CHAPTER 6

Binary BCH Codes

The Bose, Chaudhuri, and Hocquenghem (BCH) codes form a large class of powerful random error-correcting cyclic codes. This class of codes is a remarkable generalization of the Hamming codes for multiple-error correction. Binary BCH codes were discovered by Hocquenghem in 1959 [1] and independently by Bose and Chaudhuri in 1960 [2]. The cyclic structure of these codes was proved by Peterson in 1960 [3]. Binary BCH codes were generalized to codes in p^m symbols (where p is a prime) by Gorenstein and Zierler in 1961 [4]. Among the nonbinary BCH codes, the most important subclass is the class of Reed–Solomon (RS) codes. The RS codes were discovered by Reed and Solomon in 1960 [5] independently of the work by Hocquenghem, Bose, and Chaudhuri.

The first decoding algorithm for binary BCH codes was devised by Peterson in 1960 [3]. Then, Peterson's algorithm was generalized and refined by Gorenstein and Zierler [4], Chien [6], Forney [7], Berlekamp [8, 9], Massey [10, 11], Burton [12], and others. Among all the decoding algorithms for BCH codes, Berlekamp's iterative algorithm, and Chien's search algorithm are the most efficient ones.

In this chapter we consider primarily a subclass of the binary BCH codes that is the most important subclass from the standpoint of both theory and implementation. Nonbinary BCH codes and Reed–Solomon codes will be discussed in Chapter 7. For a detailed description of the BCH codes, and their algebraic properties and decoding algorithms, the reader is referred to [9] and [13–17].

6.1 BINARY PRIMITIVE BCH CODES

For any positive integers $m (m \geq 3)$ and $t (t < 2^{m-1})$, there exists a binary BCH code with the following parameters:

Block length:	$n = 2^m - 1$,
Number of parity-check digits:	$n - k \leq mt$,
Minimum distance:	$d_{min} \geq 2t + 1$.

Clearly, this code is capable of correcting any combination of t or fewer errors in a block of $n = 2^m - 1$ digits. We call this code a t-error-correcting BCH code. The generator polynomial of this code is specified in terms of its roots from the Galois field $GF(2^m)$. Let α be a primitive element in $GF(2^m)$. The generator polynomial $g(X)$ of the t-error-correcting BCH code of length $2^m - 1$ is the *lowest-degree polynomial* over $GF(2)$ that has

$$\alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2t} \tag{6.1}$$

as its roots [i.e., $g(\alpha^i) = 0$ for $1 \leq i \leq 2t$]. It follows from Theorem 2.11 that $g(X)$ has $\alpha, \alpha^2, \cdots, \alpha^{2t}$ and their conjugates as all its roots. Let $\phi_i(X)$ be the

minimal polynomial of α^i. Then, $g(X)$ must be the *least common multiple* (LCM) of $\phi_1(X), \phi_2(X), \cdots, \phi_{2t}(X)$, that is,

$$g(X) = \text{LCM}\{\phi_1(X), \phi_2(X), \cdots, \phi_{2t}(X)\}. \tag{6.2}$$

If i is an even integer, it can be expressed as a product of the following form:

$$i = i'2^l,$$

where i' is an odd number, and $l \geq 1$. Then, $\alpha^i = (\alpha^{i'})^{2^l}$ is a conjugate of $\alpha^{i'}$, and therefore α^i and $\alpha^{i'}$ have the same minimal polynomial; that is,

$$\phi_i(X) = \phi_{i'}(X).$$

Hence, every even power of α in the sequence of (6.1) has the same minimal polynomial as some preceding odd power of α in the sequence. As a result, the generator polynomial $g(X)$ of the binary t-error-correcting BCH code of length $2^m - 1$ given by (6.2) can be reduced to

$$g(X) = \text{LCM}\{\phi_1(X), \phi_3(X), \cdots, \phi_{2t-1}(X)\}. \tag{6.3}$$

Because the degree of each minimal polynomial is m or less, the degree of $g(X)$ is at most mt; that is, the number of parity-check digits, $n - k$, of the code is at most equal to mt. There is no simple formula for enumerating $n - k$, but if t is small, $n - k$ is exactly equal to mt [9, 18]. The parameters for all binary BCH codes of length $2^m - 1$ with $m \leq 10$ are given in Table 6.1. The BCH codes just defined are usually called *primitive* (or *narrow-sense*) BCH codes.

TABLE 6.1: BCH codes generated by primitive elements of order less than 2^{10}.

n	k	t	n	k	t	n	k	t
7	4	1	127	50	13	255	71	29
15	11	1		43	14		63	30
	7	2		36	14		55	31
	5	3		29	21		47	42
31	26	1		22	23		45	43
	21	2		15	27		37	45
	16	3		8	31		29	47
	11	5	255	247	1		21	55
	6	7		239	2		13	59
63	57	1		231	3		9	63
	51	2		223	4	511	502	1
	45	3		215	5		493	2
	39	4		207	6		484	3
	36	5		199	7		475	4
	30	6		191	8		466	5
	24	7		187	9		457	6

(*continued overleaf*)

TABLE 6.1: (*continued*)

n	k	t	n	k	t	n	k	t
	18	10		179	10		448	7
	16	11		171	11		439	8
	10	13		163	12		430	9
	7	15		155	13		421	10
127	120	1		147	14		412	11
	113	2		139	18		403	12
	106	3		131	19		394	13
	99	4		123	21		385	14
	92	5		115	22		376	15
	85	6		107	23		367	16
	78	7		99	24		358	18
	71	9		91	25		349	19
	64	10		87	26		340	20
	57	11		79	27		331	21
511	322	22	511	166	47	511	10	121
	313	23		157	51	1023	1013	1
	304	25		148	53		1003	2
	295	26		139	54		993	3
	286	27		130	55		983	4
	277	28		121	58		973	5
	268	29		112	59		963	6
	259	30		103	61		953	7
	250	31		94	62		943	8
	241	36		85	63		933	9
	238	37		76	85		923	10
	229	38		67	87		913	11
	220	39		58	91		903	12
	211	41		49	93		893	13
	202	42		40	95		883	14
	193	43		31	109		873	15
	184	45		28	111		863	16
	175	46		19	119		858	17
1023	848	18	1023	553	52	1023	268	103
	838	19		543	53		258	106
	828	20		533	54		249	107
	818	21		523	55		238	109
	808	22		513	57		228	110
	798	23		503	58		218	111
	788	24		493	59		208	115
	778	25		483	60		203	117
	768	26		473	61		193	118
	758	27		463	62		183	119
	748	28		453	63		173	122
	738	29		443	73		163	123

TABLE 6.1: (*continued*)

n	k	t	n	k	t	n	k	t
	728	30		433	74		153	125
	718	31		423	75		143	126
	708	34		413	77		133	127
	698	35		403	78		123	170
	688	36		393	79		121	171
	678	37		383	82		111	173
	668	38		378	83		101	175
	658	39		368	85		91	181
	648	41		358	86		86	183
	638	42		348	87		76	187
	628	43		338	89		66	189
	618	44		328	90		56	191
	608	45		318	91		46	219
	598	46		308	93		36	223
	588	47		298	94		26	239
	578	49		288	95		16	147
	573	50		278	102		11	255
	563	51						

From (6.3), we see that the single-error-correcting BCH code of length $2^m - 1$ is generated by

$$g(X) = \phi_1(X).$$

Because α is a primitive element of $GF(2^m)$, $\phi_1(X)$ is a primitive polynomial of degree m. Therefore, the single-error-correcting BCH code of length $2^m - 1$ is a Hamming code.

EXAMPLE 6.1

Let α be a primitive element of the Galois field $GF(2^4)$ given by Table 2.8 such that $1 + \alpha + \alpha^4 = 0$. From Table 2.9 we find that the minimal polynomials of α, α^3, and α^5 are

$$\phi_1(X) = 1 + X + X^4,$$
$$\phi_3(X) = 1 + X + X^2 + X^3 + X^4,$$

and

$$\phi_5(X) = 1 + X + X^2,$$

respectively. It follows from (6.3) that the double-error-correcting BCH code of length $n = 2^4 - 1 = 15$ is generated by

$$g(X) = \text{LCM}\{\phi_1(X), \phi_3(X)\}.$$

Because $\phi_1(X)$ and $\phi_3(X)$ are two distinct irreducible polynomials,

$$\begin{aligned} g(X) &= \phi_1(X)\phi_3(X) \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4) \\ &= 1 + X^4 + X^6 + X^7 + X^8. \end{aligned}$$

Thus, the code is a (15, 7) cyclic code with $d_{min} \geq 5$. Since the generator polynomial is a code polynomial of weight 5, the minimum distance of this code is exactly 5.

The triple-error-correcting BCH code of length 15 is generated by

$$\begin{aligned} g(X) &= \text{LCM}\{\phi_1(X), \phi_3(X), \phi_5(X)\} \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)(1 + X + X^2) \\ &= 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}. \end{aligned}$$

This triple-error-correcting BCH code is a (15, 5) cyclic code with $d_{min} \geq 7$. Because the weight of the generator polynomial is 7, the minimum distance of this code is exactly 7.

Using the primitive polynomial $p(X) = 1 + X + X^6$, we may construct the Galois field $GF(2^6)$, as shown in Table 6.2. The minimal polynomials of the elements

TABLE 6.2: Galois field $GF(2^6)$ with $p(\alpha) = 1 + \alpha + \alpha^6 = 0$.

0	0									(0 0 0 0 0 0)
1	1									(1 0 0 0 0 0)
α			α							(0 1 0 0 0 0)
α^2					α^2					(0 0 1 0 0 0)
α^3							α^3			(0 0 0 1 0 0)
α^4								α^4		(0 0 0 0 1 0)
α^5									α^5	(0 0 0 0 0 1)
α^6	1	+	α							(1 1 0 0 0 0)
α^7			α	+	α^2					(0 1 1 0 0 0)
α^8					α^2	+	α^3			(0 0 1 1 0 0)
α^9							α^3	$+\alpha^4$		(0 0 0 1 1 0)
α^{10}								α^4	+ α^5	(0 0 0 0 1 1)
α^{11}	1	+	α						+ α^5	(1 1 0 0 0 1)
α^{12}	1			+	α^2					(1 0 1 0 0 0)
α^{13}			α				α^3			(0 1 0 1 0 0)
α^{14}					α^2			$+\alpha^4$		(0 0 1 0 1 0)
α^{15}							α^3		+ α^5	(0 0 0 1 0 1)
α^{16}	1	+	α					$+\alpha^4$		(1 1 0 0 1 0)
α^{17}			α	+	α^2				+ α^5	(0 1 1 0 0 1)
α^{18}	1	+	α	+	α^2	+	α^3			(1 1 1 1 0 0)
α^{19}			α	+	α^2	+	α^3	$+\alpha^4$		(0 1 1 1 1 0)
α^{20}					α^2	+	α^3	$+\alpha^4$	+ α^5	(0 0 1 1 1 1)

TABLE 6.2: (*continued*)

α^{21}	1	+	α			+	α^3	$+\alpha^4$	+	α^5		(1 1 0 1 1 1)
α^{22}	1			+	α^2			$+\alpha^4$	+	α^5		(1 0 1 0 1 1)
α^{23}	1					+	α^3		+	α^5		(1 0 0 1 0 1)
α^{24}	1							$+\alpha^4$				(1 0 0 0 1 0)
α^{25}			α						+	α^5		(0 1 0 0 0 1)
α^{26}	1	+	α	+	α^2							(1 1 1 0 0 0)
α^{27}			α	+	α^2	+	α^3					(0 1 1 1 0 0)
α^{28}					α^2	+	α^3	$+\alpha^4$				(0 0 1 1 1 0)
α^{29}							α^3	$+\alpha^4$	+	α^5		(0 0 0 1 1 1)
α^{30}	1	+	α									(1 1 0 0 1 1)
α^{31}	1			+	α^2					+	α^5	(1 0 1 0 0 1)
α^{32}	1					+	α^3					(1 0 0 1 0 0)
α^{33}			α						α^4			(0 1 0 0 1 0)
α^{34}					α^2					+	α^5	(0 0 1 0 0 1)
α^{35}	1	+	α			+	α^3					(1 1 0 1 0 0)
α^{36}			α	+	α^2			+	α^4			(0 1 1 0 1 0)
α^{37}					α^2		α^3			+	α^5	(0 0 1 1 0 1)
α^{38}	1	+	α			+	α^3	+	α^4			(1 1 0 1 1 0)
α^{39}			α	+	α^2			+	α^4	+	α^5	(0 1 1 0 1 1)
α^{40}	1	+	α	+	α^2	+	α^3			+	α^5	(1 1 1 1 0 1)
α^{41}	1			+	α^2	+	α^3	+	α^4			(1 0 1 1 1 0)
α^{42}			α			+	α^3	+	α^4	+	α^5	(0 1 0 1 1 1)
α^{43}	1	+	α	+	α^2			+	α^4	+	α^5	(1 1 1 0 1 1)
α^{44}	1			+	α^2	+	α^3			+	α^5	(1 0 1 1 0 1)
α^{45}	1					+	α^3	+	α^4			(1 0 0 1 1 0)
α^{46}			α					+	α^4	+	α^5	(0 1 0 0 1 1)
α^{47}	1	+	α	+	α^2					+	α^5	(1 1 1 0 0 1)
α^{48}	1			+	α^2	+	α^3					(1 0 1 1 0 0)
α^{49}			α			+	α^3	+	α^4			(0 1 0 1 1 0)
α^{50}					α^2			+	α^4		α^5	(0 0 1 0 1 1)
α^{51}	1	+	α			+	α^3			+	α^5	(1 1 0 1 0 1)
α^{52}	1			+	α^2			+	α^4			(1 0 1 0 1 0)
α^{53}			α			+	α^3			+	α^5	(0 1 0 1 0 1)
α^{54}	1	+	α	+	α^2			+	α^4			(1 1 1 0 1 0)
α^{55}			α	+	α^2	+	α^3			+	α^5	(0 1 1 1 0 1)
α^{56}	1	+	α	+	α^2	+	α^3	+	α^4			(1 1 1 1 1 0)
α^{57}			α	+	α^2	+	α^3	+	α^4	+	α^5	(0 1 1 1 1 1)
α^{58}	1	+	α	+	α^2	+	α^3	+	α^4	+	α^5	(1 1 1 1 1 1)
α^{59}	1			+	α^2	+	α^3	+	α^4	+	α^5	(1 0 1 1 1 1)
α^{60}	1					+	α^3	+	α^4	+	α^5	(1 0 0 1 1 1)
α^{61}	1							+	α^4	+	α^5	(1 0 0 0 1 1)
α^{62}	1									+	α^5	(1 0 0 0 0 1)
										$\boxed{\alpha^{63} = 1}$		

TABLE 6.3: Minimal polynomials of the elements in $GF(2^6)$.

