符号に劣るが，REC 符号が元となる符号として用いる LSECC の符号長よりも Relaxed-REC 符号で元となる符号として用いる Relaxed-LSECC の符号長は小さい．そのため Relaxed-REC 符号は符号長が小さく，さらに 6.6 節に示すように書き込みビット数を REC 符号に比べ削減できる．

6.6 評価実験

本節では，Relaxed-REC 符号の性質とそれを用いて構成した不揮発メモリとそのエンコーダ/デコーダの性能を評価する．

6.6.1 符号の性質

提案手法の性質を示すため，複数の Relaxed-REC 符号を生成した．Relaxed-REC 符号として，$(17, 8, 5, 1)$-relaxed-REC，$(19, 8, 5, 2)$-relaxed-REC，$(21, 8, 5, 4)$-relaxed-REC，$(26, 8, 5, 8)$-relaxed-REC を用いる．これらの Relaxed-REC 符号はそれぞれ $(17, 9, 5)$-relaxed-LSECC，$(19, 10, 5)$-relaxed-LSECC，$(21, 12, 5)$-relaxed-LSECC，$(26, 16, 5)$-relaxed-LSECC から生成される．比較対象の REC 符号として $(23, 8, 5, 1)$-REC，$(25, 8, 5, 2)$-REC，$(29, 8, 5, 4)$-REC，$(37, 8, 5, 8)$-REC を用いる．これらの REC 符号はそれぞれすべてのビットが 1 の符号語を含む $(23, 9, 5)$-LSECC，$(25, 10, 5)$-LSECC，$(29, 12, 5)$-LSECC，$(37, 16, 5)$-LSECC から生成される．

表 6.1 に符号の性質を示す．すべての符号は情報量 $k_r = 8$，誤り訂正能力 $t = 2$ である．最大書き込みビット数，最小書き込みビット数，平均書き込みビット数はすべての書き込みの組み合わせから計算して得られる．表 6.1 より，(37, 8, 5, 8)-REC 符号の平均書き込みビット数が 7.18 ビットに対して (26, 8, 5, 8)-relaxed-REC 符号では 5.18 ビットに削減される．Relaxed-REC 符号の最大書き込みビット，平均書き込みビット数は理論的な制約である $n - \lceil k_p/2 \rceil$ よりも非常に小さい．Relaxed-REC 符号では値を 2^r 個の符号語に割り当てている．r を大きくすると符号長は大きくなるが，値に割り当てられる符号語の数が増え，最大書き込みビットと平均書き込みビット数が削減できる．

6.6.2 書き込みビット数評価実験

マルチメディア系の音声や画像を生成するベンチマークである MediaBench [26] を SimpleScalar [1] を用いてトレースを取得した．対象の誤り訂正符号を用いて構成したメモリをトレースに適用して書き込みビット数を計測した．

表 6.1: 符号の性質.

	符号長 n [ビット]	誤り訂正能力 t [ビット]	最大書き込みビット	最小書き込みビット数	平均書き込みビット数
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	20	2	20	5	10.0 (1.00)
(23, 8, 5, 1)-REC [23]	23	2	11	5	8.92 (0.89)
(17, 8, 5, 1)-relaxed-REC	17	2	8	5	6.77 (0.68)
(25, 8, 5, 2)-REC [23]	25	2	12	5	8.67 (0.87)
(19, 8, 5, 2)-relaxed-REC	19	2	9	5	6.89 (0.69)
(29, 8, 5, 4)-REC [23]	29	2	12	5	7.80 (0.78)
(21, 8, 5, 4)-relaxed-REC	21	2	9	5	6.41 (0.64)
(37, 8, 5, 8)-REC [23]	37	2	10	5	7.18 (0.72)
(26, 8, 5, 8)-relaxed-REC	26	2	8	5	5.83 (0.58)

表 6.2: 書き込みビット数評価実験結果.

	adpcmE	adpcmD	epicE	epicD	g721E	g721D
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	2,902,865 (1.00)	1,022,642 (1.00)	1,213,777 (1.00)	670,655 (1.00)	3,030,135 (1.00)	1,140,475 (1.00)
(23, 8, 5, 1)-REC	2,492,757 (0.86)	974,215 (0.95)	1,240,812 (1.02)	695,669 (1.04)	2,625,590 (0.87)	1,073,272 (0.94)
(25, 8, 5, 2)-REC	2,303,056 (0.79)	929,465 (0.91)	1,168,265 (0.96)	649,003 (0.97)	2,426,246 (0.80)	1,012,208 (0.89)
(29, 8, 5, 4)-REC	2,146,736 (0.74)	873,537 (0.85)	1,111,116 (0.92)	629,212 (0.94)	2,264,785 (0.75)	929,645 (0.82)
(37, 8, 5, 8)-REC	2,085,224 (0.72)	811,245 (0.79)	1,038,296 (0.86)	592,397 (0.88)	2,199,998 (0.73)	888,672 (0.78)
(17, 8, 5, 1)-relaxed-REC	2,046,513 (0.70)	767,730 (0.75)	985,159 (0.81)	563,423 (0.84)	2,159,348 (0.71)	847,269 (0.74)
(19, 8, 5, 2)-relaxed-REC	1,993,295 (0.69)	774,639 (0.76)	996,721 (0.82)	570,375 (0.85)	2,105,284 (0.69)	854,668 (0.75)
(21, 8, 5, 4)-relaxed-REC	1,862,289 (0.64)	736,402 (0.72)	937,886 (0.77)	537,731 (0.80)	1,965,608 (0.65)	792,833 (0.70)
(26, 8, 5, 8)-relaxed-REC	1,774,395 (0.61)	677,714 (0.66)	878,235 (0.72)	510,916 (0.76)	1,873,010 (0.62)	745,191 (0.65)

メモリへの書き込みは同じビットは書き込まないものとする手法 [51] を想定する．表 6.2 に結果を示す．(17, 8, 5, 1)-relaxed-REC を用いて構成したメモリは (23, 8, 5, 1)-REC 符号を用いて構成したメモリと比較して最大 20.6%の書き込みビット数を削減した．(26, 8, 5, 8)-relaxed-REC を用いて構成したメモリは (15, 7, 5)-BCH 符号+(5, 1, 5) 反復符号を用いて構成したメモリと比較して最大 38.9%の書き込みビット数を削減した．

6.6.3 メモリ評価実験

メモリのエンコーダ/デコーダをハードウェア記述言語 Verilog HDL で記述し，Synopsys 社の Design Compiler のトポグラフィカルモードを用いて論理合成した．セルライブラリは STARC90nm を用いた．エンコーダ/デコーダは REC 符号と同じ構成である．エンコーダ/デコーダで用いるルックアップテーブルの面積とエネルギーは MRAM を仮定し，あらかじ

表 6.3: ルックアップテーブルのセル数.

