A Fast Technique for Computing Syndromes of BCH and RS Codes

I. S. Reed
Department of Electrical Engineering
University of Southern California

T. K. Truong, R. L. Miller
Communication Systems Research Section

In this article, a combination of the Chinese Remainder Theorem and Winograd's algorithm is used to compute transforms of odd length over $GF(2^m)$. Such transforms are used to compute the syndromes needed for decoding BCH and RS codes. The present scheme requires substantially fewer multiplications and additions than the conventional method of computing the syndromes directly.

I. Introduction

It is shown in Ref. 1 that the finite field transform can be used to compute the syndromes for BCH and RS codes. The disadvantage of the transform method over $GF(2^m)$ is that the transform length is an odd number, so that the most efficient FFT algorithm cannot be used. In this article a combination of the Chinese Remainder Theorem and Winograd's algorithm is used to develop a fast finite field transform over $GF(2^m)$. Such a fast transform can be used to compute the syndromes needed when decoding BCH and RS codes (Ref. 2). It is also shown by example that the number of multiplications and additions of this new scheme for computing the syndromes is substantially fewer than used in the direct method.

II. The Computation of the Syndromes for BCH and RS Codes

Let n be the block length of a BCH or RS code of designed distance d in $GF(2^m)$. Also denote the received vector by $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$. If α^j $1 \leqslant j \leqslant d-1$ are roots of the code's generator polynominal, then the decoder needs to evaluate the syndromes S_j for $1 \leqslant j \leqslant d-1$ from the received vector by

$$S_j = \sum_{i=0}^{n-1} r_i \alpha^{ij} \text{ for } 1 \leqslant j \leqslant d-1 \qquad (1)$$

Evidently, a direct computation of S_j for $1 \leqslant j \leqslant d-1$ involves $(d-1)(n-1)$ multiplications and $(d-1)(n-1)$ additions. Using the Chinese Remainder Theorem, it is shown in this section that this number of multiplications and additions can be reduced substantially. To see this, if $n = n_1 n_2 \ldots n_r$, where $(n_i, n_j) = 1$ for $i \neq j$, then by the Chinese Remainder Theorem it follows (Refs. 3, 4) that the n-point transform over $GF(2^m)$ defined in (1) can be decomposed into a multidimensional transform over $GF(2^m)$ as follows:

Given an integer j, define $f(j) = (j_1, j_2, \ldots, j_r)$, where $j_k = j \bmod n_k$ $(1 \leqslant k \leqslant r)$. Then $\alpha_k = \alpha^{(0,0,\ldots,0,1,0,\ldots,0)}$, where 1 is in the k^{th} position, is a primitive $n_k{}^{\text{th}}$ root of

unity. The computation of S_j given by (1) can now be written as

$$S_j = S_{(j_1, j_2, \ldots, j_r)}$$

$$= \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \cdots \sum_{i_r=0}^{n_r-1} a_{(i_1, i_2, \ldots, i_r)} \alpha_1^{i_1 j_1} \alpha_2^{i_2 j_2} \cdots \alpha_r^{i_r j_r} \quad (2)$$

$$\text{for } 1 \leqslant j \leqslant d-1$$

From Eq. (2) this algorithm consists of the following r stages:

Stage 1:

$$S^1_{(i_1, i_2, \ldots, j_r)} = \sum_{i_r=0}^{n_r-1} a_{(i_1, i_2, \ldots, i_r)} \alpha_r^{i_r j_r}$$

$$\text{for } 0 \leqslant j_r \leqslant n_r - 1$$

Stage 2:

$$S^2_{(i_1, i_2, \ldots, j_{r-1}, j_r)} = \sum_{i_{r-1}=0}^{n_{r-1}-1} S^1_{(i_1, i_2, \ldots, j_r)} \alpha_{r-1}^{i_{r-1} j_{r-1}}$$

$$\text{for } 0 \leqslant j_{r-1} \leqslant n_{r-1} - 1$$

$$\vdots$$

Stage r:

$$S_j = S^r_{(j_1, \ldots, j_{r-1}, j_r)} = \sum_{i_1=0}^{n_1-1} S^{r-1}_{(i_1, j_2, \ldots, j_r)} \alpha_1^{i_1 j_1}$$