Elements	Minimal polynomials
$\alpha, \alpha^2, \alpha^4, \alpha^{16}, \alpha^{32}$	$1 + X + X^6$
$\alpha^3, \alpha^6, \alpha^{12}\alpha^{24}, \alpha^{48}\alpha^{33}$	$1 + X + X^2 + X^4 + X^6$
$\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{17}, \alpha^{34}$	$1 + X + X^2 + X^5 + X^6$
$\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56}, \alpha^{49}, \alpha^{35}$	$1 + X^3 + X^6$
$\alpha^9, \alpha^{18}, \alpha^{36}$	$1 + X^2 + X^3$
$\alpha^{11}, \alpha^{22}, \alpha^{44}, \alpha^{25}, \alpha^{50}, \alpha^{37}$	$1 + X^2 + X^3 + X^5 + X^6$
$\alpha^{13}, \alpha^{26}, \alpha^{52}, \alpha^{41}, \alpha^{19}, \alpha^{38}$	$1 + X + X^3 + X^4 + X^6$
$\alpha^{15}, \alpha^{30}, \alpha^{60}, \alpha^{57}, \alpha^{51}, \alpha^{39}$	$1 + X^2 + X^4 + X^5 + X^6$
α^{21}, α^{42}	$1 + X + X^2$
$\alpha^{23}, \alpha^{46}, \alpha^{29}, \alpha^{58}, \alpha^{53}, \alpha^{43}$	$1 + X + X^4 + X^5 + X^6$
$\alpha^{27}, \alpha^{54}, \alpha^{45}$	$1 + X + X^3$
$\alpha^{31}, \alpha^{62}, \alpha^{61}, \alpha^{59}, \alpha^{55}, \alpha^{47}$	$1 + X^5 + X^6$

TABLE 6.4: Generator polynomials of all the BCH codes of length 63.

n	k	t	$\mathrm{g}(X)$
63	57	1	$\mathrm{g}_1(X) = 1 + X + X^6$
	51	2	$\mathrm{g}_2(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)$
	45	3	$\mathrm{g}_3(X) = (1 + X + X^2 + X^5 + X^6)\mathrm{g}_2(X)$
	39	4	$\mathrm{g}_4(X) = (1 + X^3 + X^6)\mathrm{g}_3(X)$
	36	5	$\mathrm{g}_5(X) = (1 + X^2 + X^3)\mathrm{g}_4(X)$
	30	6	$\mathrm{g}_6(X) = (1 + X^2 + X^3 + X^5 + X^6)\mathrm{g}_5(X)$
	24	7	$\mathrm{g}_7(X) = (1 + X + X^3 + X^4 + X^6)\mathrm{g}_6(X)$
	18	10	$\mathrm{g}_{10}(X) = (1 + X^2 + X^4 + X^5 + X^6)\mathrm{g}_7(X)$
	16	11	$\mathrm{g}_{11}(X) = (1 + X + X^2)\mathrm{g}_{10}(X)$
	10	13	$\mathrm{g}_{13}(X) = (1 + X + X^4 + X^5 + X^6)\mathrm{g}_{11}(X)$
	7	15	$\mathrm{g}_{15}(X) = (1 + X + X^3)\mathrm{g}_{13}(X)$

in $GF(2^6)$ are listed in Table 6.3. Using (6.3), we find the generator polynomials of all the BCH codes of length 63, as shown in Table 6.4. The generator polynomials of all binary primitive BCH codes of length $2^m - 1$ with $m \leq 10$ are given in Appendix C.

It follows from the definition of a t-error-correcting BCH code of length $n = 2^m - 1$ that each code polynomial has $\alpha, \alpha^2, \cdots, \alpha^{2t}$ and their conjugates as roots. Now, let $\mathrm{v}(X) = v_0 + v_1 X + \cdots + v_{n-1}X^{n-1}$ be a polynomial with coefficients from $GF(2)$. If $\mathrm{v}(X)$ has $\alpha, \alpha^2, \cdots, \alpha^{2t}$ as roots, it follows from Theorem 2.14 that $\mathrm{v}(X)$ is divisible by the minimal polynomials $\phi_1(X), \phi_2(X), \cdots, \phi_{2t}(X)$ of $\alpha, \alpha^2, \cdots, \alpha^{2t}$. Obviously, $\mathrm{v}(X)$ is divisible by their least common multiple (the generator polynomial),

$$\mathrm{g}(X) = \mathrm{LCM}\{\phi_1(X), \phi_2(X), \cdots, \phi_{2t}(X)\}.$$

Hence, $v(X)$ is a code polynomial. Consequently, we may define a t-error-correcting BCH code of length $n = 2^m - 1$ in the following manner: a binary n-tuple $\mathbf{v} = (v_0, v_1, v_2, \cdots, v_{n-1})$ is a codeword if and only if the polynomial $v(X) = v_0 + v_1 X + \cdots + v_{n-1}X^{n-1}$ has $\alpha, \alpha^2, \cdots, \alpha^{2t}$ as roots. This definition is useful in proving the minimum distance of the code.

Let $v(X) = v_0 + v_1 X + \cdots + v_{n-1}X^{n-1}$ be a code polynomial in a t-error-correcting BCH code of length $n = 2^m - 1$. Because α^i is a root of $v(X)$ for $1 \le i \le 2t$, then

$$v(\alpha^i) = v_0 + v_1\alpha^i + v_2\alpha^{2i} + \cdots + v_{n-1}\alpha^{(n-1)i} = 0. \tag{6.4}$$

This equality can be written as a matrix product as follows:

$$(v_0, v_1, \cdots, v_{n-1}) \cdot \begin{bmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \vdots \\ \alpha^{(n-1)i} \end{bmatrix} = 0 \tag{6.5}$$

for $1 \le i \le 2t$. The condition given by (6.5) simply says that the inner product of $(v_0, v_1, \cdots, v_{n-1})$ and $(1, \alpha^i, \alpha^{2i}, \cdots, \alpha^{(n-1)i})$ is equal to zero. Now, we form the following matrix:

$$\mathbb{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & (\alpha^2) & (\alpha^2)^2 & (\alpha^2)^3 & \cdots & (\alpha^2)^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ \vdots & & & & & \vdots \\ 1 & (\alpha^{2t}) & (\alpha^{2t})^2 & (\alpha^{2t})^3 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix}. \tag{6.6}$$

It follows from (6.5) that if $\mathbf{v} = (v_0, v_1, \cdots, v_{n-1})$ is a codeword in the t-error-correcting BCH code, then

$$\mathbf{v} \cdot \mathbb{H}^{\mathrm{T}} = \mathbb{0}. \tag{6.7}$$

On the other hand, if an n-tuple $\mathbf{v} = (v_0, v_1, \cdots, v_{n-1})$ satisfies the condition of (6.7), it follows from (6.5) and (6.4) that, for $1 \le i \le 2t$, α^i is a root of the polynomial $v(X)$. Therefore, $\mathbf{v}$ must be a codeword in the t-error-correcting BCH code. Hence, the code is the null space of the matrix $\mathbb{H}$, and $\mathbb{H}$ is a parity-check matrix of the code. If for some i and j, α^j is a conjugate of α^i, then $v(\alpha^j) = 0$ if and only if $v(\alpha^i) = 0$ (see Theorem 2.11); that is, if the inner product of $\mathbf{v} = (v_0, v_1, \cdots, v_{n-1})$ and the ith row of $\mathbb{H}$ is zero, the inner product of $\mathbf{v}$ and the jth row of $\mathbb{H}$ is also zero. For this reason, the jth row of $\mathbb{H}$ can be omitted. As a result, the $\mathbb{H}$ matrix given by (6.6) can be reduced to the following form:

$$\mathbb{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ 1 & (\alpha^5) & (\alpha^5)^2 & (\alpha^5)^3 & \cdots & (\alpha^5)^{n-1} \\ \vdots & & & & & \vdots \\ 1 & (\alpha^{2t-1}) & (\alpha^{2t-1})^2 & (\alpha^{2t-1})^3 & \cdots & (\alpha^{2t-1})^{n-1} \end{bmatrix}. \tag{6.8}$$

Note that the entries of $\mathbb{H}$ are elements in $GF(2^m)$. We can represent each element in $GF(2^m)$ by an m-tuple over $GF(2)$. If we replace each entry of $\mathbb{H}$ with its corresponding m-tuple over $GF(2)$ arranged in column form, we obtain a binary parity-check matrix for the code.

EXAMPLE 6.2

Consider the double-error-correcting BCH code of length $n = 2^4 - 1 = 15$. From Example 6.1 we know that this is a (15, 7) code. Let α be a primitive element in $GF(2^4)$. Then, the parity-check matrix of this code is

$$\mathbb{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^{15} & \alpha^{18} & \alpha^{21} & \alpha^{24} & \alpha^{27} & \alpha^{30} & \alpha^{33} & \alpha^{36} & \alpha^{39} & \alpha^{42} \end{bmatrix}$$

[by (6.8)]. Using Table 2.8 and the fact that $\alpha^{15} = 1$, and representing each entry of $\mathbb{H}$ with its corresponding 4-tuple, we obtain the following binary parity-check matrix for the code:

$$\mathbb{H} = \left[\begin{array}{cccccccccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

Now, we are ready to prove that the t-error-correcting BCH code just defined indeed has a minimum distance of at least $2t + 1$. To prove this, we need to show that no $2t$ or fewer columns of $\mathbb{H}$ given by (6.6) sum to zero (Corollary 3.2.1). Suppose that there exists a nonzero codeword $\mathbf{v} = (v_0, v_1, \cdots, v_{n-1})$ with weight $\delta \leq 2t$. Let $v_{j_1}, v_{j_2}, \cdots, v_{j_\delta}$ be the nonzero components of $\mathbf{v}$ (i.e., $v_{j_1} = v_{j_2} = \cdots = v_{j_\delta} = 1$). Using (6.6) and (6.7), we have

$$\begin{aligned} \mathbb{0} &= \mathbf{v} \cdot \mathbb{H}^{\mathrm{T}} \\ &= (v_{j_1}, v_{j_2}, \cdots, v_{j_\delta}) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^2)^{j_1} & \cdots & (\alpha^{2t})^{j_1} \\ \alpha^{j_2} & (\alpha^2)^{j_2} & \cdots & (\alpha^{2t})^{j_2} \\ \alpha^{j_3} & (\alpha^2)^{j_3} & \cdots & (\alpha^{2t})^{j_3} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^2)^{j_\delta} & \cdots & (\alpha^{2t})^{j_\delta} \end{bmatrix} \end{aligned}$$

$$= (1, 1, \cdots, 1) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^{2t} \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^{2t} \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \cdots & (\alpha^{j_3})^{2t} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \cdots & (\alpha^{j_\delta})^{2t} \end{bmatrix}.$$

The preceding equality implies the following equality:

$$(1, 1, \cdots, 1) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^{\delta} \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^{\delta} \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \cdots & (\alpha^{j_3})^{\delta} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \cdots & (\alpha^{j_\delta})^{\delta} \end{bmatrix} = \mathbb{0}, \tag{6.9}$$

where the second matrix on the left is a $\delta \times \delta$ square matrix. To satisfy the equality of (6.9), the *determinant* of the $\delta \times \delta$ matrix must be zero; that is,

$$\begin{vmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^{\delta} \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^{\delta} \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \cdots & (\alpha^{j_3})^{\delta} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \cdots & (\alpha^{j_\delta})^{\delta} \end{vmatrix} = 0.$$

Taking out the common factor from each row of the foregoing determinant, we obtain

$$\alpha^{(j_1+j_2+\cdots+j_\delta)} \cdot \begin{vmatrix} 1 & \alpha^{j_1} & \cdots & \alpha^{(\delta-1)j_1} \\ 1 & \alpha^{j_2} & \cdots & \alpha^{(\delta-1)j_2} \\ 1 & \alpha^{j_3} & \cdots & \alpha^{(\delta-1)j_3} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 1 & \alpha^{j_\delta} & \cdots & \alpha^{(\delta-1)j_\delta} \end{vmatrix} = 0. \tag{6.10}$$

The determinant in the preceding equality is a *Vandermonde determinant* that is *nonzero*. Therefore, the product on the left-hand side of (6.10) cannot be zero. This is a contradiction, and hence our assumption that there exists a nonzero codeword $\mathbf{v}$ of weight $\delta \leq 2t$ is *invalid*. This implies that the minimum weight of the t-error-correcting BCH code defined previously is at least $2t + 1$. Consequently, the minimum distance of the code is at least $2t + 1$.

The parameter $2t + 1$ is usually called the *designed distance* of the t-error-correcting BCH code. The true minimum distance of a BCH code may or may not be equal to its designed distance. In many cases the true minimum distance of a

BCH code is equal to its designed distance; however, there are also cases in which the true minimum distance is greater than the designed distance.

Binary BCH codes with length $n \neq 2^m - 1$ can be constructed in the same manner as for the case $n = 2^m - 1$. Let β be an element of order n in the field $GF(2^m)$. We know that n is a factor of $2^m - 1$. Let $\mathbf{g}(X)$ be the binary polynomial of minimum degree that has

$$\beta, \beta^2, \cdots, \beta^{2t}$$

as roots. Let $\Psi_1(X), \Psi_2(X), \cdots, \Psi_{2t}(X)$ be the minimal polynomials of $\beta, \beta^2, \cdots, \beta^{2t}$, respectively. Then,

$$\mathbf{g}(X) = \text{LCM}\{\Psi_1(X), \Psi_2(X), \cdots, \Psi_{2t}(X)\}.$$

Because $\beta^n = 1, \beta, \beta^2, \cdots, \beta^{2t}$ are roots of $X^n + 1$. We see that $\mathbf{g}(X)$ is a factor of $X^n + 1$. The cyclic code generated by $\mathbf{g}(X)$ is a t-error-correcting BCH code of length n. In a manner similar to that used for the case $n = 2^m - 1$, we can prove that the number of parity-check digits of this code is at most mt, and the minimum distance of the code is at least $2t + 1$. If β is not a primitive element of $GF(2^m)$, $n \neq 2^m - 1$, and the code is called a *nonprimitive* BCH code.

