RC-011

巡回符号向きプロセッサの設計と評価
Design and Evaluation of a Processor for Cyclic Code

君家 一紀 † 浜辺 崇 † 坂主 圭史 † 武内 良典 † 今井 正治 †
Kazuki Ohya Takashi Hamabe Keishi Sakanushi Yoshinori Takeuchi Masaharu Imai

1. はじめに

今日, 無線通信技術は我々にとって必要不可欠なものとなっている. 技術の発達に伴い, 無線通信システムの普及が進んでおり, 今後も用途の拡大が期待されている. 一例として, 無線通信を利用したセンサシステムが挙げられる. センサシステムはセンサを用いて様々な情報を収集し活用するシステムで, 医療, 農業, セキュリティなどの分野への応用が研究されている [1–4].

図 1 に, 通信システムにおける通信手順の一例を示す. 情報通信ではノイズなどの影響により, 伝送路において情報が変化する通信誤りが起こりうる. ゆえに, 通信誤りを検知し, 通信される情報の信頼性を保証する誤り制御が必要となる. 誤り制御の手法として, FEC(前方誤り訂正：Forward Error Correction) 方式 [5] および ARQ(自動再送要求：Automatic Repeat reQuest) 方式 [6] の 2 つが広く用いられている. FEC 方式や ARQ 方式を実現する符号の一つとして巡回符号が知られている. 巡回符号は, 利用する生成多項式に応じてハミング距離を様々に変化させることが可能である. 例えば, パリティビット方式では, 生成多項式 $x+1$ を用いることで, ハミング距離が 2 の符号となり, 奇数個のビットの誤りを検出することができる. 一方, (23, 12) ゴレイ符号 [7] は, 生成多項式 $x^{11}+x^9+x^7+x^6+x^5+x+1$ を用いることで, ハミング距離が 7 の符号となり, 3bit までの誤りを訂正することができる. 要求されるハミング距離は, 通信環境や用途に依存するが, 巡回符号を用いることで様々な要求を満たしうる.

巡回符号には, 符号化, 復号, そして誤り訂正の 3 つの処理が存在するが, これらの処理は実行サイクル数が大きいという問題点を持つ. そこで本研究では, 任意の生成多項式に対応した, 巡回符号を効率よく処理する手法, およびこの手法を用いた巡回符号向きの ASIP（Application Specific Instruction-set Processor）を提案する.

本稿の構成は以下の通りである. 第 2 節では, 巡回符号およびその関連研究について述べる. 第 3 節では, 巡回符号を効率よく処理する手法, およびその実装について述べる. 第 4 節において, 行った評価実験とその結果について述べ, 最後に第 5 節で本研究をまとめる.

2. 巡回符号

本節では, 巡回符号とその関連研究について述べる. 巡回符号では, 生成多項式を用いて情報記号から検査記号を計算後, これを情報記号へと付加することで符号語を得る. 受信側では, 受信語からシンドロームを求め, 誤りが検出された場合, 誤り訂正を行う.

本稿で用いる生成多項式 $g(x)$ を式 (1) に示す.

$$g(x) = x^8 + x^2 + x + 1 \tag{1}$$

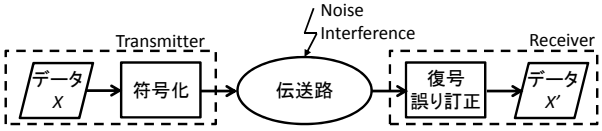

図 1: 通信システムにおける通信手順

式 (1) に示す $g(x)$ はセンサシステムへの応用が期待されている通信規格 IEEE1902.1 [8] で用いられており, 符号長が 127bit までの情報記号において, 1bit 誤りの訂正, および 2bit 誤りの検出が可能である.

2.1 アルゴリズム

本節では, 巡回符号における符号化, 復号, そして誤り訂正の 3 つの処理のアルゴリズムについて述べる. なお, より詳細な処理手順は文献 [9,10] に示されている.

