社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－ＳＬＤＭ－126（18）
2006／10／26

C言語設計によるリードソロモン符号復号化回路の最適化

小田島 賢和† 小西 徹也‡ 神戸 尚志††

†近畿大学大学院 総合理工学研究科 エレクトロニクス系工学専攻 〒577-8502 大阪府東大阪市小若江 3-4-1
‡近畿大学大学院 総合理工学研究科 東大阪モノづくり専攻 〒577-8502 大阪府東大阪市小若江 3-4-1
††近畿大学 理工学部 電気電子工学科 〒577-8502 大阪府東大阪市小若江 3-4-1

E-mail: †0633340415r@kindai.ac.jp, ‡0532360601x@kindai.ac.jp, ††tkambe@ele.kindai.ac.jp

あらまし リードソロモン符号はバイト単位で誤り訂正を行うことができ，CDやDVDなどの記憶装置や，ADSLや宇宙通信などの通信分野などで多く使用されている．しかし，ガロア体を使用したバースト誤り訂正を得意とする高度な訂正能力を持つ反面，処理に複雑な演算を多用するため，データの処理速度が低いという問題がある．本稿ではRTL設計よりも抽象度の高いC言語設計が可能な動作合成システムであるBachシステムを用いて，リードソロモン符号の復号化回路をC言語記述によるハードウェアに適した記述を行い，最適化を図り，その結果を評価する．

キーワード C言語設計，リードソロモン符号，誤り訂正，動作合成

C-based Design and its Optimization for a Reed-Solomon Decoder

Yoshikazu ODAJIMA† Tetsuya KONISHI‡ and Takashi KAMBE††

† ‡ Graduate School of Science and Engineering Kinki University, 3-4-1Kowakae, Higashi-Osaka-shi, Osaka, 577-8502 Japan
† † School of Science and Engineering Department of Electric and Electronic Engineering Kinki University, 3-4-1Kowakae, Higashi-Osaka-shi, Osaka, 577-8502 Japan

E-mail: †0633340415r@kindai.ac.jp, ‡0532360601x@kindai.ac.jp, ††tkambe@ele.kindai.ac.jp

Abstract Reed-Solomon decoder can correct byte errors and it has been a popular technology for various device such as CD, DVD, communication. But the speed of data correction is low since it needs many Galois field based calculations. This paper proposes a design and its optimization of a Reed-Solomon decoder using "Bach" system that can do C-based design.

Keyword C-based Design, Reed-Solomon Code, error correction, behavior synthesis

1. はじめに

誤り訂正技術は様々な分野で活用されている．CDやDVDなどの記録媒体においては記録面の劣化やキズなどがついたものからデータを取り出し，通信分野では通信時に受けた外乱によるデータの誤りを訂正するなど日常生活の中で必要不可欠な技術である．本稿ではリードソロモン符号の復号化回路の設計を行った．リードソロモン符号は巡回符号の一種であり，バースト誤りに強いという特長がある．

本研究ではリードソロモン符号復号化回路の回路化において，RTレベル設計より抽象度の高い設計が可能となる動作合成システムであるBachシステムを用いた．Bachシステムではハードウェアの並列動作等を記述できるようにANSI-Cを拡張したBach-C言語を入力とし，動作合成によって論理合成可能なVHDLを自動生成するC言語設計ツールである．また，ソフトウェア用に記述されたANSI-CをそのままBach-C言語の記述に変更するだけではハードウェアに適した記述ではないため，これをハードウェア向きの記述に変更することで回路の最適化を行い，性能評価を行った．

本稿の構成は2章でリードソロモン符号の復号手順，3章でリードソロモン符号復号化回路のハードウェア設計，4章でC言語設計における最適化手法について，5章で性能評価，6章でまとめと今後の課題について述べる．

2. リードソロモン符号の復号手順

リードソロモン符号はtバイトの誤り訂正能力があ

る場合，t重バイト誤り訂正と呼ばれる．本研究で取り扱うリードソロモン符号の誤り訂正能力は入力データ$n=182$バイト中$t=5$バイト，及び入力データ$n=208$バイト中$t=8$バイトまでの誤りを訂正するものであり，これは DVD の誤り訂正に使用されている．

