## 1.8 ERROR DETECTION AND CORRECTION

### Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal. The term single-bit error means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1 as shown in figure 1.8.1. The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.

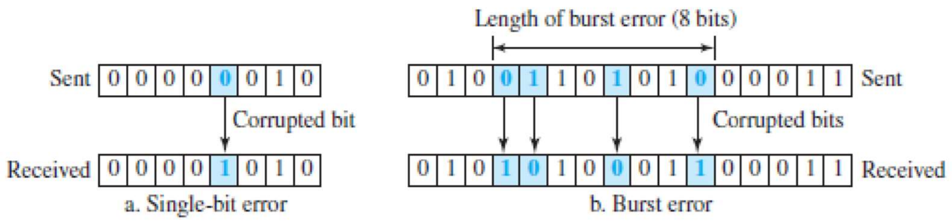

**Fig1.8.1: Effect of a single-bit and a burst error on a data unit.**
*[Source :"Data Communications and Networking" by Behrouz A. Forouzan,Page-258]*

### Redundancy

The central concept in detecting or correcting errors is redundancy. To detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

### Detection versus Correction

The correction of errors is more difficult than the detection. In error detection, if any error has occurred. A single-bit error is the same for us as a burst error. In error correction, we need to know the exact number of bits that are corrupted and, their location in the message.

### Coding

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits.The receiver checks the relationships between the two sets of bits to detect errors.

### Linear Block Codes

Almost all block codes used today belong to a subset of block codes called linear block codes.

### Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The minimum hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

## Parity-Check Code

The most familiar error-detecting code is the parity-check code. This code is a linear block code. In this code, a k-bit dataword is changed to an n-bit codeword where n = k + 1. The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even as shown in figure1.8.2 . Some implementations specify an odd number of 1s.The minimum Hamming distance for this category is dmin = 2, which means that the code is a single-bit error-detecting code.

The code in Table (below) is a parity-check code with k = 4 and n = 5.

| *Dataword* | *Codeword* | *Dataword* | *Codeword* |
|---|---|---|---|
| 0000 | **00000** | 1000 | **10001** |
| 0001 | **00011** | 1001 | **10010** |
| 0010 | **00101** | 1010 | **10100** |
| 0011 | **00110** | 1011 | **10111** |
| 0100 | **01001** | 1100 | **11000** |
| 0101 | **01010** | 1101 | **11011** |
| 0110 | **01100** | 1110 | **11101** |
| 0111 | **01111** | 1111 | **11110** |

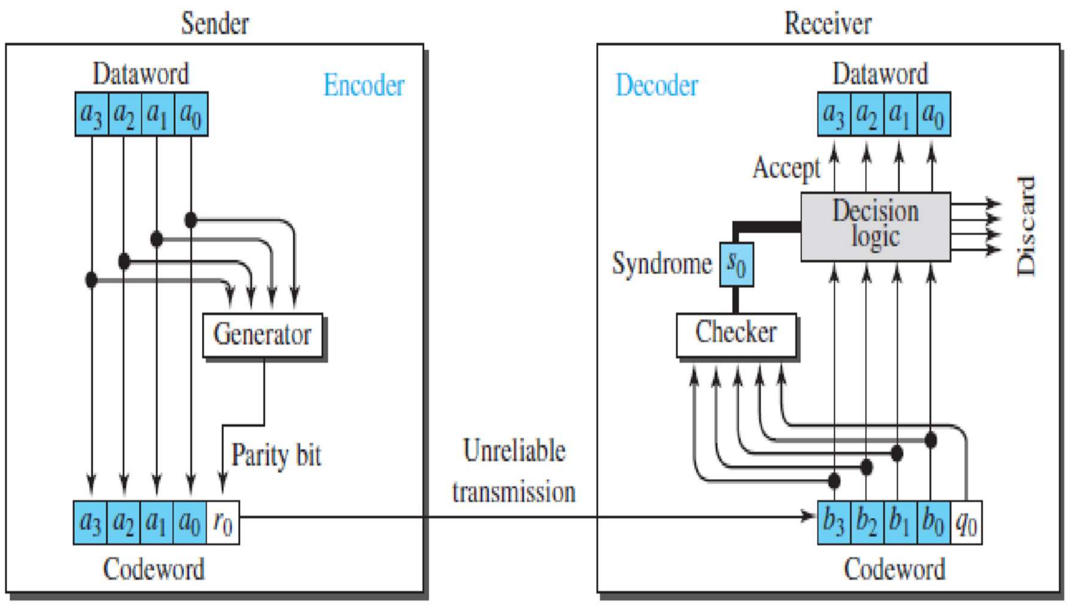

**Fig1.8.2: Encoder and decoder for simple parity-check code**
*[Source :"Data Communications and Networking" by Behrouz A. Forouzan,Page-263]*