	エンコーダ	デコーダ
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	0	1,808
(23, 8, 5, 1)-REC	512	147,456
(25, 8, 5, 2)-REC	3,072	327,684
(29, 8, 5, 4)-REC	20,480	1,572,880
(37, 8, 5, 8)-REC	589,824	33,554,688
(17, 8, 5, 1)-relaxed-REC	512	2,304
(19, 8, 5, 2)-relaxed-REC	3,072	5,124
(21, 8, 5, 4)-relaxed-REC	20,480	6,160
(26, 8, 5, 8)-relaxed-REC	589,824	16,640

表 6.4: 面積評価結果.

	エンコーダ $[um^2]$		デコーダ $[um^2]$		合計 $[um^2]$
	本体	LUT	本体	LUT	
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	108.0	-	2.1	1,115	1,225
(23, 8, 5, 1)-REC	174.3	316	45.2	90,980	91,515
(25, 8, 5, 2)-REC	203.9	1,895	79.7	202,181	204,360
(29, 8, 5, 4)-REC	136.9	12,636	86.1	970,467	983,326
(37, 8, 5, 8)-REC	129.8	363,921	86.1	20,703,242	21,067,379
(17, 8, 5, 1)-relaxed-REC	245.5	316	79.7	1,421	2,062
(19, 8, 5, 2)-relaxed-REC	263.2	1,895	79.7	3,161	5,399
(21, 8, 5, 4)-relaxed-REC	264.6	12,636	86.1	3,800	16,787
(26, 8, 5, 8)-relaxed-REC	145.4	363,921	92.4	10,266	374,426

めデータが保存されているとする．実装したメモリの面積と MediaBench を動作させたときのエネルギーを評価した．

ルックアップテーブルの 1 ビットあたりの面積は $6.17 \times 10^{-1} um^2$ とする [10]．ルックアップテーブルのセル数を表 6.3 に示す．論理合成したエンコーダ本体とルックアップテーブル部分の面積を表 6.4 に示す．Relaxed-REC 符号を用いて構成したメモリの面積は既存の誤り訂正符号を用いて構成したメモリと比較して最大で 305 倍となった．REC 符号の面積は既存の誤り訂正と比較して最大で 17,000 倍だったため，REC 符号と比較すると大きく面積を削減できた．

次に回路のエネルギー評価を示す．メモリへの書き込みは同じ値は書き込まないものとす

る手法 [51] を想定する．N_r，N_w，B_w をアプリケーション動作時の総読み込みアクセス回数，総書き込みアクセス回数，総書き込みビット数とする．メモリは動作周波数が 100MHz の MRAM を想定した．MRAM の 1 ビットあたりのリードエネルギー E_r は 1.06×10^{-15}J/bit，ライトエネルギー E_w は 2.60×10^{-12}J/bit とする [10]．表 6.5 にメディアベンチアプリケーションの N_r，N_w を示す．B_w は表 6.2 で与えられる．L_c を符号長とする．

BCH 符号+反復符号を用いて構成したメモリのエネルギー E_t は式 (6.13) で，Relaxed-REC 符号を用いて構成したメモリのエネルギー E_t は式 (6.14) でそれぞれ与えられる．

$$E_t = (N_r \times L_c \times E_r) + (B_w \times E_w) + E_o \tag{6.13}$$

$$E_t = \{(N_r + N_w) \times L_c \times E_r\} + (B_w \times E_w) + E_o \tag{6.14}$$

初項はメモリから符号語を読み出すエネルギー，第二項はメモリへ符号語を書き込むエネルギー，第三項はエンコーダ/デコーダのオーバーヘッドである．Relaxed-REC 符号では，書き込む符号語をメモリに書き込まれている符号語から決定するためにメモリから符号語の読み出しが必要である．エンコーダ/デコーダのオーバーヘッドは式 (6.15) で求める．

$$\begin{aligned} E_o &= N_w \times (d_E \times P_{dynamic,E} + P_{leakage,E}/100\text{MHz}) \\ &+ N_w \times \{(r+1) \times E_r\} \\ &+ N_r \times (d_D \times P_{dynamic,D} + P_{leakage,D}/100\text{MHz}) \\ &+ N_r \times \{(k_p+1) \times E_r\} \end{aligned} \tag{6.15}$$

d_E と d_D はエンコーダとデコーダのクリティカルパス遅延，$P_{dynamic,E}$ と $P_{dynamic,D}$ はエンコーダとデコーダのダイナミック電力，$P_{leakage,E}$ と $P_{leakage,D}$ はエンコーダとデコーダのリーク電力を表す．$d_E \times P_{dynamic,E}$，$d_D \times P_{dynamic,D}$ はそれぞれエンコーダ/デコーダのダイナミックエネルギー，$P_{leakage,E}$/100MHz，$P_{leakage,D}$/100MHz はそれぞれエンコーダ/デコーダのリークエネルギーを表す [6]．表 6.6 に各符号を用いて構成したメモリのエンコーダ/デコーダのクリティカルパス遅延，ダイナミック電力，リーク電力を示す．第二項と第四項は，エンコーダ/デコーダで用いているルックアップテーブルから値を読み出すエネルギーを表す．

表 6.7 にエネルギー評価実験結果を示す．(17, 8, 5, 1)-relaxed-REC を用いて構成したメモ

表 6.5: MediaBench の N_r と N_w.

	adpcmE	adpcmD	epicE	epicD	g721E	g721D
N_r	147,756	368,985	150,462	128,829	162,925	384,238
N_w	450,692	229,316	357,576	276,827	482,019	260,804

表 6.6: エンコーダ/デコーダのクリティカルパス遅延/ダイナミック電力/リーク電力.