リードソロモン符号は特別な BCH 符号であり，その復号方法も BCH 符号に似ている．本稿では復号手順を“シンドローム計算”，“ユークリッド処理”，“チェンサーチ”の 3 つのブロックに分けて述べる．シンドローム計算は受信多項式$Y(x)$のシンドロームを求める計算を行い，シンドローム多項式$S(x)$を得る．シンドローム多項式が零であれば，その時点で処理を終了する(no error)．シンドローム多項式が非零であれは，ユークリッド処理を行う．ユークリッド処理ではシンドローム多項式よりユークリッド法を用いて誤り位置多項式$\sigma(x)$と誤り数値多項式$\omega(x)$を求める．チェンサーチでは誤り位置多項式を用いてチェンサーチを行い誤り位置を求め，誤り位置多項式と誤り数値多項式から誤りパターンを求め，誤り訂正を行う．これらの処理について以下にその概要を述べる．

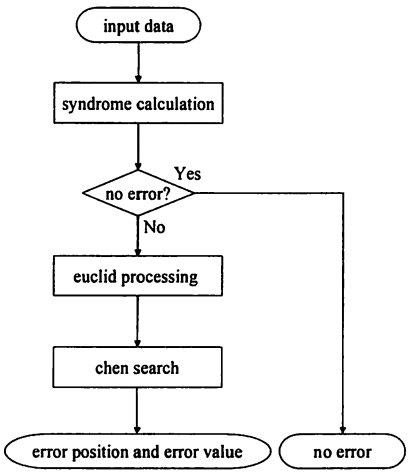

図 1　リードソロモン符号復号アルゴリズム

2.1. シンドローム計算

リードソロモン符号は根としてガロア体の元，α^0，α^1，…，α^{2t-1}を持つから復号回路における入力である受信多項式$Y(x)$に対してシンドローム

$$s_i = Y(\alpha^i) \quad (i=0,1,\cdots,2t-1) \tag{1}$$

を求める．シンドローム計算の出力はs_iを係数としたシンドローム多項式$S(x)$である．s_iがすべて零であれば，受信多項式$Y(x)$に誤りが含まれていないと判断する．

2.2. ユークリッド処理

リードソロモン符号は誤り位置を誤りロケータとして表す．これは入力データのi番目が誤っている場合，誤りロケータはガロア体の元α^iと表現される．euclid 関数では誤りロケータを根に持つ誤り位置多項式$\sigma(x)$と誤りパターンを求めるために必要となる誤り数値多項式$\omega(x)$を求める．これらを求めるアルゴリズムとしてユークリッド法を用いた．これはシンドローム多項式と誤り位置多項式の連立方程式を多項式を使って解く逐次的計算法で，最大公約数を求めるユークリッド互除法を利用したアルゴリズムである．入力データのj_1番目，j_2番目，…，j_ℓ番目のℓか所に誤りが起きたとすると誤り位置多項式$\sigma(x)$は(2)式のように定義される．

$$\begin{aligned}\sigma(x) &= (1-\alpha^{j_1}x)(1-\alpha^{j_2}x)\cdots(1-\alpha^{j_\ell}x) \\ &= \sigma_\ell x^\ell + \sigma_{\ell-1}x^{\ell-1} + \cdots + \sigma_1 x + \sigma_0\end{aligned} \tag{2}$$

また，シンドローム多項式，誤り位置多項式，誤り数値多項式の間には(3)式の関係式が成立する．

$$\sigma(x)S(x) \equiv \omega(x) \tag{3}$$

2.3. チェンサーチ

チェンサーチとはユークリッド処理で求めた誤り位置多項式$\sigma(x)$を用いて誤り位置を求める処理と誤り数値多項式$\omega(x)$を用いて誤りパターンを求める処理を行う処理である．誤り位置多項式は誤りロケータの逆元を根に持つことから$\sigma(x)$にα^{-i} $(i=0,1,2\cdots,n-1)$を順次代入し，$\sigma(\alpha^{-i})=0$であればα^iが誤りロケータである．この後，誤りパターンを求めるが，j_i番目に対応する誤りパターンe_{j_i}は(4)の関係式から求めることができる．

$$e_{j_i} = -\frac{\omega(\alpha^{-j_i})}{\sigma'(\alpha^{-j_i})\alpha^{-j_i}} \tag{4}$$