EXAMPLE 6.3

Consider the Galois field $GF(2^6)$ given in Table 6.2. The element $\beta = \alpha^3$ has order $n = 21$. Let $t = 2$. Let $\mathbf{g}(X)$ be the binary polynomial of minimum degree that has

$$\beta, \beta^2, \beta^3, \beta^4$$

as roots. The elements β, β^2, and β^4 have the same minimal polynomial, which is

$$\Psi_1(X) = 1 + X + X^2 + X^4 + X^6.$$

The minimal polynomial of β^3 is

$$\Psi_3(X) = 1 + X^2 + X^3.$$

Therefore,

$$\begin{aligned}\mathbf{g}(X) &= \Psi_1(X)\Psi_3(X) \\ &= 1 + X + X^4 + X^5 + X^7 + X^8 + X^9.\end{aligned}$$

We can check easily that $\mathbf{g}(X)$ divides $X^{21} + 1$. The (21, 12) code generated by $\mathbf{g}(X)$ is a double-error-correcting nonprimitive BCH code.

Now, we give a general definition of binary BCH codes. Let β be an element of $GF(2^m)$. Let l_0 be any nonnegative integer. Then, a binary BCH code with designed distance d_0 is generated by the binary polynomial $\mathbf{g}(X)$ of minimum degree that has as roots the following consecutive powers of β:

$$\beta^{l_0}, \beta^{l_0+1}, \cdots, \beta^{l_0+d_0-2}$$

For $0 \leq i < d_0 - 1$, let $\Psi_i(X)$ and n_i be the minimum polynomial and order of β^{l_0+i}, respectively. Then,

$$\mathbb{g}(X) = \text{LCM}\{\Psi_0(X), \Psi_1(X), \cdots, \Psi_{d_0-2}(X)\}$$

and the length of the code is

$$n = \text{LCM}\{n_0, n_1, \cdots, n_{d_0-2}\}.$$

The BCH code just defined has a minimum distance of at least d_0 and has no more than $m(d_0 - 1)$ parity-check digits (the proof of these is left as an exercise). Of course, the code is capable of correcting $\lfloor (d_0 - 1)/2 \rfloor$ or fewer errors. If we let $l_0 = 1$, $d_0 = 2t + 1$, and β be a primitive element of $GF(2^m)$, the code becomes a t-error-correcting primitive BCH code of length $2^m - 1$. If $l_0 = 1$, $d_0 = 2t + 1$, and β is not a primitive element of $GF(2^m)$, the code is a nonprimitive t-error-correcting BCH code of length n that is the order of β. We note that in the definition of a BCH code with designed distance d_0, we require that the generator polynomial $\mathbb{g}(X)$ has $d_0 - 1$ consecutive powers of a field element β as roots. This requirement guarantees that the code has a minimum distance of at least d_0. This lower bound on the minimum distance is called the *BCH bound*. Any cyclic code whose generator polynomial has $d - 1$ consecutive powers of a field element as roots has a minimum distance of at least d.

In the rest of this chapter, we consider only the primitive BCH codes.

6.2 DECODING OF BCH CODES

Suppose that a codeword $\mathbb{v}(X) = v_0 + v_1X + v_2X^2 + \cdots + v_{n-1}X^{n-1}$ is transmitted, and the transmission errors result in the following received vector:

$$\mathbb{r}(X) = r_0 + r_1X + r_2X^2 + \cdots + r_{n-1}X^{n-1}.$$

Let $\mathbb{e}(X)$ be the error pattern. Then,

$$\mathbb{r}(X) = \mathbb{v}(X) + \mathbb{e}(X). \tag{6.11}$$

As usual, the first step of decoding a code is to compute the syndrome from the received vector $\mathbb{r}(X)$. For decoding a t-error-correcting primitive BCH code, the syndrome is a $2t$-tuple:

$$\mathbb{S} = (S_1, S_2, \cdots S_{2t}) = \mathbb{r} \cdot \mathbb{H}^T, \tag{6.12}$$

where $\mathbb{H}$ is given by (6.6). From (6.6) and (6.12) we find that the ith component of the syndrome is

$$\begin{aligned} S_i &= \mathbb{r}(\alpha^i) \\ &= r_0 + r_1\alpha^i + r_2\alpha^{2i} + \cdots + r_{n-1}\alpha^{(n-1)i} \end{aligned} \tag{6.13}$$

for $1 \leq i \leq 2t$. Note that the syndrome components are elements in the field $GF(2^m)$. We can compute from $\mathbb{r}(X)$ as follows. Dividing $\mathbb{r}(X)$ by the minimal polynomial $\phi_i(X)$ of α^i, we obtain

$$\mathbb{r}(X) = \mathbb{a}_i(X)\phi_i(X) + \mathbb{b}_i(X),$$

where $\mathbf{b}_i(X)$ is the remainder with degree less than that of $\phi_i(X)$. Because $\phi_i(\alpha^i) = 0$, we have

$$S_i = \mathbf{r}(\alpha^i) = \mathbf{b}_i(\alpha^i). \tag{6.14}$$

Thus, the syndrome component S_i can be obtained by evaluating $\mathbf{b}_i(X)$ with $X = \alpha^i$.

EXAMPLE 6.4

Consider the double-error-correcting (15, 7) BCH code given in Example 6.1. Suppose that the vector

$$\mathbf{r} = (1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0)$$

is received. The corresponding polynomial is

$$\mathbf{r}(X) = 1 + X^8.$$

The syndrome consists of four components:

$$\mathbf{S} = (S_1, S_2, S_3, S_4).$$

The minimal polynomials for α, α^2, and α^4 are identical, and

$$\phi_1(X) = \phi_2(X) = \phi_4(X) = 1 + X + X^4.$$

The minimal polynomial of α^3 is

$$\phi_3(X) = 1 + X + X^2 + X^3 + X^4.$$

Dividing $\mathbf{r}(X) = 1 + X^8$ by $\phi_1(X) = 1 + X + X^4$, we obtain the remainder

$$\mathbf{b}_1(X) = X^2.$$

Dividing $\mathbf{r}(X) = 1 + X^8$ by $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$, we have the remainder

$$\mathbf{b}_3(X) = 1 + X^3.$$

Using $GF(2^4)$ given by Table 2.8 and substituting α, α^2, and α^4 into $\mathbf{b}_1(X)$, we obtain

$$S_1 = \alpha^2, \quad S_2 = \alpha^4, \quad S_4 = \alpha^8.$$

Substituting α^3 into $\mathbf{b}_3(X)$, we obtain

$$S_3 = 1 + \alpha^9 = 1 + \alpha + \alpha^3 = \alpha^7.$$

Thus,

$$S = (\alpha^2, \alpha^4, \alpha^7, \alpha^8).$$

Because $\alpha^1, \alpha^2, \cdots, \alpha^{2t}$ are roots of each code polynomial, $v(\alpha^i) = 0$ for $1 \leq i \leq 2t$. The following relationship between the syndrome and the error pattern follows from (6.11) and (6.13):

$$S_i = e(\alpha^i) \tag{6.15}$$

for $1 \leq i \leq 2t$. From (6.15) we see that the syndrome $\mathbb{S}$ depends on the error pattern $\mathbf{e}$ only. Suppose that the error pattern $e(X)$ has ν errors at locations $X^{j_1}, X^{j_2}, \cdots X^{j_\nu}$; that is,

$$e(X) = X^{j_1} + X^{j_2} + \cdots + X^{j_\nu}, \tag{6.16}$$

where $0 \leq j_1 < j_2 < \cdots < j_\nu < n$. From (6.15) and (6.16) we obtain the following set of equations:

$$\begin{aligned}
S_1 &= \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_\nu} \\
S_2 &= (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \cdots + (\alpha^{j_\nu})^2 \\
S_3 &= (\alpha^{j_1})^3 + (\alpha^{j_2})^3 + \cdots + (\alpha^{j_\nu})^3 \\
&\vdots \\
S_{2t} &= (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \cdots + (\alpha^{j_\nu})^{2t},
\end{aligned} \tag{6.17}$$

where $\alpha^{j_1}, \alpha^{j_2}, \cdots, \alpha^{j_\nu}$ are unknown. *Any method for solving these equations is a decoding algorithm for the BCH codes.* Once we have found $\alpha^{j_1}, \alpha^{j_2}, \cdots, \alpha^{j_\nu}$, the powers $j_1, j_2, \cdots, j_\nu$ tell us the error locations in $e(X)$, as in (6.16). In general, the equations of (6.17) have many possible solutions (2^k of them). Each solution yields a different error pattern. If the number of errors in the actual error pattern $e(X)$ is t or fewer (i.e., $\nu \leq t$), the solution that yields an error pattern with the *smallest number of errors* is the right solution; that is, the error pattern corresponding to this solution is the most probable error pattern $e(X)$ caused by the channel noise. For large t, solving the equations of (6.17) directly is difficult and ineffective. In the following, we describe an effective procedure for determining α^{j_l} for $l = 1, 2, \cdots, \nu$ from the syndrome components S_i's.

For convenience, let

$$\beta_l = \alpha^{j_l} \tag{6.18}$$

for $1 \leq l \leq \nu$. We call these elements the *error location numbers*, since they tell us the locations of the errors. Now, we can express the equations of (6.17) in the following form:

$$\begin{aligned}
S_1 &= \beta_1 + \beta_2 + \cdots + \beta_\nu \\
S_2 &= \beta_1^2 + \beta_2^2 + \cdots + \beta_\nu^2 \\
&\vdots \\
S_{2t} &= \beta_1^{2t} + \beta_2^{2t} + \cdots + \beta_\nu^{2t}.
\end{aligned} \tag{6.19}$$

These $2t$ equations are symmetric functions in $\beta_1, \beta_2, \cdots, \beta_\nu$, which are known as *power-sum symmetric functions*. Now, we define the following polynomial:

$$\begin{aligned}\sigma(X) &\triangleq (1+\beta_1 X)(1+\beta_2 X)\cdots(1+\beta_\nu X)\\ &= \sigma_0 + \sigma_1 X + \sigma_2 X^2 + \cdots + \sigma_\nu X^\nu.\end{aligned} \tag{6.20}$$

The roots of $\sigma(X)$ are $\beta_1^{-1}, \beta_2^{-1}, \cdots, \beta_\nu^{-1}$, which are the inverses of the error-location numbers. For this reason, $\sigma(X)$ is called the *error-location polynomial*. Note that $\sigma(X)$ is an unknown polynomial whose coefficients must be determined. The coefficients of $\sigma(X)$ and the error-location numbers are related by the following equations:

$$\begin{aligned}\sigma_0 &= 1\\ \sigma_1 &= \beta_1 + \beta_2 + \cdots + \beta_\nu\\ \sigma_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{\nu-1}\beta_\nu\\ &\vdots\\ \sigma_\nu &= \beta_1\beta_2\cdots\beta_\nu.\end{aligned} \tag{6.21}$$

The σ_i's are known as *elementary symmetric functions* of β_l's. From (6.19) and (6.21), we see that the σ_i's are related to the syndrome components S_j's. In fact, they are related to the syndrome components by the following *Newton's identities*:

$$\begin{aligned}S_1 + \sigma_1 &= 0\\ S_2 + \sigma_1 S_1 + 2\sigma_2 &= 0\\ S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 &= 0\\ &\vdots\\ S_\nu + \sigma_1 S_{\nu-1} + \cdots + \sigma_{\nu-1} S_1 + \nu\sigma_\nu &= 0\\ S_{\nu+1} + \sigma_1 S_\nu + \cdots + \sigma_{\nu-1} S_2 + \sigma_\nu S_1 &= 0\\ &\vdots\end{aligned} \tag{6.22}$$

For the binary case, since $1 + 1 = 2 = 0$, we have

$$i\sigma_i = \begin{cases}\sigma_i & \text{for odd } i,\\ 0 & \text{for even } i.\end{cases}$$

If it is possible to determine the elementary symmetric functions $\sigma_1, \sigma_2, \cdots, \sigma_\nu$ from the equations of (6.22), the error-location numbers $\beta_1, \beta_2, \cdots, \beta_\nu$ can be found by determining the roots of the error-location polynomial $\sigma(X)$. Again, the equations of (6.22) may have many solutions; however, we want to find the solution that yields a $\sigma(X)$ of minimal degree. This $\sigma(X)$ will produce an error pattern with a minimum number of errors. If $\nu \le t$, this $\sigma(X)$ will give the actual error pattern $\mathbf{e}(X)$. Next, we describe a procedure to determine the polynomial $\sigma(X)$ of minimum degree that satisfies the first $2t$ equations of (6.22) (since we know only S_1 through S_{2t}).

At this point, we outline the error-correcting procedure for BCH codes. The procedure consists of three major steps:

1. Compute the syndrome $\mathbb{S} = (S_1, S_2, \cdots, S_{2t})$ from the received polynomial $\mathbb{r}(X)$.
2. Determine the error-location polynomial $\sigma(X)$ from the syndrome components $S_1, S_2, \cdots, S_{2t}$.
3. Determine the error-location numbers $\beta_1, \beta_2, \cdots, \beta_v$ by finding the roots of $\sigma(X)$, and correct the errors in $\mathbb{r}(X)$.

The first decoding algorithm that carries out these three steps was devised by Peterson [3]. Steps 1 and 3 are quite simple; step 2 is the most complicated part of decoding a BCH code.

6.3 ITERATIVE ALGORITHM FOR FINDING THE ERROR-LOCATION POLYNOMIAL $\sigma(X)$

Here we present Berlekamp's iterative algorithm for finding the error-location polynomial. We describe only the algorithm, without giving any proof. The reader who is interested in details of this algorithm is referred to Berlekamp [9], Peterson and Weldon [13], and MacWilliams and Sloane [14].

The first step of iteration is to find a minimum-degree polynomial $\sigma^{(1)}(X)$ whose coefficients satisfy the first Newton's identity of (6.22). The next step is to test whether the coefficients of $\sigma^{(1)}(X)$ also satisfy the second Newton's identity of (6.22). If the coefficients of $\sigma^{(1)}(X)$ do satisfy the second Newton's identity of (6.22), we set

$$\sigma^{(2)}(X) = \sigma^{(1)}(X).$$

If the coefficients of $\sigma^{(1)}(X)$ do not satisfy the second Newton's identity of (6.22), we add a correction term to $\sigma^{(1)}(X)$ to form $\sigma^{(2)}(X)$ such that $\sigma^{(2)}(X)$ has minimum degree and its coefficients satisfy the first two Newton's identities of (6.22). Therefore, at the end of the second step of iteration, we obtain a minimum-degree polynomial $\sigma^{(2)}(X)$ whose coefficients satisfy the first two Newton's identities of (6.22). The third step of iteration is to find a minimum-degree polynomial $\sigma^{(3)}(X)$ from $\sigma^{(2)}(X)$ such that the coefficients of $\sigma^{(3)}(X)$ satisfy the first three Newton's identities of (6.22). Again, we test whether the coefficients of $\sigma^{(2)}(X)$ satisfy the third Newton's identity of (6.22). If they do, we set $\sigma^{(3)}(X) = \sigma^{(2)}(X)$. If they do not, we add a correction term to $\sigma^{(2)}(X)$ to form $\sigma^{(3)}(X)$. Iteration continues until we obtain $\sigma^{(2t)}(X)$. Then, $\sigma^{(2t)}(X)$ is taken to be the error-location polynomial $\sigma(X)$, that is,

$$\sigma(X) = \sigma^{(2t)}(X).$$

This $\sigma(X)$ will yield an error pattern $\mathbb{e}(X)$ of minimum weight that satisfies the equations of (6.17). If the number of errors in the received polynomial $\mathbb{r}(X)$ is t or less, then $\sigma(X)$ produces the true error pattern.