2.1.1 符号化

符号化の処理を式 (2)(3) に示す. なお, $m(x)$, $c(x)$, $y(x)$ は, それぞれ情報記号 M, 検査記号 C, 符号語 Y を多項式表現したものを表す. 例えば, ビット列“ 10100001 ”は, 多項式表現 x^7+x^5+1 に対応する. また, 演算は GF(2) 上で行われるため, 加算および減算は XOR 演算によって実現される.

$$c(x) = (m(x)\cdot x^8) \bmod g(x) \tag{2}$$
$$y(x) = m(x)\cdot x^8 + c(x) \tag{3}$$

式 (2) は, 一度に処理可能なデータ量に応じて情報記号を分割して処理できる. 図 2 に, 32bit の情報記号を 4 つに分割して検査記号を計算する例を示す. ここで, 分割されたデータのうち, 上位から i 番目のデータを $m_i(x)$ とする. 符号化処理は, 繰り返し XOR 演算を必要とするため, 実行サイクル数が大きい.

2.1.2 復号

本稿では, 受信語 Y' からシンドローム S を求める処理を復号と呼ぶ. シンドロームは式 (5) により計算される. なお, $y'(x)$, $s(x)$, $e(x)$ はそれぞれ受信語 Y', シンドローム S, 外乱による通信誤り E を多項式表現したものを表す.

$$y'(x) = y(x) + e(x) \tag{4}$$
$$s(x) = y'(x) \bmod g(x) \tag{5}$$

受信側では, 復号により通信誤りの有無を判断する. すなわち $s(x)=0$ のとき, 通信誤りが発生しなかったと判断し, そうでなければ, 誤り訂正を行う.

†大阪大学大学院情報科学研究科

```
c(x) = 0;
for(i = 0; i < 4; i++){
    d(x) = c(x) + m_i(x);          —(A)
    c(x) = (d(x) · x^8) mod g(x); —(B)
}
```

図 2: 検査記号の計算

```
for(i = 0; i < 127; i++){
    d(x) = x^i mod g(x);
    if(d(x) == s(x)){break;}
}
return i;
```

図 3: 誤り位置の特定

2.1.3 誤り訂正

誤り訂正では, 復号により得られたシンドロームから誤り位置を特定し, 訂正する. 1bit 誤りが発生した際の, シンドロームの値と誤り位置 n との関係を式（6）に示す. なお, $s_n(x)$ は下位から n 番目に誤りが発生した際のシンドローム値を表す.

$$s_n(x) = x^n \bmod g(x)\ (0 \leq n \leq 126) \tag{6}$$

1bit 誤りの場合, $s_n(x)$ の値はすべて異なる. すなわち, シンドロームの値から誤り位置が一意に定まるため, 受信側は誤り位置を特定し, 訂正することができる. 一方, 2bit 誤りの場合, 誤り位置が一意に定まらないが, $s(x) \neq 0$ となるため, 誤りを検出することはできるが, 訂正できない.

誤り位置を特定する処理を図 3 に示す. 式（6）の結果とシンドロームを逐次比較していくことで, 誤り位置を特定できるが, 繰り返し剰余を計算する必要があるため, 実行サイクル数が大きい.

2.2 関連研究

巡回符号処理は実行サイクル数が大きいという問題点を持つため, これを解決する様々な手法が提案されている. 関連研究は, 以下の 3 つに分類できる.

・アルゴリズムの改良

文献 [11] は, 用いる生成多項式を限定することでアルゴリズムを改良し, 符号化と復号処理の高速化する手法を提案している. この手法では, 演算器やメモリを追加する必要はないが, 多くの生成多項式に適用できない.

・テーブルの利用

テーブルを利用することで処理を高速化する手法 [7, 12, 13] が提案されている. 図 2 に示した符号化処理において, 剰余を計算する処理 (B) が実行時間の大部分を占める. このため, テーブルに $d(x)$ の全 $256(= 2^8)$ パターンに対する処理 (B) の結果を格納することで, 剰余の計算を省略することができる. 図 4 に, 符号化と復号処理に用いるテーブルを, 図 5 にテーブルを用いた符号化処理を示す. テーブルは, 8bit データを入力とし, 処理（B) の結果を出力する. 誤り訂正においても, シンドロームの全 $256(= 2^8)$ パターンに対応する誤り位置をテーブルに格納しておくことで, 誤り位置を高速に特定できる. テーブルを用いる手法は, 処理が高速である反面, 外部メモリを必要とし, 面積や消費電力の増大が問題となる. また, メモリ容量に制限があるシステムでは, この手法を適用できない可能性がある.