ここで$\sigma'(x)$は$\sigma(x)$の形式微分

$$\sigma'(x) = \ell\sigma_\ell x^{\ell-1} + \sigma_{\ell-1}x^{\ell-2} + \cdots + 2\sigma_2 x + \sigma_1 \tag{5}$$

である．

3. ハードウェア設計

3.1. Bach システムを用いた設計工程

ハードウェアの設計手法は，RT レベルのハードウェア設計言語である VHDL，Verilog-HDL を用いた設計が一般的である．半導体技術の進歩により，1 つの LSI

に数千万ゲートの回路が搭載可能となったため，HDLよりもさらに抽象度の高い記述での回路設計が注目されている．これは，動作合成システムと言われ，C言語やそれに近いアルゴリズム記述からのハードウェアの設計を行う．本研究で使用した Bach システムもこの動作合成を実現したシステムの1つである．

図2に示すように，従来の HDL によるハードウェア設計は，まず，C言語などによるアルゴリズムの開発を行う，その後ソフトウェアを使った検証，HDL での回路記述，RTL 論理合成，シミュレーションといった手順で行われる．そのため，HDL で回路設計の不具合が見つかった場合，修正に大きな手間が必要となる．

Bach を用いた設計では，アルゴリズムの設計段階から抽象度の高い Bach-C 言語を用い，機能設計・検証を行うことで，HDL による機能検証が削減されるだけでなく，Bach-C 記述から RTL の VHDL が自動生成される．また，各処理の計算時間や回路規模見積り，シミュレーションによるボトルネックの分析，機能の分割や並列化，中間メモリの自動生成，スループット指定によるパイプライン回路の自動生成などにより，各種アーキテクチャを短期間に探索できる．

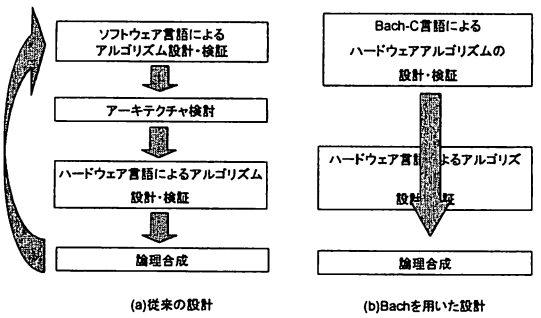

図 2　Bach システムの設計工程

3.2. リードソロモン符号復号化回路のハードウェア化

リードソロモン符号復号化回路を設計するにあたり，回路を以下のような構成とした．

受信多項式を入力しシンドローム計算を行い，シンドローム多項式を出力する部分を syndrome 関数，シンドローム多項式を入力しユークリッド処理を行い，誤り位置多項式および誤り数値多項式を出力する部分を euclid 関数，誤り位置多項式および誤り数値多項式を入力しチェンサーチを行い誤り位置を出力し，誤りパターンを出力する部分を chen 関数とする．また，回路外部との同期は同期チャネル in，out を用いて行い，回路外部とのデータの授受は非同期チャネルを用いて行う．

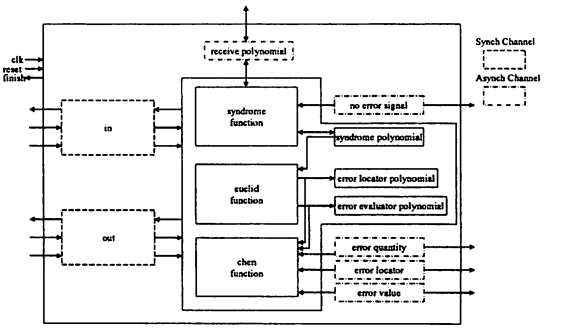

図 3　符号復号化回路アーキテクチャ

4. ハードウェアの最適化

Bach システムを用いて設計を行うにあたり，設計の元になるものはC言語で記述されたソフトウェア用のソースである．これはハードウェアを意識せずソフトウェアで実行されることを前提に記述されているものであるから，このまま動作合成したのでは最適な回路を得ることは難しい．本章では具体的にソフトウェア向けのソースをどのように動作合成に適した記述に変更を行ったかについて述べる．