Let

$$\sigma^{(\mu)}(X) = 1 + \sigma_1^{(\mu)} X + \sigma_2^{(\mu)} X^2 + \cdots + \sigma_{l_\mu}^{(\mu)} X^{l_\mu} \tag{6.23}$$

be the minimum-degree polynomial determined at the μth step of iteration whose coefficients satisfy the first μ Newton's identities of (6.22). To determine $\sigma^{(\mu+1)}(X)$,

we compute the following quantity:

$$d_\mu = S_{\mu+1} + \sigma_1^{(\mu)} S_\mu + \sigma_2^{(\mu)} S_{\mu-1} + \cdots + \sigma_{l_\mu}^{(\mu)} S_{\mu+1-l_\mu}. \tag{6.24}$$

This quantity d_μ is called the μth *discrepancy*. If $d_\mu = 0$, the coefficients of $\sigma^{(\mu)}(X)$ satisfy the $(\mu+1)$th Newton's identity. In this event, we set

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X).$$

If $d_\mu \neq 0$, the coefficients of $\sigma^{(\mu)}(X)$ do not satisfy the $(\mu+1)$th Newton's identity, and we must add a correction term to $\sigma^{(\mu)}(X)$ to obtain $\sigma^{(\mu+1)}(X)$. To make this correction, we go back to the steps *prior to* the μth step and determine a polynomial $\sigma^{(\rho)}(X)$ such that the ρth discrepancy $d_\rho \neq 0$, and $\rho - l_\rho$ [l_ρ is the degree of $\sigma^{(\rho)}(X)$] has the largest value. Then,

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X), \tag{6.25}$$

which is the minimum-degree polynomial whose coefficients satisfy the first $\mu+1$ Newton's identities. The proof of this is quite complicated and is omitted from this introductory book.

To carry out the iteration of finding $\sigma(X)$, we begin with Table 6.5 and proceed to fill out the table, where l_μ is the degree of $\sigma^{(\mu)}(X)$. Assuming that we have filled out all rows up to and including the μth row, we fill out the $(\mu+1)$th row as follows:

1. If $d_\mu = 0$, then $\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X)$, and $l_{\mu+1} = l_\mu$.
2. If $d_\mu \neq 0$, we find another row ρ prior to the μth row such that $d_\rho \neq 0$ and the number $\rho - l_\rho$ in the last column of the Table has the largest value. Then, $\sigma^{(\mu+1)}(X)$ is given by (6.25), and

$$l_{\mu+1} = \max(l_\mu, l_\rho + \mu - \rho). \tag{6.26}$$

In either case,

$$d_{\mu+1} = S_{\mu+2} + \sigma_1^{(\mu+1)} S_{\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+2-l_{\mu+1}}, \tag{6.27}$$

where the $\sigma_i^{(\mu+1)}$'s are the coefficients of $\sigma^{(\mu+1)}(X)$. The polynomial $\sigma^{(2t)}(X)$ in the last row should be the required $\sigma(X)$. If its degree is greater than t, there are more

TABLE 6.5: Berlekamp's iterative procedure for finding the error-location polynomial of a BCH code.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1				
2				
$\vdots$				
$2t$				

than t errors in the received polynomial $\mathbb{r}(X)$, and generally it is not possible to locate them.

EXAMPLE 6.5

Consider the (15, 5) triple-error-correcting BCH code given in Example 6.1. Assume that the codeword of all zeros,

$$\mathbb{v} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$

is transmitted, and the vector

$$\mathbb{r} = (0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,1\,0\,0)$$

is received. Then, $\mathbb{r}(X) = X^3 + X^5 + X^{12}$. The minimal polynomials for α, α^2, and α^4 are identical, and

$$\phi_1(X) = \phi_2(X) = \phi_4(X) = 1 + X + X^4.$$

The elements α^3 and α^6 have the same minimal polynomial,

$$\phi_3(X) = \phi_6(X) = 1 + X + X^2 + X^3 + X^4.$$

The minimal polynomial for α^5 is

$$\phi_5(X) = 1 + X + X^2.$$

Dividing $\mathbb{r}(X)$ by $\phi_1(X)$, $\phi_3(X)$, and $\phi_5(X)$, respectively, we obtain the following remainders:

$$\begin{aligned} \mathbb{b}_1(X) &= 1, \\ \mathbb{b}_3(X) &= 1 + X^2 + X^3, \\ \mathbb{b}_5(X) &= X^2. \end{aligned} \tag{6.28}$$

Using Table 2.8 and substituting α, α^2, and α^4 into $\mathbb{b}_1(X)$, we obtain the following syndrome components:

$$S_1 = S_2 = S_4 = 1.$$

Substituting α^3 and α^6 into $\mathbb{b}_3(X)$, we obtain

$$\begin{aligned} S_3 &= 1 + \alpha^6 + \alpha^9 = \alpha^{10}, \\ S_6 &= 1 + \alpha^{12} + \alpha^{18} = \alpha^5. \end{aligned} \tag{6.29}$$

Substituting α^5 into $\mathbb{b}_5(X)$, we have

$$S_5 = \alpha^{10}.$$

Using the iterative procedure described previously, we obtain Table 6.6. Thus, the error-location polynomial is

$$\sigma(X) = \sigma^{(6)}(X) = 1 + X + \alpha^5 X^3.$$

TABLE 6.6: Steps for finding the error-location polynomial of the (15,5) BCH code given in Example 6.5.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	1	0	0
1	$1+X$	0	1	0 (take $\rho=-1$)
2	$1+X$	α^5	1	1
3	$1+X+\alpha^5X^2$	0	2	1 (take $\rho=0$)
4	$1+X+\alpha^5X^2$	α^{10}	2	2
5	$1+X+\alpha^5X^3$	0	3	2 (take $\rho=2$)
6	$1+X+\alpha^5X^3$	—	—	—

We can easily check that α^3, α^{10}, and α^{12} are the roots of $\sigma(X)$. Their inverses are α^{12}, α^5, and α^3, which are the error-location numbers. Therefore, the error pattern is

$$\mathbf{e}(X) = X^3 + X^5 + X^{12}.$$

Adding $\mathbf{e}(X)$ to the received polynomial $\mathbf{r}(X)$, we obtain the all-zero vector.

If the number of errors in the received polynomial $\mathbf{r}(X)$ is less than the designed error-correcting capability t of the code, it is not necessary to carry out the $2t$ steps of iteration to find the error-location polynomial $\sigma(X)$. Let $\sigma^{(\mu)}(X)$ and d_μ be the solution and discrepancy obtained at the μth step of iteration. Let l_μ be the degree of $\sigma^{(\mu)}(X)$. Chen [19] has shown that if d_μ and the discrepancies at the next $t - l_\mu - 1$ steps are all zero, $\sigma^{(\mu)}(X)$ is the error-location polynomial. Therefore, if the number of errors in the received polynomial $\mathbf{r}(X)$ is $\nu(\nu \leq t)$, only $t + \nu$ steps of iteration are needed to determine the error-location polynomial $\sigma(X)$. If ν is small (this is often the case), the reduction in the number of iteration steps results in an increase in decoding speed.

The described iterative algorithm for finding $\sigma(X)$ applies not only to binary BCH codes but also to nonbinary BCH codes.

6.4 SIMPLIFIED ITERATIVE ALGORITHM FOR FINDING THE ERROR-LOCATION POLYNOMIAL $\sigma(X)$

The described iterative algorithm for finding $\sigma(X)$ applies to both binary and nonbinary BCH codes, including Reed–Solomon codes; however, for binary BCH codes, this algorithm can be simplified to t-steps for computing $\sigma(X)$.

Recall that for a polynomial $f(X)$ over $GF(2)$,

$$f^2(X) = f(X^2),$$

[see (2.10)]. Because the received polynomial $\mathbf{r}(X)$ is a polynomial over $GF(2)$, we have

$$\mathbf{r}^2(X) = \mathbf{r}(X^2).$$

Substituting α^i for X in the preceding equality, we obtain

$$\mathbb{r}^2(\alpha^i) = \mathbb{r}(\alpha^{2i}). \tag{6.30}$$

Because $S_i = \mathbb{r}(\alpha^i)$, and $S_{2i} = \mathbb{r}(\alpha^{2i})$ [see (6.13)], we obtain the following relationship between S_{2i} and S_i:

$$S_{2i} = S_i^2. \tag{6.31}$$

Suppose the first Newton's identity of (6.22) holds. Then,

$$S_1 + \sigma_1 = 0.$$

This result says that

$$\sigma_1 = S_1. \tag{6.32}$$

It follows from (6.31) and (6.32) that

$$\begin{aligned} S_2 + \sigma_1 S_1 + 2\sigma_2 &= S_1^2 + S_1 \cdot S_1 + 0 \\ &= 0. \end{aligned} \tag{6.33}$$

The foregoing equality is simply the Newton's second equality. This result says that if the first Newton's identity holds, then the second Newton's identity also holds. Now, suppose the first and the third Newton's identities of (6.22) hold; that is,

$$S_1 + \sigma_1 = 0, \tag{6.34}$$

$$S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0. \tag{6.35}$$

The equality of (6.34) implies that the second Newton's identity holds:

$$S_2 + \sigma_1 S_1 + 2\sigma_2 = 0. \tag{6.36}$$

Then,

$$(S_2 + \sigma_1 S_1 + 2\sigma_2)^2 = 0,$$

$$S_2^2 + \sigma_1^2 S_1^2 = 0. \tag{6.37}$$

It follows from (6.31) that (6.37) becomes

$$S_4 + \sigma_1^2 S_2 = 0. \tag{6.38}$$

Multiplying both sides of (6.35) by σ_1, we obtain

$$\sigma_1 S_3 + \sigma_1^2 S_2 + \sigma_1 \sigma_2 S_1 + 3\sigma_1 \sigma_3 = 0. \tag{6.39}$$

Adding (6.38) and (6.39) and using the equalities of (6.31) and (6.32), we find that the fourth Newton's identity holds:

$$S_4 + \sigma_1 S_3 + \sigma_2 S_2 + \sigma_3 S_1 + 4\sigma_4 = 0.$$

This result says that if the first and the third Newton's identities hold, the second and the fourth Newton's identities also hold.

With some effort, it is possible to prove that if the first, third, $\cdots$, $(2t-1)$th Newton's identities hold, then the second, fourth, $\cdots$, $2t$th Newton's identities also hold. This implies that with the iterative algorithm for finding the error-location polynomial $\sigma(X)$, the solution $\sigma^{(2\mu-1)}(X)$ at the $(2\mu-1)$th step of iteration is also the solution $\sigma^{(2\mu)}(X)$ at the 2μth step of iteration; that is,

$$\sigma^{(2\mu)}(X) = \sigma^{(2\mu-1)}(X). \tag{6.40}$$

This suggests that the $(2\mu-1)$th and the 2μth steps of iteration can be combined. As a result, the foregoing iterative algorithm for finding $\sigma(X)$ can be reduced to t steps. Only the even steps are needed.

The simplified algorithm can be carried out by filling out a table with only t rows, as illustrated in Table 6.7.

Assuming that we have filled out all rows up to and including the μth row, we fill out the $(\mu+1)$th row as follows:

1. If $d_\mu = 0$, then $\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X)$.
2. If $d_\mu \neq 0$, we find another row preceding the μth row, say the ρth, such that the number $2\rho - l_\rho$ in the last column is as large as possible and $d_\rho \neq 0$. Then,

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{2(\mu-\rho)} \sigma^{(\rho)}(X). \tag{6.41}$$

In either case, $l_{\mu+1}$ is exactly the degree of $\sigma^{(\mu+1)}(X)$, and the discrepancy at the $(\mu+1)$th step is

$$d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{2\mu+3-l_{\mu+1}}. \tag{6.42}$$

The polynomial $\sigma^{(t)}(X)$ in the last row should be the required $\sigma(X)$. If its degree is greater than t, there were more than t errors, and generally it is not possible to locate them.

The computation required in this simplified algorithm is half of the computation required in the general algorithm; however, we must remember that the simplified algorithm applies only to binary BCH codes. Again, if the number of errors in the

TABLE 6.7: A simplified Berlekamp iterative procedure for finding the error-location polynomial of a binary BCH code.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$-\frac{1}{2}$	1	1	0	-1
0	1	S_1	0	0
1				
2				
$\vdots$				
t				

TABLE 6.8: Steps for finding the error-location polynomial of the (15,5) binary BCH code given in Example 6.6.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$-\frac{1}{2}$	1	1	0	-1
0	1	$S_1 = 1$	0	0
1	$1 + S_1X = 1 + X$	$S_3 + S_2S_1 = \alpha^5$	1	1 (take $\rho = -\frac{1}{2}$)
2	$1 + X + \alpha^5X^2$	α^{10}	2	2 (take $\rho = 0$)
3	$1 + X + \alpha^5X^3$	—	3	3 (take $\rho = 2$)

received polynomial $\mathbb{r}(X)$ is less than t, it is not necessary to carry out the t steps of iteration to determine $\sigma(X)$ for a t-error-correcting binary BCH code. Based on Chen's result [19], if for some μ, d_μ and the discrepancies at the next $\lceil (t - l_\mu - 1)/2 \rceil$ steps of iteration are zero, $\sigma^{(\mu)}(X)$ is the error-location polynomial. If the number of errors in the received polynomial is $\nu (\nu \leq t)$, only $\lceil (t + \nu)/2 \rceil$ steps of iteration are needed to determine the error-location polynomial $\sigma(X)$.

EXAMPLE 6.6

The simplified table for finding $\sigma(X)$ for the code considered in Example 6.5 is given in Table 6.8. Thus, $\sigma(X) = \sigma^{(3)}(X) = 1 + X + \alpha^5X^3$, which is identical to the solution found in Example 6.5.