	エンコーダ			デコーダ		
	d_E [ns]	$P_{dynamic,E}$ [μW]	$P_{leakage,E}$ [μW]	d_D [ns]	$P_{dynamic,D}$ [μW]	$P_{leakage,D}$ [μW]
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	0.64	35.3	0.96	0.29	0.30	0.018
(23, 8, 5, 1)-REC	0.52	78.9	1.73	0.38	12.7	0.26
(25, 8, 5, 2)-REC	0.51	82.2	2.03	0.33	31.5	0.79
(29, 8, 5, 4)-REC	0.60	55.2	1.31	0.33	34.3	0.87
(37, 8, 5, 8)-REC	0.58	54.0	1.26	0.33	34.2	0.87
(17, 8, 5, 1)-relaxed-REC	0.50	109.9	2.46	0.33	31.8	0.79
(19, 8, 5, 2)-relaxed-REC	0.50	109.3	2.62	0.33	31.4	0.79
(21, 8, 5, 4)-relaxed-REC	0.50	114.3	2.64	0.33	33.9	0.87
(26, 8, 5, 8)-relaxed-REC	0.51	58.1	1.35	0.33	36.9	0.94

リは (23, 8, 5, 1)-REC 符号を用いて構成したメモリと比較して最大 20.0%のエネルギーを削減した．(26, 8, 5, 8)-relaxed-REC を用いて構成したメモリは (15, 7, 5)-BCH 符号+(5, 1, 5) 反復符号を用いて構成したメモリと比較して最大 38.3%のエネルギーを削減した.

6.7 本章のまとめ

本章では，すべてのビットが 1 の符号語 $11\cdots1$ を含む必要のない Relaxed-線形組織誤り訂正符号を用いた Relaxed-REC 符号の構成手法を提案した．提案手法を用いて符号を生成すると，符号長が n ビット，元となる誤り訂正符号の情報量が k_p のとき，メモリに符号語を書き込むときの最大書き込みビット数は $n - \lceil k_p/2 \rceil$ に制約でき，平均書き込みビット数を削減できる．提案アルゴリズムにより生成した Relaxed-REC 符号を用いて構成したメモリをアプリケーションに適用した結果，既存の REC 符号を用いて構成したメモリに比べ最大 20.0%，BCH 符号と反復符号を連接した符号を用いて構成したメモリに比べ最大 38.3%のエネルギーを削減した.

表 6.7: エネルギー評価実験結果 (単位：nJ).

		adpcmE	adpcmD	epicE	epicD	g721E	g721D
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	リード	3.1	7.8	3.2	2.7	3.5	8.1
	ライト	7,547.4	2,658.9	3,155.8	1,743.7	7,878.4	2,965.2
	オーバーヘッド	15.8	10.6	12.8	10.1	17.0	11.8
	合計	7,566.4 (1.00)	2,677.3 (1.00)	3,171.9 (1.00)	1,756.5 (1.00)	7,898.8 (1.00)	2,985.1 (1.00)
(23, 8, 5, 1)-REC	リード	14.6	9.0	3.7	3.1	4.0	9.4
	ライト	6,481.2	2,533.0	3,226.1	1,808.7	6,826.5	2,790.5
	オーバーヘッド	29.6	19.7	24.0	18.8	31.7	21.9
	合計	6,525.3 (0.86)	2,561.7 (0.96)	3,253.8 (1.03)	1,830.7 (1.04)	6,862.2 (0.87)	2,821.8 (0.95)
(25, 8, 5, 2)-REC	リード	15.9	9.8	4.0	3.4	4.3	10.2
	ライト	5,987.9	2,416.6	3,037.5	1,687.4	6,308.2	2,631.7
	オーバーヘッド	33.9	25.1	27.8	21.9	36.4	27.6
	合計	6,037.7 (0.80)	2,451.5 (0.92)	3,069.3 (0.97)	1,712.7 (0.98)	6,349.0 (0.80)	2,669.5 (0.89)
(29, 8, 5, 4)-REC	リード	18.4	11.3	4.6	4.0	5.0	11.8
	ライト	5,581.5	2,271.2	2,888.9	1,636.0	5,888.4	2,417.1
	オーバーヘッド	27.9	22.6	23.1	18.2	30.0	24.7
	合計	5,627.8 (0.74)	2,305.1 (0.86)	2,916.6 (0.92)	1,658.1 (0.94)	5,923.4 (0.75)	2,453.5 (0.82)
(37, 8, 5, 8)-REC	リード	23.5	14.5	5.9	5.1	6.4	15.1
	ライト	5,421.6	2,109.2	2,699.6	1,540.2	5,720.0	2,310.5
	オーバーヘッド	28.8	23.0	23.8	18.8	30.9	25.1
	合計	5,473.8 (0.72)	2,146.7 (0.80)	2,729.2 (0.86)	1,564.0 (0.89)	5,757.3 (0.73)	2,350.8 (0.79)
(17, 8, 5, 1)-relaxed-REC	リード	10.8	6.6	2.7	2.3	2.9	6.9
	ライト	5,320.9	1,996.1	2,561.4	1,464.9	5,614.3	2,202.9
	オーバーヘッド	40.8	28.6	33.2	26.1	43.7	31.6
	合計	5,372.5 (0.71)	2,031.4 (0.76)	2,597.4 (0.82)	1,493.3 (0.85)	5,661.0 (0.72)	2,241.4 (0.75)
(19, 8, 5, 2)-relaxed-REC	リード	12.1	7.4	3.0	2.6	3.3	7.7
	ライト	5,182.6	2,014.1	2,591.5	1,483.0	5,473.7	2,222.1
	オーバーヘッド	42.3	29.4	34.5	27.0	45.4	32.5
	合計	5,236.9 (0.69)	2,050.9 (0.77)	2,629.0 (0.83)	1,512.6 (0.86)	5,522.4 (0.70)	2,262.4 (0.76)
(21, 8, 5, 4)-relaxed-REC	リード	13.3	8.2	3.3	2.9	3.6	8.6
	ライト	4,842.0	1,914.6	2,438.5	1,398.1	5,110.6	2,061.4
	オーバーヘッド	44.7	31.1	36.4	28.5	48.0	34.3
	合計	4,900.0 (0.65)	1,953.9 (0.73)	2,478.3 (0.78)	1,429.5 (0.81)	5,162.2 (0.65)	2,104.3 (0.70)
(26, 8, 5, 8)-relaxed-REC	リード	16.5	10.2	4.1	3.6	4.5	10.6
	ライト	4,613.4	1,762.1	2,283.4	1,328.4	4,869.8	1,937.5
	オーバーヘッド	28.6	23.4	23.7	18.7	30.8	25.6
	合計	4,658.6 (0.62)	1,795.6 (0.67)	2,311.3 (0.73)	1,350.7 (0.77)	4,905.1 (0.62)	1,973.6 (0.66)