The calculation is done in modular arithmetic . The encoder uses a generator that takes a copy of a 4-bit dataword ($a0$, $a1$, $a2$, and $a3$) and generates a parity bit r0. The dataword bits and the parity bit create the 5-bit codeword.

The parity bit that is added makes the number of 1s in the codeword even. This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In otherwords, r0 =a3+a2+a1+a0 (modulo-2)

If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1.

In both cases, the total number of 1s in the codeword is even. The sender sends the codeword, which may be corrupted during transmission. The receiver receives a 5-bit word.The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits. The result is called the syndrome, is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1. s0 = b3 + b2 + b1+ b0 + q0 (modulo-2)

The syndrome is passed to the decision logic analyzer.

**If the syndrome is 0**, there is no detectable error in the received codeword; the data portion of the received codeword is accepted as the dataword; **if the syndrome is 1**, the data portion of the received codeword is discarded. The dataword is not created.

## CYCLIC CODES

Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.

For example,if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word *a*0 to *a*6, and the bits in the second word*b*0 to *b*6, we can shift the bits by using the following: *b*1=*a*0*b*2=*a*1*b*3 =*a*2*b*4=*a*3*b*5=*a*4*b*6=*a*5*b*0=*a*6

## Cyclic Redundancy Check

The **cyclic redundancy check (CRC)**, is used in networks such as LANs and WANs.

In the encoder, the dataword has *k* bits (4 here); the codeword has *n* bits (7 here).

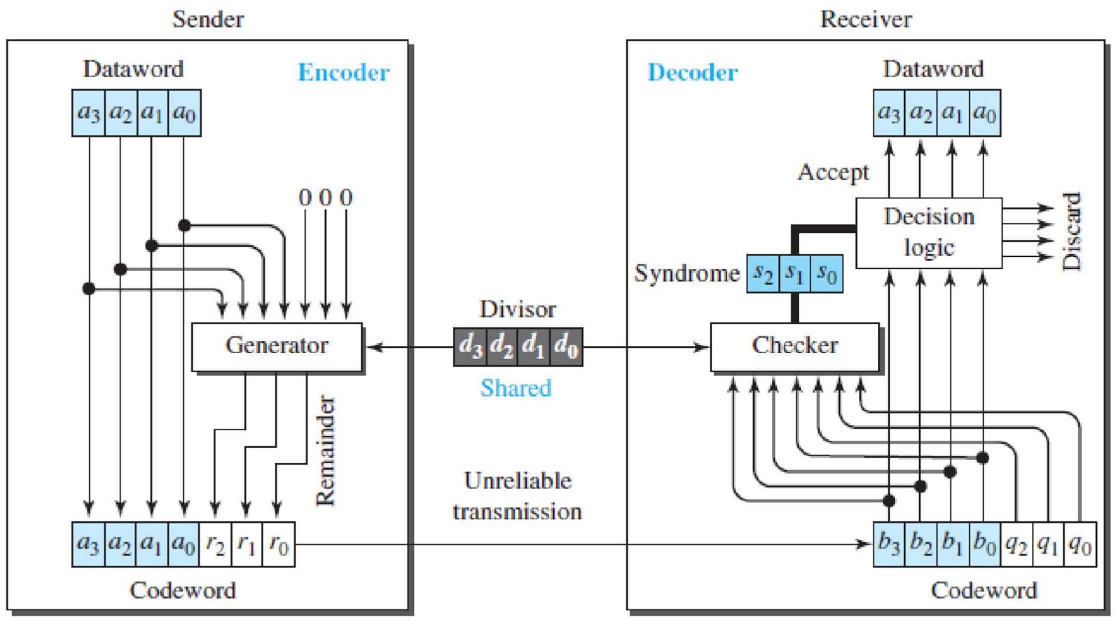

**Fig1.8.3: CRC encoder and decoder.**

*[Source :"Data Communications and Networking" by Behrouz A. Forouzan,Page-265]*

The size of the dataword is augmented by adding n -k (3 here) 0s to the right-hand sideof the word. The n-bit result is fed into the generator. The generator uses a divisor of size n -k (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division).