6.5 FINDING THE ERROR-LOCATION NUMBERS AND ERROR CORRECTION

The last step in decoding a BCH code is to find the error-location numbers that are the reciprocals of the roots of $\sigma(X)$. The roots of $\sigma(X)$ can be found simply by substituting $1, \alpha, \alpha^2, \cdots, \alpha^{n-1} (n = 2^m - 1)$ into $\sigma(X)$. Since $\alpha^n = 1, \alpha^{-l} = \alpha^{n-l}$. Therefore, if α^l is a root of $\sigma(X)$, α^{n-l} is an error-location number, and the received digit r_{n-l} is an erroneous digit. Consider Example 6.6, where the error-location polynomial was found to be

$$\sigma(X) = 1 + X + \alpha^5X^3.$$

By substituting $1, \alpha, \alpha^2, \cdots, \alpha^{14}$ into $\sigma(X)$, we find that α^3, α^{10}, and α^{12} are roots of $\sigma(X)$. Therefore, the error-location numbers are α^{12}, α^5, and α^3. The error pattern is

$$\mathbb{e}(X) = X^3 + X^5 + X^{12},$$

which is exactly the assumed error pattern. The decoding of the code is completed by adding (modulo-2) $\mathbb{e}(X)$ to the received vector $\mathbb{r}(X)$.

The described substitution method for finding the roots of the error-location polynomial was first used by Peterson in his algorithm for decoding BCH codes [3]. Later, Chien [6] formulated a procedure for carrying out the substitution and error correction. Chien's procedure for searching for error-location numbers is described next. The received vector

$$\mathbb{r}(X) = r_0 + r_1X + r_2X^2 + \cdots + r_{n-1}X^{n-1}$$

is decoded bit by bit. The high-order bits are decoded first. To decode r_{n-1}, the decoder tests whether α^{n-1} is an error-location number; this is equivalent to testing whether its inverse, α, is a root of $\sigma(X)$. If α is a root, then

$$1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_\nu\alpha^\nu = 0.$$

Therefore, to decode r_{n-1}, the decoder forms $\sigma_1\alpha, \sigma_2\alpha^2, \cdots, \sigma_\nu\alpha^\nu$. If the sum $1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots, \sigma_\nu\alpha^\nu = 0$, then α^{n-1} is an error-location number, and r_{n-1} is an erroneous digit; otherwise, r_{n-1} is a correct digit. To decode r_{n-l}, the decoder forms $\sigma_1\alpha^l, \sigma_2\alpha^{2l}, \cdots, \sigma_\nu\alpha^{\nu l}$ and tests the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \cdots + \sigma_\nu\alpha^{\nu l}.$$

If this sum is 0, then α^l is a root of $\sigma(X)$, and r_{n-l} is an erroneous digit; otherwise, r_{n-l} is a correct digit.

The described testing procedure for error locations can be implemented in a straightforward manner by a circuit such as that shown in Figure 6.1 [6]. The t σ-registers are initially stored with $\sigma_1, \sigma_2, \cdots, \sigma_t$ calculated in step 2 of the decoding ($\sigma_{\nu+1} = \sigma_{\nu+2} = \cdots = \sigma_t = 0$ for $\nu < t$). Immediately before r_{n-1} is read out of the buffer, the t multipliers $\otimes$ are pulsed once. The multiplications are performed, and $\sigma_1\alpha, \sigma_2\alpha^2, \cdots, \sigma_\nu\alpha^\nu$ are stored in the σ-registers. The output of the logic circuit A is 1 if and only if the sum $1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_\nu\alpha^\nu = 0$; otherwise, the output of A is 0. The digit r_{n-1} is read out of the buffer and corrected by the output of A. Once r_{n-1} is decoded, the t multipliers are pulsed again. Now, $\sigma_1\alpha^2, \sigma_2\alpha^4, \cdots, \sigma_\nu\alpha^{2\nu}$ are stored in the σ-registers. The sum

$$1 + \sigma_1\alpha^2 + \sigma_2\alpha^4 + \cdots + \sigma_\nu\alpha^{2\nu}$$

is tested for 0. The digit r_{n-2} is read out of the buffer and corrected in the same manner as r_{n-1} was corrected. This process continues until the whole received vector is read out of the buffer.

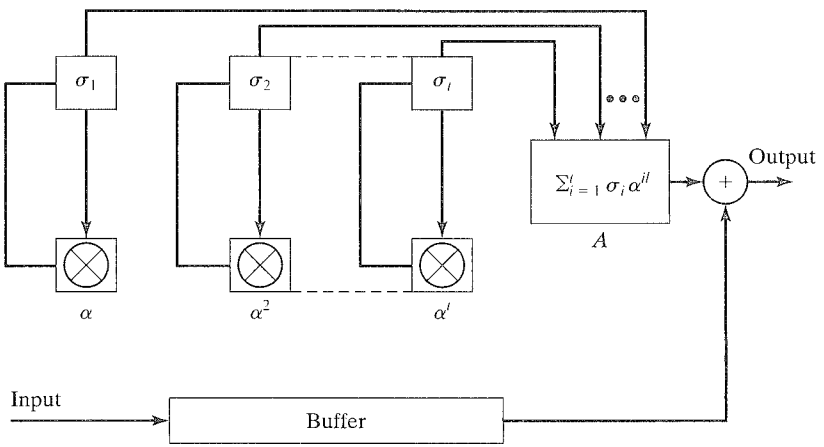

FIGURE 6.1: Cyclic error location search unit.

The described decoding algorithm also applies to nonprimitive BCH codes. The $2t$ syndrome components are given by

$$S_i = \mathbb{r}(\beta^i)$$

for $1 \leq i \leq 2t$.

6.6 CORRECTION OF ERRORS AND ERASURES

If the channel is the binary symmetric erasure channel as shown in Figure 1.6(b), the received vector may contain both errors and erasures. It was shown in Section 3.4, that a code with minimum distance $d_{\min}$ is capable of correcting all combinations of ν errors and e erasures provided that

$$2\nu + e + 1 \leq d_{\min}. \tag{6.43}$$

Erasure and error correction with binary BCH codes are quite simple. Suppose a BCH code is designed to correct t errors, and the received polynomial $\mathbb{r}(X)$ contains ν (unknown) random errors and e (known) erasures. The decoding can be accomplished in two steps. First, the erased positions are replaced with 0's and the resulting vector is decoded using the standard BCH decoding algorithm. Next, the e erased positions are replaced with 1's, and the resulting vector is decoded in the same manner. The decodings result in two codewords. The codeword with the smallest number of errors corrected outside the e erased positions is chosen as the decoded codeword. If the inequality of (6.43) holds, this decoding algorithm always results in correct decoding. To see this, we write (6.43) in terms of the error-correcting capability t of the code as

$$\nu + e/2 \leq t. \tag{6.44}$$

Assume that when e erasures are replaced with 0's, $e^* \leq e/2$ errors are introduced in those e erased positions. As a result, the resulting vector contains a total of $\nu + e^* \leq t$ errors that are guaranteed to be correctable if the inequality of (6.44) (or (6.43)) holds; however, if $e^* > e/2$ then only $e - e^* < e/2$ errors are introduced when the e erasures are replaced with 1's. In this case the resultant vector contains $\nu + (e - e^*) < t$ errors. Such errors are also correctable. Therefore, with the described decoding algorithm, at least one of the two decoded codewords is correct.

6.7 IMPLEMENTATION OF GALOIS FIELD ARITHMETIC

Decoding of BCH codes requires computations using Galois field arithmetic. Galois field arithmetic can be implemented more easily than ordinary arithmetic because there are no carries. In this Section we discuss circuits that perform addition and multiplication over a Galois field. For simplicity, we consider the arithmetic over the Galois field $GF(2^4)$ given by Table 2.8.

To add two field elements, we simply add their vector representations. The resultant vector is then the vector representation of the sum of the two field elements. For example, we want to add α^7 and α^{13} of $GF(2^4)$. From Table 2.8 we find that their vector representations are (1 1 0 1) and (1 0 1 1), respectively. Their vector sum is (1 1 0 1) + (1 0 1 1) = (0 1 1 0), which is the vector representation of α^5.

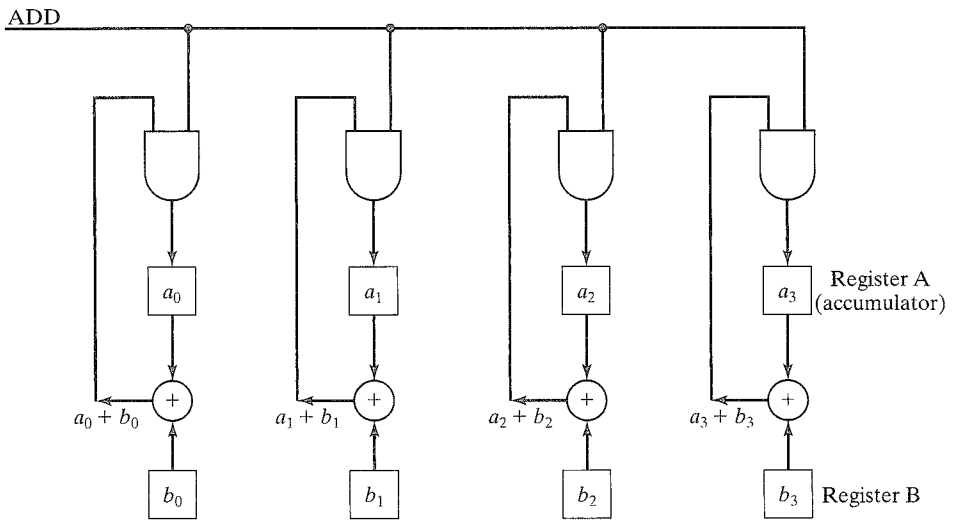

FIGURE 6.2: Galois field adder.

Two field elements can be added with the circuit shown in Figure 6.2. First, the vector representations of the two elements to be added are loaded into registers A and B. Their vector sum then appears at the inputs of register A. When register A is pulsed (or clocked), the sum is loaded into register A (register A serves as an accumulator).

For multiplication, we first consider multiplying a field element by a fixed element from the same field. Suppose that we want to multiply a field element β in $GF(2^4)$ by the primitive element α whose minimal polynomial is $\phi(X) = 1 + X + X^4$. We can express element β as a polynomial in α as follows:

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3.$$

Multiplying both sides of this equality by α and using the fact that $\alpha^4 = 1 + \alpha$, we obtain the following equality:

$$\alpha\beta = b_3 + (b_0 + b_3)\alpha + b_1\alpha^2 + b_2\alpha^3.$$

This multiplication can be carried out by the feedback shift register shown in Figure 6.3. First, the vector representation (b_0, b_1, b_2, b_3) of β is loaded into the register, then the register is pulsed. The new contents in the register form the

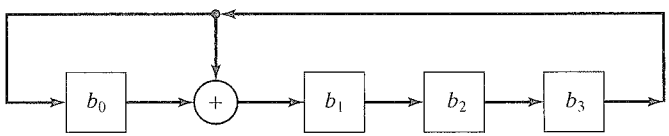

FIGURE 6.3: Circuit for multiplying an arbitrary element in $GF(2^4)$ by α.

vector representation of $\alpha\beta$. For example, let $\beta = \alpha^7 = 1 + \alpha + \alpha^3$. The vector representation of β is (1 1 0 1). We load this vector into the register of the circuit shown in Figure 6.3. After the register is pulsed, the new contents in the register will be (1 0 1 0), which represents α^8, the product of α^7 and α. The circuit shown in Figure 6.3 can be used to generate (or count) all the nonzero elements of $GF(2^4)$. First, we load (1 0 0 0) (vector representation of $\alpha^0 = 1$) into the register. Successive shifts of the register will generate vector representations of successive powers of α, in exactly the same order as they appear in Table 2.8. At the end of the fifteenth shift, the register will contain (1 0 0 0) again.

As another example, suppose that we want to devise a circuit to multiply an arbitrary element β of $GF(2^4)$ by the element α^3. Again, we express β in polynomial form:

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3.$$

Multiplying both sides of the preceding equation by α^3, we have

$$\begin{aligned}\alpha^3\beta &= b_0\alpha^3 + b_1\alpha^4 + b_2\alpha^5 + b_3\alpha^6 \\ &= b_0\alpha^3 + b_1(1+\alpha) + b_2(\alpha+\alpha^2) + b_3(\alpha^2+\alpha^3) \\ &= b_1 + (b_1+b_2)\alpha + (b_2+b_3)\alpha^2 + (b_0+b_3)\alpha^3.\end{aligned}$$

Based on the preceding expression, we obtain the circuit shown in Figure 6.4, which is capable of multiplying any element β in $GF(2^4)$ by α^3. To multiply, we first load the vector representation (b_0, b_1, b_2, b_3) of β into the register, then we pulse the register. The new contents in the register will be the vector representation of $\alpha^3\beta$. Next, we consider multiplying two arbitrary field elements. Again, we use $GF(2^4)$ for illustration. Let β and γ be two elements in $GF(2^4)$. We express these two elements in polynomial form:

$$\begin{aligned}\beta &= b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3, \\ \gamma &= c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3.\end{aligned}$$

Then, we can express the product $\beta\gamma$ in the following form:

$$\beta\gamma = (((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta)\alpha + c_0\beta \tag{6.45}$$

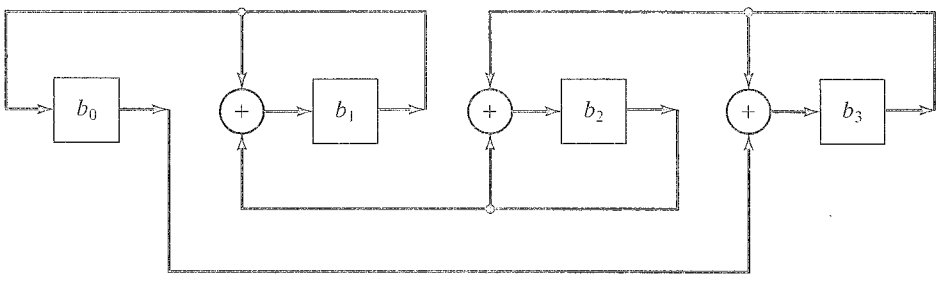

FIGURE 6.4: Circuit for multiplying an arbitrary element in $GF(2^4)$ by α^3.

This product can be evaluated with the following steps:

1. Multiply $c_3\beta$ by α and add the product to $c_2\beta$.
2. Multiply $(c_3\beta)\alpha + c_2\beta$ by α and add the product to $c_1\beta$.
3. Multiply $((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta$ by α and add the product to $c_0\beta$.