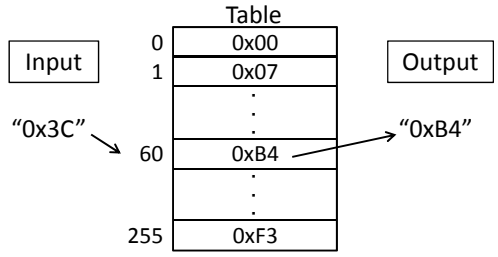

図 4: 符号化と復号処理に用いるテーブル

```
c(x) = 0;
for(i = 0; i < 4; i++){
    d(x) = c(x) + m_i(x);
    c(x) = table[d(x)]; //図 4 のテーブルから値を参照
}
```

図 5: テーブルを用いた検査記号の計算

・専用演算器の実装

巡回符号処理を高速化するために, 専用の演算器を実装したプロセッサ [14] が提案されている. しかしながら, 8bit データの処理に 2 サイクル要し, 入力データのビット位置を予め反転させておく必要がある. また, 専用のレジスタを必要とし, 実行サイクル数や消費電力の面で改善の余地がある.

本研究では, 任意の生成多項式に対応した, 巡回符号を効率的に処理する演算器, およびこれを実装した ASIP を提案する.

3. 巡回符号向き ASIP の設計

本研究では, ASIP 設計ツールとして, ASIP Solutions 社の ASIP Meister [15, 16] を用い, ベースプロセッサは同社の Brownie Micro 16(以下 BM16 と呼ぶ) を用いる. BM16 は 16bit の RISC プロセッサで, 表 1 に示す命令セットを持つ. BM16 へ巡回符号処理用の命令を追加することで, 巡回符号向きの ASIP を設計する.

3.1 巡回符号処理の効率化

巡回符号には, 符号化, 復号, そして誤り訂正の 3 つの処理が存在する. これらの処理では, 図 2, 3 に示すように, 生成多項式との剰余を計算する. 本節では, 式 (7) に

表 1: Brownie Micro 16 の命令セット

命令の種類	実装されている命令
算術演算	加算, 減算, 算術シフト
論理演算	論理和, 論理積, 排他的論理和, 否定, 論理シフト
比較演算	等号比較, 不等号比較, 符号無し比較
即値演算	即値加算, 即値論理シフト, 即値ロード
ビット演算	ビットセット, ビットクリア ビットテスト
ロード演算	ハーフワードロード, バイトロード
ストア演算	ハーフワードストア, バイトストア
分岐演算	条件分岐
ジャンプ演算	即値ジャンプ, レジスタ間接ジャンプ
割り込み演算	割り込み発行, 割り込み復帰
特殊演算	NOP, スリープ

示す剰余の計算を行列演算へ帰着することで, 巡回符号処理を効率化する手法を示す.

$$\left(x^n \sum_{k=0}^{7} \alpha_k x^k \right) \bmod g(x) \qquad (\alpha_k \in \{0,1\}) \tag{7}$$

3.2 行列演算への帰着

式 (7) において, $n = 1$ の場合を考える. このとき, 式 (7) は式 (8) のように計算される.

$$\left(x \sum_{k=0}^{7} \alpha_k x^k \right) \bmod g(x) = \begin{cases} \sum_{k=1}^{7} \alpha_{k-1} x^k & (\alpha_7 = 0) \\ \sum_{k=1}^{7} \alpha_{k-1} x^k + (x^2 + x + 1) & (\alpha_7 = 1) \end{cases} \tag{8}$$

式 (8) に示すように, 最上位ビット α_7 の値に応じて処理が異なる. $\alpha_7 = 0$ のとき 1bit の左シフトが, $\alpha_7 = 1$ のとき 1bit の左シフトおよび $g(x)$ との XOR 演算が計算される. これは, 図 6 に示す, 生成多項式 $g(x)$ に対応した行列 G を用いることで, 最上位ビットの値に依存せず, 式 (9) のように計算することができる.

$$(\alpha_7\ \alpha_6\ \alpha_5\ \alpha_4\ \alpha_3\ \alpha_2\ \alpha_1\ \alpha_0) \times \mathrm{G} \tag{9}$$

