

マイクロコンピュータ上での Berlekamp-Massey アルゴリズムを用いた BCH 符号の復号†

岡 野 博 一††

誤り訂正符号が衛星通信、電子計算機などのデジタルシステムの信頼性向上のために実用されている。BCH 符号は高い誤り訂正能力をもち実用上重要な符号であるが、誤り訂正能力の増大につれて復号器が複雑になる。とくに、Berlekamp-Massey (以下、B-M と略す) アルゴリズムは複雑なので、このアルゴリズムを用いたハードによる実用的復号器は開発されていないようであり、ガロア体を扱う専用計算機を用いてソフトウェアによって実現することが提案されている¹⁾。本論文において、まずマイクロコンピュータ上でソフトウェアによって実現した B-M アルゴリズムを用いた BCH 符号の復号プログラムとその復号性能を示す。そして、これまでの著者の成果をも加えて、マイクロコンピュータによるソフトとハード回路の両方を用いる場合の BCH 符号復号のための有益な設計データを提供し、さらに実用的復号法を明らかにする。たとえば、符号長 $n=127$ 、誤り訂正数 t が 4 以下であれば、すべてをソフトウェアによって復号すると、復号符号速度は 25 k ビット/秒以上となる。また、 $n=127$ 、 $t=5\sim 10$ のときは、Chien のハード回路を用いて一部を処理し、あとは B-M アルゴリズム等を用いてソフトウェアによって復号すれば 16 k $\sim$ 4.5 k ビット/秒の復号符号速度が得られる。なお、以上の値は Z80 程度のマイクロコンピュータを用いた場合である。

1. はじめに

誤り訂正符号が衛星通信、デジタルオーディオディスク (DAD)、電子計算機などのデジタルシステムの信頼性向上のために実用されている。そして、より効率のよい復号法、復号器の研究がなされてきた^{5), 11)}。

著者はマイクロコンピュータ上でソフト的に誤り訂正符号を実現することを試み、Fire 符号の復号法^{6), 7)}、BCH 符号の復号法^{8), 9)}を提案している。そして、シンドロームを高速に算出する方法として、シフト演算法⁶⁾、テーブル・ルックアップ法⁹⁾の有効性を示し、さらにガロア体上の 4 次以下の方程式の効率のよい解法を提案している⁹⁾。

さて、BCH 符号は誤り訂正能力が高く優れた符号であるが、誤り訂正能力の増大につれて復号法が複雑になる。誤り訂正能力が 4 重誤り以下の場合には、効率的な復号法が開発されている^{5), 8), 9), 10)}。しかし、誤り訂正能力が 5 重誤り以上になると Berlekamp-Massey (以下、B-M と略す) アルゴリズム^{1), 2), 10)}、Chien アルゴリズム^{1), 4)}を用いねばならなくなり、復号法は複雑となる。この場合は、すべてをハード的に復号器を

構成することは困難でもあり、経済的でないと考えられる。したがって、とくに B-M アルゴリズムの部分にはガロア体を扱う専用計算機を用いてソフトウェアによって実現することが提案されている¹²⁾。

本論文において、B-M アルゴリズムを用いた 4 重および 10 重誤り訂正 BCH 符号の復号プログラムとその復号性能を示す。言語はマイクロコンピュータ PFL-16 A のアセンブラである。B-M アルゴリズムをソフトウェアによって実現し、他の部分をハード回路を用いた場合、符号長 $n=127$ 、誤り訂正数 $t=10$ の場合、復号符号速度 $S=5.2$ k ビット/秒が得られ、 $n=1,023$ 、 $t=10$ のとき $S=42$ k ビット/秒となる。したがって、B-M アルゴリズムを実現するために高価な専用計算機あるいは復号器を用いなくても、低中速のデータ回線等には本稿で示した復号プログラムが十分実用的である。また、マイクロコンピュータ程度のもので、10 重誤り訂正 BCH 符号が実現できることは工学的に重要であろう。

さらに、著者のこれまでの成果をも加えて、マイクロコンピュータによるソフトウェアとハード回路の両方を用いる場合の BCH 符号復号のための有益な設計データを提供し、同時に実用的復号法を明らかにする。

たとえば、 $n=127$ 、 $t\leq 4$ の場合、すべてをソフトウェアによって復号すると $S>25$ k ビット/秒となる。また、 $n=127$ 、 $t=5\sim 10$ の場合、Chien アルゴリズム

† Decoding of BCH Codes Using Berlekamp-Massey Algorithms on a Microcomputer by HIROKAZU OKANO (Department of Information Electronics, Tokuyama Technical College).