Multiplication by α can be carried out by the circuit shown in Figure 6.3. This circuit can be modified to carry out the computation given by (6.45). The resultant circuit is shown in Figure 6.5. In operation of this circuit, the feedback shift register A is initially empty, and (b_0, b_1, b_2, b_3) and (c_0, c_1, c_2, c_3), the vector representations of β and γ, are loaded into registers B and C, respectively. Then, registers A and C are shifted four times. At the end of the first shift, register A contains $(c_3b_0, c_3b_1, c_3b_2, c_3b_3)$, the vector representation of $c_3\beta$. At the end of the second shift, register A contains the vector representation of $(c_3\beta)\alpha + c_2\beta$. At the end of the third shift, register A contains the vector representation of $((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta$. At the end of the fourth shift, register A contains the product $\beta\gamma$ in vector form. If we express the product $\beta\gamma$ in the form

$$\beta\gamma = (((c_0\beta) + c_1\beta\alpha) + c_2\beta\alpha^2) + c_3\beta\alpha^3,$$

we obtain a different multiplication circuit, as shown in Figure 6.6. To perform the multiplication, β and γ are loaded into registers B and C, respectively, and register A is initially empty. Then, registers A, B, and C are shifted four times. At the end of

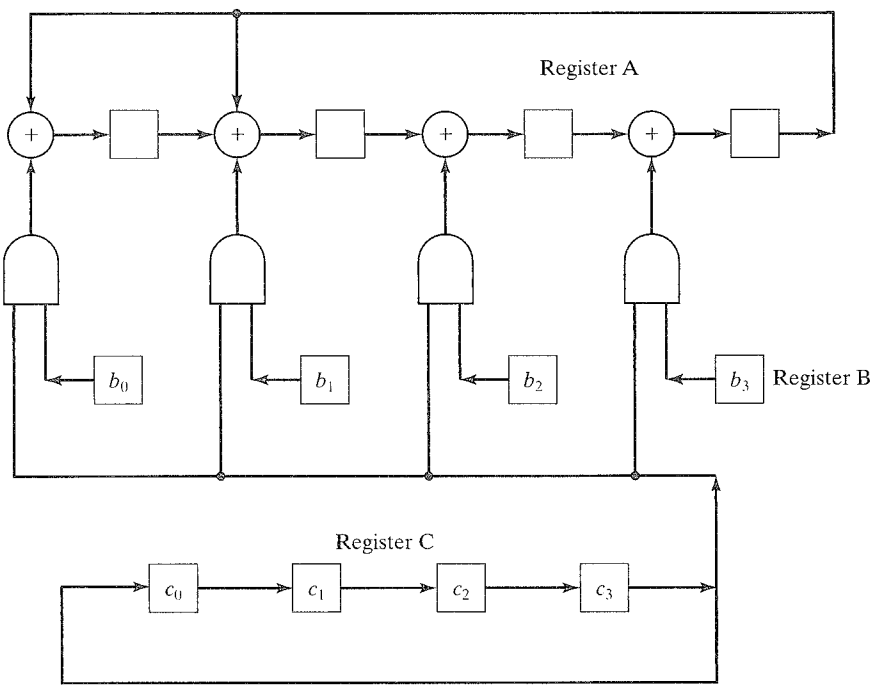

FIGURE 6.5: Circuit for multiplying two elements of $GF(2^4)$.

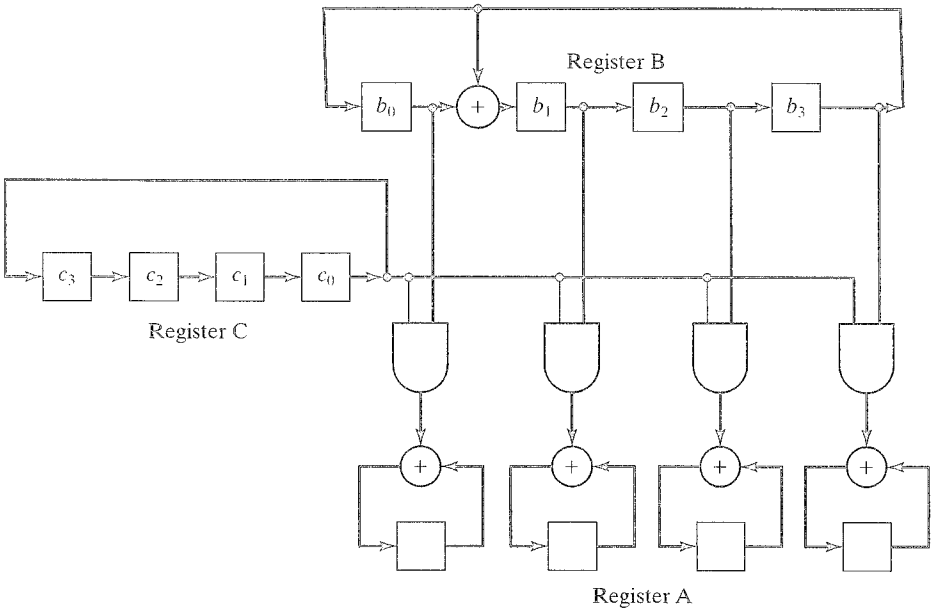

FIGURE 6.6: Another circuit for multiplying two elements of $GF(2^4)$.

the fourth shift, register A holds the product $\beta\gamma$. Both multiplication circuits shown in Figures 6.5 and 6.6 are of the same complexity and require the same amount of computation time.

Two elements from $GF(2^m)$ can be multiplied with a combinational logic circuit with $2m$ inputs and m outputs. The advantage of this implementation is its speed; however, for large m, it becomes prohibitively complex and costly. Multiplication can also be programmed in a general-purpose computer; it requires roughly $5m$ instruction executions.

Let $\mathbb{r}(X)$ be a polynomial over $GF(2)$. We consider now how to compute $\mathbb{r}(\alpha^i)$. This type of computation is required in the first step of decoding of a BCH code. It can be done with a circuit for multiplying a field element by α^i in $GF(2^m)$. Again, we use computation over $GF(2^4)$ for illustration. Suppose that we want to compute

$$\mathbb{r}(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + \cdots + r_{14}\alpha^{14}, \tag{6.46}$$

where α is a primitive element in $GF(2^4)$ given by Table 2.8. We can express the right-hand side of (6.46) in the form

$$\mathbb{r}(\alpha) = (\cdots(((r_{14})\alpha + r_{13})\alpha + r_{12})\alpha + \cdots)\alpha + r_0.$$

Then, we can compute $\mathbb{r}(\alpha)$ by adding an input to the circuit for multiplying by α shown in Figure 6.3. The resultant circuit for computing $\mathbb{r}(\alpha)$ is shown in Figure 6.7. In operation of this circuit, the register is initially empty. The vector $(r_0, r_1, \cdots, r_{14})$ is shifted into the circuit one digit at a time. After the first shift, the register contains $(r_{14}, 0, 0, 0)$. At the end of the second shift, the register contains the vector representation of $r_{14}\alpha + r_{13}$. At the completion of the third shift, the

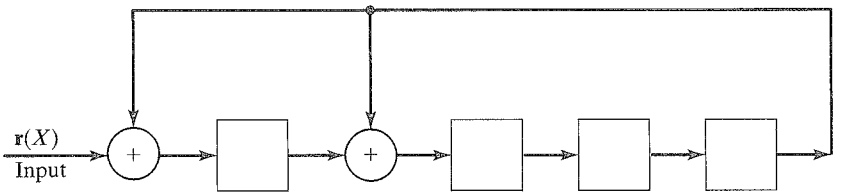

FIGURE 6.7: Circuit for computing $\mathbf{r}(\alpha)$.

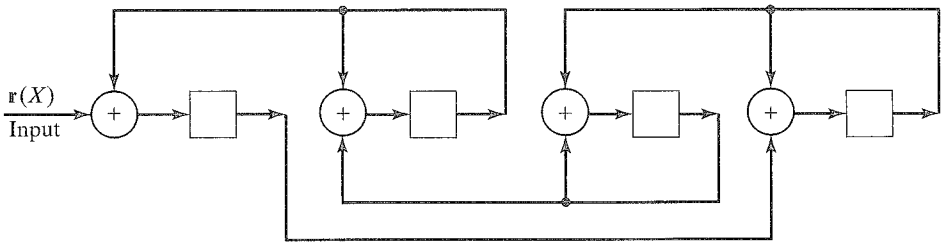

FIGURE 6.8: Circuit for computing $\mathbf{r}(\alpha^3)$.

register contains the vector representation of $(r_{14}\alpha + r_{13})\alpha + r_{12}$. When the last digit r_0 is shifted into the circuit, the register contains $\mathbf{r}(\alpha)$ in vector form.

Similarly, we can compute $\mathbf{r}(\alpha^3)$ by adding an input to the circuit for multiplying by α^3 of Figure 6.4. The resultant circuit for computing $\mathbf{r}(\alpha^3)$ is shown in Figure 6.8.

There is another way of computing $\mathbf{r}(\alpha^i)$. Let $\phi_i(X)$ be the minimal polynomial of α^i. Let $\mathbf{b}(X)$ be the remainder resulting from dividing $\mathbf{r}(X)$ by $\phi_i(X)$. Then,

$$\mathbf{r}(\alpha^i) = \mathbf{b}(\alpha^i).$$

Thus, computing $\mathbf{r}(\alpha^i)$ is equivalent to computing $\mathbf{b}(\alpha^i)$. A circuit can be devised to compute $\mathbf{b}(\alpha^i)$. For illustration, we again consider computation over $GF(2^4)$. Suppose that we want to compute $\mathbf{r}(\alpha^3)$. The minimal polynomial of α^3 is $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$. The remainder resulting from dividing $\mathbf{r}(X)$ by $\phi_3(X)$ has the form

$$\mathbf{b}(X) = b_0 + b_1 X + b_2 X^2 + b_3 X^3.$$

Then,

$$\begin{aligned} \mathbf{r}(\alpha^3) &= \mathbf{b}(\alpha^3) \\ &= b_0 + b_1\alpha^3 + b_2\alpha^6 + b_3\alpha^9 \\ &= b_0 + b_1\alpha^3 + b_2(\alpha^2 + \alpha^3) + b_3(\alpha + \alpha^3) \\ &= b_0 + b_3\alpha + b_2\alpha^2 + (b_1 + b_2 + b_3)\alpha^3. \end{aligned} \tag{6.47}$$

From the preceding expression we see that $\mathbf{r}(\alpha^3)$ can be computed by using a circuit that divides $\mathbf{r}(X)$ by $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$ and then combining

the coefficients of the remainder $\mathbb{b}(X)$ as given by (6.47). Such a circuit is shown in Figure 6.9, where the feedback connection of the shift register is based on $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$. Because α^6 is a conjugate of α^3, it has the same minimal polynomial as α^3, and therefore $\mathbb{r}(\alpha^6)$ can be computed from the same remainder $\mathbb{b}(X)$ resulting from dividing $\mathbb{r}(X)$ by $\phi_3(X)$. To form $\mathbb{r}(\alpha^6)$, we combine

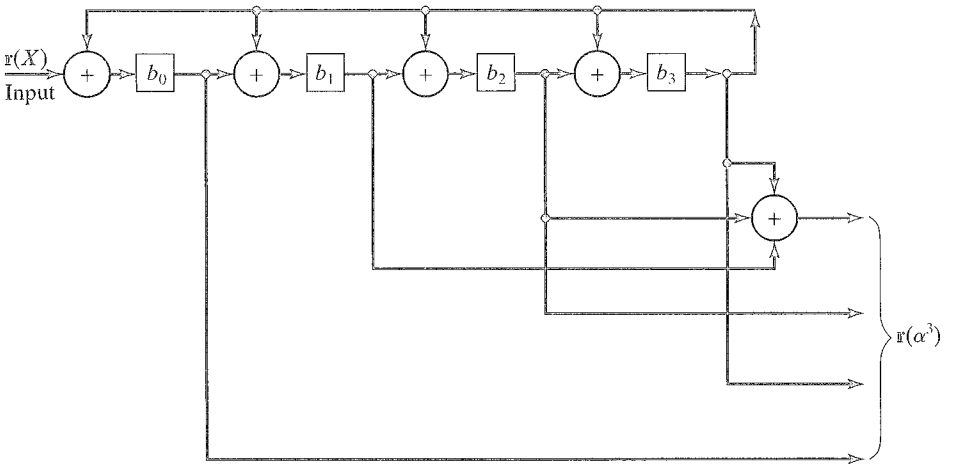

FIGURE 6.9: Another circuit for computing $\mathbb{r}(\alpha^3)$ in $GF(2^4)$.

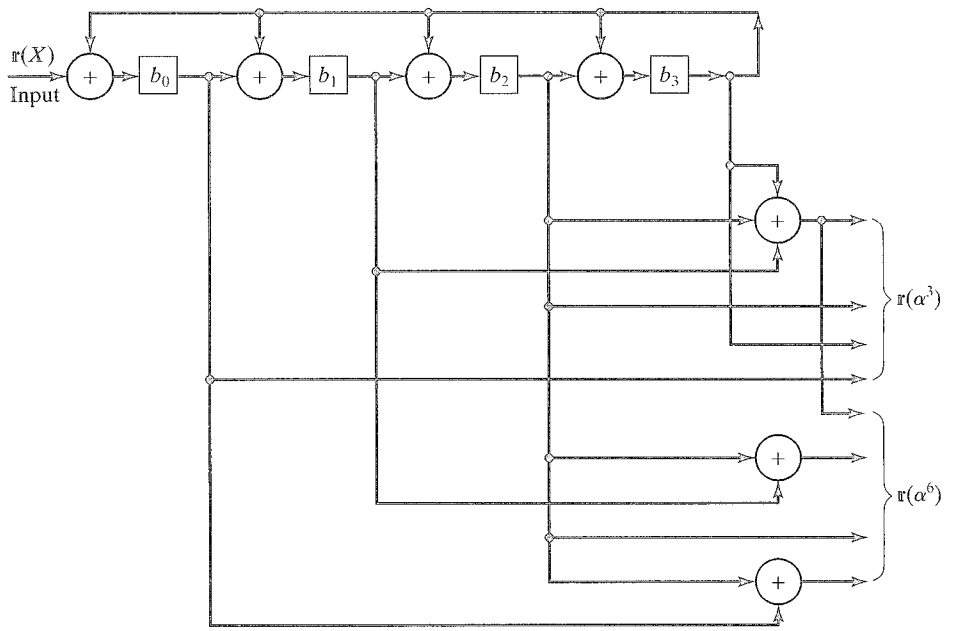

FIGURE 6.10: Circuit for computing $\mathbb{r}(\alpha^3)$ and $\mathbb{r}(\alpha^6)$ in $GF(2^4)$.

the coefficients of $\mathbb{b}(X)$ in the following manner:

$$\begin{aligned}\mathbb{r}(\alpha^6) &= \mathbb{b}(\alpha^6)\\ &= b_0 + b_1\alpha^6 + b_2\alpha^{12} + b_3\alpha^{18}\\ &= b_0 + b_1(\alpha^2 + \alpha^3) + b_2(1 + \alpha + \alpha^2 + \alpha^3) + b_3\alpha^3\\ &= (b_0 + b_2) + b_2\alpha + (b_1 + b_2)\alpha^2 + (b_1 + b_2 + b_3)\alpha^3.\end{aligned}$$

The combined circuit for computing $\mathbb{r}(\alpha^3)$ and $\mathbb{r}(\alpha^6)$ is shown in Figure 6.10.