$n > 1$ の場合も同様に行列演算に帰着することができる. 式 (7) において, $n = l$ のときの計算結果を $d_l(x)$ とする. このとき, $d_{l+1}(x)$ を式 (10) から得る.

$$\begin{aligned} d_{l+1}(x) &= \left(x^{l+1} \sum_{k=0}^{7} \alpha_k x^k \right) \bmod g(x) \\ &= \left\{ x \left(x^l \sum_{k=0}^{7} \alpha_k x^k \bmod g(x) \right) \right\} \bmod g(x) \\ &= \{ x \times d_l(x) \} \bmod g(x) \end{aligned} \tag{10}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

図 6: 行列 G

$n = 1$ のとき, 式 (7) は式 (9) の演算から得られる. したがって, 行列 G との乗算を再帰的に計算することで, $n > 1$ の場合も同様に求めることができる.

以上のことから, 式 (7) は式 (11) に示す行列演算に帰着することができる.

$$(\alpha_7\ \alpha_6\ \alpha_5\ \alpha_4\ \alpha_3\ \alpha_2\ \alpha_1\ \alpha_0) \times \mathrm{G}^n \tag{11}$$

3.3 巡回符号処理用の追加命令

本節では, 前節にて述べた行列演算を用いた, 巡回符号処理に用いる追加命令について述べる. 追加命令は次の 3 命令である.

CCAL(Cyclic code CALculation) 命令
検査記号およびシンドロームを計算する命令.

ERPS(ERror Position Specify) 命令
誤り位置を特定する命令.

FEPS(Fast Error Position Specity) 命令
64bit までの情報記号に対し, 1 サイクルで誤り位置を特定する命令.

3.3.1 CCAL 命令

CCAL 命令は, 検査記号およびシンドロームを計算する. CCAL 命令を処理する演算器の構成を図 7 に示す. 図 7 中の G^8, G^{16} は, 8bit の入力に対し, それぞれ G^8, G^{16} を乗算する回路を表す. rd レジスタは計算された検査記号を記憶するレジスタで, 初期値は 0 とする. rs レジスタへ情報記号を入力することで, 検査記号を計算し, rd レジスタへと格納する. 情報記号が 16bit より多い場合は, rd レジスタに記憶されている検査記号の途中結果と情報記号との XOR 演算の後, 行列演算が行われる.

CCAL 命令では, G^8 および G^{16} を乗算する演算器を利用している. これは, 式 (12) に示すように, 16bit の情報記号を上位と下位に分割することで, 剰余の計算を行列演算に帰着できるためである.

$$\begin{aligned} &\left(\sum_{k=0}^{15} \alpha_k x^{k+8} \right) \bmod g(x) \\ &= \left\{ \left(x^8 \sum_{k=0}^{7} \alpha_k x^k \right) \bmod g(x) \right\} \\ &+ \left\{ \left(x^{16} \sum_{k=0}^{7} \alpha_{k+8} x^k \right) \bmod g(x) \right\} \end{aligned} \tag{12}$$

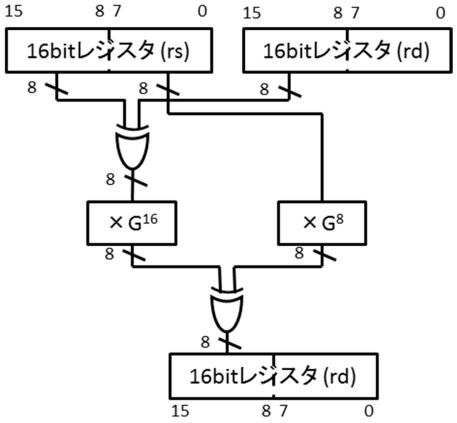

図 7: CCAL 命令における演算器の構成

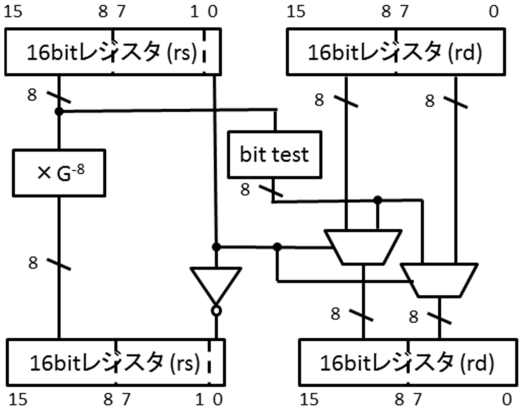

図 8: ERPS 命令における演算器の構成

3.3.2 ERPS 命令

ERPS 命令は, 行列演算により誤り位置を特定する. 1bit 誤り発生時のシンドロームと誤り位置との関係は式 (6) に示した.