†† 徳山工業高等専門学校情報電子工学科

をハード回路で処理し、あとのシンドローム算出、B-M アルゴリズム等はソフトウェアによって処理すれば、 $S=16k\sim 4.5k$ ビット/秒となる。なお、以上の復号符号度は Z80, i8080 程度のマイクロコンピュータを用いた値であるので、Z8000, i8086 等を用いれば、さらに5倍程度の高速処理が実現されよう。

2. BCH 符号

2元 BCH 符号について簡単に述べる^{1),4)}。生成多項式 $g(x)$ は、 $GF(2^m)$ の原始元を α 、最小距離を d とすると、 $\alpha, \alpha^2, \dots, \alpha^{d-1}$ を根とする最小次のモニック多項式で与えられる。符号長は $n=2^m-1$ であり、誤り訂正可能数を t とすると、 $d=2t+1$ である。

さて、 t 個の誤りが生じたとして、 j 番目の誤り位置を X_j で表すと、シンドロームは次式で与えられる。

$$S_i = \sum_{j=1}^t X_j^i, \quad i=1, 2, \dots, d-1 \quad (1)$$

ただし、MOD 2 の性質から $S_{2t} = S_t^2$ である。

また、誤り位置を X_j ($j=1, 2, \dots, t$) とすると、誤り位置多項式は次式となる。

$$\sigma(D) = \prod_{j=1}^t (1 - X_j D) = \sigma_0 + \sigma_1 D + \dots + \sigma_t D^t$$

ただし、 $\sigma_0 = 1$ (2)

シンドローム S_i ($i=1, 2, \dots, 2t-1$) と誤り位置多項式の係数 σ_i ($i=1, 2, \dots, t$) との間には次式の関係がある。

$$\begin{bmatrix} S_1 & 1 & 0 & 0 & 0 & 0 & \dots \\ S_3 & S_2 & S_1 & 1 & 0 & 0 & 0 & \dots \\ S_5 & S_4 & S_3 & S_2 & S_1 & 1 & 0 & \dots \\ \vdots & & & & & & \vdots & \\ S_{2t-3} & S_{2t-4} & \dots & \dots & \dots & \dots & S_{t-2} & S_{t-3} \\ S_{2t-1} & S_{2t-2} & \dots & \dots & \dots & \dots & S_t & S_{t-1} \end{bmatrix} \begin{bmatrix} 1 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{t-1} \\ \sigma_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad (3)$$

式(3)より、誤り位置多項式の係数を求めるのが Peterson の方法^{1),4)}である。もう一つの優れた方法として、B-M アルゴリズムがあるが、これについては次章で述べる。

さて、式(2)の根が誤り位置を示すが、式(2)の一般解法として Chien アルゴリズム^{1),4)}がある。この方法は有限体上の方程式を解くために原理的にすべての元を代入して根か否かを確かめるものである。直線的に処理をする場合の Chien アルゴリズムによる巡回誤り訂正ユニットを図 1¹²⁾に示す。まず、 $\sigma_1 \alpha + \sigma_2 \alpha^2 + \dots + \sigma_t \alpha^t$ を計算し 1 ならば式(2)において、 $\sigma(\alpha) = 0$ なので A の出力を 1 にしてバッファからの受

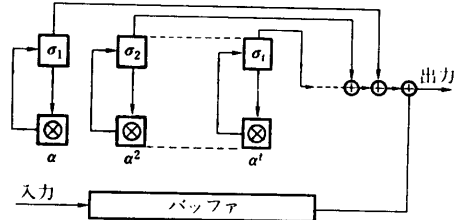


図 1 巡回誤り訂正ユニット
Fig. 1 Cyclic error location unit.

信出力を訂正する。次のクロックでは $\sigma_1 \alpha^2 + \sigma_2 \alpha^4 + \dots + \sigma_t \alpha^{2t}$ が計算され、 $\sigma(\alpha^2) = 0$ かを確かめる。この動作をすべての元に対して行う。なお、ハードウェア量が増加するが、並列的にすべての元を代入する方法も容易に行え、演算時間を短くすることができる。

さて、復号手順を要する。

(ステップ1): 受信符号よりシンドローム S_i を算出する。

(ステップ2): シンドローム S_i から誤り位置多項式の係数 σ_i を算出する。

(ステップ3): 誤り位置多項式を解き、誤り位置 X_j を算出する。

(ステップ4): 誤り位置 X_j により誤りビットを反転して訂正する。

ここで、各ステップの演算手法について述べておく。ステップ1はハードウェアによれば、シフトレジスタまたは EX OR (排他論理和) Tree を用いた回路によって実行される^{1),4),5)}。ソフトウェアによる場合は、シフト演算法⁶⁾およびテーブル・ルックアップ法⁶⁾が有効である。シフト演算法とは、シフトレジスタによる除算をそのまま模擬したものであり、受信符号を生成多項式のパターンで EX OR をとりながらシフトしていくものである。テーブル・ルックアップ法は数ビット (ここでは8ビット) ごとの剰余をあらかじめシフト演算法等で求めて、数ビットごとの入力と剰余との対応表である剰余表を作っておき、この表を使って除算を行うものであり、したがって、シフト演算法に比べて 5~8 倍程度高速に演算を行うことができる。