第7章　結論

本論文では，不揮発メモリの書き込みビット数削減と誤り訂正を同時に実現する手法を提案した．

第2章**「研究動向」**では，書き込みビット数削減手法と誤り訂正手法を紹介した．本論文では，書き込みビット数削減手法の1つである Early Write Termination を用いることを前提とした．書き込みビット数削減手法の Early Write Termination，Flip-N-Write，最大ハミング距離を制限した符号，書き込み削減符号では書き込みビット数は削減できるが，誤り訂正を考慮していないことを述べた．また，誤り訂正手法の既存研究としてハミング符号を紹介し，ハミング符号はメモリにデータを書き込む際の書き込みビット数の削減を考慮していないことを述べた．

第3章**「ドーナツ符号」**では，最悪書き込みビット数削減と誤り訂正を同時に実現する符号として拡張ドーナツ符号を提案した．最初に，最大ハミング距離と最小ハミング距離を制約する符号として偶数パリティ符号を用いたドーナツ符号を提案した．しかし，ドーナツ符号単体では効果的な書き込みビット数削減と誤り訂正能力を有さない．そこで，ドーナツ符号の最大ハミング距離と最小ハミング距離をより現実的な制約とするために符号の性質を拡張する符号拡張手法を提案した．符号拡張手法は，符号を連接させることで新たな符号 (拡張符号) を生成する手法である．実際にドーナツ符号に符号拡張手法を適用することで，ハミング符号と同等の誤り訂正能力を持ち，最悪書き込みビット数が削減される拡張ドーナツ符号が生成できる．

拡張ドーナツ符号を用いて構成した不揮発メモリを計算機上で実装し，書き込みビット数を評価した結果を示した．書き込みビット数評価実験の結果から，拡張ドーナツ符号を用いて構成した不揮発メモリはハミング符号を用いて構成した不揮発メモリと比較して最大35.7%の最悪書き込みビット数を削減した．また，拡張ドーナツ符号を用いたメモリの回路を実装

し，エンコーダ/デコーダならびに不揮発メモリのエネルギーを評価した結果を示した．エネルギー評価実験の結果から，拡張ドーナツ符号を用いて構成した不揮発メモリはハミング符号を用いて構成した不揮発メモリと比較して最大 35.7%のエネルギーを削減した．最後に拡張ドーナツ符号は入力データの反転ビット数が大きい場合に有効であることを確認した．

第 4 章**「一対多符号」**では，最悪書き込みビット数削減と誤り訂正を同時に実現する符号として一対多符号を提案した．一対多符号は誤り訂正符号を元に，値と符号語を一対多に対応させた符号である．一対多符号は最悪書き込みビット数を最小化し，誤り訂正能力を維持したまま最大誤り訂正能力が増加することを示した．

一対多符号を用いて構成した不揮発メモリを計算機上で実装し，書き込みビット数を評価した結果を示した．書き込みビット数評価実験の結果から，一対多符号を用いて構成した不揮発メモリはハミング符号を用いて構成した不揮発メモリと比較して最大 57.1%の最悪書き込みビット数を削減した．また，一対多符号を用いたメモリの回路を実装し，エンコーダ/デコーダならびに不揮発メモリのエネルギーを評価した結果を示した．エネルギー評価実験の結果から，一対多符号を用いて構成した不揮発メモリはハミング符号を用いて構成した不揮発メモリと比較して最大 56.8%のエネルギーを削減した．

第 5 章**「REC 符号」**では，平均書き込みビット数削減と誤り訂正を同時に実現する符号として REC 符号を提案した．最初に，元となる誤り訂正符号の符号語をクラスタリングして，各クラスタに値を割り当てることで，値と符号語が 1 対多に対応した符号が生成される．生成された符号は書き込みビット数削減と誤り訂正を実現する符号であることを証明した．次に符号を生成するための効果的なクラスタリング手法を提案した．提案手法を用いて生成した符号を REC 符号と呼ぶ．REC 符号では訂正能力を維持したまま，メモリに符号語を書き込むときの最大書き込みビット数を符号長の半分以下に制約でき，平均書き込みビット数を削減できることを示した．

REC 符号を用いて構成した不揮発メモリを計算機上で実装し，書き込みビット数を評価した結果を示した．書き込みビット数評価実験の結果から，REC 符号を用いて構成した不揮発メモリは BCH 符号と反復符号を連接した符号を用いて構成した不揮発メモリと比較して，平均書き込みビット数を最大 28.2%削減した．また，REC 符号を用いたメモリの回路を

実装し，エンコーダ/デコーダならびに不揮発メモリのエネルギーを評価した結果を示した．エネルギー評価実験の結果から，REC 符号を用いて構成した不揮発メモリは BCH 符号と反復符号を連接した符号を用いて構成した不揮発メモリと比較して最大 27.5%のエネルギーを削減した．REC 符号は書き込みビット数削減と誤り訂正を同時に実現する効果的な符号であることを示した． REC 符号は線形組織誤り訂正符号を元にして生成するが，この線形組織誤り訂正符号にはすべてのビットが 1 の符号語 $11\cdots1$ を含む必要があるという制約がある．REC 符号ではこの制約を満たすために符号長が増加するという欠点がある．