式 (13) は, 行列の性質から明らかに成立する. ここで, G^{-n} は G^n の逆行列を表す.

$$\begin{aligned} & (\alpha_7\ \alpha_6\ \alpha_5\ \alpha_4\ \alpha_3\ \alpha_2\ \alpha_1\ \alpha_0) \times G^n \times G^{-n} \\ = \ & (\alpha_7\ \alpha_6\ \alpha_5\ \alpha_4\ \alpha_3\ \alpha_2\ \alpha_1\ \alpha_0) \end{aligned} \tag{13}$$

これは, 式 (7) において, 演算結果から入力ビット列 α_k を求められることを示している. すなわち, 式 (14) において, 行列 G^n を用いることで α_k から β_k を, 行列 G^{-n} を用いることで β_k から α_k を求めることができる.

$$\sum_{k=0}^{7} \beta_k x^k = \left(x^n \sum_{k=0}^{7} \alpha_k x^k \right) \bmod g(x) \tag{14}$$
$$(\alpha_k \in \{0,1\}, \beta_k \in \{0,1\})$$

ERPS 命令では, G^{-8} を計算する演算器を利用することで, 誤り位置を特定する. 以下, 検査記号に誤りが発生した場合と情報記号に誤りが発生した場合に分けて, 誤り位置特定手法を述べる.

(i) 検査記号に誤りが発生した場合

式 (6) は式 (15) のようになる.

$$s_n(x) = x^n \ (0 \le n \le 7) \tag{15}$$

被除数の次数が除数の次数より小さいため被除数と剰余が等しくなる. また, $s_n(x)$ は誤り位置に等しいビットのみ 1 になるため, シンドロームと誤りを含んだ検査記号との XOR 演算を計算することで, 誤りを訂正できる.

(ii) 情報記号に誤りが発生した場合

式 (6) は式 (16) のようになる.

$$\begin{aligned} s_n(x) &= x^{8p+q} \bmod g(x) \ (0 \le q \le 7) \\ &= (x^8)^p x^q \bmod g(x) \end{aligned} \tag{16}$$

ただし, $n = 8p + q$ とし, p は取りうる中の最大値とする. このとき, 行列 G^{-8} を用いることで p と q を特定する. G^{-8} を繰り返し乗算することで, シンドロームの値は, 検査記号に誤りが発生した場合と同様, 1bit のみが 1 となる. したがって, 1bit のみが 1 となるまでに G^{-8} を乗算した回数から p を, 1 の位置から q を算出し, 誤り位置を特定することができる.

ERPS 命令を処理する演算器の構成を図 8 に示す. 図 8 中の G^{-8} は, 8bit の入力に対し, G^{-8} を乗算する回路を表す. また, 図中の bit test は入力された値のうち 1 個のみが 1 である場合に入力値を, それ以外の場合は全零を出力する回路を表す. rs レジスタの上位 8bit へシンドロームを与えることで, G^{-8} を乗算し, rs レジスタの上位 8bit へと格納する. また, rs レジスタの下位 1bit は誤りが 16bit のうち, 上位に存在するか下位に存在するかを保持するフラグであり, 命令が実行される度に反転する. rd レジスタには, 誤り位置を特定できた場合には, その位置のみ 1 のビット列が格納される. 前述したフラグを用いることで, 上位であるか下位であるかを決定し, rd レジスタへ格納する.

この手法では, 誤り位置の特定に要する時間は誤り位置に依存する. すなわち, 検査行列に誤りが発生した際は, XOR 演算を 1 度計算することにより訂正できるが, p の値が大きくなるに従い, G^{-8} を乗算する回数が増加するため実行サイクル数も大きくなる.