ステップ2は Peterson の方法と B-M アルゴリズムが用いられることはすでに述べた。さて、誤り訂正数 t が小さい場合 ($t \leq 4$)、あらかじめ式(3)を解いて、 S_i によって直接 σ_i を表す式を使用する (以下、直線公式法と称す) のがソフト的にもハード的にも有効である。たとえば、 $t=2$ とすると次のようになる。

$$1 + \sigma_1 D + \sigma_2 D^2 = 0 \quad (4)$$

ただし,

$$\sigma_1 = S_1, \sigma_2 = S_1^2 + S_3/S_1$$

ステップ 3 はすでに述べたように Chien のアルゴリズムが用いられるが, 方程式を直接解法しないので効率がよくない. 誤り訂正数 t が小さい場合 ($t \leq 4$), Polkinghorn の方法^{3), 5)} および著者の直接解法⁹⁾ が実用的で, ソフト的にもハード的にも効率がよい. Polkinghorn の方法は, 2 次および 3 次方程式をうまく変形し, 定数と根のテーブルを用いて解く方法である. 直接解法は古典代数学の方法を適用し, 4 次方程式を 3 次以下の方程式の解法に帰着させて, Polkinghorn の方法を用いて根を求めるものである.

ステップ 4 は簡単なので説明を省く.

3. Berlekamp-Massey アルゴリズム

復号手順のなかで最も重要なシンδροーム S_i から誤り位置多項式の係数 σ_i を求める (ステップ 2) B-M アルゴリズム^{1), 2), 4), 10)} について述べる. とくに本論文は文献 10) を発展させてまとめたものであり, B-M アルゴリズムについては文献 10) に詳述したので, ここでは要点のみを述べるに止める.

まず, 誤り位置多項式の係数 σ_i ($i=1, 2, \dots, t$) はシンδροーム S_i ($i=1, 2, \dots, 2t-1$) を生成する最小長さの線形フィードバック・シフトレジスタの結合係数であることを示すことができる^{4), 10)}.

さて, B-M アルゴリズム, つまり, 与えられた系列を生成する最小長さの線形フィードバック・シフトレジスタ (以下, LFSR と略す) 生成アルゴリズムを要約する. 図 2 に LFSR を示す. 結合係数は $C_1, C_2, \dots, C_L$ であり, L 段に格納される初期値 $S_0, S_1, \dots, S_{L-1}$ が最初の出力となる. 続く出力は次式で与えられる.

$$S_j = - \sum_{i=1}^L C_i S_{j-i}, \quad j = L, L+1, \dots \quad (5)$$

いま, $S_0, S_1, \dots, S_{n-1}$ までは生成する LFSR が決定しているとする. この LFSR の長さを L_n とすると次

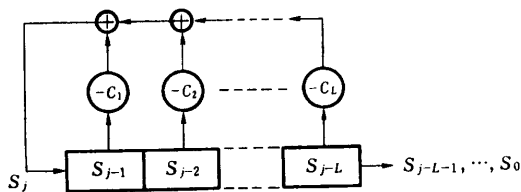


図 2 L 段の線形フィードバックシフトレジスタ
Fig. 2 L -stage linear feedback shift-register (LFSR).

式 $C^{(n)}(D)$ で表される.

$$C^{(n)}(D) = 1 + D_1^{(n)}(D) + \dots + C_L^{(n)} D^L \quad (6)$$

したがって, 次式が成立する.

$$S_j + \sum_{i=1}^{L_n} C_i^{(n)} S_{j-i} = \begin{cases} 0, & j = L_n, L_n+1, \dots, n-1 \\ d_n, & j = n \end{cases} \quad (7)$$

ここで d_n は S_n と回路の第 n 番目の出力との差である. $d_n=0$ のときはこの回路は $S_0, S_1, \dots, S_{n-1}, S_n$ を生成するから,

$$C^{(n+1)}(D) = C^{(n)}(D), \quad L_{n+1} = L_n \quad (8)$$

である. もし, $d_n \neq 0$ ならば, $S_0, S_1, \dots, S_n$ を出力する回路を新しく作らなければならない. さて, m を最後に LFSR の長さを変えた直前の系列の長さとする. したがって, $C^{(m)}(D)$, 長さ L_m の回路が, $S_0, S_1, \dots, S_{m-1}$ は出力するが, 続く S_m は出力しないことを意味するから,

$$S_j + \sum_{i=1}^{L_m} C_i^{(m)} S_{j-i} = \begin{cases} 0, & j = L_m, L_m+1, \dots, m-1 \\ d_m \neq 0, & j = m \end{cases} \quad (9)$$

このとき, $S_0, S_1, \dots, S_n$ を出力する LFSR は次式で与えられる.

$$C^{(n+1)}(D) = C^{(n)}(D) - d_n d_m^{-1} D^{n-m} C^{(m)}(D) \quad (10)$$

また, この LFSR の長さは次式となる.