The quotient of the division is discarded;the remainder (r2r1r0) is appended to the data word to create the codeword.The decoder receives the codeword (possibly corrupted in transition). A copy of all n bits is fed to the checker, which is a replica of the generator.

The remainder produced by the checker is a syndrome of n-k (3 here) bits, which is fed to the decision logic analyzer.The analyzer has a simple function. If the syndrome bits are all 0s, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise,the 4 bits are discarded (error).

**Encoder**

The encoder shown in figure 1.8.4, takes a dataword and augments it with n-k number of 0s. It then divides the augmented dataword by the divisor.

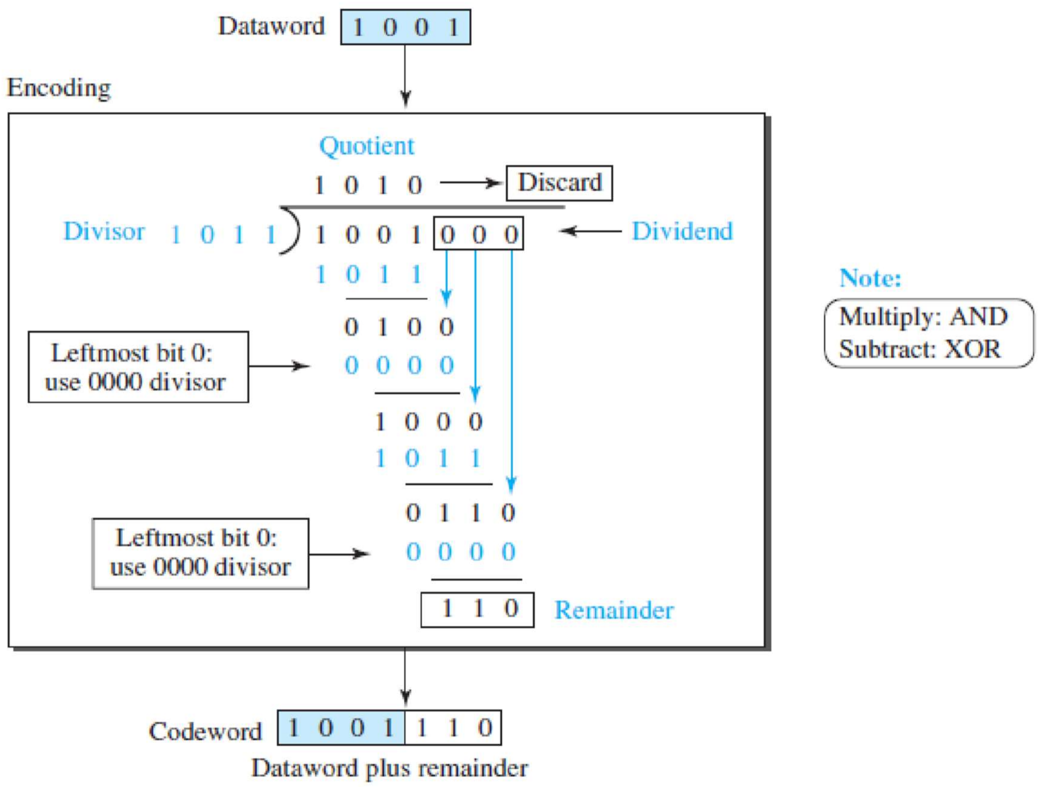

**Fig1.8.4: Division in CRC encoder.**

*[Source :"Data Communications and Networking" by Behrouz A. Forouzan,Page-266]*

As in decimal division, the process is done step by step. In each step, a copy of the divisor is XOR ed with the 4 bits of the dividend. The result of the XOR operation(remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division.

If the leftmost bit of the dividend (or the part used in each step)is 0, the step cannot use the regular divisor; we need to use an all-0s divisor.When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits (r2, r1, and r0). They are appended to the dataword to create the codeword.

**Decoder**

The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error with a high probability; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded.

