# Software Implementation of Binary BCH Decoder Using Microcontroller

**Lect. Mohammed Kasim Al-Haddad**
**Electronics and Communication Department**
**College of Engineering - Baghdad University**
**mkmih12@gmail.com**

## ABSTRACT:

In this paper a decoder of binary BCH code is implemented using a PIC microcontroller for code length n=127 bits with multiple error correction capability, the results are presented for correcting errors up to 13 errors. The Berkelam-Massey decoding algorithm was chosen for its efficiency. The microcontroller PIC18f45k22 was chosen for the implementation and programmed using assembly language to achieve highest performance. This makes the BCH decoder implementable as a low cost module that can be used as a part of larger systems. The performance evaluation is presented in terms of total number of instructions and the bit rate.

**Key Words: BCH code, Berlekamp-Massey algorithm, Chien search, PIC microcontroller.**

## التنفيذ البرمجي لفك ترميز شفرة BCH الثنائية باستعمال المسيطر الدقيق

**م. محمد قاسم محمد الحداد**
**قسم الهندسة الالكترونية والاتصالات**
**كلية الهندسة – جامعة بغداد**

## الخلاصة

في هذا البحث تم تنفيذ منظومة لفك ترميز BCH الثنائية باستعمال المسيطر الدقيق نوع PIC لطول ترميز يساوي 127 مرتبة ثنائية مع قدرة على تصحيح اخطاء متعددة، وقد تم عرض النتائج لتصحيح الاخطاء يصل عددها حتى 13 خطأ. تم اختيار خوارزمية بيركيلام-ميسي في عملية فك الترميز لكفاءتها. تم اختيار المسيطر الدقيق PIC18f45k22 للتنفيذ وقد تمت برمجته بواسطة لغة التجميع للحصول على اعلى اداء. وهذا يجعل فك ترميز BCH قابل للتنفيذ كوحدة بنائية واطئة الكلفة وممكن استعمالها كجزئ من انظمة اكبر. وكان تقيم الاداء عن طريق استعراض عدد التعليمات الكلي ومعدل الترميز.

**الكلمات الرئيسية: ترميز BCH، خوارزمية بيركيلام-ميسي، بحث تشين، المعالج الدقيق PIC.**

## 1. INTRODUCTION

Bose-Chaudhuri-Hocquenghem (BCH) codes are an important subclass of the linear cyclic block codes that can be considered as a generalization of Hamming codes. Unlike Hamming codes, BCH codes has multiple error correction capability. There are efficient decoding algorithms for the BCH codes due to their special algebraic structure، **Jiang, 2010**. Two important subclasses of the BCH Codes are the Reed-Solomon Codes, which are nonbinary and the binary BCH codes. This paper is about the binary BCH code.

**Chu** and **Sung, 2009** presented a technique for improving Chien search base on Constant Galois Field Multiplication (CGFM) where a multiplication by a constant is implemented by shift register. Multiplications can be performed in a single shift operation if proper lookup tables are used. Higher speed performance can be achieved by larger lookup tables. **Taki El-Din *et al*., 3013** proposed an algorithm for detecting the undetectable received codewords i.e. when the number of errors is greater than the design error correction capability $t$. This is achieved by transforming the error locating polynomial to another polynomial whose coefficients must satisfy certain conditions. If the conditions are satisfied, the decoder can proceed to Chien search and correct the received codeword, otherwise, the Chien search is skipped and decoding failure is declared and repeat request is transmitted. **Schipani *et al.*, 2011** made a comparison study of the different decoding algorithms of the generalized nonbinary BCH codes with the binary BCH codes as a special case. Schipani proposed an algorithm for syndrome evaluation that is based on rearranging the received codeword polynomial as a linear combination of polynomials with special grouping of powers of $x$. This arrangement reduces the number of multiplications over the direct evaluation of the received codeword polynomial. **Lin *et at*., 2010** proposed a hardware-based soft decision decoding algorithm of BCH codes where the syndrome calculations and error location stages are processed in parallel. Simulation was for the $t$=2 error correcting code (255,239,2), Lin suggested that the proposed algorithm can correct at most $2t$+1 errors. **Wu, 2008** presented a modification to the Berkelamp-Massey algorithm that can produce a list of candidates of correct codewords (list decoding) for both Reed-Solomon and BCH codes where the number of correctable errors of these codes is extended beyond their design error correcting capability.

## 2. MATHEMATICAL BACKGROUND

Finite fields especially Galois Field (GF) play an important role in coding theory, here is a brief introduction. A finite field has a finite number of elements, their number is either a prime $p$ or a power of a prime q=$p^m$. If $p$=2, the GF field is referred to by GF($2^m$), a special but yet important case where $m$=1 which yields GF(2) has the only two elements 0 and 1. There are to operations defined over the field; addition (+) and multiplication ($\cdot$). Other than number of elements being finite all properties if fields apply, like closure, commutative and associative operations, the identity and inverse elements of addition and multiplication. The elements of a GF($2^m$) are defined as remainder obtained when dividing the powers of $x$ by a generator polynomial $g(x)$ whose coefficients $\in$ GF(2), i.e. 0 or 1, the roots of $g(x) \in$ GF($2^m$). A polynomial with coefficients $\in$ GF(2) is called a binary polynomial. A generator polynomial must be a binary polynomial and irreducible that is it cannot be factored into two or more binary polynomials. Such a polynomial is also called a primitive polynomial and its root $\alpha$ is called primitive element because all elements can be represented in terms of powers of $\alpha$. The above can be stated mathematically as

$$x^i = s(x)g(x) + r(x) \qquad \deg(r(x))<\deg(g(x)) \tag{1}$$

Equation (1) tells us simply that $x^{i}$ (considered as a polynomial) when divided by $g(x)$, the remainder $r(x)$ represents the element of the GF($2^m$) field in polynomial form. **Table 1.** shows an example of GF($2^3$) field generated using the primitive polynomial $g(x)=x^3+x+1$. **Table 1.** also shows the two forms representing the GF elements, one is the power of $\alpha$, here the element 0 cannot be represented as an integer power of $\alpha$, so, it is represented as $\alpha^{-\infty}$ by convention. The second representation is by the binary polynomial $r(x)$. The third representation will be explained later in system implantation section. The addition operation is performed by adding the polynomial form of the elements where the coefficients are added modulo-2 for example: $\alpha^3 + \alpha^4 = (x+1)+(x^2+x) = x^2+1 = \alpha^6$. Note that modulo-2 addition implies that addition is the same as subtraction i.e., $x+x=x-x=0$. Multiplication is performed in the power for with $\alpha^{q-1}=1$ where $q=2^m$, for example: $\alpha^4\alpha^5 = \alpha^9 = \alpha^7\alpha^2 = 1\cdot\alpha^2 = \alpha^2$. From the above it is clear that every element is the additive inverse of itself and the multiplicative inverse of $\alpha^i$ is $\alpha^{q-1-i}$for more details and proofs of claims stated above the reader may refer to references **Justesen, Høholdt, 2004** and **Jiang, 2010**. The elements of a GF field can be generated by hardware using a linear feedback shift register (LFSR), **Jiang, 2010**, as shown in **Fig. 1**, where the branches represent the coefficients of $g(x)$, the contents of the LFSR can be initialized by any element other than 0. A single clock to the LFSR represents a multiplication by $x$ when the LFSR overflows; $g(x)$ is subtracted from the content. Clocking $j$ times is equivalent to multiplication by $x^j$. One more thing that needs to be addressed before leaving this section is the minimal polynomials. A minimal polynomial $\phi_i(x)$ of element $\alpha^i$ is a binary irreducible polynomial with $\alpha^i$ as its root. The polynomial $\phi_1(x)=(x+\alpha)(x+\alpha^2)(x+\alpha^4)=x^3+x^2+1$ is a minimal polynomial of $\alpha$ because it is binary, irreducible and has $\alpha$ as root, and there is no binary polynomial of degree less than 3 with $\alpha$ as a root, the polynomial $x+\alpha$ is not minimal although it is of degree 1 because it has the coefficient $\alpha$ which is not a binary value. The roots of $\phi_1(x)$: $\alpha$, $\alpha^2$, $\alpha^4$ are called conjugates.