4.1. 変数化

リードソロモン符号の復号では，符号を多項式表現で扱っている．C言語プログラムでは，多項式の各係数を配列として表現している．配列を用いて表現すると，配列をアクセスするためのアドレスデコーダが必要となり，回路規模が増加する．しかも，配列への同時アクセスができないため，待ち時間が発生してしまう．そこで，配列の各要素を独立した変数にすることでこの問題を解決した．これにより，アドレスデコーダが必要なくなり，さらにアクセス競合も無くなる為，より高速な処理が可能となる．

例として syndrome 関数内のシンドローム計算を挙げる．syndrome 関数では受信多項式にガロア体の元を代入していき，結果が零か否かの判定を行っている((1)式)．ソフトウェア記述であれば多項式の係数を配列を用いて記述するのが一般的であるが，ハードウェアを意識すると配列に対するアクセス競合によるオーバヘッドが大きい．これを解消するために計算式に含まれている配列の要素をそれぞれ独立した変数に書き換える．これにより，配列に格納されていた各値は独立したレジスタとして保持されるため，アクセス競合が発生しない．図4はシンドローム計算の記述の一部であるが，変更前では入力値が格納されている配列 buffer を変数 buffer_value に，シンドローム計算時に使

用するガロア体元のテーブル GFtable を定数に，出力値が格納されている配列 s を変数 s に変更することで，処理の独立性を高めた．

```
unsigned#8 GFtable[10] =
{0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1d, 0x3a};

for(j=1; j<=181; j++){
  s[0]=buffer[j]^GF_MUL(s[0], GFtable[0]);
  s[1]=buffer[j]^GF_MUL(s[1], GFtable[1]);
  s[2]=buffer[j]^GF_MUL(s[2], GFtable[2]);
  s[3]=buffer[j]^GF_MUL(s[3], GFtable[3]);
  s[4]=buffer[j]^GF_MUL(s[4], GFtable[4]);
  s[5]=buffer[j]^GF_MUL(s[5], GFtable[5]);
  s[6]=buffer[j]^GF_MUL(s[6], GFtable[6]);
  s[7]=buffer[j]^GF_MUL(s[7], GFtable[7]);
  s[8]=buffer[j]^GF_MUL(s[8], GFtable[8]);
  s[9]=buffer[j]^GF_MUL(s[9], GFtable[9]);
}
```

(a) 変更前

```
for(j=1; j<=181; j++){
  buffer_value=buffer[j];
  s0=buffer_value^GF_MUL(s0, 0x01);
  s1=buffer_value^GF_MUL(s1, 0x02);
  s2=buffer_value^GF_MUL(s2, 0x04);
  s3=buffer_value^GF_MUL(s3, 0x08);
  s4=buffer_value^GF_MUL(s4, 0x10);
  s5=buffer_value^GF_MUL(s5, 0x20);
  s6=buffer_value^GF_MUL(s6, 0x40);
  s7=buffer_value^GF_MUL(s7, 0x80);
  s8=buffer_value^GF_MUL(s8, 0x1d);
  s9=buffer_value^GF_MUL(s9, 0x3a);
}
```

(b) 変更後

図 4　シンドローム計算部分記述

配列を用いたデータ構造はリードソロモン回路復号化回路の全体で使用されているため，同様の最適化を euclid 関数，chen 関数にも適用した．

4.2. データ入力方法の変更

syndrome 関数の入力である受信多項式は C 言語ソースでは配列として保持されている．Bach-C 記述では当初，非同期チャネルの配列を用いて表現し，この配列を介してデータの入力を行った．しかし，受信多項式の全係数を保持するレジスタを用意することになり，回路規模が大きくなってしまう．回路規模を抑えるためにこの部分を以下のように変更を行った．同期チャネル通信を用いて受信多項式の係数を 1 クロック毎に 1 個入力するようにし，入力されたクロックでその係数に関するシンドローム計算を行う．このように変更することで入力データを保持しておく配列が必要なくなり，レジスタが削減できた．