$$L_{n+1} = \max[L_n, n+1-L_m] \quad (11)$$

以上述べた LFSR の生成アルゴリズムを図 3 に示す. 図 3 で, $2L > N$ のとき LFSR の長さはもとのままであり, そうでないときは LFSR の長さを増や

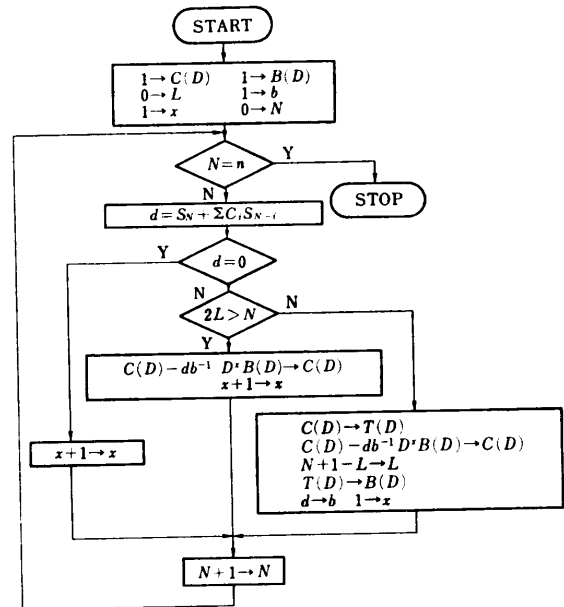


図 3 LFSR の生成アルゴリズム
Fig. 3 LFSR synthesis algorithm.

す。これは式(11)より、長さが変わらないのは、 $l_n \geq n+1-l_n$ 、すなわち、 $2l_n \geq n+1 > n$ となるからである。 $d=d^*$ 、 $b=d^*$ 、 $x=n-m$ 、 $B(D)=C^{(m)}(D)$ 、 $C(D)$ は $C^{(*)}(D)$ または $C^{(*)+1}(D)$ であり、 $T(D)$ は $C(D) \rightarrow B(D)$ とするために一時的に用いるものである。

なお、2元 BCH 符号の場合、 n が奇数のときには $d_n=0$ がつねに成立する。

4. 10 ビット誤り訂正 BCH 符号の復号プログラム

マイクロコンピュータのアセンブリ言語で作成した 10 ビット誤り訂正 BCH 符号の復号プログラムについて述べる。ここで用いる符号は (63, 18) 符号、つまり符号長 63、情報ビット 18、チェックビット 45 であり、符号長 127 の符号を短縮化して用いる。生成多項式は次式で与えられる。

$$G(x) = m_1(x) \cdot m_3(x) \cdot m_5(x) \cdot m_7(x) \cdot m_9(x) \cdot m_{11}(x) \cdot m_{13}(x) \cdot m_{15}(x) \quad (12)$$

ただし、

$$\begin{aligned} m_1(x) &= x^6 + x + 1 \\ m_3(x) &= x^6 + x^4 + x^2 + x + 1 \\ m_5(x) &= x^6 + x^5 + x^2 + x + 1 \\ m_7(x) &= x^6 + x^3 + 1 \\ m_9(x) &= x^3 + x^2 + 1 \\ m_{11}(x) &= x^6 + x^5 + x^3 + x^2 + 1 \\ m_{13}(x) &= x^6 + x^4 + x^3 + x + 1 \\ m_{15}(x) &= x^6 + x^5 + x^4 + x^2 + 1 \end{aligned} \quad (13)$$

シンδροームは受信系列を $m_i(x)$ ($i=1, 3, \dots, 15$) で割ったときの剰余を $R_i(x)$ ($i=1, 3, \dots, 15$) とし、 $GF(2^7)$ の原始元を α とすると次式で求められる。

$$\begin{aligned} S_i &= R_i(\alpha^i), \quad i=1, 3, \dots, 15 \\ S_{17} &= R_8(\alpha^{17}) \\ S_{19} &= R_{13}(\alpha^{19}) \end{aligned} \quad (14)$$

剰余を算出するにはシフト演算法またはテーブル・ルックアップ法を用いればよい。実際には、効率をよくするために、受信符号を $m_1(x) \cdot m_3(x)$ で割ったときの剰余 $R_{13}(x)$ を求め、 $S_1=R_{13}(\alpha)$ 、 $S_3=R_{13}(\alpha^3)$ のようにしてシンδροームを算出している。

さて、シンδροームから誤り位置多項式の係数の算出には 3 章で述べた B-M アルゴリズムを用いる。図 3 のアルゴリズムに従って作成したプログラム BERL 10 のリストを図 4 に示す。 t (≤ 10) 重誤りのとき、シンδροーム $S_1, S_3, \dots, S_{2t-1}$ より誤り位置多項式の係数 $\sigma_1, \sigma_2, \dots, \sigma_t$ を算出する。プログラムの