## 3. BINARY BCH CODES

Generalized BCH codes are of length $n=q^v-1$ where $q$ is a power of prime and $v$ is positive integer. Two important subclasses of the generalized BCH codes are the binary BCH codes when $q=2$ and the Reed-Solomon (RS) codes when $v=1$, **Moreira** and **Farrell, 2006**. In this paper we are interested in the binary BCH codes, which can be described as follows; for $m\geq3$ and $t<2^m-1$ there exist a binary BCH code C($n,k,t$) with the following properties:

| | |
|---|---|
| 1- Code length: | $n=2^m-1$ |
| 2- Number of parity bits: | $n-k\leq mt$ |
| 3- Minimum hamming distance: | $d_{min}\geq2t+1$ |
| 4- Error correcting capability: | $t$ errors in a codeword |

The generator polynomial $g(x)$ of binary BCH code with above parameters is the minimum-degree binary polynomial that has $\alpha$, $\alpha^2$, $\alpha^3$,...., $\alpha^{2t}$ as roots, i.e.

$$g(\alpha^i) = 0 \qquad i = 1,2,3,\cdots,2t \tag{2}$$

The binary polynomial that satisfies Eq. (2) can be described in terms of the minimal polynomials of the roots $\alpha^i$ as

$$g(x) = \text{LCM}(\phi_1(x), \phi_2(x), \cdots, \phi_{2t}(x)) \tag{3}$$

Where LCM(*a*,*b*) is the least common multiple of *a* and *b*. It can be shown that GF elements $\alpha^i$ and $\alpha^{2i}$ are both conjugates, that is they are both roots of the same minimal polynomial. Therefore the even indexed minimal polynomials can be removed and Eq. (3) becomes

$$g(x) = \text{LCM}(\phi_1(x), \phi_3(x), \cdots, \phi_{2t-1}(x)) \tag{4}$$

Hamming codes are special case of BCH codes with $g(x) = (\phi_1(x))$ and $t=1$. If the massage *k*-bits are represented in a polynomial form

$$m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1} \tag{5}$$

The BCH code can be generated by multiplying $m(x)$ by the generator polynomial

$$c(x) = m(x) g(x) \tag{6}$$

This is a nonsystematic encoding, for the systematic encoding the codeword $c(x)$ is given by Eq. (7), **Moreira** and **Farrell, 2006**

$$c(x) = x^{n-k} m(x) + p(x) = q(x) g(x) \tag{7}$$

Where, $p(x)$ and $q(x)$ are the remainder and quotient when dividing $x^{n-k}m(x)$ by $g(x)$. In both equations (7) & (8) the codeword $c(x)$ is a multiple of the generator polynomial $g(x)$, this means that the roots of $g(x)$ (Eq. (2)) are also roots of $c(x)$, i.e.

$$c(\alpha^i) = c_0 + c_1\alpha^i + c_2\alpha^{2i} + \cdots + c_{n-1}\alpha^{i(n-1)} = 0 \qquad i = 1, 2, \cdots, 2t \tag{8}$$

These 2*t* equations can be written in matrix form

$$[c_0 \quad c_1 \quad c_2, \ldots, c_{n-1}] \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \cdots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix}^T = CH^T = \mathbf{0} \tag{9}$$

Equation (9) gives the parity check matrix *H*. It has been mentioned earlier that if $\alpha^j$ is a root of one of the minimal polynomials in Eq. (4) and hence *g*(*x*), then its conjugates are also roots, these conjugates appear in the even rows of Eq. (9), so, these rows can be removed from *H*

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \cdots & (\alpha^3)^{n-1} \\ 1 & \alpha^5 & (\alpha^5)^2 & \cdots & (\alpha^5)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{2t-1} & (\alpha^{2t-1})^2 & \cdots & (\alpha^{2t-1})^{n-1} \end{bmatrix} \tag{10}$$

where, Each element of *H* is a GF($2^m$) element which can be represented by a vector of *m* bits.

## 4. DECODING OF BINARY BCH CODES

The process of decoding BCH codes is described by the block diagram shown in **Fig. 2**. It consists of 3 main stages; the first is the syndrome calculations, the second is to use these syndrome values to find a polynomial whose roots represents the error locations, the third step is to solve the error locating polynomial to find the error positions. Since we are dealing with a binary code (unlike RS codes) there is no need for an error evaluation step as the error values in the binary case are always 1's. These 3 main blocks are describes next.

### 4.1 Syndrome Calculations

Like any other block code, the received word and the error pattern are related in polynomial form by

$$r(x) = c(x) + e(x) \tag{11}$$

Since $CH^T$=0, the syndrome can be calculated by

$$S_j = r(\alpha^j) = \sum_{i=0}^{n-1} r_j \alpha^{ij} \qquad j = 1, 2, \cdots, 2t \tag{12}$$

Equation (12) involves calculating the syndrome values either using polynomial evaluation over GF($2^m$) field or using matrix multiplication as in

$$S = RH^T \tag{13}$$

where, *R* is the received word as a vector. There is another way of calculation which involves polynomial division. The received word as a polynomial is divided by the minimal polynomials and the remainder polynomials are evaluated for the values of the roots $\alpha^j$ as below

$$r(x) = a_j(x)\phi_j(x) + b_j(x) \tag{14}$$

Recall that $\alpha^j$, $j$=1,2,…, 2$t$ are roots of $\phi_j(x)$ therefore Eq. (12) becomes

$$S_j = r(\alpha^j) = b_j(\alpha^j) = \sum_{i=0}^{m-1} b_j \alpha^{ij} \tag{15}$$

The difference between Eq. (12) and Eq. (15) is that Eq. (12) is about evaluating the polynomial $r(x)$ which of degree $n$-1 while Eq. (15) is about evaluating the polynomials $b_j(x)$ which of degree $m$ which is less than the order of $r(x)$. But in order to obtain the polynomials $b_j(x)$, 2$t$ divisions must be performed in order to obtain the 2$t$ polynomials $b_j(x)$. One last thing is that in binary case Eq. (12) and Eq. (15) can be used to calculate the syndromes of odd indices and the syndromes of even indices can be obtained using the equation below which applies for any binary polynomial, **Moreira** and **Farrell, 2006**

$$S_{2i} = (S_i)^2 \tag{16}$$

### 4.2 Error Locating Polynomial

The error locating polynomial is of the form

$$(1-\beta_1 x)(1-\beta_2 x)\cdots(1-\beta_v x) = 1 + \sigma_1 x + \cdots + \sigma_v x^v = 0 \tag{17}$$

Where $\beta_j$, $j$=1, 2,…, $v$ are GF($2^m$) elements that represent the error location through their power representation, i.e.

$$\beta_j = \alpha^{i_j} \tag{18}$$

The value of $i_j$ in Eq. (18) is the error location in the codeword vector $R$. The roots of Eq. (17) are the multiplicative inverses of $\beta_j$. The value of $v$ is the order of the equation and the number of errors and it is unknown. There are different algorithms for finding the error locating polynomial

1- Peterson's Algorithm
2- Euclidean Algorithm
3- Berlekamp-Massey (BM) Algorithm

There are other algorithms which are beyond the scope of this work. These 3 algorithms share the common ground that the syndrome values and the coefficient of the error locating polynomial are related through the so-called Newton's identity

$$\begin{aligned}
&S_1 + \sigma_1 = 0\\
&S_2 + \sigma_1 S_1 + 2\sigma_2 = 0\\
&S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 2\sigma_2 = 0\\
&S_v + \sigma_1 S_{v-1} + \sigma_2 S_{v-2} + \cdots + \sigma_{v-1} S_1 + v\sigma_v = 0\\
&S_{v+1} + \sigma_1 S_v + \sigma_2 S_{v-1} + \cdots + \sigma_{v-1} S_2 + \sigma_v S_1 = 0
\end{aligned} \tag{19}$$

$$S_{v+2} + \sigma_1 S_{v+1} + \sigma_2 S_v + \cdots + \sigma_{v-1} S_3 + \sigma_v S_2 = 0$$
$$S_{2t} + \sigma_1 S_{2t-1} + \sigma_2 S_{2t-2} + \cdots + \sigma_{v-1} S_{2t-v+1} + \sigma_v S_{2t-v} = 0$$