The arithmetic operation of division over $GF(2^m)$ can be performed by first forming the multiplicative inverse of the divisor β and then multiplying this inverse β^{-1} by the dividend, thus forming the quotient. The multiplicative inverse of β can be found by using the fact that $\beta^{2^m-1} = 1$. Thus,

$$\beta^{-1} = \beta^{2^m-2}.$$

6.8 IMPLEMENTATION OF ERROR CORRECTION

Each step in the decoding of a BCH code can be implemented either by digital hardware or by software. Each implementation has certain advantages. We consider these implementations next.

6.8.1 Syndrome Computations

The first step in decoding a t-error-correction BCH code is to compute the $2t$ syndrome components $S_1, S_2, \cdots, S_{2t}$. these syndrome components may be obtained by substituting the field elements $\alpha, \alpha^2, \cdots, \alpha^{2t}$ into the received polynomial $\mathbb{r}(X)$. For software implementation, α^i into $\mathbb{r}(X)$ is best substituted as follows:

$$\begin{aligned}S_i &= \mathbb{r}(\alpha^i) = r_{n-1}(\alpha^i)^{n-1} + r_{n-2}(\alpha^i)^{n-2} + \cdots + r_1\alpha^i + r_0\\ &= (\cdots((r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i + \cdots + r_1)\alpha^i + r_0.\end{aligned}$$

This computation takes $n - 1$ additions and $n - 1$ multiplications. For binary BCH codes, we have shown that $S_{2i} = S_i^2$. With this equality, the $2t$ syndrome components can be computed with $(n - 1)t$ additions and nt multiplications.

For hardware implementation, the syndrome components may be computed with feedback shift registers as described in Section 6.7. We may use either the type of circuits shown in Figures 6.7 and 6.8 or the type of circuit shown in Figure 6.10. The second type of circuit is simpler. From the expression of (6.3), we see that the generator polynomial is a product of at most t minimal polynomials. Therefore, at most t feedback shift registers, each consisting of at most m stages, are needed to form the $2t$ syndrome components. The computation is performed as the received polynomial $\mathbb{r}(X)$ enters the decoder. As soon as the entire $\mathbb{r}(X)$ has entered the decoder, the $2t$ syndrome components are formed. It takes n clock cycles to complete the computation. A syndrome computation circuit for the double-error-correcting (15, 7) BCH code is shown in Figure 6.11, where two feedback shift registers, each with four stages, are employed.

The advantage of hardware implementation of syndrome computation is speed; however, software implementation is less expensive.

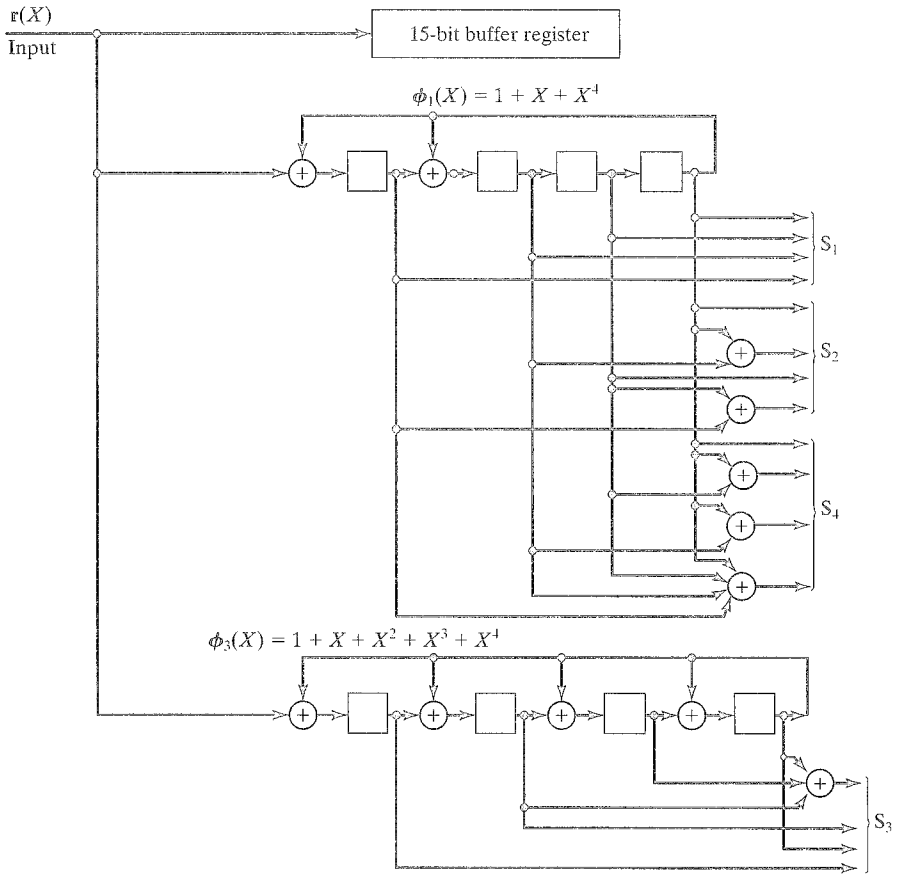

FIGURE 6.11: Syndrome computation circuit for the double-error-correcting (15, 7) BCH code.

6.8.2 Finding the Error-Location Polynomial $\sigma(X)$

For this step the software computation requires somewhat fewer than t additions and t multiplications to compute each $\sigma^{(\mu)}(X)$ and each d_μ, and since there are t of each, the total is roughly $2t^2$ additions and $2t^2$ multiplications. A pure hardware implementation requires the same total, and the speed depends on how much is done in parallel. The type of circuit shown in Figure 6.2 may be used for addition, and the type of circuits shown in Figures 6.5 and 6.6 may be used for multiplications. A very fast hardware implementation of finding $\sigma(X)$ would probably be very expensive, whereas a simple hardware implementation would probably be organized much like a general-purpose computer, except with a wired rather than a stored program.

6.8.3 Computation of Error-Location Numbers and Error Correction

In the worst case, this step requires substituting n field elements into an error-location polynomial $\sigma(X)$ of degree t to determine its roots. In software this requires nt multiplications and nt additions. This step can also be performed in hardware using Chien's

searching circuit, shown in Figure 6.1. Chien's searching circuit requires t multipliers for multiplying by $\alpha, \alpha^2, \cdots, \alpha^t$, respectively. These multipliers may be the type of circuits shown in Figures 6.3 and 6.4. Initially, $\sigma_1, \sigma_2, \cdots, \sigma_t$ found in step 2 are loaded into the registers of the t multipliers. Then, these multipliers are shifted n times. At the end of the lth shift, the t registers contain $\sigma_1\alpha^l, \sigma_2\alpha^{2l}, \cdots, \sigma_t\alpha^{tl}$. Then, the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \cdots + \sigma_t\alpha^{tl}$$

is tested. If the sum is zero, α^{n-l} is an error-location number; otherwise, α^{n-l} is not an error-location number. This sum can be formed by using t m-input modulo-2 adders. An m-input OR gate is used to test whether the sum is zero. It takes n clock cycles to complete this step. If we want to correct only the message digits, only k clock cycles are needed. A Chien's searching circuit for the double-error-correcting (15, 6) BCH code is shown in Figure 6.12.

For large t and m, the cost for building t wired multipliers for multiplying $\alpha, \alpha^2, \cdots, \alpha^t$ in one clock cycle becomes substantial. For more economical but

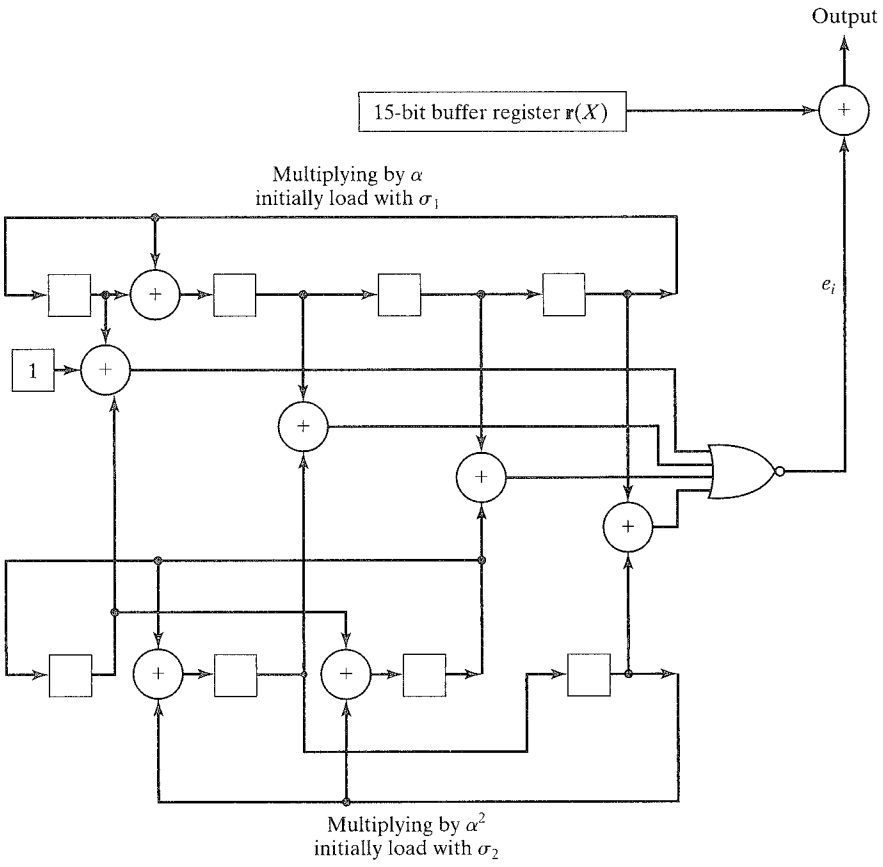

FIGURE 6.12: Chien's searching circuit for the double-error-correcting (15, 7) BCH code.

slower multipliers, we may use the type of circuit shown in Figure 6.5 (or shown in Figure 6.6). Initially, σ_i is loaded into register B, and α^i is stored in register C. After m clock cycles, the product $\sigma_i\alpha^i$ is in register A. To form $\sigma_i\alpha^{2i}$, $\sigma_i\alpha^i$ is loaded into register B. After another m clock cycles, $\sigma_i\alpha^{2i}$ will be in register A. Using this type of multiplier, nm clock cycles are needed to complete the third step of decoding a binary BCH code.

Steps 1 and 3 involve roughly the same amount of computation. Because n is generally much larger than t, $4nt$ is much larger than $4t^2$, and steps 1 and 3 involve most of the computation. Thus, hardware implementation of these steps is essential if high decoding speed is needed. With hardware implementation, step 1 can be done as the received polynomial $r(X)$ is read in, and step 3 can be accomplished as $r(X)$ is read out. In this case the computation time required in steps 1 and 3 is essentially negligible.

6.9 WEIGHT DISTRIBUTION AND ERROR DETECTION OF BINARY BCH CODES

The weight distributions of double-error-correcting, triple-error-correcting, and some low-rate primitive BCH codes have been completely determined; however, the weight distributions for the other BCH codes are still unknown. The weight distribution of a double-error-correcting or a triple-error-correcting primitive BCH code can be determined by first computing the weight distribution of its dual code and then applying the MacWilliams identity of (3.32). The weight distribution of the dual of a double-error-correcting primitive BCH code of length $2^m - 1$ is given in

TABLE 6.9: Weight distribution of the dual of a double-error-correcting primitive binary BCH code of length $2^m - 1$.

Odd $m \geq 3$	
Weight, i	Number of vectors with weight i, B_i
0	1
$2^{m-1} - 2^{(m+1)/2-1}$	$[2^{m-2} + 2^{(m-1)/2-1}](2^m - 1)$
2^{m-1}	$(2^m - 2^{m-1} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m+1)/2-1}$	$[2^{m-2} - 2^{(m-1)/2-1}](2^m - 1)$

TABLE 6.10: Weight distribution of the dual of a double-error-correcting primitive binary BCH code of length $2^m - 1$.

Even $m \geq 4$	
Weight, i	Number of vectors with weight i, B_i
0	1
$2^{m-1} - 2^{(m+2)/2-1}$	$2^{(m-2)/2-1}[2^{(m-2)/2} + 1](2^m - 1)/3$
$2^{m-1} - 2^{m/2-1}$	$2^{(m+2)/2-1}(2^{m/2} + 1)(2^m - 1)/3$
2^{m-1}	$(2^{m-2} + 1)(2^m - 1)$
$2^{m-1} + 2^{m/2-1}$	$2^{(m+2)/2-1}(2^{m/2} - 1)(2^m - 1)/3$
$2^{m-1} + 2^{(m+2)/2-1}$	$2^{(m-2)/2-1}[2^{(m-2)/2} - 1](2^m - 1)/3$

TABLE 6.11: Weight distribution of the dual of a triple-error-correcting primitive binary BCH code of length $2^m - 1$.

Odd $m \geq 5$	
Weight, i	**Number of Vectors with weight i, B_i**
0	1
$2^{m-1} - 2^{(m+1)/2}$	$2^{(m-5)/2}[2^{(m-3)/2} + 1](2^{m-1} - 1)(2^m - 1)/3$
$2^{m-1} - 2^{(m-1)/2}$	$2^{(m-3)/2}[2^{(m-1)/2} + 1](5 \cdot 2^{m-1} + 4)(2^m - 1)/3$
2^{m-1}	$(9 \cdot 2^{2m-4} + 3 \cdot 2^{m-3} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m-1)/2}$	$2^{(m-3)/2}[2^{(m-1)/2} - 1](5 \cdot 2^{m-1} + 4)(2^m - 1)/3$
$2^{m-1} + 2^{(m+1)/2}$	$2^{(m-5)/2}[2^{(m-3)/2} - 1](2^{m-1} - 1)(2^m - 1)/3$

TABLE 6.12: Weight distribution of the dual of a triple-error-correcting primitive binary BCH code of length $2^m - 1$.