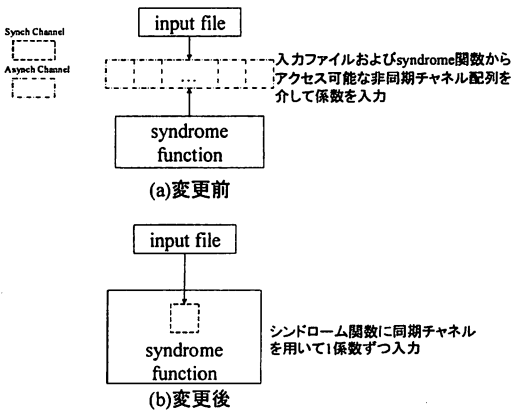

図 5　データ入力方法変更

4.3. 並列化

4.3.1. ユークリッド法の並列化

ユークリッド処理はシンドローム多項式から誤り位置多項式，誤り数値多項式を求める．euclid 関数の処理を大きく 2 つにわけると多項式同士の除算(多項式除算)と多項式同士の乗算(多項式乗算)処理にわけることができ，これらの処理を 1 セットとしたループ処理となっている．この多項式除算と多項式乗算を高速化するために並列処理を実現した．図 6 はユークリッド法のアルゴリズム中の多項式除算と多項式乗算をイメージした図である．図 6 上側が多項式除算，下側が多項式乗算を表している．□一つひとつが多項式の係数を表しており，その中の Q0，Q1 は多項式除算の商を表している．多項式乗算の乗数は多項式除算の商であり，多項式除算が終了していなくても，商さえ求まっていれば，多項式乗算を開始することができる．よって，まず商を出力するために必要な部分(図 6 一点鎖線部分)の処理を行ってから，多項式除算のそれ以外の部分の処理(図 6 二点鎖線部分)と，多項式乗算処理の並列に処理する．

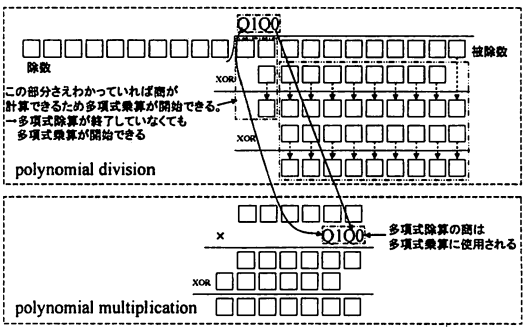

図 6　ユークリッド法の並列化

4.3.2. チェンサーチの並列化

チェンサーチは誤り位置多項式にガロア体の元を代入して誤り位置を探索する処理である．誤り訂正能力が $t=5$ の場合，代入する元は $\alpha^{-0} \sim \alpha^{-181}$ である．この処理はそれぞれの元に対して独立しているため，$\alpha^{-0} \sim \alpha^{-90}$ を代入する部分と，$\alpha^{-91} \sim \alpha^{-181}$ を代入する部分を並列に処理するように変更をし，高速化を行った．元に対してのさらなる並列化は可能であるが，並列化による回路規模増加を避け，2 並列とした．

4.4. チェンサーチに必要となるガロア体元の算出方法変更

チェンサーチは誤り位置多項式((2)式))にガロア体の元 α^{-i} を代入するが，α^{-i} を用意するだけではなく高次に代入するガロア体の元を用意する必要がある．具体的に 5 バイトの誤りを訂正する場合について述べ

る．この場合誤り位置多項式は

$$\sigma(x)=\sigma_5x^5+\sigma_4x^4+\sigma_3x^3+\sigma_2x^2+\sigma_1x+\sigma_0 \quad (6)$$

である．

(6)式に入力データの3番目が誤っているか調べているために α^{-3} を代入する場合，実際には x^2 には $\alpha^{-3\times2}=\alpha^{-6}$，$x^3$ には $\alpha^{-3\times3}=\alpha^{-9}$，$x^4$ には $\alpha^{-3\times4}=\alpha^{-12}$，$x^5$ には $\alpha^{-3\times5}=\alpha^{-15}$ と4つの代入する元を α^{-3} から計算をして求めている．これでは $\sigma(\alpha^{-3})$ を求めるために α^{-6} から α^{-15} を求める計算が終了するまで待たなければならない．この待ち時間を回避するために代入する5つの元，この場合 α^{-3}，α^{-6}，α^{-9}，α^{-12}，α^{-15} の値をレジスタに保持するよう変更を行った．これにより，項毎の計算の独立性が高まり，待ち時間無く，誤り位置多項式の計算を1クロックで求めることができる．例えば，次のループで $\sigma(\alpha^{-4})$ を求める場合に，x 項は α^{-1}，x^2 項は α^{-2}，x^3 項は α^{-3}，x^4 項は α^{-4}，x^5 項は α^{-5} を，各レジスタから独立に乗算し，1クロックで求めることができる．