In the binary case the term $v\sigma_v$ reduces to 0 when $v$ is even or 1 when $v$ is odd. In the above equations the syndrome values $S_i$ are given and the coefficients $\sigma_i$ are the unknowns. Peterson's algorithm solves these linear equations for $\sigma_i$. Although Peterson's algorithm seems straight forward but it becomes very complex in terms of amount of computations for large number of errors, especially the degree of the error locating polynomial $v$ which is the number of actual errors is unknown, so the algorithms has to start with the maximum number of correctable errors allowed by the design i.e., $v=t$ and go backwards by reducing the order $v$ until the actual number of errors is reached. For details the reader can refer to **Jiang, 2010**. The Euclidean algorithm is a general algorithm with many applications. Here the Euclidean algorithm can be applied to find the so-called key equation given by

$$\sigma(x)S(x) - \mu(x)x^{n-k} = -W(x) \tag{20}$$

The algorithm involves repeated polynomial divisions, it starts with $x^{n-k}$ and $S(x)$ as inputs, First $x^{n-k}$ is divided by $S(x)$ then $S(x)$ is divided by the remainder of the first step then the remainder of the first step is divided by the remainder of the second step and so on recursively. The algorithm stops when the remainder of the last step which becomes $W(x)$ has a degree such that

$$\deg(W(x)) \leq \lfloor (n-k)/2 \rfloor + 1 \tag{21}$$

Although the Euclidean algorithm is more efficient than the Peterson's algorithm in terms of amount of calculations but the Berlekamp-Massey algorithm is preferred over the Euclidean algorithm because the Euclidean algorithm starts with polynomials of degrees larger than the actual degree of the error locating polynomial $\sigma(x)$ and goes down till the actual degree of $\sigma(x)$ is reached, While BM Algorithm starts with polynomial of order 1 and goes up to the actual degree of $\sigma(x)$ and this makes it more efficient than the Euclidean algorithm. Since in this work, the BM algorithm is adopted in the implantation of the decoder, this algorithm is presented with more details.
The idea of BM algorithm depends on the Newton identity given by Eq. (19) where the $\sigma(x)$ is suppose to satisfy the Newton identity for syndrome values up to $S_{2t}$. in the beginning $\sigma(x)$ is initialized by

$$\sigma^{(1)}(x) = 1 + S_1 x,\ B^{(1)}(x) = S_1^{-1},\ v^{(1)} = 1, j=2$$

Where $B^{(j)}(x)$ is the correction polynomial and the superscript in parentheses is the iteration count. Newton identity in Eq. (19) is used to estimate the next syndrome value

$$\tilde{S}_j = \sum_{i=1}^{v_{j-1}} \sigma_i^{j-1} S_{j-i} \tag{22}$$

The discrepancy between the estimated value of syndrome and the actual one is calculated

$$\Delta^{(j)} = S_j - \tilde{S}_j \tag{23}$$

If the value of $\Delta^{(j)}=0$ then $\sigma^{(j)}(x)$ left unchanged, but $B^{(j)}(x)$ is updated as below

$$B^{(j)}(x) = xB^{(j-1)}(x) \tag{24}$$

If $\Delta^{(j)}\neq 0$, $\sigma^{(j)}(x)$ is updated as below

$$\sigma^{(j)}(x) = \sigma^{(j-1)}(x) - \Delta^{(j)} xB^{(j-1)}(x) \tag{25}$$

In addition to the condition $\Delta^{(j)} \neq 0$, if the condition $2v^{(j-1)}<j-1$ is satisfied both $B(x)$ and $v$ are updated as in Eq. (26) and Eq. (27) below otherwise $B^{(j)}(x)$ is updates by Eq. (24) and $v$ is kept the same

$$B^{(j)}(x) = (\Delta^{(j)})^{-1} \sigma^{(j-1)}(x) \tag{26}$$

$$v^{(j)} = j - v^{(j-1)} \tag{27}$$

The process is repeated 2$t$ steps. The block diagram of BM algorithm is shown in **Fig. 3.** The reader may find that this block diagram and variable initialization are different than what is found in literature, this is because the block diagram has been modified in a way to be suitable for efficient programming. So, Basically **Fig. 3** shows the flowchart of the actual implemented program for decoding binary BCH codes in this work.

### 4.3 Error Position Locating and Chien Search

The final step of the decoding is to find the roots of the error locating polynomial and from the roots values, the error position will be found. The algorithm used to find the roots of a GF polynomial is Chien search, **Jiang, 2010**. Chien search is simply to evaluate the GF polynomial over all the elements of the GF($2^m$) field. Recursive calculation is performed in order to make the search efficient. Suppose that the error locating polynomial $\sigma(x)$ is evaluated for the GF element $\alpha^i$

$$\sigma(\alpha^i) = 1 + \sigma_1\alpha^i + \sigma_2\alpha^{2i} + \cdots + \sigma_t\alpha^{vi} \tag{28}$$

In the next iteration $\sigma(x)$ is evaluated for the next GF element $\alpha^i$

$$\sigma(\alpha^{i+1}) = 1 + \sigma_1\alpha^{i+1} + \sigma_2\alpha^{2(i+1)} + \cdots + \sigma_t\alpha^{v(i+1)}$$
$$\sigma(\alpha^{i+1}) = 1 + \sigma_1\alpha^i\alpha^1 + \sigma_2\alpha^{2i}\alpha^2 + \cdots + \sigma_t\alpha^{vi}\alpha^v \tag{29}$$

So, by defining each term of the $i^{\text{th}}$ iteration of Eq. (28) by

$$\sigma_j^{(i)} = \sigma_j \alpha^{ji} \tag{30}$$

These terms will be updated in the next iteration by

$$\sigma_j^{(i+1)} = \sigma_j^{(i)} \alpha^{j} \tag{31}$$

With initial values equal the coefficients of $\sigma(x)$ i.e. $\sigma_j^{(0)} = \sigma_j$. Recalling Eq. (17), the error positions are obtained from the multiplicative inverses of the roots of $\sigma(x)$, so, if the roots were found as $\alpha^{i_1}$, $\alpha^{i_2}$,…, $\alpha^{i_v}$, the error positions will be at $2^m$-1-$i_1$, $2^m$-1-$i_2$,…, $2^m$-1-$i_v$. In other words if $\alpha^{i_k}$ is a root, then the error position $ep_k$ will be given by

$$ep_k = 2^m - 1 - i_k \tag{32}$$

## 5. THE PIC18F45K22 MICROCONTROLLER

The PIC18F45K22 is one of the PIC18(L)F4X/5XK22 family that has many features suitable for many applications. In this section a brief overview is presented about its features that are relevant to application of this paper. One of the most attractive features of the PIC18F45K22 is that its price is around 3$ and the system of binary BCH decoder is entirely implemented by this microcontroller without any external hardware. The PIC18F45K22 will be referred to throughout this paper by (PIC) for simplicity. The PIC has a 16-bit wide instructions and 8-bit wide data. It can operate up to 64MHz clock frequency using internal multiply-by-4 PLL. Each instruction is executed by 4 clock cycles that makes the instruction cycle execution frequency 16MHz which is referred to as Million Instructions Per Second (MIPS). The Program memory is 32kB (16k instructions), the data memory is 1536 bytes SRAM, this RAM contains both special Function Registers (SFR) and General Purpose Registers (GPR). FSR's are used for control and status of the controller and peripheral functions, GPR used for general data storage for user applications. The RAM address range is from 0 to 4k addressable with 12-bit data address bus. This address range is divided to 16 banks of 256B. The FSR's are located at the end of the addressable range F38h-FFFh. The GPR are implemented in the first 6 banks 000h-5FFh, the remaining addressable range is not implemented. When a location in the RAM is to be addressed, the lower 8 bits are specified in the instruction addressing the byte location within a specific bank, the higher 4 pits should be loaded in the Bank Select Register (BSR), a special function register, to specify one bank out of the 16 banks. There is an alternative way to address part of the RAM called the Access RAM. The Access RAM contains the first 96 bytes of the RAM (00h-5Fh) in bank 0 and the last 160 bytes of the RAM (60h-FFh) in bank 15. The combined 256 bytes are treated as one bank (Access Bank) and the BSR is ignored. This addressing mode is specified by an operand 'a' in the instruction, if a=0, access bank is used and the content of BSR is ignored, if a=1 the bank is specified by the BSR contents.
Another mode of RAM addressing is the indirect mode. Indirect addressing allows the user to access a location in data memory without giving a fixed address in the instruction. This is done by using File Select Registers (FSR's) as pointers to the locations which are to be accessed. Indirect addressing is implemented through either of 3 registers FSR0, FSR1, FSR2, each is a pair of 8-bit registers, FSRnL and FSRnH. Each FSR holds the 12-bit address value, so, the upper 4 bits of the register is not used. In a given instruction, the memory location is accessed by indirect mode using

the operands INDF0 through INDF2. These are thought of as virtual registers, the actual register is defined by the FSRn contents. In the implemented program indirect addressing is widely used along with access bank mode. Direct addressing using the BSR is never needed within the program core.