Even $m \geq 6$	
Weight, i	**Number of vectors with weight i, B_i**
0	1
$2^{m-1} - 2^{(m+4)/2-1}$	$[2^{m-1} + 2^{(m+4)/2-1}](2^m - 4)(2^m - 1)/960$
$2^{m-1} - 2^{(m+2)/2-1}$	$7[2^{m-1} + 2^{(m+2)/2-1}]2^m(2^m - 1)/48$
$2^{m-1} - 2^{m/2-1}$	$2(2^{m-1} + 2^{m/2-1})(3 \cdot 2^m + 8)(2^m - 1)/15$
2^{m-1}	$(29 \cdot 2^{2m} - 4 \cdot 2^m + 64)(2^m - 1)/64$
$2^{m-1} + 2^{m/2-1}$	$2(2^{m-1} - 2^{m/2-1})(3 \cdot 2^m + 8)(2^m - 1)/15$
$2^{m-1} + 2^{(m+2)/2-1}$	$7[2^{m-1} - 2^{(m+2)/2-1}]2^m(2^m - 1)/48$
$2^{m-1} + 2^{(m+4)/2-1}$	$[2^{m-1} - 2^{(m+4)/2-1}](2^m - 4)(2^m - 1)/960$

Tables 6.9 and 6.10. The weight distribution of the dual of a triple-error-correcting primitive BCH code is given in Tables 6.11 and 6.12. Results presented in Tables 6.9 to 6.11 were mainly derived by Kasami [22]. For more on the weight distribution of primitive binary BCH codes, the reader is referred to [9] and [22].

If a double-error-correcting or a triple-error-correcting primitive BCH code is used for error detection on a BSC with transition probability p, its probability of an undetected error can be computed from (3.36) and one of the weight distribution tables, Tables 6.9 to 6.12. It has been proved [23] that the probability of an undetected error, $P_u(E)$, for a double-error-correcting primitive BCH code of length $2^m - 1$ is upper bounded by 2^{-2m} for $p \leq \frac{1}{2}$, where $2m$ is the number of parity-check digits in the code. The probability of an undetected error for a triple-error-correcting primitive BCH code of length $2^m - 1$ with odd m satisfies the upper bound 2^{-3m}, where $3m$ is the number of parity-check digits in the code [24].

It would be interesting to know how a general t-error-correcting primitive BCH code performs when it is used for error detection on a BSC with transition probability p. It has been proved [25] that for a t-error-correcting primitive BCH of

length $2^m - 1$, if the number of parity-check digits is equal to mt, and m is greater than a certain constant $m_0(t)$, the number of codewords of weight i satisfies the following equalities:

$$A_i = \begin{cases} 0 & \text{for } 0 < i \leq 2t \\ (1 + \lambda_0 \cdot n^{1/10}) \begin{pmatrix} n \\ i \end{pmatrix} 2^{-(n-k)} & \text{for } i > 2t, \end{cases} \tag{6.48}$$

where $n = 2^m - 1$, and λ_0 is upper bounded by a constant. From (3.19) and (6.48), we obtain the following expression for the probability of an undetected error:

$$P_u(E) = (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)} \sum_{i=2t+1}^{n} \begin{pmatrix} n \\ i \end{pmatrix} p^i (1-p)^{n-i}. \tag{6.49}$$

Let $\varepsilon = (2t+1)/n$. Then the summation of (6.49) can be upper bounded as follows [13]:

$$\sum_{i=n\varepsilon}^{n} \begin{pmatrix} n \\ i \end{pmatrix} p^i (1-p)^{n-i} \leq 2^{-nE(\varepsilon, p)} \tag{6.50}$$

provided that $p < \varepsilon$, where

$$E(\varepsilon, p) = H(p) + (\varepsilon - p)H'(p) - H(\varepsilon)$$
$$H(x) = -x \log_2 x - (1-x) \log_2 (1-x),$$

and

$$H'(x) = \log_2 \frac{1-x}{x}.$$

$E(\varepsilon, p)$ is positive for $\varepsilon > p$. Combining (6.49) and (6.50), we obtain the following upper bound on $P_u(E)$ for $\varepsilon > p$:

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-nE(\varepsilon, p)} 2^{-(n-k)}$$

For $p < \varepsilon$ and sufficient large n, $P_u(E)$ can be made very small. For $p \geq \varepsilon$, we can also derive a bound on $P_u(E)$. It is clear from (6.49) that

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)} \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} p^i (1-p)^{n-i}.$$

Because

$$\sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} p^i (1-p)^i = 1,$$

we obtain the following upper bound on $P_u(E)$:

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)}. \tag{6.51}$$

We see that for $p \geq \varepsilon$, P_u still decreases exponentially with the number of parity-check digits, $n - k$. If we use a sufficiently large number of parity-check digits, the probability of an undetected error $P_u(E)$ will become very small. Now, we may

summarize the preceding results above as follows: For a t-error-correcting primitive BCH code of length $n = 2^m - 1$ with number of parity-check digits $n - k = mt$ and $m \geq m_0(t)$, its probability of an undetected error on a BSC with transition probability p satisfies the following bounds:

$$P_u(E) \leq \begin{cases} (1 + \lambda_0 \cdot n^{-1/10})2^{-n[1-R+E(\varepsilon,p)]} & \text{for } p < \varepsilon \\ (1 + \lambda_0 \cdot n^{-1/10})2^{-n(1-R)} & \text{for } p \geq \varepsilon \end{cases} \tag{6.52}$$

where $\varepsilon = (2t + 1)/n$, $R = k/n$, and λ_0 is a constant.

The foregoing analysis indicates that primitive BCH codes are very effective for error detection on a BSC.

6.10 REMARKS

BCH codes form a subclass of a very special class of linear codes known as *Goppa codes* [21, 22]. It has been proved that the class of Goppa codes contains good codes. Goppa codes are in general noncyclic (except the BCH codes), and they can be decoded much like BCH codes. The decoding also consists of four steps: (1) compute the syndromes; (2) determine the error-location polynomial $\sigma(X)$; (3) find the error-location numbers; and (4) evaluate the error values (this step is not needed for binary Goppa codes). Berlekamp's iterative algorithm for finding the error-location polynomial for a BCH code can be modified for finding the error-location polynomial for Goppa codes [26]. Discussion of Goppa codes is beyond the scope of this introductory book. Moreover, implementation of BCH codes is simpler than that of Goppa codes, and no Goppa codes better than BCH codes have been found. For details on Goppa codes, the reader is referred to [26]–[30].

Our presentation of BCH codes and their implementation is given in the time domain. BCH codes also can be defined and implemented in the frequency domain using Fourier transforms over Galois fields. Decoding BCH codes in the frequency domain sometimes offers computational or implementation advantages. This topic will be discussed in Chapter 7.

PROBLEMS

6.1 Consider the Galois field $GF(2^4)$ given by Table 2.8. The element $\beta = \alpha^7$ is also a primitive element. Let $\mathbf{g}_0(X)$ be the lowest-degree polynomial over $GF(2)$ that has

$$\beta, \beta^2, \beta^3, \beta^4$$

as its roots. This polynomial also generates a double-error-correcting primitive BCH code of length 15.

a. Determine $\mathbf{g}_0(X)$.

b. Find the parity-check matrix for this code.

c. Show that $\mathbf{g}_0(X)$ is the reciprocal polynomial of the polynomial $\mathbf{g}(X)$ that generates the (15, 7) double-error-correcting BCH code given in Example 6.1.

6.2 Determine the generator polynomials of all the primitive BCH codes of length 31. Use the Galois field $GF(2^5)$ generated by $\mathbf{p}(X) = 1 + X^2 + X^5$.

6.3 Suppose that the double-error-correcting BCH code of length 31 constructed in Problem 6.2 is used for error correction on a BSC. Decode the received polynomials $\mathbb{r}_1(X) = X^7 + X^{30}$ and $\mathbb{r}_2(X) = 1 + X^{17} + X^{28}$.

6.4 Consider a t-error-correcting primitive binary BCH code of length $n = 2^m - 1$. If $2t + 1$ is a factor of n, prove that the minimum distance of the code is exactly $2t + 1$. (*Hint*: Let $n = l(2t + 1)$. Show that $(X^n + 1)/(X^l + 1)$ is a code polynomial of weight $2t + 1$.)

6.5 Is there a binary t-error-correcting BCH code of length $2^m + 1$ for $m \geq 3$ and $t < 2^{m-1}$? If there is such a code, determine its generator polynomial.

6.6 Consider the field $GF(2^4)$ generated by $\mathbb{p}(X) = 1 + X + X^4$ (see Table 2.8). Let α be a primitive element in $GF(2^4)$ such that $\mathbb{p}(\alpha) = 0$. Devise a circuit that is capable of multiplying any element in $GF(2^4)$ by α^7.

6.7 Devise a circuit that is capable of multiplying any two elements in $GF(2^5)$. Use $\mathbb{p}(X) = 1 + X^2 + X^5$ to generate $GF(2^5)$.

6.8 Devise a syndrome computation circuit for the binary double-error-correcting (31, 21) BCH code.

6.9 Devise a Chien's searching circuit for the binary double-error-correcting (31, 21) BCH code.

6.10 Consider the Galois field $GF(2^6)$ given by Table 6.2. Let $\beta = \alpha^3, l_0 = 2$, and $d = 5$. Determine the generator polynomial of the BCH code that has

$$\beta^2, \beta^3, \beta^4, \beta^5$$

as its roots (the general form presented at the end of Section 6.1). What is the length of this code?

6.11 Let $l_0 = -t$ and $d = 2t + 2$. Then we obtain a BCH code of designed distance $2t + 2$ whose generator polynomial has

$$\beta^{-t}, \cdots, \beta^{-1}, \beta^0, \beta^1, \cdots, \beta^t$$

and their conjugates as all its roots.

𝕒. Show that this code is a reversible cyclic code.

𝕓. Show that if t is odd, the minimum distance of this code is at least $2t + 4$. (*Hint:* Show that $\beta^{-(t+1)}$ and β^{t+1} are also roots of the generator polynomial.)

BIBLIOGRAPHY

1. A. Hocquenghem, "Codes corecteurs d'erreurs," *Chiffres*, 2: 147–56, 1959.

2. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Inform. Control*, 3: 68–79, March 1960.

3. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose–Chaudhuri Codes," *IRE Trans. Inform. Theory*, IT-6: 459–70, September 1960.

4. D. Gorenstein and N. Zierler, "A Class of Cyclic Linear Error-Correcting Codes in p^m Symbols," *J. Soc. Ind. Appl. Math.*, 9: 107–214, June 1961.

5. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Ind. Appl. Math.*, 8: 300–304, June 1960.

6. R. T. Chien, "Cyclic Decoding Procedure for the Bose–Chaudhuri–Hocquenghem Codes," *IEEE Trans. Inform. Theory*, IT-10: 357–63, October 1964.

7. G. D. Forney, "On Decoding BCH Codes," *IEEE Trans. Inform. Theory*, IT-11: 549–57, October 1965.

8. E. R. Berlekamp, "On Decoding Binary Bose–Chaudhuri–Hocquenghem Codes," *IEEE Trans. Inform. Theory*, IT-11: 577–80, October 1965.

9. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

10. J. L. Massey, "Step-by-Step Decoding of the Bose–Chaudhuri–Hocquenghem Codes," *IEEE Trans. Inform. Theory*, IT-11: 580–85, October 1965.

11. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Inform. Theory*, IT-15: 122–27, January 1969.

12. H. O. Burton, "Inversionless Decoding of Binary BCH Codes," *IEEE Trans. Inform. Theory*, IT-17: 464–66, July 1971.

13. W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, 2d ed., MIT Press, Cambridge, 1970.

14. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.

15. G. C. Clark, Jr., and J. B. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1981.

16. R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Reading Mass., 1983.

17. S. B. Wicker, *Error Control Systems for Digital Communication and Storage,* Prentice Hall, Englewood Cliffs, N.J., 1995.

18. H. B. Mann, "On the Number of Information Symbols in Bose–Chaudhuri Codes," *Inform. Control*, 5: 153–62, June 1962.

19. C. L. Chen, "High-Speed Decoding of BCH Codes," *IEEE Trans. Inform. Theory*, IT-27(2): 254–56, March 1981.

20. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Weight Distributions of BCH Codes," *IEEE Trans. Inform. Theory*, IT-12(2): 274, April 1966.

21. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Cyclic Codes Which are Invariant under the Affine Group," *Scientific Report AFCRL-66-622*, Air Force Cambridge Research Labs, Bedford, Mass., 1966.

22. T. Kasami, "Weight Distributions of Bose–Chaudhuri–Hocquenghem Codes," *Proc. Conf. Combinatorial Mathematics and Its Applications*, R. C. Bose and T. A. Dowling, eds., University of North Carolina Press, Chapel Hill, 1968.

23. S. K. Leung-Yan-Cheong, E. R. Barnes, and D. U. Friedman, "On Some Properties of the Undetected Error Probability of Linear Codes," *IEEE Trans. Inform. Theory*, IT-25(1): 110–12, January 1979.

24. G. T. Ong and C. Leung, "On the Undetected Error Probability of Triple-Error-Correcting BCH Codes," *IEEE Trans. Inform. Theory*, IT-37: 673–78, 1991.

25. V. M. Sidelńikov, "Weight Spectrum of Binary Bose–Chaudhuri–Hocquenghem Code." *Probl. Inform. Transm.*, 7(1): 11–17, 1971.

26. V. D. Goppa, "A New Class of Linear Codes," *Probl. Peredachi Inform.*, 6(3): 24–30, September 1970.

27. V. D. Goppa, "Rational Representation of Codes and (L, g) Codes," *Probl. Peredachi Inform.*, 7(3): 41–49, September 1971.

28. N. J. Patterson, "The Algebraic Decoding of Goppa Codes," *IEEE Trans. Inform. Theory*, IT-21: 203–7, March 1975.

29. E. R. Berlekamp, "Goppa Codes," *IEEE Trans. Inform. Theory*, IT-19(5): 590–92, September 1973.

30. Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Inform. Control*, 27: 87–99, January 1975.

31. R. E. Blahut, "Transform Techniques for Error Control Codes," *IBM J. Res. Dev.*, 23(3): 299–315 May 1979.