4.5. ガロア体乗算器の共有

チェン関数の並列化を行ったことでガロア体の元同士の乗算回路が増加した．このガロア体乗算器はチェンサーチ部分でしか使用していない．他の処理部でのガロア体乗算と回路を共有化し，回路規模の削減を図る．

誤り訂正能力が5バイトの場合，チェンサーチ実行部分で5個，次ループで使用するガロア体の元を計算するために5個で計10個のガロア体乗算回路が必要である．この処理部分を並列化しているため合計20個のガロア体乗算回路を使用した．

syndrome 関数内のシンドローム計算はシンドローム多項式の 10 個の係数を計算するためにガロア体乗算回路を10個使用している．共有できる乗算器はまだ10個があるので，1クロックで2個の係数を処理するように変更し，高速化を図った．

図7はシンドローム計算部の記述である図7(a)は変更前の記述である．シンドローム多項式の係数 $S_0 \sim S_9$ を計算している．これを図7(b)のように変更した．図7(a)では buffer_value の1個の係数しか処理を行っていないが，図7(b)では buffer_value1 と buffer_value2 の2個の係数を1クロックで処理している．この処理でガロア体乗算回路 GF_MUL が20個使用しており，チェンサーチで増加したガロア体乗算回路をすべて共有している．

```
s0=buffer_value^GF_MUL(s0, 0x01);
s1=buffer_value^GF_MUL(s1, 0x02);
s2=buffer_value^GF_MUL(s2, 0x04);
s3=buffer_value^GF_MUL(s3, 0x08);
s4=buffer_value^GF_MUL(s4, 0x10);
s5=buffer_value^GF_MUL(s5, 0x20);
s6=buffer_value^GF_MUL(s6, 0x40);
s7=buffer_value^GF_MUL(s7, 0x80);
s8=buffer_value^GF_MUL(s8, 0x1d);
s9=buffer_value^GF_MUL(s9, 0x3a);
```

(a)1係数/クロック処理

⇩

```
s0=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s0, 0x01)), 0x01);
s1=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s1, 0x02)), 0x02);
s2=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s2, 0x04)), 0x04);
s3=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s3, 0x08)), 0x08);
s4=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s4, 0x10)), 0x10);
s5=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s5, 0x20)), 0x20);
s6=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s6, 0x40)), 0x40);
s7=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s7, 0x80)), 0x80);
s8=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s8, 0x1d)), 0x1d);
s9=buffer_value2^GF_MUL((buffer_value1^GF_MUL(s9, 0x3a)), 0x3a);
```

(b)2係数/クロック処理

図 7　ガロア体乗算回路の共有

5. 実験結果

最適化手法を各関数に適用した結果を示す．syndrome 関数には変数化，データ入力方法の変更とガロア体乗算回路の共有化を，euclid 関数には変数化と多項式除算，多項式乗算の並列化を，chen 関数には変数化とチェンサーチの並列化，チェンサーチに必要となるガロア体元の算出方法の変更を適用した．動作クロックは 14ns であり，回路規模は Synopsys Design Compiler(日立 0.18μm ライブラリを使用)，処理時間は Mentor Graphics Model Sim を用いて測定を行った．

表1に誤り訂正能力が5バイトの回路の結果を示す．syndrome 関数に関する最適化では，回路規模はシンドローム計算の変数化を行うことでシンドローム計算の並列処理が行われ，使用するガロア体乗算回路の個数が増加するが，データ入力方法を変更したことで必要なレジスタ数が減ったためほぼ回路規模の変化は無く，処理時間が約 2.02 倍の高速化となった．

euclid 関数に関する最適化では，諸多項式を配列から変数に変更したことでアドレスデコーダが削減されるため，回路規模が約 2,400 ゲート削減され，処理時間は 20%の高速化となった．

chen 関数に関する最適化では，誤り位置の計算を並列化したため回路規模が増加しているが，処理時間が約 5.68 倍の高速化となっている．

さらに，ガロア体乗算器の共有を適用したものは，chen 関数で使用されているガロア体乗算回路を共有して使用しているため，回路規模はほとんど増加せず処理時間が約 28%の高速化となった．

全体としては，回路規模が約 10%増加したが，処理

時間が約 21 倍の高速化となり最適化の効果があらわれていることがわかる.