## 6. SYSTEM IMPLEMENTATION

The implemented system was designed to correct binary BCH codes of length $n$=127 bits for any value of t, only the results shown at the end are for up to $t$=13 errors. The programming was made by assembly language using MPLAB IDE V8.70. The start is to find a GF generator polynomial, and since $n$=127 the generator polynomial should be of degree $m$=7. The value of $m$ also represent the number of bits required to represent the GF($2^m$) elements. There are tables for GF generator polynomials as in **Justesen** and **Høholdt, 2004** and **Lin** and **Costello1, 1983**, the GF generator polynomial that is commonly used is

$$g(x) = 1 + x^3 + x^7 \tag{33}$$

One important issue that needs to be addressed is that which representation we should use for the GF elements? As has been discussed in **section 2**, there are two representations of the GF elements; one is the polynomial representation and the other is the powers of primitive element, see **Table 1**. Each has its advantages and disadvantages and these are discussed in the following

1- The polynomial representation: Advantages is that all $2^m$ elements can be represented in binary without special treatment and the addition is simply bitwise XOR. Disadvantage is that the multiplication is not straightforward; it can only be implemented if one of the multiplicands is in the power form and the other operand is in polynomial as described in **section 2**. Since the power of the elements has a maximum of $2^{m-2}$, this means that there are cases where $2^m$-2 (126 in our case) sub-operations may be needed for a single multiplication. This is not practical and therefore the polynomial representation is not used in the system.

2- The power representation: Advantages is that the multiplication is made by addition of powers mod $2^{m-1}$, which can be implemented by few steps. Disadvantages are that the addition is not possible and requires lookup table and since there are two operands the lookup table will be of $2^m$X$2^m$ dimension and this is not practical. The other disadvantage is that the zero has no integer power representation, therefore special treatment is required.

3- The modified representation: It is decided to adopt a representation close to the power representation where the powers are represented by adding 1 to the actual power and the GF element 0 is represented by 0. In this way all elements are represented by 0 through $2^m$-1. The addition will be implemented by converting the power representation of the operand to polynomials and converting the result to power (if needed). The conversion from power to polynomial and vice versa is implemented by lookup tables, so, there will be 2 size $2^m$=128 lookup tables.

**Table 2.** shows the memory organization of the program data, the first four entries of the table are with the range of 0h-60h to make it possible to access them by access bank mode (without specifying the BSR content) this is useful for reducing program instructions especially these contents are accessed frequently within loops in the program. The lookup tables are located each in a separate bank for ease of access, where each table starts at address 0 within the respective bank. The received codeword although contains 127 bits but is stored in 127 bytes where each bit is stored in a separate memory location. This might seem like a waist of memory but it is actually not. If the received codeword is stored compactly in 16 bytes, it would be very difficult and time consuming (in terms of number of instructions) to access an individual bit. While storing each bit in a separate memory location (byte) makes it much easier to access the individual bits, especially accessing the individual bits is needed in the syndrome calculations and Chien search, which are time consuming parts of the program.

### 6.1 Implementation of Syndrome Calculations

The syndrome calculation has been discussed in **Section 4.1** and it was mentioned that direct calculation involves evaluation of polynomials, Eq. (12), of higher degree than the remainder polynomials given by Eq. (15). Because the method of Eq. (15) requires *t* polynomial divisions in advance it cannot be considered better than the direct calculation method unless parallelism is adopted in the implementation which is possible by hardware implementation, where the *t* polynomial divisions can be implemented by *t* separate LFSR's. Since the implantation of the system in this work is a software implantation, the direct calculation is adopted with the following considerations:

1- All calculations are made by using the polynomial representation so there will be no need for conversion from one form to the other.

2- The values of $\alpha^{ij}$ are calculated in advance in polynomial form using MATLAB and stored in a lookup table in the program memory because of its large size.

3- Only the syndromes of odd indices are calculated using Eq. (12) and the syndromes of even indices are calculated using Eq. (16).

**Figure 4.** shows a flowchart used to implement the syndrome calculation. The values $\alpha^{ij}$ are the elements of the parity check matrix given by Eq. (10) which are stored as a table in the end of the program memory in polynomial form. These values are addressed by the table pointer register TBLPTR which is a 3-byte register. The content of the table is read by the instruction TBLRD*+ that reads the content of the table into a memory register TABLAT which is an SFR and at the same time increments the contents of TBLPTR making it ready for the next read instruction. Since the received codeword $C$ is binary, the multiplication $c_i\alpha^{ij}$ of Eq. (9) is simple where the value of $c_i$ is checked for zero value, if so, nothing is done; otherwise the value of $\alpha^{ij}$ is used to calculate $S_j$. When the inner loop is completed the value of $S_j$ is obtained in polynomial form and since the calculations of the next stage of the BM algorithm requires that the values of syndromes to be in the power form the value of $S_j$ is converted immediately to power form. This is made easily by lookup table that is stored in the RAM as in **Table 2**. The lookup table used to convert from polynomial form to power form is called GFINV. The polynomial form of $S_j$ is stored in the FSR2L register with FSR2H loaded with a fixed value of 2 which is the bank address of GFINV table. The contents of the FSR2 is retrieved and stored in place of the old $S_j$, and the value of $S_j$ becomes in power form. The counter of the outer loop is incremented by 2 to calculate the odd indexed syndromes. The even indexed syndromes are later calculated using Eq. (16) which is much faster than the direct calculations.

### 6.2 Implementation of BM Algorithm

Before going into the details of implementing the BM algorithm, the operation of GF elements multiplication is considered first because there are few details that need to be considered. It has been mentioned before that the multiplication in power form is easier than in polynomial form because it is basically addition of powers mod $2^m$-1. An algorithm presented by **Deschamps *et al.*, 2009** for

mod *n* addition that is suitable for hardware implementation. Since the addition used in this work is mod $2^m$-1 and mod *n* is more general, the algorithm is slightly modified for mod $2^m$-1 addition to make it easier than the mod n addition as shown in **Fig. 5**. In this algorithm the two integers *a* and *b* are added (which are both represented by *m* bits) and the result, *c*, is incremented by 1, then the result is tested for overflow, if $c<M=2^m$ (no overflow) then the addition without incrimination is the correct result so *c* is decremented, otherwise when there is an overflow and the *m*th bit is set then the *m*th bit of *c* is simply cleared and the algorithm terminates. It can be seen that the clearing of the *m*th bit is common for both sides of the condition this does not affect the other branch of the condition because the *m*th bit is already 0. The test of the *m*th bit is simply made using the instruction (BTFSS f,b,a) or (BTFSC f,b,a), where, the bit whose address (b (0-7)) within the file register (f) is tested if set (or clear) the next instruction is skipped. The operand a=0,1 is for the memory access mode. The clearing of a specific bit (b) of a file register (f) is cleared by the instruction BCF f,b,a. The mod addition algorithm is not implemented in the way described above because the powers of the GF elements are stored as $i$+1 rather than $i$. This requires the algorithm to be further modified as in **Fig. 6**. This algorithm is used throughout the main program whenever is needed and it is first used for evaluation of the even indexed syndromes using Eq. (16).
Now some details about BM algorithm implementation is presented by providing more insight for the main blocks of the algorithms block diagram shown in **Fig. 3**. The first quantity to calculate is the estimate of the *j*th syndrome as in Eq. (22), see **Fig. 7**. Here a temporary variable ($T$) is assigned the value of $\sigma_i S_{j\text{-}1}$ obtained as a multiplication of two GF elements which are represented by the alternative power form, so, the algorithm of mod addition shown in **Fig. 5** is used. Before carrying out the multiplication as mod addition of powers, the values of $\sigma_i$ and $S_{j\text{-}1}$ are checked if either one equals to 0. If so, the whole multiplication and addition is skipped and a new iteration is started. Since the value of $T$ should be added to $\tilde{S}_j$, it should be converted to polynomial form in a way similar to what has been described earlier using lookup table. Here the lookup table used is GF lookup table where the value of $T$ is used as an address stored in FSR1 and the value pointed by that address is retrieved as the value of $T$ in polynomial form. Now the addition is performed as a simple XOR of the two quantities. Next the discrepancy $\Delta^{(j)}$ is calculated as in Eq. (23) which also requires the conversion of $S_j$ to polynomial form.
The implementation of the block related to Eq. (25) is separated into two parts; the first is to calculate the discrepancy part (denoted as $D(x)$) in a temporary location