最初の EQU 文でテーブルを管理する。EV, VE はガロア体のベクトル⇄指数変換表を格納する先頭番地である。その他、図 3 に対応して、 B は $B(D)$ の係数、 C は $C(D)$ の係数、 T は $T(D)$ の係数、 D は d 、 X は x 、 BB は b を表す。また、サブルーチン SUB 10 は、 $C(D) - db^{-1}D^*B(D) \rightarrow C(D)$ を計算する部分であるが紙面の都合でリストは省略する。

つぎに、誤り位置多項式の解法は Chien のアルゴリズムを用いるが、このプログラムは簡単なので説明は省略する。

さて、求めた誤り位置多項式の根が誤り位置を示すので、対応するビットを反転させれば誤り訂正が終了する。

4 重誤り訂正 BCH 符号の復号プログラムも同様であるので、説明は省略する。

さて、マイクロコンピュータ PFL-16 A のアセンブリ言語で作成した、4 重誤り訂正 (63, 39) BCH 符号および 10 重誤り訂正 (63, 18) BCH 符号の復号プログラムの復号性能を表 1 に示す。

Step 1 はテーブル・ルックアップ法とシフト演算法 (() 内の値) による値を示す。前者のほうが 5~8 倍高速であるが、剰余表のためのテーブルを必要とする。

Step 2 は B-M アルゴリズムと直接公式による方法 (() 内の値) による値を示す。4~5 重誤り訂正までは後者が高速で効率がよいが、それ以上になると、プログラムが長大になり前者のほうが有利になると思われる。

Step 3 は Chien のアルゴリズムと直接解法 (() 内の値) による値を示す。4 重誤り訂正までは直接解法が有効ではるかに高速である。

Step 4 はとくに説明を要しない。

表 1 BCH 符号の復号性能

Table 1 Decoding performance of BCH codes.

(単位 ms)

	4重 誤り訂正	10重 誤り訂正
(符号長, 情報ビット)	(63, 39)	(63, 18)
Step 1. シンδροームの算出	0.9	2.0 (13.3)
Step 2. σ_i の算出と誤りビット数の判定	4.1 (1.0)	24.0 —
Step 3. 誤り位置多項式の解法	32.0 (2.5)	85.0 —
Step 4. 誤りの訂正	0.25	1.6
復号時間の合計	37.25 (4.65)	112.6 (123.9)

STMT. LABEL	OP.	OPERANDS/COMMENT	
1	BGN	BERL10	63 J3
2 EV	EQU	X*0000*	64
3 VE	EQU	X*0001*	65
4 R	EQU	X*0002*	66
5 C	EQU	X*0003*	67
6 D	EQU	X*0004*	68 **
7 X	EQU	X*0005*	69
8 T	EQU	X*0006*	70
9 S	EQU	X*0007*	71
10 BR	EQU	X*0008*	72
11 **			73
12	CLEAR	R0	74
13	CLEAR	X0	75
14	CLEAR	R1	76 **
15	MVI	R1,X*42*	77
16 L00P	ST	R0,(R)(X0)	78
17	AI	X0,1	79 L00P2
18	SI	R1,1,Z	80
19	B	L00P	81
20	ST	R0,N	82
21	ST	R0,LX	83
22	MVI	R0,X*01*	84
23	ST	R0,(C)	85
24	ST	R0,(B)	86
25	ST	R0,(BR)	87
26	ST	R0,(X)	88
27 **			89
28 J1	L	R0,N	90
29	SI	R0,10	91
30	SI	R0,9,NZ	92
31	B	JE	93
32	CLEAR	R0	94
33	ST	R0,SNN	95 L00P3
34	L	R0,LX	96
35	SKIP	R0,NZ	97
36	B	J3	98
37 **			99
38	CLEAR	X0	100
39	CLEAR	X1	101
40 L00P1	AI	X0,1	102
41	L	R1,(C)(X0)	103
42	SKIP	R1,NZ	104
43	B	J2	105
44	L	X1,N	106
45	S	X1,X0	107 J4
46	L	X1,(S)(X1)	108
47	SKIP	X1,NZ	109
48	B	J2	110 J5
49	L	R2,(VE)(X1)	111
50	MV	X1,R1	112
51	L	R1,(VE)(X1)	113
52	A	R1,R2	114 **
53	BAL	(OVFN)	115
54	MV	X1,R1	116 SUB10
55	L	R1,(EV)(X1)	117 OVFN
56	L	R2,SNN	118 LX
57	EOR	R1,R2	119 N
58	ST	R1,SNN	120 SNN
59 J2	L	R0,LX	121
60	S	R0,X0,Z	
61	B	L00P1	
62 **			

図 4 Berlekamp-Massey アルゴリズムのプログラム (BERL 10)
Fig. 4 Program for Berlekamp-Massey algorithms (BERL 10).

