C-011

FPGA を用いた LRPC 復号器の試作
Prototype LRPC decoder using FPGA

坂本　翔太† 黒川　恭一† 松原　隆† 岩井　啓輔†
Shota Sakamoto Takakazu Kurokawa Takashi Matsubara Keisuke Iwai

1 はじめに

IoT による通信量の増加に対応するため、ワイヤレスセンサーネットワーク (WSN) にランダム線形ネットワーク符号 (RLNC) を利用し通信効率を改善する研究が行われている。RLNC は中間ノードで同時に受信した信号をランダムな係数を用いて線形結合した後、次のノードに送信することによって通信効率を上昇させる方式である。しかし、中間ノードで線形結合に失敗すると、エラーが符号全体に広がり復号できなくなる。上記のような状態になっても復号できる手段として RLNC と Low Rank Parity Check codes(LRPC 符号) を併用する方法がある [1]。

しかし、LRPC 符号の処理速度が明確に書かれた論文がなく、RLNC に組み合わせた場合、現行の通信のように高速処理することができるのかは明らかになっていない。そこで、LRPC 符号の有用性を高めるため、FPGA に LRPC 復号器の高速化実装を行う。本稿では、試作した FPGA を用いた LRPC 復号器を評価した。

2 LRPC 符号

LRPC 符号はランク距離符号の 1 種である。ランク距離符号の特性として、ハミング符号と異なり誤り訂正能力がビットエラー数で決まるのでは無く、(n,k)-符号のエラーベクトル $e=(e_1,e_2,\ldots,e_n)$ に対するベクトル空間 $E=\langle e_1,e_2,\ldots,e_n\rangle$ の次元 r によって定まる。ここで $\langle e_1,e_2,\ldots,e_n\rangle$ は $GF(q)$ 上で $e_1,e_2,\ldots,e_n$ により張られるベクトル空間を示す。LRPC 符号はパリティ検査行列の各要素 $h_{ij}\in GF(q^m)$ が張るベクトル空間 $F=\langle h_{11},h_{12},\ldots,h_{(n-k)n}\rangle$ の次元 d が小さいものである。復号限界は $r<(n-k)/2$ である。復号アルゴリズムをアルゴリズム 1 に示す。

LRPC 符号の復号においてベクトル空間 E を求めるまでの処理をランクサポートリカバリ (RSR) という。RSR までの処理については C 言語のライブラリ rbc_lib [2] が存在しているが、LRPC 復号全体のライブラリは現在存在しない。

アルゴリズム 1 LRPC の復号

Input 符号長:n, 情報長:k , 受信符号:$\boldsymbol{c}'=\boldsymbol{c}+\boldsymbol{e}$,
生成行列:$G\in GF(q^m)^{k\times n}$,
パリティ検査行列:$H=\begin{pmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{(n-k)1} & \cdots & h_{(n-k)n}\end{pmatrix}$,
F:$\{f_1,\ldots,f_d\}=base(\langle h_{11},\ldots,h_{(n-k)n}\rangle)$
($base(\cdot)$ は $GF(q)$ 上の基底を示す)
$h_{ijl}\in GF(q)$ $s.t.$ $h_{ij}=\sum_{l=1}^{d}h_{ijl}f_l$
Output 情報ベクトル:$\boldsymbol{v}$

$\boldsymbol{s}^t=(s_1,s_2,\ldots,s_{(n-k)})\leftarrow H\boldsymbol{c}'$
$S=\{s'_1,s'_2,\ldots,s'_t\}\leftarrow base(\langle s_1,\ldots,s_{(n-k)}\rangle)$
▷ $t=\dim_{GF(q)}(\langle S\rangle)$
for $i=1,2,3,\ldots,d$ **:**
  $S_i\leftarrow\langle s'_1f_i^{-1},s'_2f_i^{-1},\ldots,s'_tf_i^{-1}\rangle$
$E=\{E_1,E_2,\ldots,E_r\}\leftarrow base(S_1\cap S_2\cap\cdots\cap S_d)$
▷ $r=\dim_{GF(q)}(\langle E\rangle)$
if $t\neq rd$ **:**
  $failure$
for $i=1,2,3,\ldots,n-k$ **:**
  $s_{ijl}\in GF(q)\leftarrow$**Solve** $s_i=\sum_{j=1}^{r}\sum_{l=1}^{d}s_{ijl}E_jf_l$
$x\in GF(q)^{n\times r}\leftarrow$ **Solve**