$$D(x) = \Delta^{(j)} x B^{(j-1)}(x) \tag{34}$$

The Flowchart of implementation of Eq. (34) is shown in **Fig. 8**, here the coefficient $B_i$ of the term $x^i$ is multiplied by $\Delta^{(j)}$ and assigned for $D(x)$ as the coefficient $D_{i+1}$ of the term $x^{i+1}$. This is obvious because of the term $x$ in the expression of Eq. (34). Of course the coefficient $B_i$ should be checked for zero value as explained earlier. Later the two terms of $D(x)$ and $\sigma^{(j\text{-}1)}(x)$ are added to update $\sigma^{(j)}(x)$. No special treatment is required except for changing the coefficient of $D(x)$ and $\sigma^{(j\text{-}1)}(x)$ to polynomial form for addition and back to power form using GF and GFINV lookup tables. One last point is that the block of Eq. (25) appears twice in **Fig. 3**. A natural way to do it is to position it before the ($2v^{(j\text{-}1)}$-$j$<0) condition test and on the (No) branch of the ($\Delta^{(j)}$=0) condition test. This is not suitable for in-place calculation of $\sigma^{(j)}(x)$, since updating $\sigma^{(j\text{-}1)}(x)$ to $\sigma^{(j)}(x)$ would cause loosing the value of $\sigma^{(j\text{-}1)}(x)$ that is needed for the $B^{(j)}(x)$ calculation (as in Eq. (26)) which is implemented in a later stage. The block of Eq. (26) is simply implemented by multiplying the multiplicative inverse of $\Delta^{(j)}$ by $\sigma^{(j-1)}(x)$. The multiplicative inverse $(\Delta^{(j)})^{-1}$ is obtained easily when $\Delta^{(j)}$ is in power form ($\Delta^{(j)}=\alpha^k$) as $(\Delta^{(j)})^{-1}=(a^k)^{-1}=a^{q-1-k}$ where $q=2^m-1$. The quantity $q$-1-$k$ is simply the 1's complement of $k$. Since the $\Delta^{(j)}$ is represented in the alternative power form, where the stored value is the power plus 1, a decrement is needed before the binary inversion. Implementation of Eq. (24) is only a shifting of coefficients and no calculations are involved. The remaining blocks of the BM algorithm flowchart are straight forward and there is no need to elaborate.

### 6.3 Implementation of Chien Search

The efficient implementation of Chien search described earlier can be implemented by updating the coefficients of the error locating polynomial $\sigma_j$ recursively for each iteration as in Eq. (31). Where, $i$th iteration represents the evaluation of $\sigma(x)$ for $\alpha^i$. This is faster than the direct evaluation using Eq. (28) and it suitable for in-place calculations because the initial coefficients are no longer needed. The flowchart of Chien Search is shown in **Fig. 9**. The product part is implemented using the alternate power for of the GF elements and since this product is on one hand needed for the other iterations, and on the other hand is required to be converted polynomial form for summation part, a temporary variable $T$ is used to convert to the polynomial form. For practical considerations the evaluation of $\sigma(x)$ for the element 1 (i=0) is made separately outside the loop where the coefficients $\sigma_j$ are simply summed and the result is check if equal to zero. When all of the roots of $\sigma(x)$ are found before reaching the final value of the counter $i$, the process of the Chien search can be terminated to save the extra unnecessary computations time as suggested by **Wu, 2004**. For each root found, the error position is found as in Eq. (32) which is simply the 1's complement of the roots power representation.

## 7. PERFORMANCE EVALUATION

In general the implementation of decoding algorithms can be made either by hardware or software. Hardware implementation is advantageous in terms of speed so higher bit rates can be accomplished, but on the other hand its cost is high because of the amount of hardware components used. The software implementation is better in terms of cost because it needs only a processor and possibly some input-output peripherals. Fortunately the PIC microcontrollers became a very cost effective engineering solution because of their high performance and low cost with on chip input-output peripherals. The implantation of the decoder being entirely software running by a PIC microcontroller makes the decoder cost only the market price of the used microcontroller (PIC18F45K22) which is around 3$. The disadvantage is that the speed performance is lower than the direct hardware implementation because of the sequential execution of instructions and the limitation of clock frequency. Now the performance analysis of the implemented system will be given in terms of number of instructions cycles for the three main blocks of the system: syndrome calculations, BM algorithm and Chien Search, for different values of the decoding capability $t$. The code length is fixed as $n$=127, and the values of the number of correctable errors $t$ is varied from 4 to 13. The choice of the code parameters: $n$ (no. of code bits), $k$ (no. of message bits) and $t$ (no. of correctable errors) is not arbitrary and these values can be found in tables as in **Lin** and **Costello**, **1983**. **Table 3** shows the number of instructions for each component of the decoder. The first 8 rows of the table show the number of instructions for different values of $t$ with the number of actual errors is maximum i.e. $v$=$t$. In the next 7 rows $t$ is fixed to 13 and the number of actual errors is varied from

12 to 0. It can be seen that the syndrome calculations depends only on $t$ and not on the actual number of errors $v$, this is expected because syndrome calculations is made of two nested loops one of length $n$ and one of length $t$ which makes the number of instructions proportional to $nt$. The BM algorithm requires a number of instructions that depends on the number of actual errors occurred in the received codeword in spite of the fact that it has a fixed number of iterations ($2t$) but the different paths of the algorithm (see **Fig. 3**) have different amount of calculations. In general it has a number of calculations proportional to $t^2$ **Schipani *et al.*, 2011**. The number of instructions of Chien search depends not only on the number of actual errors $v$ (which represents the degree of the error locating polynomial) but also on the errors locations. Since the loop of Chien search is terminated when all the error locations are found (see **Fig. 9**). The number of instructions of Chien search is given as minimum and maximum, the minimum number of the loop iterations is $v$, this is a special case when all errors are found in the first $v$ iterations. The maximum number of iterations is when the Chien search loop has to go throw all the $n$-1 GF elements i.e. when the last error position is found by the last iteration. The next column of **Table 3** shows the total number of instruction of the three decoder components with the Chien search number of instructions taken as maximum. Finally the achievable bit rate $R_b$ in kilo bits per second (kbps) is given by the last column it is calculated as

$$R_b = nF / N_t \tag{35}$$

Where $F$ is the instruction cycle execution frequency which is in our case 16 MHz and $N_t$ is the total instructions of the decoder for decoding $n$ bits. It is important to note that the system bit rate $R_b$ is calculated based on the worst case when the number of actual errors is maximum i.e. $v$=$t$ and the Chien search iterations is maximum. Therefore, the values of $R_b$ given in **Table 3**, where $v \neq t$ is not realistic but it is shown for the sake of comparison especially the last entry where a correct codeword is received $v$=0. In this case (which is usually the most probable caser) the execution time is very small (the rate is high) because only the first syndrome value is needed to decide if the codeword is correct which is when $S_1$=0, there will be no need to calculate the other syndrome values and this is more easily seen from Peterson’s algorithm, **Jiang, 2010**.