復号時間の合計は、() なしとありの値を別々に加えた値である。高速なほうの復号符号速度 (n /復号時間) は、4 重誤り訂正 BCH 符号で約 40 k ビット/秒、10 重誤り訂正 BCH 符号で約 500 ビット/秒である。最も時間を要するのは Chien のアルゴリズムの部分である。

なお、実際にプログラムは組んでいないが、汎用コンピュータを用いたとして概算した表 1 と同様のデータが文献 12) に示されている。

5. 復号法の設計

BCH 符号を復号する場合に各復号ステップの演算時間は、元の加算の演算時間と元の乗算の演算時間の和を K とすると次のようになることが、文献 12) で示されている。

Step 1: $ntK - t \times$ (元の加算の演算時間) $\approx ntK$

ただし、 n は符号長、 t は誤り訂正数。

Step 2: $2t^2K$

Step 3: $ntK - n \times$ (元の乗算の演算時間) $\approx ntK$

実際に復号プログラムに組むと、分岐、判断、データのロード等のいろいろな処理が必要となる。しかし、各ステップの演算時間は、種々の処理を K の値に取り込んだ形で上記の式で表される。本稿のプログラムの場合、Step 2, 3 は上記、文献 12) と同じ演算を行っていると考えられる。Step 1 は文献 12) では、受信符号 $r(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \dots + r_1x + r_0$ に $\alpha, \alpha^3, \dots, \alpha^{2t-1}$ を代入して $S_1, S_3, \dots, S_{2t-1}$ を求めているが、本稿ではシフト演算法、またはテーブル・ルックアップ法を用いているので異なった演算処理をする、しかし演算時間はやはり ntK の形になる。

さて、各ステップをソフトとハードの混成で行う復号法のための設計データを提供し、ついで、設計例を示す。この場合、ハードによる部分は高速なので、ソフトによる部分に比較して、復号時間を無視するものとする。

設計データは L-16 A をもとにしている。Z 80, i 8080, M 6800 等も演算速度は同程度である。したがって、Z 8000, i 8086, M 68000 等を用いる場合は、ここで示す設計データの 5 倍程度の演算速度を考慮すればよいであろう。

表 2 演算符号速度 $S_1 (=S_2)$
Table 2 Calculation code rate $S_1 (=S_2)$.
(単位: k ボー)

t	K_1, K_2		
	3 (μ s) (テーブル・ルックアップ法)	25 (μ s) (シフト演算法)	120 (μ s) (チェインの方法)
2	166	20	4.1
3	111	13	2.7
4	83	10	2.0
5	66	8.0	1.6
6	55	6.6	1.3
7	47	5.7	1.1
8	41	5.0	1.0
9	37	4.4	0.9
10	33	4.0	0.8

さて、各ステップの復号時間を実際に作成した復号プログラムから算出し、文献 12) の結果を考慮すると次のことがいえる。まず、Step 1 の復号時間は ntK_1 で表され、シフト演算法を用いたときは $K_1=25 \mu$ s、テーブル・ルックアップ法を用いたときは $K_1=3 \mu$ s、となる。また、演算符号速度 ($n/\text{演算時間}$) $=S_1=1/(tK_1)$ である。 t, K_1 をパラメータとした S_1 を表 2 に示す。また、Step 3 の演算符号速度は、 $S_3=1/(tK_3)$ となり S_1 と同様に表されるので表 2 に含める。ただし、Chien アルゴリズムを用いて、 $K_3=120 \mu$ s となる。つぎに、Step 2 は B-M アルゴリズムを用いると、演算時間は $2t^2K_2$ で表され、演算符号速度は $S_2=n/(2t^2K_2)$ となる。なお、実際のプログラムから演算時間は、 $240t^2+151t \approx 240t^2=2t^2K_2$ となり、 $K_2=120 \mu$ s が算出される。 n, t をパラメータとした S_2 を表 3 に示す。

ここで、誤り訂正数が小さい場合に有効な方法につ

表 3 演算符号速度 S_2 ($K=120 \mu$ s)
Table 3 Calculation code rate S_2 ($K=120 \mu$ s).

t	演算時間 (ms)	S_2 (k ボー)				
		$n=15$	$n=31$	$n=63$	$n=127$	$n=1,023$
2	0.96	15	32	65	132	1,065
3	2.16	6.9	14	24	58	473
4	3.84	3.9	8.0	16	33	266
5	6.0	2.5	5.1	10	21	170
6	8.6	1.7	3.5	7.2	14	118
7	11.76	1.2	2.6	5.3	10	86
8	15.3	0.9	2.0	4.1	8.2	66
9	19.4	0.7	1.5	3.2	6.5	52
10	24.0	0.6	1.2	2.6	5.2	42

表 4 直接公式法による演算符号速度 S_2
Table 4 Calculation code rate S_2 using direct formulas methods.

t	演算時間 T_A (ms)	S_2 (k ボー)				
		$n=15$	$n=31$	$n=63$	$n=127$	$n=1,023$
2	0.1	150	310	630	1,270	10,230
3	0.4	37	77	157	317	2,557
4	1.0	15	31	63	127	1,023

表 5 直接解法による演算符号速度
Table 5 Calculation code rate S_3 using direct solving methods.