第 6 章**「Relaxed-REC 符号」**では，REC 符号の元となる線形組織誤り訂正符号の制約を緩めた Relaxed-REC 符号を提案した．Relaxed-REC 符号で用いる線形組織誤り訂正符号はすべてのビットが 1 の符号語 $11\cdots1$ を含む制約がないため，REC 符号に比べて符号長が小さく，それに伴い平均書き込みビット数も大きく削減できる．Relaxed-REC 符号を生成するための新たなクラスタリング手法を提案した．提案手法を用いて生成した Relaxed-REC 符号では訂正能力を維持したまま，メモリに符号語を書き込むときの最大書き込みビット数を制約し，平均書き込みビット数を削減できることを示した．

Relaxed-REC 符号を用いて構成した不揮発メモリを計算機上で実装し，書き込みビット数を評価した結果を示した．書き込みビット数評価実験の結果から，Relaxed-REC 符号を用いて構成したメモリは BCH 符号と反復符号を連接した符号を用いて構成した不揮発メモリと比較して，平均書き込みビット数を最大 38.9%削減した．また，Relaxed-REC 符号を用いたメモリの回路を実装し，エンコーダ/デコーダならびに不揮発メモリのエネルギーを評価した結果を示した．エネルギー評価実験の結果から，Relaxed-REC 符号を用いて構成したメモリは BCH 符号と反復符号を連接した符号を用いて構成した不揮発メモリと比較して最大 38.3%のエネルギーを削減した．

表 7.1 に既存符号と提案符号の性質の比較を示す．表 7.1 より，Relaxed-REC 符号は他の符号に比べ，符号長の増大を抑えつつ平均書き込みビット数が大きく削減できる．よって，Relaxed-REC 符号は本論文で提案した符号の中では最も性能が高いと考える．ただし，無視できるほどではあるが Relaxed-REC 符号では既存符号や拡張ドーナツ符号に比べてエンコード/デコードが複雑となる．

表 7.1: 符号の性質の比較.

	符号長	情報量	誤り訂正能力	最大書き込みビット数	最小書き込みビット数	平均書き込みビット数
(7, 4, 3) ハミング符号	7	4	1	7	3	3.50 (1.00)
拡張ドーナツ符号	9	4	1	6	3	4.50 (1.29)
一対多符号	10	4	1–2	4	3	3.25 (0.93)
(9, 4, 3, 1)-REC	9	4	1	4	3	3.25 (0.93)
(10, 4, 3, 2)-REC	10	4	1	4	3	3.12 (0.89)
(12, 4, 3, 4)-REC	12	4	1	4	3	2.93 (0.83)
(15, 7, 5)-BCH 符号 + (5, 1, 5) 反復符号	20	8	2	20	5	10.0 (1.00)
(23, 8, 5, 1)-REC	23	8	2	11	5	8.92 (0.89)
(25, 8, 5, 2)-REC	25	8	2	12	5	8.67 (0.87)
(29, 8, 5, 4)-REC	29	8	2	12	5	7.80 (0.78)
(37, 8, 5, 8)-REC	37	8	2	10	5	7.18 (0.72)
(17, 8, 5, 1)-relaxed-REC	17	8	2	8	5	6.77 (0.68)
(19, 8, 5, 2)-relaxed-REC	19	8	2	9	5	6.89 (0.69)
(21, 8, 5, 4)-relaxed-REC	21	8	2	9	5	6.41 (0.64)
(26, 8, 5, 8)-relaxed-REC	26	8	2	8	5	5.83 (0.58)

本論文で提案した符号はいずれも符号長が増える方向に設計しており，メモリに用いる符号の符号長が増大することで，メモリの面積が増大する．現在，主にキャッシュに用いられている SRAM を不揮発メモリである MRAM に代替することが本論文の目標のひとつである．本論文の評価実験では MRAM 同士を比較しているが，SRAM と MRAM を比較した場合には既存符号を用いて構成した SRAM に対して提案符号を用いて構成した MRAM の面積が下回っていることが必要だと考えられる．SRAM のメモリセルサイズは MRAM のメモリセルサイズの最大 30 倍と言われている．一般に符号長とメモリの面積は比例関係にあるため，提案符号の符号長が既存符号の符号長に比べて最大 30 倍を下回っていれば問題ないと考える．実際に，本論文で提案した符号の符号長は既存符号の符号長と比べて，(15, 7, 5)-BCH 符号+(5, 1, 5) 反復符号と (37, 8, 5, 8)-REC 符号を比較した場合で最大 1.85 倍である．つまり，本論文で提案した符号の符号長は既存符号の符号長と比べて最大でも 30 倍を下回っているため，符号長が増える方向に設計はしているもののハードウェア設計においては問題ないと考える．また，一般的なメモリでは面積増大によりダイナミック電力とリーク電力の増加が考えられる．ダイナミック電力のうち，ライトエネルギーは書き込みビット数に関係し，リードエネルギーは符号長に関係する．符号長増加によってリードエネルギーは増加するが，MRAM のリードエネルギーはライトエネルギーの約 1/1000 と極めて小さく，総エネ

ルギーはライトエネルギーが支配的となる．また，MRAM の使用時のリーク電力はほぼ 0 に近く，さらに不揮発メモリの特性を利用して使用していないときには電力を供給しない．以上より，面積増大による消費エネルギー増加は非常に小さいと考える．

以下に今後の課題を挙げる．

マルチレベルセル MRAM

本論文では，メモリセルに 1 ビットのデータを保存するシングルレベルセルを対象とした．それに対して，1 つのメモリセルに多ビットのデータを保存するマルチレベルセルがある [48]．マルチレベルセルの MRAM はシングルレベルセルの MRAM よりも高集積化できる一方，1 セルあたりに書き込むエネルギーは増加する．また，シングルレベルセル MRAM よりも書き込みエネルギーと書き込みエラー率のトレードオフがより厳しくなり，そのためシングルレベルセル MRAM よりも書き込みエラー率が上がると言われている．そこで，マルチレベルセルの MRAM に対応する書き込み削減誤り訂正符号を考案することで上記問題を解決できる可能性がある．