As a final discussion, the increase of the bit rate is considered. To increase the bit rate, one possibility is to increase the system frequency and this can be done by using a higher level PIC microcontroller like PIC24 family where frequency can be increased up to 40 Mhz. This option is simple but still the outcome is limited. Another way is to use a sort of distributed system like the one described in **Fig. 10** where $N$ decoders are used in parallel and fed by a buffer that is filled from the incoming data stream which can be of bit rate equals $MR_b$ where $M$>1 is a factor, and $R_b$ is the operating bit rate of the individual decoder as given in **Table 3**.The input buffer feeds the individual decoders asynchronously, that is whenever a decoder is free, an $n$-bit block is fed to it. The output buffer is to rearrange the decoded messages according to their original sequence. It is assumed that the occurrence of the cases where $v$ is high and error positions requiring high number of iterations have a low probability. This arrangement should give the sufficient time for and individual decoder to decode up to the maximum time required by the worst case but also less overall time for the while $N$ decoders to decode $N$ codewords that are not necessarily require the maximum time of decoding. Of course, there can be situations where the cases of high decoding time can occur more than once in the buffer and the time imposed by the incoming bit rate ($MR_b$) is insufficient. This will create the situation when the input buffer encounter an all busy decoders when trying to allocate a decoder to specific codeword, in this case a decoding failure is declared and a retransmission request is issued. This will make the decoder a suboptimal in the sense that there are situations where the number of errors $v \leq t$ but the time assigned to the decoder is insufficient. The performance evaluation of this system can be investigated as a future work to find the best choice of the parameters *M* and *N*.

## CONCLUSION

A binary BCH decoder is implemented in PIC microcontroller platform using assembly language with n=127 bits and adjustable error correction capability t, the number of code bits n can be increased to 255 bits with slight modifications to the program. Results shows that the time consumption of the decoding algorithm depend on the actual number of errors and that suggests using the designed decoder in a distributed system fashion in order to enhance the performance.

## REFERENCES

- Cho, J. and Sung, W., 2009, *Efficient Software-Based Encoding and Decoding of BCH Codes*, IEEE Transactions On Computers, Vol. 58, No. 7 pp 878-889.
- Deschamps, J. P., Imaña, J. J., Sutter, G. D., 2009, *Hardware Implementation of Finite-Field Arithmetic*, McGrow Hill.
- Jiang, Y., 2010, *A Practical Guide to Error-Control Coding Using MATLAB*, Artech House.
- Justesen, J., and Høholdt, T., 2004, *A Course In Error-Correcting Codes*, European Mathematical Society.
- Lin, S. and Costello, D. J., Jr., 1983, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey.
- Lin, Y., Chang, H., and Lee, C., 2010, *An Improved Soft BCH Decoder with One Extra Error Compensation*, Proceedings of 2010 IEEE International Symposium on Circuits and Systems, Paris, pp 3941-3944.
- Lee, Y., Yoo, H. , Yoo, I. and Park, I., 2014, *High-Throughput and Low-Complexity BCH Decoding Architecture for Solid-State Drives*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 22 , No. 5. 1183-1187.
- Moreira, J. C, and Farrell, P. G., 2006, *Essentials of Error-Control Coding,* John Wiley.
- Namba, K., Pontarelli, S.,Ottavi, M. and Lombardi, F, 2014, *A Single-Bit and Double-Adjacent Error Correcting Parallel Decoder for Multiple-Bit Error Correcting BCH Codes*, IEEE Transactions on Device and Materials Reliability, Vol. 14 , No. 2, pp 664–671.
- Schipani, D., Elia, M., Rosenthal, J., 2011 *On the Decoding Complexity of Cyclic Codes Up to the BCH Bound* IEEE International Symposium on Information Theory Proceedings, St. Petersburg, pp 835-839.
- Taki El-Din, R. F., El-Hassani, R. M. and El-Ramly, S. H., 2013, *Optimizing Chien Search Usage in the BCH Decoder for High Error Rate Transmission*, IEEE Communications Letters, Vol. 17, No. 4, pp 741-744.
- Wu, Y., New, 2008, *List Decoding Algorithms for Reed-Solomon and BCH Codes*, IEEE Transactions on Information Theory, Vol. 54, No. 8, pp 3611-3630.
- Wu, Y., *Low Power Decoding of BCH Codes*, 2004, Proceedings of 2010 IEEE International Symposium on Circuits and Systems, Canada pp 369-372.
- www.microchip.com, 2012, PIC18(*L)F2X/4XK22 Data Sheet*, Microchip.

**Table 1.** GF elements and their representation.

| $i$ | GF elements power form | | GF elements polynomial form | | GF elements alternative form | |
|---|---|---|---|---|---|---|
| | $\alpha^i$ | $i$ in Binary Representation | $r(x)$ | Binary Representation | $\alpha^i$ | $i$+1 Binary Representation |
| - | $0=\alpha^{-\infty}$ | - | 0 | 000 | $0=\alpha^{-\infty}$ | 000 |
| 0 | $1=\alpha^0$ | 000 | 1 | 001 | $1=\alpha^0$ | 001 |
| 1 | $\alpha^1$ | 001 | $x$ | 010 | $\alpha^1$ | 010 |
| 2 | $\alpha^2$ | 010 | $x^2$ | 100 | $\alpha^2$ | 011 |
| 3 | $\alpha^3$ | 011 | $x$+1 | 011 | $\alpha^3$ | 100 |
| 4 | $\alpha^4$ | 100 | $x^2+x$ | 110 | $\alpha^4$ | 101 |
| 5 | $\alpha^5$ | 101 | $x^2+x+1$ | 111 | $\alpha^5$ | 110 |
| 6 | $\alpha^6$ | 110 | $x^2+1$ | 101 | $\alpha^6$ | 111 |

**Table 2.** Program Memory Organization.

| Bank | Address | Contents |
|---|---|---|
| 0 | 0h-1Fh | General variables |
| 0 | 20h-3Fh | Syndrome values |
| 0 | 40h-4Fh | Error locating polynomial coefficients $\sigma(x)$ |
| 0 | 50h-5Fh | Correction polynomial coefficients $B(x)$ |
| 1 | 00-7Fh | GF elements lookup table |
| 2 | 0h-7Fh | GFINV elements lookup table |
| 3 | 0h-7Eh | Received/Corrected code in bits |

**Table 3.** Performance analysis of the implemented binary BCH decoder.

| n=127, Instruction Cycle Frequency=16 Mhz | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| t | k | v | Syndrome Calculations | BM Algorithm | Chien Search | | Total Instructions | Bit Rate (kbps) |
| | | | | | Min. | Max. | | |
| 4 | 99 | 4 | 4702 | 1455 | 453 | 11799 | 17956 | 113.17 |
| 5 | 92 | 5 | 5874 | 2087 | 658 | 14210 | 22171 | 91.66 |
| 6 | 85 | 6 | 7046 | 2833 | 901 | 16621 | 26500 | 76.68 |
| 7 | 78 | 7 | 8218 | 3693 | 1182 | 19032 | 30943 | 65.67 |
| 9 | 71 | 9 | 10562 | 5711 | 1858 | 23854 | 40127 | 50.64 |
| 10 | 64 | 10 | 11734 | 6957 | 2253 | 26265 | 44956 | 45.20 |
| 11 | 57 | 11 | 12906 | 8273 | 2686 | 28676 | 49855 | 40.76 |
| 13 | 50 | 13 | 15250 | 11184 | 3666 | 33498 | 59934 | 33.90 |
| 13 | 50 | 12 | 15250 | 10547 | 3157 | 31087 | 56884 | 35.72 |
| 13 | 50 | 10 | 15250 | 9195 | 2253 | 26265 | 50710 | 40.07 |
| 13 | 50 | 8 | 15250 | 7907 | 1501 | 21443 | 44600 | 45.56 |
| 13 | 50 | 6 | 15250 | 6683 | 901 | 16621 | 38554 | 52.71 |
| 13 | 50 | 4 | 15250 | 5523 | 453 | 11799 | 32572 | 62.38 |
| 13 | 50 | 2 | 15250 | 4427 | 157 | 6977 | 26654 | 76.24 |
| 13 | 50 | 0 | 15250 | - | - | - | 1178 | 1724.96 |

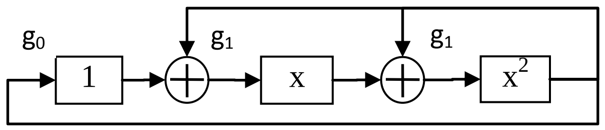


**Figure 1.** Implementation of LFSR for generating GF elements.

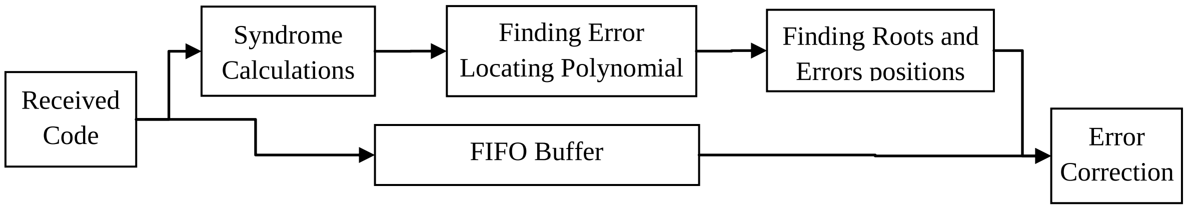

Received Code
Syndrome Calculations
Finding Error Locating Polynomial
Finding Roots and Errors positions
FIFO Buffer
Error Correction