t	演算時間 T_B (ms)	S_3 (k ボー)				
		$n=15$	$n=31$	$n=63$	$n=127$	$n=1,023$
2	0.15	100	206	419	846	6,819
3	0.48	31	64	131	264	2,131
4	0.50	6	12	25	50	409

いて述べる。まず、Step 2 は 4 重誤り訂正程度までは直接公式法が演算時間が早い。演算時間を T_A とすると、演算符号速度は $S_2=n/T_A$ となる。著者の文献 8), 9) を参照して演算符号速度 S_2 を求めたものを表 4 に示す。つぎに、Step 3 は 4 重誤り訂正までは直接解法^{8), 9)}を用いることが可能であり、Chien アルゴリズムに比しはるかに高速である。上記と同様の文献を参照して演算符号速度 S_3 を求めたものを表 5 に示す。

さて、以上の表 2~表 5 を用いれば BCH 符号の復号法の設計 (各ステップの復号手法の組合せ) を容易に行うことができる。4 重誤り訂正以内の場合は今井ら⁶⁾および著者ら¹¹⁾によって ROM を用いた効率のよい復号器がすでに提案されている。しかし、5 重誤り訂正以上の場合は Step 2 の B-M アルゴリズムを用いることになるが、このアルゴリズムは、むしろ、復号時間の許される範囲で、ソフト的に実現するのが現実的である¹²⁾。もし、Step 2 のみをマイクロコンピュータのソフトウェアにより実現し、他をすべてハードによって実現すれば演算符号速度は S_2 のみを考えればよい。この場合の復号器の構成図を図 5 に示す。

さて、各ステップの演算符号速度を S_1, S_2, S_3 とすると、総合の復号符号速度 S_T は次式となる。

$$\frac{1}{S_T} = \frac{1}{S_1} + \frac{1}{S_2} + \frac{1}{S_3} \quad (15)$$

次に設計例を示す。

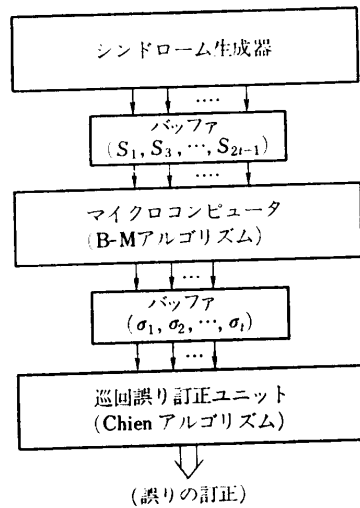


図 5 BCH 符号の復号器
Fig. 5 Decoder for BCH codes.

(1) L-16 A, Z80 等を用いて, $t=10$, $n=127$ の場合の BCH 符号の復号法を求めよ。

解) Step 1 はテーブル・ルックアップ法を用いるとすると, 表 2 より $S_1=33$ k ビット/秒. Step 2 は B-M アルゴリズムを用いて, 表 3 より $S_2=5.2$ k ビット/秒. Step 3 は Chien のアルゴリズムを用いて, 表 1 より $S_3=0.8$ k ビット/秒. したがって, 式(15)より, $S_T=0.68$ k ビット/秒となる. もし, Step 3 をハードで実現すれば S_3 を無視して, $S_T=4.5$ k ビット/秒となる。

(2) Z8000, i8086, M68000 等を用いて, $t=4$, $n=63$ のときの BCH 符号の復号法を求めよ。

解) Step 1 はテーブル・ルックアップ法を用いて, 表 2 より $S_1=83$ k ビット/秒. Step 2 は直接公式による方法を用いて, 表 4 より $S_2=63$ k ビット/秒. Step 3 は直接解法を用いて, 表 5 より $S_3=25$ k ビット/秒. したがって, 式(15)より $S_T=14.7$ k ビット/秒となる. しかし, この値は Z80 程度の場合であるから, 約 5 倍と考えると $S_T=73$ k ビット/秒となる. なお, Step 1, Step 3 をハード化すると, $S_T=315$ k ビット/秒である。

さて, 以上のような復号法の考察を行い, 標準的な復号法による復号符号速度を示すと 図 6 となる. $t=2\sim4$ では Step 1 はテーブル・ルックアップ法, Step 2 は直接公式法, Step 3 は直接解法を用い, すべてソフトウェアによって実現する. $t=5\sim10$ では, Step 1 はテーブル・ルックアップ法, Step 2 は B-M アルゴ

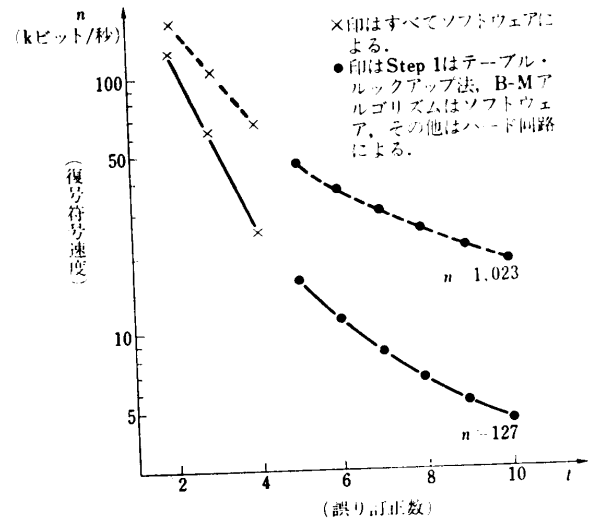


図 6 標準的な復号法の場合の $n-t$ 曲線
Fig. 6 $n-t$ curves for standard decoding.