本論文での提案手法は書き込みエネルギー増大とエラー増大への対応を目標とする手法であるため，マルチレベルセル MRAM と相性が良いと考えられる．そこで，今後の課題として本論文で提案した符号をマルチレベルセル MRAM 用に最適化することが考えられる．また，今回は情報量 4 ビットに対して 1 ビットの誤り訂正能力を担保したが，マルチレベルセル MRAM で書き込みエラー率が上がった場合には誤り訂正能力の強化が必要になると考えられる．

面積削減

本論文では，省電力化と信頼性向上に焦点を置いて符号構成手法を提案した．しかし本論文で提案したいずれの手法も既存手法と比較して面積が増大した．SRAM を MRAM に代替することを仮定すると増加した面積は許容できる範囲ではあるが，ワーキングメモリでは面積は小さければ小さいほど良いとされる．そこで今後は，面積効率の良い書き込み削減誤り訂正符号を考えたい．

アプリケーションの局所性

動作するアプリケーションによってビット反転の局所性が生じる可能性があるため，局

所性を考慮して符号を選択することでより省電力化できる可能性がある．今後の課題として，このような符号の研究，もしくは値と符号語の効果的なマッピング方法の研究が考えられる．

謝辞

本研究は，筆者が早稲田大学大学院基幹理工学研究科博士後期課程在学中に，早稲田大学大学院基幹理工学研究科 戸川望教授の指導のもとに行ったものである．

本論文の執筆にあたり多大なる御指導，御助言を賜りました戸川望教授に心より感謝いたします．学部，修士課程，博士後期課程の 6 年間にわたり丁寧にご指導頂き多くを学び成長することができました．同じく常日頃から適切なご指導を頂きました早稲田大学大学院基幹理工学研究科 柳澤政生教授に深く御礼申し上げます．研究に関する鋭いご指摘を頂くことで多くの課題を発見することができました．加えて本論文に関する貴重な御助言を頂きました早稲田大学大学院基幹理工学研究科 前原文明教授に深く感謝致します．お忙しい中多岐に亘るご指導，ご指摘を頂き，本論文の質を高めることができました．

また，研究全般に亘り指導ならびに貴重な御助言を頂きました早稲田大学大学院基幹理工学研究科 史又華教授に心より感謝申し上げます．常日頃から的確なご指摘を頂き，研究の質を高めることが出来ました．終始，適切な御指導ならびに御助言を頂きました，本学講師の多和田雅師氏に深く感謝いたします．学部時代より研究に関する鋭いご意見を頂き，研究者としての姿勢を学ぶことができました．研究室同期の大屋優氏，藤原晃一氏，井川昂輝氏，伊東光希氏，北山遼育氏，島崎健太氏，鮑思雅氏に謝意を表します．日常の議論を通じて多くの知識や示唆を頂き，また公私にわたって多くの時間を共有でき充実した研究生活を送ることができました．そして研究生活を温かく見守りそして辛抱強く支援してくださった家族・親戚・友人に深い感謝の意を表します．

最後に，日頃より多方面にわたり様々な御意見を頂き支援していただいた戸川研究室，柳澤研究室の皆様に心より感謝致します．この論文を新たな出発として，今後さらに精進することで，皆様へのご恩返しとしたいと存じます．

参考文献

[1] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *IEEE Trans. Computers*, vol. 35, issue 2, pp. 59–67, 2002.

[2] B. D. Bel, J. Kim, C. H. Kim *et al.*, "Improving STT-MRAM density through multibit error correction," in *Proc. DATE 2014*, pp. 1–6, 2014.

[3] X. Bi, Z. Sun, H. Li *et al.*, "Probabilistic design methodology to improve run-time stability and performance of STT-RAM caches," in *Proc. ICCAD 2012*, pp. 88–94, 2012.

[4] P. Bikki and P. Karuppanan, "SRAM cell leakage control techniques for ultra low power application: A survey," *Circuits and Systems*, vol. 8, pp. 23–52, 2017.

[5] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, 1960.

[6] T. D. Burd and R. W. Brodersen, "Energy efficient design," in *Energy Efficient Microprocessor Design*, pp. 7–44, Springer Science & Business Media, New York, 2012.

[7] E. Chen, D. Apalkov, Z. Diao *et al.*, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Trans. on Magnetics*, vol. 46, pp. 1873–1878, 2010.

[8] S. Cho and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. MICRO 2009*, pp. 347–357, 2009.

[9] S.-W. Chung, T. Kishi, J. W. Park *et al.*, "4Gbit density STT-MRAM using perpendicular MTJ realized with compact cell structure," in *Proc. IEDM 2016*, pp. 27.1.1–27.1.4, 2016.

[10] X. Dong, X. Wu, G. Sun *et al.*, "Circuit and microarchtechture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement," in *Proc. DAC 2008*, pp. 554–559, 2008.

[11] F. Frustaci, M. Khayatzadeh, D. Blaauw *et al.*, "SRAM for error-tolerant applications with dynamic energy-quality management in 28 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, pp. 1310–1323, 2015.

[12] S. Gravano, M. C. Doggett, and P. J. Mcdougall, "Comparison of a cyclic code and a repetition code with the same code rate in the presence of single-bit errors," *Int. Journal of Electronics*, vol. 67, no. 4, pp. 495–502, 1989.

[13] A. Guler and N. K. Jha, "Ultra-low-leakage, robust FinFET SRAM design using multiparameter asymmetric FinFETs," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, pp. 1–25, 2017.

[14] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[15] G. Hu, J. J. Nowak, G. Lauer *et al.*, "Low-current spin transfer torque MRAM," in *Proc. VLSI-DAT 2017*, pp. 1–2, 2017.

[16] A. Jog, A. K. Mishra, C. Xu *et al.*, "Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. DAC 2012*, pp. 243–252, 2012.

[17] E. Kültürsay, M. Kandemir, A. Sivasubramaniam *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. ISPASS 2013*, pp. 256–267, 2013.