$$\begin{pmatrix} h_{111} & \cdots & h_{1n1} \\ h_{112} & \cdots & h_{1n2} \\ \vdots & \ddots & \vdots \\ h_{211} & \cdots & h_{2n1} \\ \vdots & \ddots & \vdots \\ h_{(n-k)1d} & \cdots & h_{(n-k)nd}\end{pmatrix}x=\begin{pmatrix} s_{111} & \cdots & s_{1r1} \\ s_{112} & \cdots & s_{1r2} \\ \vdots & \ddots & \vdots \\ s_{211} & \cdots & s_{2r1} \\ \vdots & \ddots & \vdots \\ s_{(n-k)d1} & \cdots & s_{(n-k)dr}\end{pmatrix}$$

for $i=1,2,3,\ldots,n$ **:**
  $e_i\leftarrow\sum_{j=1}^{r}x_{ij}E_j$
Solve $G\boldsymbol{v}=\boldsymbol{c}'-\boldsymbol{e}$ ▷ $\boldsymbol{e}^t=(e_1,e_2,\ldots,e_n)$
return $\boldsymbol{v}$

3 実装

3.1 実装環境

本研究では [1] で使用されているパラメータ $n=30$, $k=15,q=2,m=30,0\le r\le 7$ を用いて LRPC 復号器を FPGA 実装した。Verilog-HDL で回路を記述し、Xilinx 社の提供する開発環境ソフトウェア Vivado2021.2 を用いて xc7vx485tffg1761-2 に実装した。

3.2 モジュールの作成

FPGA 実装を行うにあたり復号のアルゴリズムのモジュール化を行った。LRPC 全体のブロック図を図 1 に、アルゴリズム 1 の行番号とのモジュールの対応および各モジュールの必要サイクル数を表 1 にそれぞれ示す。LRPC 復号器の主な処理は $GF(2^{30})$ 上の乗算と $GF(2)$ 上のガウスジョルダンの消去法となり、図 1 中の GM は $GF(2^{30})$ 上の乗算回路、GJ はガウスジョルダンの消去法回路を表す。各モジュールに計算状態、入力可能状態および出力状態を持たせ、前後のモジュールと状態の情報を共有している。サイクル数に関しては、GJ_no4 が最大で、またエラーベクトルの次元 r が小さいときも不変であることが分かる。今後、パイプライン実装する際のボトルネックとなるため、改善の必要があると考えられる。

3.2.1 $GF(2^{30})$ 上の乗算回路 (GM)

$GF(2^{30})$ 上の乗算回路については 1 サイクルで計算させるため、配線と排他的論理和のみで記述した。

† 防衛大学校

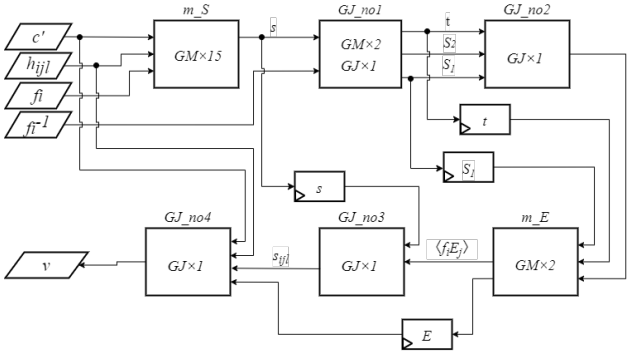

図 1　LRPC 復号器のブロック図

表 1　モジュールとアルゴリズムの対応、サイクル数

アルゴリズム 1 の行番号	モジュール名	サイクル数
1	m_s	$k+2$
2,3,4,5	GJ_no1	$rd+2$
6,7,8,9	GJ_no2 m_E	$2rd+2$ $rd+2$
10,11	GJ_no3	$rd+2$
12,13,14,15,16	GJ_no4	$n+2$
-	合計	$5rd+n+k+12$