**Figure 2.** Block diagram for Binary BCH Decoding.

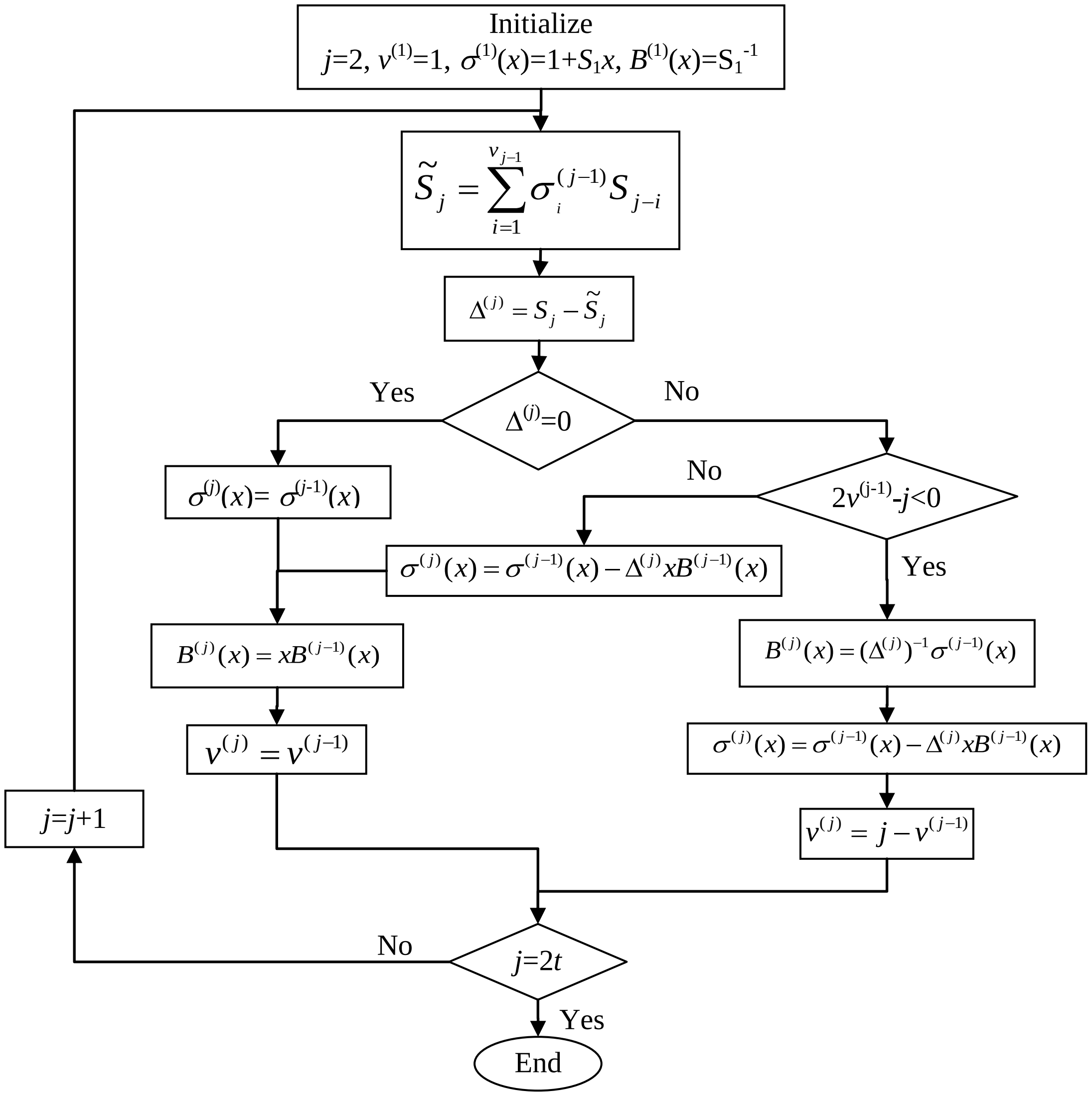

Initialize
$j=2,\ v^{(1)}=1,\ \sigma^{(1)}(x)=1+S_1x,\ B^{(1)}(x)=S_1^{-1}$
$\tilde{S}_j=\sum_{i=1}^{v_{j-1}}\sigma_i^{(j-1)}S_{j-i}$
$\Delta^{(j)}=S_j-\tilde{S}_j$
Yes
No
$\Delta^{(j)}=0$
$\sigma^{(j)}(x)=\sigma^{(j-1)}(x)$
No
$2v^{(j-1)}-j<0$
Yes
$\sigma^{(j)}(x)=\sigma^{(j-1)}(x)-\Delta^{(j)}xB^{(j-1)}(x)$
$B^{(j)}(x)=xB^{(j-1)}(x)$
$B^{(j)}(x)=(\Delta^{(j)})^{-1}\sigma^{(j-1)}(x)$
$v^{(j)}=v^{(j-1)}$
$\sigma^{(j)}(x)=\sigma^{(j-1)}(x)-\Delta^{(j)}xB^{(j-1)}(x)$
$j=j+1$
$v^{(j)}=j-v^{(j-1)}$
No
$j=2t$
Yes
End

**Figure 3.** Block Diagram of BM algorithm.

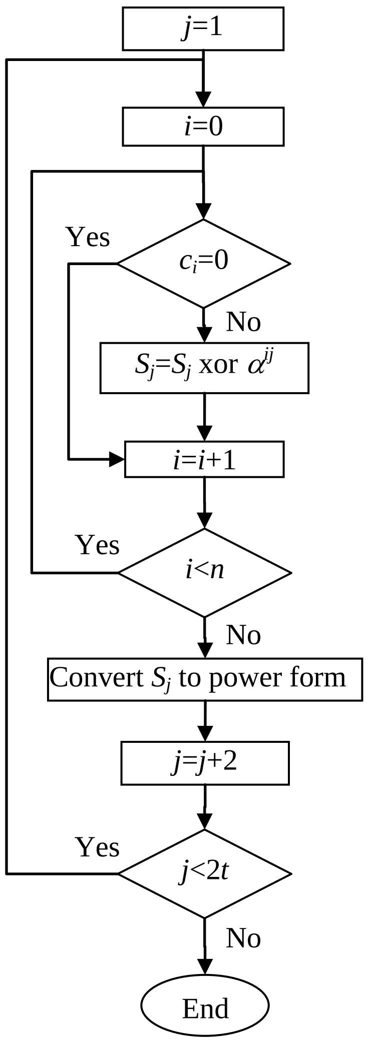
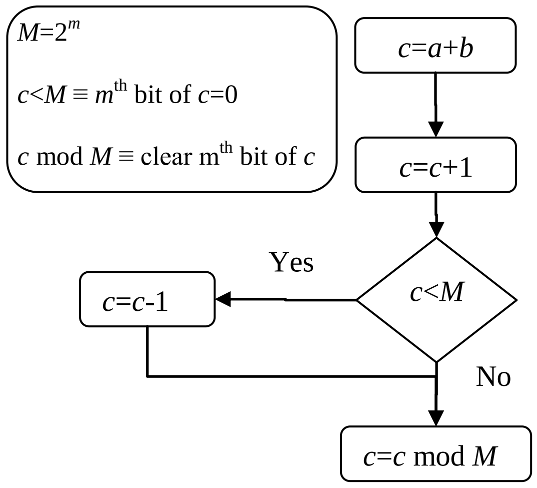
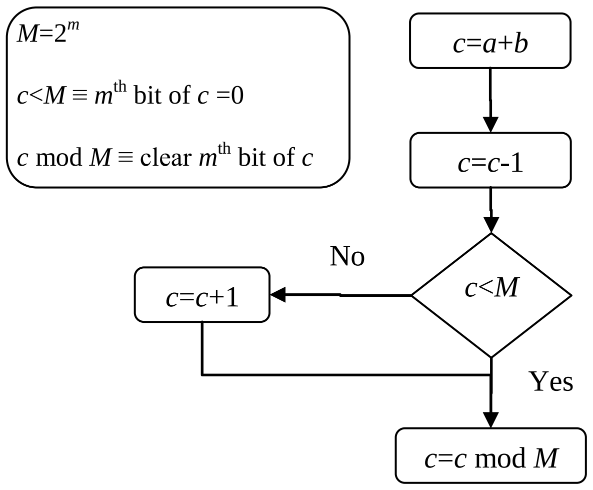