[18] P. Kisos, G. Kostopoulos, N. Sklavos *et al.*, "Hardware implementation of the RC4 stream cipher," in *Proc. ISCAS 2003*, vol. 3, pp. 1363–1366, 2003.

[19] A. Klinefelter, N. E. Roberts, Y. Shakhsheer *et al.*, "A 6.45 μ W self-powered IoT SoC with integrated energy-harvesting power management and ULP asymmetric radios," in *Proc. ISSCC 2015*, pp. 384–386, 2015.

[20] T. Kojo, M. Tawada, M. Yanagisawa *et al.*, "A write-reducing and error-correcting code generation method for non-volatile memories," in *Proc. APCCAS 2014*, pp. 304–307, 2014.

[21] ——, "Bit-write-reducing and error-correcting code generation by clustering error-correcting codewords for non-volatile memories," in *Proc. ICCAD 2015*, pp. 682–689, 2015.

[22] ——, "Code generation limiting maximum and minimum hamming distances for non-volatile memories," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 12, pp. 2484–2493, 2015.

[23] ——, "A bit-write-reducing and error-correcting code generation method by clustering ecc codewords for non-volatile memories," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99-A, no. 12, pp. 2398–2411, 2016.

[24] ——, "A relaxed bit-write-reducing and error-correcting code for non-volatile memories," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101-A, no. 7, pp. 1045–1052, 2018.

[25] B. C. Lee, E. Ipek, O. Mutlu *et al.*, "Architecting phase change memory as a scalable dram alternative," in *Proc. ISCA 2009*, vol. 37, pp. 2–13, 2009.

[26] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," in *Proc. MICRO 1997*, pp. 330–335, 1997.

[27] J. Liu, B. Jaiyen, R. Veras *et al.*, "RAIDR: Retention-aware intelligent dram refresh," in *Proc. ISCA 2012*, vol. 40, pp. 1–12, 2012.

[28] J. A. Mandelman, R. H. Dennard, G. B. Bronner *et al.*, "Challenges and future directions for the scaling of dynamic random-access memory (DRAM)," *IBM Journal of Research and Development*, vol. 46, pp. 187–212, 2002.

[29] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 27, pp. 1537–1550, 2015.

[30] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches," *IEEE Trans. on Parallel and Distributed Systems*, vol. 26, pp. 1524–1537, 2015.

[31] K. Mohanram and S. Rixner, "Context-independent codes for off-chip interconnects," in *Proc. PACS 2004*, pp. 107–119, 2004.

[32] G. E. Moore, "Cramming more components onto integrated circuits," *Electric Magazine*, vol. 38, no. 8, 1965.

[33] P. Nielsen and N. Bashara, "The reversible voltage-induced initial resistance in the negative resistance sandwich structure," *IEEE Trans. on Electron Devices*, vol. 11, pp. 243–244, 1964.

[34] F. Oboril, F. Hameed, R. Bishnoi *et al.*, "Normally-off STT-MRAM cache with zero-byte compression for energy efficient last-level caches," in *Proc. ISLPED 2016*, pp. 236–241, 2016.

[35] H. Ohnsorge, "Linear systematic code encoding and detecting devices," U.S. Patent 3 512 150, May 12, 1965.

[36] S. Raoux, G. W. Burr, M. J. Breitwisch *et al.*, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, pp. 465–479, 2008.

[37] W. Rujia, J. Lei, Z. Youtao *et al.*, "Selective restore: an energy efficient read disturbance mitigation scheme for future STT-MRAM," in *Proc. DAC 2015*, pp. 1–6, 2013.

[38] B. Schneir, "Description of a new variable-length key, 64-bit block cipher (blowfish)," in *Proc. FSE 1994*, vol. 809, pp. 191–204, 1994.

[39] Z. Shao and Y.-H. Chang, "Non-volatile memory (NVM) technologies," *Journal of Systems Architecture*, vol. 71, pp. 1, 2016.

[40] P. Singh and S. K. Vishvakarma, "Ultra-low power, process-tolerant 10T (PT10T) SRAM with improved read/write ability for internet of things (IoT) applications," *Journal of Low Power Electronics and Applications*, vol. 7, no. 3, 2017.

[41] Y. J. Song, J. H. Lee, H. C. Shin *et al.*, "Highly functional and reliable 8Mb STT-MRAM embedded in 28nm logic," in *Proc. IEDM 2016*, pp. 27.2.1–27.2.4, 2016.

[42] G. Strawn and C. Strawn, "Moore's law at fifty," *IT Professional*, vol. 17, pp. 69–72, 2015.

[43] R. Takemura, T. Kawahara, K. Ono *et al.*, "Highly-scalable disruptive reading and restoring scheme for Gb-scale SPRAM and beyond," *Solid-State Electronics*, vol. 58, pp. 28–33, 2011.

[44] M. Tawada, S. Kimura, M. Yanagisawa *et al.*, "Ecc-based bit-write reduction code generation for non-volatile memory," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 12, pp. 2494–2504, 2015.

[45] J. Wang, X. Dong, and Y. Xie, "Building and optimizing MRAM-based commodity memories," *ACM Trans. Archit. Code Optim.*, vol. 11, pp. 36.1–36.22, 2015.

[46] W. Wen, Y. Zhang, and J. Yang, "Read error resilient MLC STT-MRAM based last level cache," in *Proc. ICCD 2017*, pp. 455–462, 2017.

[47] W. Wujie, Z. Yaojun, M. Mengjie *et al.*, "State-restrict MLC STT-RAM designs for high-reliable high-performance memory system," in *Proc. DAC 2014*, pp. 1–6, 2014.

[48] C. Xunchao, K. Navid, Z. Jian *et al.*, "AOS: Adaptive overwrite scheme for energy-efficient MLC STT-RAM cache," in *Proc. DAC 2016*, pp. 1–6, 2016.

[49] Y. Zhang, F. Zhang, Y. Shakhsheer *et al.*, "A batteryless 19 μ w MICS/ISM-Band energy harvesting body sensor node SoC for ExG applications," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 199–213, 2013.