表 1 誤り訂正能力 5 バイト実験結果

	最適化方法	回路規模 [ゲート]	処理時間 [ns]
-	C 言語ソフトウェア	-	92,000
①	最適化前	18,519	70,238
②	①+syndrome 関数最適化	18,827	34,804
③	②+euclid 関数最適化	16,456	27,692
④	③+chen 関数最適化	20,306	4,564
⑤	④+ガロア体乗算器の共有	20,572	3,304

表 2 に誤り訂正能力が 8 バイトの回路の結果を示す. syndrome 関数に関する最適化では, 誤り訂正能力が 5 バイトの場合と比較して回路規模が増加しているが, 処理時間は約 2.1 倍の高速化となっている.

euclid 関数に関する最適化では, 誤り訂正能力が 5 バイトの場合とほぼ同様にアドレスデコーダが削減されるため, 回路規模が約 1,600 ゲート削減され, 処理時間は約 25%の高速化となった.

chen 関数に関する最適化では, 誤り訂正能力が 5 バイトの場合と同様, チェンサーチの並列化に伴って回路規模が増加しているが, 処理時間が約 4.8 倍の高速化となっている.

ガロア体乗算回路の共有を適用したものも, 誤り訂正能力が 5 バイトの場合と同様にガロア体乗算回路をうまく共有して使用しているため, 回路規模はほとんど増加せず処理時間が約 17%の高速化となった.

全体としては, 回路規模が約 28%増加したが, 処理時間が約 16 倍の高速化となり最適化の効果があらわれている.

この設計結果は, RT レベル設計と比較し, ほぼ同等であり, 約 3 ヶ月で設計しており, 大幅な設計期間短縮と思われる.

表 2 誤り訂正能力 8 バイト実験結果

	最適化方法	回路規模 [ゲート]	処理時間 [ns]
-	C 言語ソフトウェア	-	210,000
①	最適化前	24,180	111,272
②	①+syndrome 関数最適化	27,687	53,200
③	②+euclid 関数最適化	26,085	39,704
④	③+chen 関数最適化	30,694	8,358
⑤	④+ガロア体乗算器の共有	30,844	6,902

6. まとめと今後の課題

本研究では, 動作合成システムである Bach システムを用いてリードソロモン符号の復号化回路を設計し, 最適化を行い処理時間の向上を実現することができた.

C 言語設計によってハードウェアを設計するにあたり, ソフトウェアで動作することを前提に記述されているソースをそのまま使用しただけではハードウェア動作に向かず高速に動作することできない. これをハードウェア向きに記述を変更することで, 動作合成時のスケジューリングがスマートに行われるようになり, より無駄の少ない回路が作成しやすくなることがわかった.

今後の課題としては, 現在の復号回路は誤り訂正のみの対応であり, 入力データに消失データが含まれている場合の訂正に対応していない. 今後は消失訂正に対応した回路の作成を行う.

謝辞

エラー訂正技術を使用した研究を行うにあたり, エラー訂正プログラムを提供していただいた, 三栄ハイテックス株式会社 大石英一様, 尾崎隆介様に深くお礼申し上げます.

本研究は, 東京大学大規模集積システム設計教育研究センターを通し, シノプシス株式会社の協力で行われたものである.

文 献

[1] 江藤良純, 金子敏信 : 誤り訂正符号とその応用, オーム社, 東京, 1996 年

[2] 松井充, 山岸篤弘, 吉田英夫 : "ブロック符号のハードウェア," 実戦 誤り訂正技術, 井上徹, pp.83-129, トリケップス, 東京, 1996 年

[3] 西村芳一 : 無線データ通信におけるディジタル・エラー訂正技術入門, CQ 出版, 東京, 2004 年

[4] 今井秀樹 : 符号理論, コロナ社, 東京, 1990 年

[5] K. Okada, A. Yamada, T. Kambe : "Hardware Algorithm Optimization Using Bach C", IEICE Trans. Fundamentals vol.E85-A, No.4, (pp835-841), 2002

[6] Bach システムマニュアル : シャープ株式会社提供, 2004 年