Figure 1.8.5 shows two cases: The left-hand figure shows the value of the syndrome when no error has occurred; the syndrome is 000. The right-hand figure shows the case in which there is a single error. The syndrome is not all 0s (it is 011).

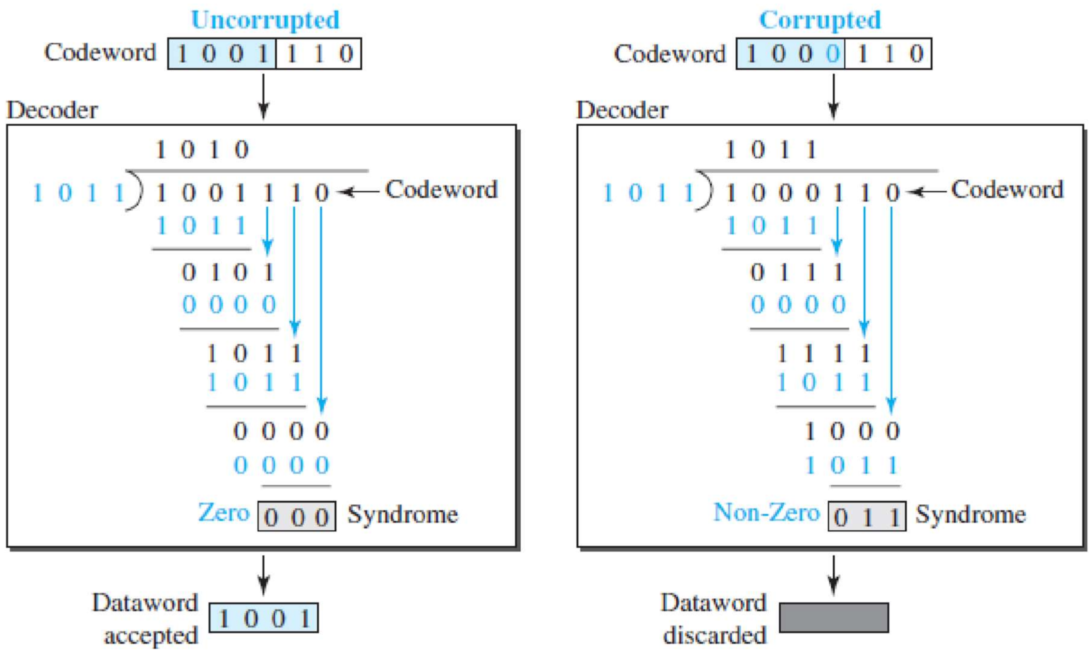

**Fig1.8.5: Division in decoder.**

*[Source :"Data Communications and Networking" by Behrouz A. Forouzan,Page-267]*

## CHECKSUM

Checksum is an error-detecting technique that can be applied to a message of any length.

At the source, the message is first divided into m-bit units as in figure 1.8.6. The generator then creates an extra m-bit unit called the checksum, which is sent with the message.

At the destination, the checker creates a new checksum from the combination of the message and sent checksum.

If the new checksum is all 0s, the message is accepted; otherwise,the message is discarded. In the real implementation, the checksum unit is not necessarily added at the end of the message; it can be inserted in the middle of the message.