3.3.3 FEPS 命令

FEPS 命令は, ERPS 命令を応用した命令で, 64bit までの情報記号に対し, 1 サイクルで誤り位置を特定する.

FEPS 命令は, G^{-8}, G^{-16}, G^{-24}, $\cdots$, G^{-64} の合計 8 種類の行列を乗算する回路, およびシンドロームや乗算結果が 1bit のみ 1 であるかどうかを判断する bit test を 9 個有する. シンドロームを入力することで, 誤り位置と誤り位置のみ 1 であるビット列の 2 つを出力する. 誤りが 1bit の場合, 9 個の bit test のうち 1 個のみが全零でない値を出力するため, 誤り位置を特定することができる. 一方, 誤りが 2bit の場合, 9 個の bit test はこの全てが全零を出力するため, 誤りが 1bit の場合と区別することができる.

3.4 任意の生成多項式への適用

3.2 節にて述べた, 剰余の計算を行列演算へと帰着する手法は, 任意の生成多項式に適用できる. プロセッサの

データ幅を W_d, 生成多項式の次数を W_g, 生成多項式に対応する, 大きさが $W_g \times W_g$ の行列を H, $l = \lceil \frac{W_d}{W_g} \rceil$ とする. このとき, 情報記号 $(\alpha_{(W_d-1)} \cdots \alpha_1\ \alpha_0)$ の検査記号は, 式 (17) より得られる.

$$\begin{aligned} (\alpha_{(W_g-1)} \cdots \alpha_0) &\times \mathrm{H}^{W_g} \\ +\,(\alpha_{(2W_g-1)} \cdots \alpha_{W_g}) &\times \mathrm{H}^{2W_g} \\ &\vdots \\ +\,(\alpha_{(lW_g-1)} \cdots \alpha_{((l-1)W_g)}) &\times \mathrm{H}^{lW_g} \end{aligned} \quad (17)$$

ただし, $W_d < lW_g$ のとき, α_{W_d}, $\alpha_{(W_d+1)}$, $\cdots$, $\alpha_{(lW_g-1)}$ は, すべて 0 とする. 生成多項式に応じた行列を用いることで, 任意の生成多項式に提案手法を適用することができる.

4. 評価実験

本節では, 前節において設計した ASIP を評価するために行った実験の方法, およびその結果について述べる. 評価項目は, 面積, 消費電力, および実行サイクル数の 3 項目である.

4.1 実験方法

面積は Synopsys 社の Design Compiler で, 0.18μm CMOS ライブラリを用いて論理合成することにより測定し, 合成結果と同社の Power Compiler を用いて 32bit の情報記号を 10 個符号化する際の消費電力を測定した.

4.2 実験結果

4.2.1 面積, 消費電力

表 2 に面積と消費電力の実験結果を示す. 面積および消費電力では, ベースプロセッサである BM16, BM16 へ CCAL 命令と ERPS 命令を追加した ASIP, そして BM16 へ CCAL 命令と FEPS 命令を追加した ASIP の 3 つを評価した. 表 2 の括弧内の数値は, BM16 に対する面積および消費電力の割合である.

命令を追加したことにより, 面積, 消費電力共に増加がみられる. ERPS 命令を追加した ASIP では, 面積の増加量が 5%, 消費電力の増加量が 2%に抑えられている. 一方, FEPS 命令を追加した ASIP では, 面積で 10%, 消費電力で 6%の増加がみられる.

CCAL 命令と ERPS 命令を追加したプロセッサでは, BM16 と比較して, 面積が約 340gate 増加している. CCAL 命令および ERPS 命令による増加量がそれぞれ約 120, 100gate であり, 残りの増加量をプロセッサの制御部やマルチプレクサが占める. なお, FEPS 命令による増加量は約 490gate で, ERPS 命令と比較して 5 倍程度大きい.

4.2.2 実行サイクル数

表 3 に符号化に要する実行サイクル数を, 表 4 に誤り訂正に要する実行サイクル数を示す. なお, 実行サイクル数が情報記号や誤り位置に依存する手法に関しては, 実行サイクル数の最小値と最大値の 2 つを示している. 評価対象は以下の 6 つの手法で, 前者 3 つが既存手法, 後者 3 つが本稿で提案する手法である. 符号化では RISC, LUT, Nguyen's, CCAL の 4 つを, 誤り訂正では RISC, LUT, ERPS, FEPS の 4 つを評価した.