[50] T. Zheng, J. Park, M. Orshansky *et al.*, "Variable-energy write STT-RAM architecture with bit-wise write-completion monitoring," in *Proc. ISLPED 2013*, pp. 229–234, 2013.

[51] P. Zhou, B. Zhao, J. Yang *et al.*, "Energy reduction for STT-RAM using write termination," in *Proc. ICCAD 2009*, pp. 264–268, 2009.

[52] J.-G. Zhu, "Magnetoresistive random access memory: The path to competitiveness and scalability," *Proceedings of the IEEE*, vol. 46, pp. 1786–1798, 2008.

[53] 古城辰朗，多和田雅師，柳澤政生，戸川望, "最大ハミング距離と最小ハミング距離を制約した符号による不揮発メモリの書き込み手法," 第 27 回 回路とシステムワークショップ, pp. 404–409, 2014.

[54] ——, "不揮発メモリを対象とした最大ハミング距離と最小ハミング距離を制約した符号による書き込み手法のエネルギー評価," 信学技報, VLD2014-105, vol. 114, no. 328, pp. 227–232, 2014.

[55] ——, "クラスタリングによる書き込みビット数削減と誤り訂正を実現する不揮発メモリを対象とした符号の構成手法," 情報処理学会 DA シンポジウム 2015 論文集, pp. 11–16, 2015.

[56] ——, "不揮発メモリを対象に最悪書込みビット数削減と誤り訂正を両立する一対多符号構成手法," 2015 年電子情報通信学会 ソサイエティ大会基礎・境界講演論文集, p. 52, 2015.

[57] ——, "不揮発メモリを対象に最悪書き込みビット数削減と誤り訂正を両立する一対多符号構成手法," 電子情報通信学会論文誌 A, vol. J99-A, no. 8, p. 52, 2016.

[58] 総務省, "IoT デバイスの急速な普及," *http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/html/nd111200.html*.

[59] 多和田 雅師, 木村 晋二, 柳澤 政生, 戸川 望, "最大ハミング距離を制限した符号とこれを用いた不揮発メモリの書き込み削減手法," 信学技報, VLD2013-28, vol. 113, no. 119, pp. 95–100, 2013.

本論文に関する発表業績

論文（学術誌原著論文）

1. ○ T. Kojo, M. Tawada, M. Yanagisawa, and N. Togawa, “A Relaxed Bit-Write-Reducing and Error-Correcting Code for Non-Volatile Memories,” *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101-A, no. 7, pp. 1045–1052, Jul. 2018.

2. ○ T. Kojo, M. Tawada, M. Yanagisawa, and N. Togawa, “A Bit-Write-Reducing and Error-Correcting Code Generation Method by Clustering ECC Codewords for Non-Volatile Memories,” *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99-A, no. 12, pp. 2398–2411, Dec. 2016.

3. ○ 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “不揮発メモリを対象に最悪書き込みビット数削減と誤り訂正を両立する一対多符号構成手法，” 電子情報通信学会論文誌 A, vol. J99-A, no. 8, pp. 336–340, Aug. 2016.

4. ○ T. Kojo, M. Tawada, M. Yanagisawa, and N. Togawa, “Code Generation Limiting Maximum and Minimum Hamming Distances for Non-Volatile Memories,” *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 12, pp. 2484–2493, Dec. 2015.

国際会議（査読付）

5. ○ T. Kojo, M. Tawada, M. Yanagisawa, and N. Togawa, “Bit-Write-Reducing and Error-Correcting Code Generation by Clustering Error-Correcting Codewords for Non-Volatile Memories,” in *Proc. 2015 International Conference On Computer Aided Design*, pp. 682–689, Nov. 2015.

6. ○ T. Kojo, M. Tawada, M. Yanagisawa, and N. Togawa, “A Write-Reducing and Error-Correcting Code Generation Method for Non-Volatile Memories,” in *Proc. 2014 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 304–307, Nov. 2014.

国内学会（査読付）

7. 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “クラスタリングによる書き込みビット数削減と誤り訂正を実現する不揮発メモリを対象とした符号の構成手法,” 情報処理学会 DA シンポジウム 2015 論文集, pp. 11–16, Aug. 2015.
8. 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “最大ハミング距離と最小ハミング距離を制約した符号による不揮発メモリの書き込み手法,” 第 27 回 回路とシステムワークショップ, pp. 404–409, Aug. 2014.

国内学会（査読無）

9. 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “不揮発メモリを対象に最悪書込みビット数削減と誤り訂正を両立する一対多符号構成手法，” 電子情報通信学会ソサイエティ大会 基礎・境界講演論文集, p. 52, Sep. 2015.
10. 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “不揮発メモリを対象とした最大ハミング距離と最小ハミング距離を制約した符号による書き込み手法のエネルギー評価,” 信学技報, VLD2014-105, vol. 114, no. 328, pp. 227–232, Nov. 2014.

国内学会（ポスター発表）

11. 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “クラスタリングによる書き込みビット数削減と誤り訂正を実現する不揮発メモリを対象とした符号の構成手法,” DA シンポジウム 2015, Aug. 2015.
12. 古城 辰朗, 多和田 雅師, 柳澤 政生, 戸川 望, “不揮発メモリを対象とした最大ハミング距離と最小ハミング距離を制約した符号による書き込み手法のエネルギー評価,” デザインガイア 2014, Nov. 2014.

受賞

13. 早稲田大学 基幹理工学研究科 情報理工・情報通信専攻 専攻賞 (本賞), 2016.
14. 早稲田大学 基幹理工学部 情報理工学科 学科賞, 2015.
15. 情報処理学会 SLDM 研究会 優秀発表学生賞, 2015.
16. 情報処理学会 SLDM 研究会 優秀論文賞, 2015.
17. 日本学生支援機構 大学院第一種奨学金 特に優れた業績による返還免除, 2015.

助成金

18. 電気通信普及財団 第 32 回電気通信普及財団賞 (テレコムシステム技術学生賞), 2016.
19. 公益財団法人 NEC C&C 財団 平成 27 年度後期国際会議論文発表者助成, 2015.