**Figure 4.** Flowchart for odd index syndrome calculations.



$M=2^m$
$c<M \equiv m^{th}$ bit of $c=0$
$c$ mod $M \equiv$ clear $m^{th}$ bit of $c$
$c=a+b$
$c=c+1$
Yes
$c<M$
$c=c-1$
No
$c=c$ mod $M$

**Figure 5.** An algorithm for mod $2^m$-1 addition.



$M=2^m$
$c<M \equiv m^{th}$ bit of $c=0$
$c$ mod $M \equiv$ clear $m^{th}$ bit of $c$
$c=a+b$
$c=c-1$
No
$c<M$
$c=c+1$
Yes
$c=c$ mod $M$

**Figure 6.** An algorithm for mod $2^m$-1 addition suitable for the used GF elements alternative form.

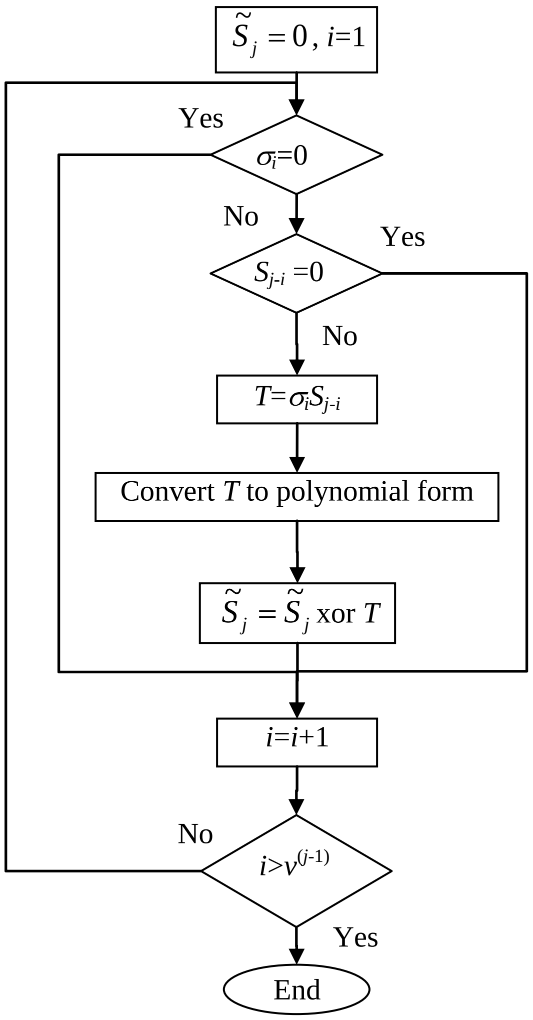


**Figure 7.** Flowchart for implementing Eq. (22) of BM algorithm.

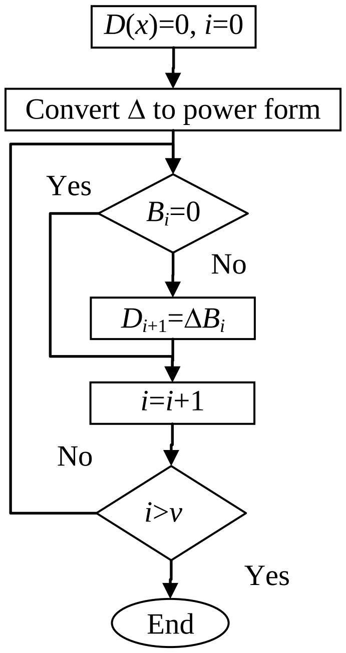


**Figure 8.** Flowchart for implementing Eq. (34) of BM algorithm.

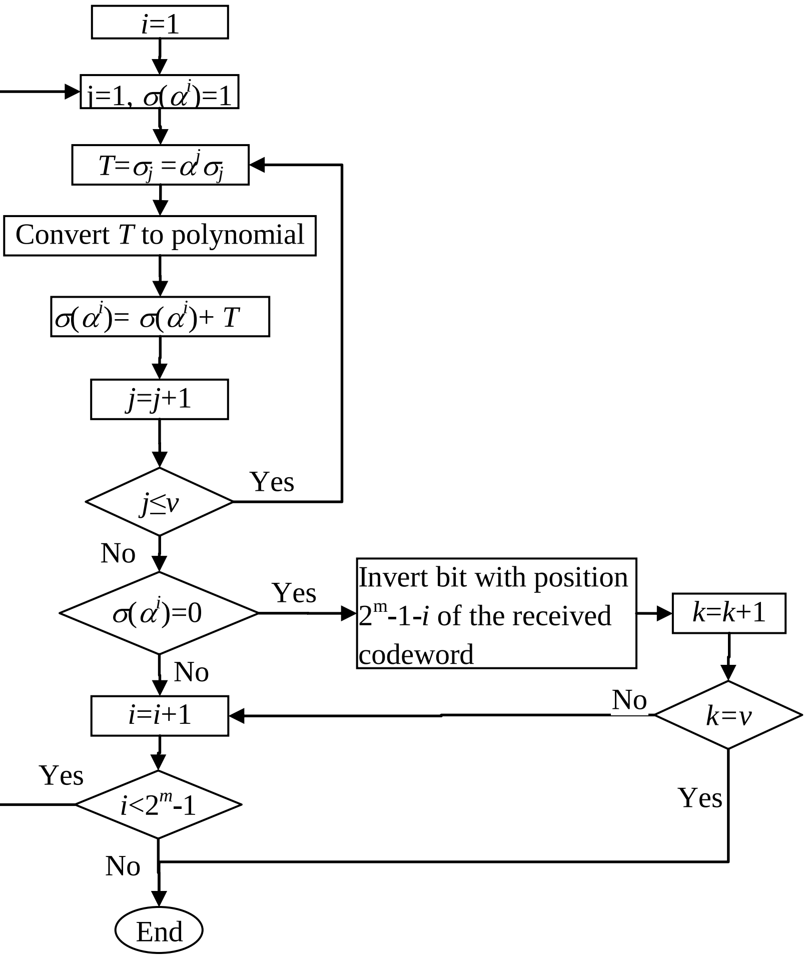


**Figure 9.** Flowchart for implementing Chien search**.**

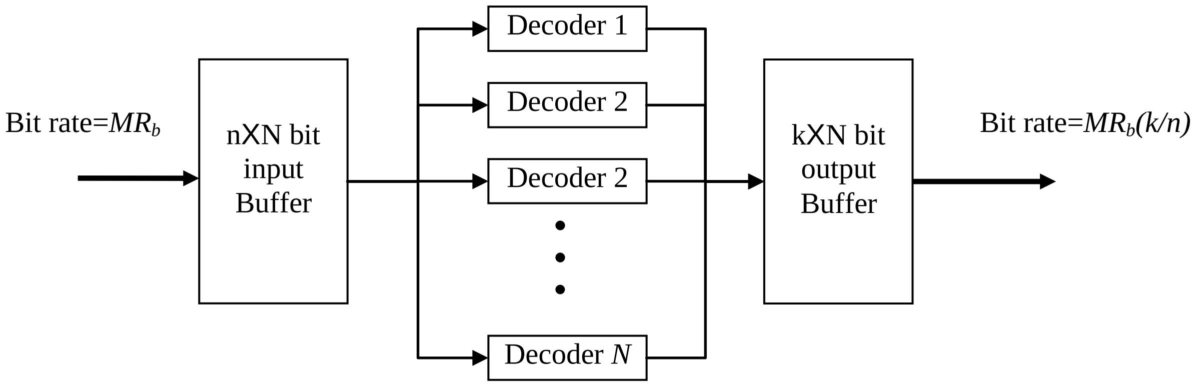


**Figure 10.** A proposed Distributive Decoder**.**