表 2: 面積, 消費電力の評価結果

	面積 [gate]	消費電力 [μW/MHz]
BM16	6905.9	43.7
BM16+ CCAL+ERPS	7244.2 (104.9%)	44.7 (102.3%)
BM16+ CCAL+FEPS	7625.4 (110.4%)	46.2 (105.7%)

表 3: 符号化に要する実行サイクル数

情報記号	16bit	32bit	64bit
RISC	142 – 158	278 – 310	550 – 614
Nguyen's	48 – 52 (32.9%)	89 – 97 (31.3%)	171 – 187 (30.5%)
LUT	26 (16.5%)	44 (14.2%)	80 (13.0%)
CCAL	12 (7.6%)	20 (6.5%)	36 (5.9%)

表 4: 誤り訂正に要する実行サイクル数

情報記号	16bit	32bit	64bit
RISC	24 – 277	24 – 453	24 – 803
LUT	15 – 29 (10.5%)	15 – 29 (6.4%)	15 – 29 (3.6%)
ERPS	5 – 17 (6.1%)	5 – 24 (5.3%)	5 – 38 (4.7%)
FEPS	6 – 10 (3.6%)	6 – 10 (2.2%)	6 – 10 (1.2%)

- RISC : RISC 命令のみを用いて処理
- LUT(Look Up Table) : テーブルを用いて処理
- Nguyen's : アルゴリズムを改良する手法 [11]
- CCAL : CCAL 命令を用いて処理
- ERPS : ERPS 命令を用いて処理
- FEPS : FEPS 命令を用いて処理

追加した命令を用いることで, 符号化, 誤り訂正ともに RISC と比較して 90%以上実行サイクル数を削減できる.

表 3 より, CCAL は LUT と比較して実行サイクル数が半分程度であることがわかる. これは, CCAL が 16bit ずつ処理するのに対し, LUT では 8bit ずつ処理するためである. ゆえに, LUT において 16bit ずつ処理するテーブルを利用することで, CCAL と同程度の実行サイクル数になることが予想されるが, このとき, テーブルのサイズは 64Kbyte と非常に大きくなる.

ERPS では, 符号長の増大に伴い実行サイクル数が増加する. 表 4 より, 情報記号が 32bit のとき, ERPS と LUT の実行サイクル数は同程度となるが, 情報記号が 32bit より多くなると, LUT の実行サイクル数の方が小さくなる. 一方, FEPS と LUT を比較すると, FEPS が符号長に依存せず 3 倍程度高速であることがわかる.

4.3 消費電力量の比較

符号化および誤り訂正に要する消費電力量を, 消費電力と実行サイクル数から算出し比較する. 表 5 に符号化に要する消費電力量を, 表 6 に誤り訂正に要する消費電力量を示す. プロセッサへ命令を追加することで, 消費電力が増加するが, 増加量が少ないため, 消費電力量は実行サイクル数におよそ比例する.

本稿の提案手法と LUT を比較すると, CCAL で 53%, FEPS で 64%の消費電力量が削減可能である. 一方, ERPS では情報記号が 32bit のとき, LUT と消費電力量が同等となる. 以上より, 提案手法を用いることで, テーブルを用いずに巡回符号を効率よく処理し, かつ消費電力量を削減できることがわかる.

表 5: 符号化に要する消費電力量 [μJ]

情報記号	16bit	32bit	64bit
RISC	6.90	13.55	26.83
Nguyen's	2.27 (32.9%)	4.24 (31.3%)	8.17 (30.5%)
LUT	1.14 (16.5%)	1.92 (14.2%)	3.50 (13.0%)
CCAL	0.54 (7.8%)	0.89 (6.6%)	1.61 (6.0%)

表 6: 誤り訂正に要する消費電力量 [μJ]

情報記号	16bit	32bit	64bit
RISC	12.10	19.80	35.09
LUT	1.27 (10.5%)	1.27 (6.4%)	1.27 (3.6%)
ERPS	0.76 (6.3%)	1.07 (5.4%)	1.70 (4.8%)
FEPS	0.46 (3.8%)	0.46 (2.3%)	0.46 (1.3%)

5. まとめ

本稿では, 巡回符号処理を効率的に行う手法, およびこれを実装した ASIP を提案した. 剰余の計算を行列演算へと帰着することで, 任意の生成多項式に対応した, テーブルを用いない手法を提案した. 評価実験より, RISC 命令のみを用いた場合と比較して 90%以上の消費電力量を削減できること, そしてテーブルを用いる手法と比較しても同等, 或いはそれ以下に消費電力量を抑えられることを示した.