$$\text{for } 1 \leqslant j \leqslant d-1 \quad (3)$$

Assume the number of multiplications and additions needed to perform the n_i-point transform for $1 \leqslant i \leqslant r-1$ is m_i and a_i, respectively. Then by Eq. (3) it is evident that the number of multiplications and additions needed to compute Stage 1 up to Stage $(r-1)$ is given by

$$M = n \sum_{i=1}^{r-1} \frac{m_i}{n_i} \text{ and } A = n \sum_{i=1}^{r-1} \frac{a_i}{n_i}, \text{ respectively.}$$

To compute the syndromes one needs only to compute the first d points of the transform over $GF(2^m)$ defined in Eq. (1). Thus, at the Stage r in Eq. (3), one needs only to compute $S_j = S_{(j_1, j_2, \ldots, j_r)}$ for $1 \leqslant j \leqslant d-1$. Hence the number of both multiplications and additions needed to compute Stage r in (3) is $(d-1)(n_1-1)$. Combining this with the above results the total number of multiplications and additions needed to compute (1) is finally $M + (d-1)(n_1-1)$ and $A + (d-1)(n_1-1)$. respectively.

Suppose that the transform length given in Eq. (3) of the first $r-1$ stages is small. Then it was shown (Ref. 1) that such short length transforms can often be computed by a modification of Winograd's algorithm. Algorithms for computing a transform over $GF(2^m)$ of n points for $n = 3,5,7,9.17$ are given in detail in Ref. (1). The detailed algorithms for computing the n-point transform over $GF(2^m)$ for $n = 3,5$ are given in the appendix. These transforms are used in the following example to compute a transform of 2^8-1 points.

Example 1: Let $n = 255$ be the block length of an RS code of designed distance $d = 33$. This code will correct t errors where $2t < 33$. The first step of the decoding process is to compute the 32 syndromes as follows:

$$S_j = \sum_{i=0}^{255-1} r_i \alpha^{ij} \text{ for } 1 \leqslant j \leqslant d-1 = 32 \quad (4)$$

where α is an element of order 255 in $GF(2^8)$. If Eq. (4) is computed directly, the number of multiplications and additions needed is $32(255-1) = 8128$ and $32(255-1) = 8128$, respectively. Since $n = 255 = n_1 n_2 n_3 = 17 \cdot 5 \cdot 3$, (4) for this case reduces to

Stage 1:

$$S^1_{(i_1, i_2, j_3)} = \sum_{i_3=0}^{3-1} a_{(i_1, i_2, i_3)} \alpha_3^{i_3 j_3} \text{ for } 0 \leqslant j_3 \leqslant 2$$

Stage 2:

$$S^2_{(i_1,j_2,j_3)} = \sum_{i_2=0}^{5-1} S^1_{(i_1,i_2,j_3)}\, \alpha_2^{i_2 j_2} \quad \text{for } 0 \leqslant j_2 \leqslant 4$$

Stage 3:

$$S_j = S^3_{(j_1,j_2,j_3)} = \sum_{i_1=0}^{17-1} S^2_{(i_1,j_2,j_3)}\, \alpha_1^{i_1 j_1} \quad \text{for } 1 \leqslant j \leqslant 32 \tag{5}$$

In Eq. (5) Stage 1 is a 3-point transform and Stage 2 is a 5-point transform. These two transforms are computed by a modification of Winograd's algorithm. It is shown in the appendix that the number of multiplications and additions needed to compute the 3-point transform is 1 and 5, respectively. Thus, the number of multiplications and additions needed to compute Stage 1 is (17)(5)(1) = 85, and (17)(5)(5) = 425, respectively. Again by the appendix one observes that the number of multiplications and additions needed to compute a 5-point transform is 5 and 17, respectively. Thus, the number of multiplications and additions needed to compute Stage 2 is (17)(3)(5) = 255 and (17)(3)(17) = 867, respectively. Since one needs to compute S_j only for $j = 1,2,\ldots,32$, the number of multiplications and additions needed to compute Stage 3 is 32(17 - 1) = 512 and 32(17 - 1) = 512, respectively. Thus the total number of multiplications and additions needed to compute the syndromes S_j for $1 \leqslant j \leqslant 32$ is 85 + 255 + 512 = 852, and 425 + 867 + 512 = 1804, respectively.