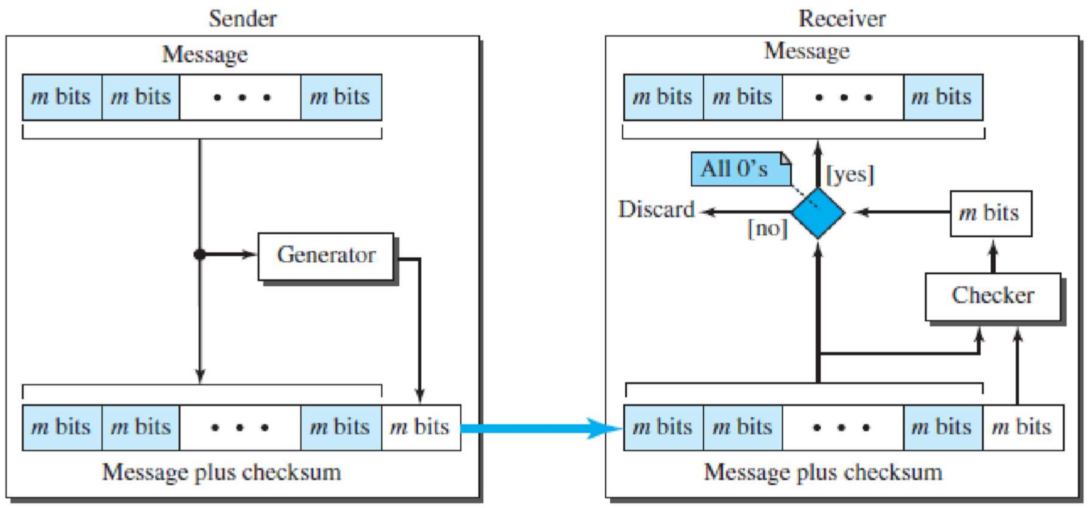

**Fig1.8.6:Checksum.**

*[Source :"Data Communications and Networking" by Behrouz A. Forouzan,Page-278]*

## Concept

The idea of the traditional checksum is simple.

## Example

Suppose the message is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers.

The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise ,there is an error somewhere and the message is not accepted.

## One's Complement Addition

The previous example has one major drawback. Each number can be written as a 4-bitword (each is less than 15) except for the sum.

One solution is to use one's complement arithmetic. In this arithmetic, we can represent unsigned numbers between 0and 2m -1 using only m bits. If the number has more than m bits, the extra leftmost bits need to be added to the m rightmost bits (wrapping).

Example

In the previous example, the decimal number 36 in binary is (100100)2. To change it to a 4bit number we add the extra leftmost bit to the right four bits as shown below. Instead of sending 36 as the sum, we can send 6 as the sum (7, 11, 12, 0, 6, 6).

The receiver can add the first five numbers in one’s complement arithmetic. If the result is 6, the numbers are accepted; otherwise, they are rejected.

We can make the job of the receiver easier if we send the complement of the sum, the **checksum**. In one’s complement arithmetic, the complement of a number is found by completing all bits (changing all 1s to 0s and all 0s to 1s). This is the same as subtracting the number from 2*m* -1.

In one’s complement arithmetic, we have two 0s: one positive and one negative, which are complements of each other. The positive zero has all m bits set to 0; the negative zero has all bits set to 1 (it is 2m -1).

If we add a number with its complement, we get a negative zero (a number with all bits set to 1). When the receiver adds all five numbers (including the checksum), it gets a negative zero. The receiver can complement the result again to get a positive zero.

**Example**

The sender adds all five numbers in one’s complement to get the sum 6. The sender then complements the result to get the checksum 9,which is 15 -6.

Note that 6 (0110)2 and 9 (1001)2; they are complements of each other.

The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, **9**). If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, **9**) and adds them in one’s complement to get 15. The sender complements 15 to get 0. This shows that data have not been corrupted. Figure 1.8.7 shows the process.

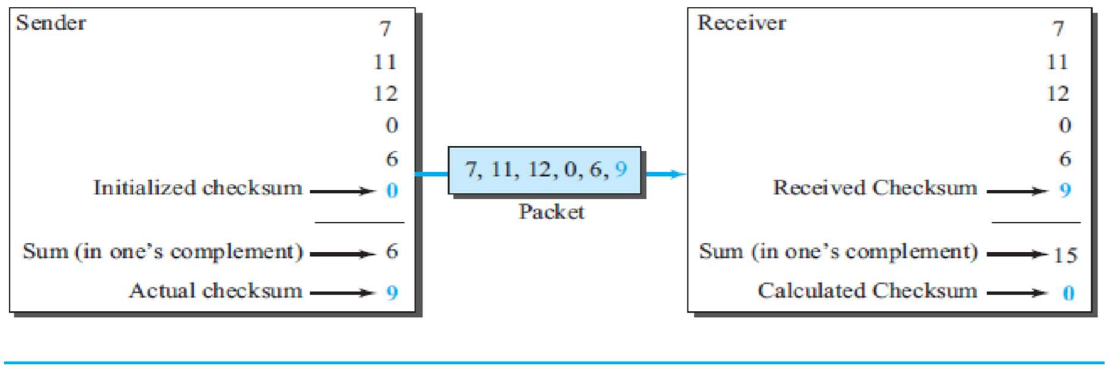

**Fig1.8.7: Checksum Process.**

*[Source :”Data Communications and Networking” by Behrouz A. Forouzan,Page-279]*