今後の課題としては, メモリを含めた評価実験を考えている. また, センサシステムなど, 実際に利用されているアプリケーションを用いた消費電力量評価が挙げられる.

謝辞

本研究の一部は, 平成 22 年度イノベーションシステム整備事業「ユビキタス生体計測ヘルスケアデバイス・システムの開発」により行われた.

参考文献

[1] F. J. Pierce and T. V. Elliott, "Regional and on-farm wireless sensor networks for agricultural systems in Eastern Washington," *Computers and Electronics in Agriculture*, Vol. 61, No. 1, pp. 32–43, 2007.

[2] I. E. Lamprinos, A. Prentza, E. Sakka, and D. Koutsouris, "A Low Power Medium Access Control Protocol for Wireless Medical Sensor Networks," *Proceedings of the 26th Annual International Conference of the IEEE EMBS*, pp. 2129–2132, 2004.

[3] Mehmet R. Yuce, Peng Choong Ng, and Jamil Y. Khan, "Monitoring of Physiological Parameters from Multiple Patients Using Wireless Sensor Network," *Journal of Medical Systems*, Vol. 32, No. 5, pp. 433–441, 2008.

[4] Paul Withington, Herbert Fluhler, and Soumya Nag, "Enhancing homeland security with advanced UWB sensors," *IEEE Microwave Magazine*, Vol. 4, No. 3, pp. 51–58, 2003.

[5] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "Forward Error Correction (FEC) Building Block," *RCF 3452*, IETF, 2002.

[6] G. Fairhurst and L. Wood, "Advice to link designers on link Automatic Repeat reQuest (ARQ)," *RCF 3366*, IETF, 2002.

[7] Tsung-Ching Lin, Hsin-Chiu Chang, Hung-Peng Lee, and Trieu-Kien Truong, "On the decoding of the (24, 12, 8) Golay code," *Information Sciences*, Vol. 180, No. 23, pp. 4729–4736, 2010.

[8] IEEE Communications Society, "IEEE Std 1902.1-2009 IEEE Standard for Long Wavelength Wireless Network Protocol," 2009.

[9] Tenkasi V. Ramabadran and Sunil S. Gaitonde, "A Tutorial on CRC Computations," *IEEE MICRO*, Vol. 8, No. 4, pp. 62–75, 1988.

[10] Chris Borrelli, "IEEE 802.3 Cyclic Redundancy Check," *XAPP209*, 2001.

[11] Gam D. Nguyen, "Fast CRCs," *IEEE Transactions on Computers*, Vol. 58, No. 10, pp. 1321–1331, 2009.

[12] Michael E. Kounavis and Frank L. Berry, "Novel Table Lookup-Based Algorithms for High-Performance CRC Generation," *IEEE Transactions on Computers*, Vol. 57, No. 11, pp. 1550–1560, 2008.

[13] Justin Ray and Philip Koopman, "Efficient High Hamming Distance CRCs for Embedded Networks," *Dependable Systems and Networks*, pp. 3–12, 2006.

[14] M16C/62P グループハードウェアマニュアル, Renesas, 2006.

[15] 今井 正治, 武内 良典, 塩見 彰睦, 佐藤 淳, 北嶋 暁, "特定用途向きプロセッサ開発システム ASIP Meister," 電子情報通信学会技術研究報告, Vol. 102, No. 399, pp. 39–44, 2002.

[16] Masaharu Imai, Yoshinori Takeuchi, Keishi Sakanushi and Nagisa Ishiura, "Advantage and Possibility of Application-domain Specific Instruction-set Processor (ASIP)," *Information and Media Technologies*, Vol. 5, No. 4, pp. 1064–1081, 2010.