3.2.2　ガウスジョルダンの消去法回路 (GJ)

ガウスジョルダンの消去法回路については、ピボットの行を入れ替えて計算するのではなく、図 2 のように計算する回路で実装した。図 2 は $GF(2)^{(y\times z)}$ の行列を計算し、ピボットが i 列目の瞬間を表す。入力側のレジスタと同じ大きさのレジスタを出力側に準備し、入力側のピボットの列で上から検索し最初に 1 が出るところをピボットにし、ピボットの行を決定する。この選択された行を用いて入力側 · 出力側両方のレジスタを更新している。

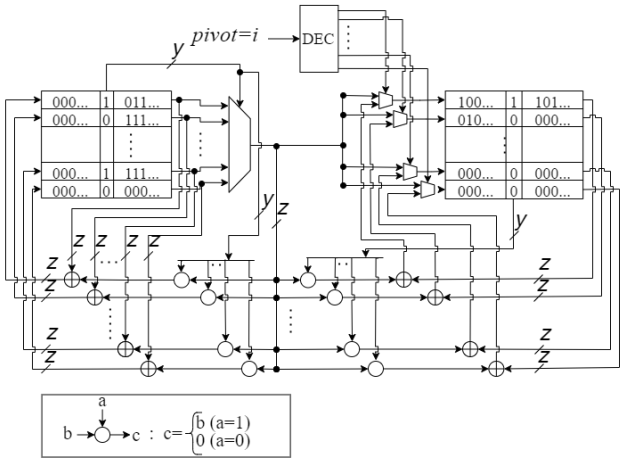

図 2　$GF(2)$ 上のガウスジョルダンの消去法回路

4　評価

4.1　処理速度

処理速度の評価については CPU と FPGA に同じ受信符号を処理させた時にかかる時間を比較した。本実装における FPGA の最大動作周波数は 149.25MHz であった。CPU の環境を表 2 に示す。

処理時間は、エラーベクトルの要素がなすベクトル空間の次元 r の大きさに伴い大きくなるため、受信符号の要素を一定数 0 にするエラーを挿入することで r の値を制御した。エラー数に対する復号速度を表 3 に示す。比較の結果、FPGA の処理速度は最小でも CPU の 12 倍であることがわかった。FPGA による LRPC 復号のスループットは $r=7$ で 517Mbps であった。

表 2　CPU 環境

CPU	Intel Core i7-9700 3.0GHz
OS	Ubuntu-20.04
メモリ	16GB
コンパイラ	gcc ver9.4.0
オプション	-O3 -mpclmul -msse2

表 3　r に対する処理時間 [μs]

r	0	1	2	3	4	5	6	7
CPU	6.98	8.08	9.57	10.88	12.15	13.57	14.96	16.52
FPGA	0.40	0.63	0.67	0.71	0.75	0.79	0.83	0.87

4.2　リソース利用数

実装時のリソース利用数を表 4 に示す。実装した際のリソース利用率は全体の 10%に収まり、本実装法でも 1 チップで最大 10 段の並列化が可能であることがわかった。

表 4　リソース利用数（率）

LUT	Register	F7 Mux	F8 Mux
28751 (9.63%)	12351 (2.36%)	870 (0.57%)	280 (0.37%)

5　結論

本稿では、FPGA による LRPC 復号器の試作を行い、処理速度およびリソース利用数の評価を行った。並列化を行わなくとも CPU の 12 倍以上の速度で復号することができ、FPGA 実装が有意であることがわかった。今後は本復号器を基に、より高いスループットを出すパイプライン実装および、省リソース設計した後の並列化を行いさらなる高速化実装を目指す。

参考文献

[1] Aragon, Nicolas, et al. “Extended Low Rank Parity Check Codes and Their Efficient Decoding for Multisource Wireless Sensor Networks,” International Symposium on Ubiquitous Networking, Springer, Cham, pp. 41-55, 2019.

[2] Aragon, Nicolas, et al. “The Rank-Based Cryptography Library,” Code-Based Cryptography Workshop, Springer, Cham, pp. 22-41, 2021.