リズムを用い, ソフトウェアで実現する. Step 3 は Chien アルゴリズムをハード回路で実現したとする。

6. あとがき

BCH 符号の復号アルゴリズムとして非常に複雑とされている B-M アルゴリズムを用いた 10 重誤り訂正 BCH 符号の復号がマイクロコンピュータのアセンブラプログラムによって実現できることを明らかにした. さらに, 従来の著者の成果をも加えて, BCH 符号の各種の復号手法を比較し, 復号時間等の検討を行い, マイクロコンピュータのソフトウェアを中心とした BCH 符号の設計法について述べた。

たとえば, B-M アルゴリズムをソフトウェアによって実現し, 他の部分をハード回路を用いた場合, 符号長 $n=127$, 誤り訂正数 $t=10$ の場合, 復号符号速度 $S=5.2$ k ビット/秒が得られ, $n=1,023$, $t=10$ のとき $S=42$ k ビット/秒となる. また, 標準的な復号法としては 図 6 より, $t=2\sim4$ では, すべてをソフトウェアによって実現すると, $n=1,023$ のとき, 64 k ビット/秒以上の復号符号速度が得られる. $t=5\sim10$ では, Step 1 はテーブル・ルックアップ法, Step 2 は B-M アルゴリズムを用いソフトウェアで実現し, Step 3 はハード回路を用いる方法によって $n=1,023$ のとき, 18 k ビット/秒以上の復号符号速度が得られる。

したがって, 本稿で示した B-M アルゴリズムを実現するプログラムは低中速のデータ回線等への実用が

可能である。

なお、上記の値は Z80 程度のマイクロコンピュータを用いた場合なので、Z8000 程度のものを用いれば、さらに約 5 倍の復号符号速度が得られよう。

謝辞 日頃、ご指導、ご助言いただく広島大学工学部市川忠男教授に深謝します。また、ご協力いただいた本学情報電子工学科義永常宏助手、学生（現安川電機）石田政美氏に感謝します。

なお、本研究の一部は文部省科研費、一般 C「誤り訂正符号の研究」（昭和 57 年度）の援助のもとに行われた。

参 考 文 献

- 1) Berlekamp, E. R.: *Algebraic Coding Theory*, pp. 176-199, McGraw-Hill Book Co., New York (1968).
- 2) Massey, J. L.: Shift Register Synthesis and BCH Decoding, *IEEE Trans.*, IT-15, pp. 122-127 (1969).
- 3) Polkinghorn, F.: Decoding of Double and Triple Correcting Bose-Chaudhuri Codes, *IEEE Trans.*, IT-12, pp. 480-481 (Oct. 1966).
- 4) 宮川, 岩垂, 今井: 符号理論, pp. 247-273, 昭晃堂, 東京 (1973).
- 5) 山岸, 今井: ROM を用いた BCH 符号の復号器の一構成法, 信学論, Vol. J 63-D, No. 12, pp. 1034-1041 (1980).
- 6) 岡野博一: マイクロコンピュータによる Fire 符号のシミュレーションプログラム—データ通信への応用—, 情報処理学会論文誌, Vol. 20, No. 6, pp. 468-473 (1979).
- 7) 岡野博一: マイクロコンピュータによる Fire 符号の高速復号法, 情報処理学会論文誌, Vol. 21, No. 5, pp. 375-382 (1980).
- 8) 岡野博一: マイクロコンピュータによる 2 重および 3 重誤り訂正 BCH 符号の復号, 情報処理学会論文誌, Vol. 21, No. 6, pp. 500-506 (1980).
- 9) 岡野博一: 誤り位置多項式の直接解法による 3 重および 4 重誤り訂正 BCH 符号の復号, 信学論, Vol. J 62-A, No. 2, pp. 137-144 (1981).
- 10) 岡野, 義永, 石田: Berlekamp-Massey アルゴリズムを用いた BCH 符号の復号, 信学技報, AL 82-55, pp. 95-103 (1982).
- 11) 岡野博一: ROM を用いた BCH 符号復号器の改良, 信学技報, AL 82-56, pp. 105-112 (1980).
- 12) Lin, S.: *An Introduction to Error-Correcting Codes*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 112-136 (1970).

(昭和 58 年 10 月 28 日受付)

(昭和 59 年 12 月 20 日採録)