Appendix

A Summary of Transform Algorithm of *n*-Points for $n = 3, 5$

For $n = 3$, let γ be a primitive cube root of unity in $GF(2^m)$, where $3|(2^m - 1)$. The 3-point transform over $GF(2^m)$ is

$$A_k = \sum_{i=0}^{3-1} a_i \gamma^{ik} \quad \text{for } 0 \leqslant k \leqslant 2.$$

Algorithm for n = 3:

$$s_1 = a_1 + a_2, s_2 = s_1 + a_0, m_0 = 1 \cdot s_2;$$

$$m_1 = s_1 \cdot \gamma^1, m_2 = 1 \cdot a_1, m_3 = 1 \cdot a_2;$$

$$s_3 = m_0 + m_1, s_4 = s_3 + m_2, s_5 = s_3 + m_3;$$

$$A_0 = m_0, A_1 = s_4, A_2 = s_5 \qquad \text{(A-1)}$$

Hence, from Eq. (A-1), the total number of multiplications and additions needed to perform a 3-point transform is 1 and 5, respectively.

Next consider the case $n = 5$. Let γ be a primitive 5^{th} root of unity in $GF(2^m)$, where $5|(2^m - 1)$. The 5-point transform is

$$A_k = \sum_{i=0}^{5-1} a_i \gamma^{ik} \quad \text{for } 0 \leqslant k \leqslant 4$$

Algorithm for n = 5:

$$s_1 = (a_2 + a_3), s_2 = a_1 + a_4, s_3 = a_1 + a_3;$$

$$s_4 = a_2 + a_4, s_5 = s_1 + s_2, s_6 = s_5 + a_0;$$

$$m_0 = 1 \cdot s_6, m_1 = (\gamma^0 + \gamma^3) \cdot s_5, m_2 = (\gamma^3 + \gamma^4) \cdot s_1;$$

$$m_3 = (\gamma^1 + \gamma^3) \cdot s_2, m_4 = (\gamma + \gamma^4) \cdot s_3, m_5 = (\gamma + \gamma^4) \cdot s_4;$$

$$m_6 = 1 \cdot a_1, m_7 = 1 \cdot a_3, m_8 = 1 \cdot a_4, m_9 = 1 \cdot a_2;$$

$$s_7 = m_0 + m_1, s_8 = s_7 + m_2, s_9 = s_7 + m_3, s_{10} = m_5 + m_9;$$

$$s_{11} = m_4 + m_6, s_{12} = m_5 + m_8, s_{13} = m_4 + m_7, s_{14} = s_9 + s_{10};$$

$$S_{15} = s_8 + s_{11}, s_{16} = s_8 + s_{12}, s_{17} = s_9 + s_{13}, A_0 = m_0;$$

$$A_1 = s_{14}, A_2 = s_{15}, A_3 = s_{16}, A_4 = s_{17}. \qquad \text{(A-2)}$$

Hence, from Eq. (A-2), the total number of multiplications and additions needed to perform a 7-point transform is 5 and 17, respectively.

References

1. Reed, I. S., Truong, T. K., Miller, R. L., and Benjauthrit, B., "Further Results on Fast Transforms for Decoding Reed-Solomon Codes over $GF(2^m)$ for $m = 4, 5, 6, 8$," in *The Deep Space Network Progress Report 42-50,* pp. 132-154, Jet Propulsion Laboratory, Pasadena, Calif., Jan. 15, 1979.
2. Berkekamp, E. R., *Algebraic Coding Theory,* New York, McGraw Hill, 1968.
3. Winograd, S., "On Computing the Discrete Fourier Transform," *Proc. Nat. Acad. Sci. USA,* Vol. 73, 1976, pp. 1005-1006.
4. Reed, I. S., and Truong, T. K., "Fast Mersene-Prime Transforms For Digital Filtering," *Proc. IEE*, Vol. 125, No. 5, May 1978, pp. 433-440.