4 데이터 전송의 기초

쉽게 배우는 데이터 통신과 컴퓨터 네트워크

학습목표

- ✓ 전송과 교환 시스템의 구조와 원리
- ✓ 프레임 전송 과정에서 발생되는 오류의 유형
- ✓ 문자 프레임과 비트 프레임의 구조
- ✓ 오류 검출 코드의 종류와 원리를 이해
- ✓ 생성 다항식을 이용한 오류 검출 방식

3절. 프레임

□ 프레임(Frame)

- 송수신 호스트 MAC주소, 제어 정보, 체크섬 등의 정보
- 내부 정보의 표현 형태 : 문자 프레임, 비트 프레임

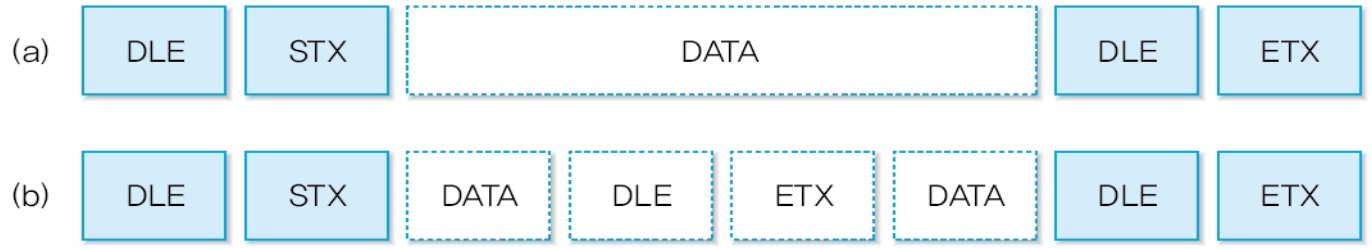
□ 문자 프레임(Character Frame)

- 프레임의 내용이 문자(8비트 ASCII 코드)로만 구성됨
- IBM의 BSC 및 ISO의 Basic 프로토콜에 사용

▪ 프레임의 구조

- 프레임의 시작과 끝에 특수 문자 사용 [그림 4-13(a)]
  - 시작: DLE / STX, 끝: DLE / ETX
- 전송 데이터에 특수 문자가 포함되면 혼선이 발생 [그림 4-13(b)]

DLE : Data Link Escape
STX : Start of Text
ETX : End od Text

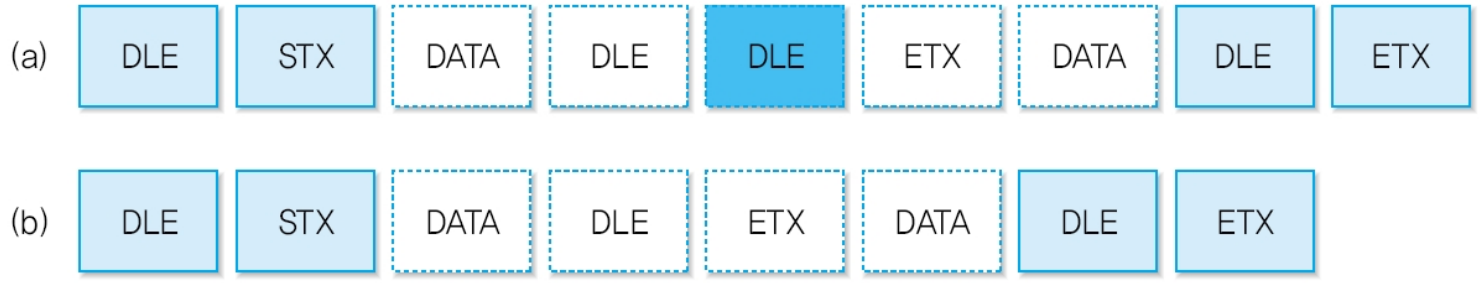


[그림 4-13] 문자 프레임의 구조

3절. 프레임

□ 문자 프레임

- **문자 스터핑(Character Stuffing)**
  - 문자 프레임 전송과정에서 제어 문자를 추가하는 기능
  - 송신 호스트: 데이터에 DLE 문자가 있으면 강제로 DLE 문자 추가 [그림 4-14(a)]
  - 수신 호스트: 데이터에 DLE 문자가 두 번 연속 있으면 DLE 문자 삭제 [그림 4-14(b)]

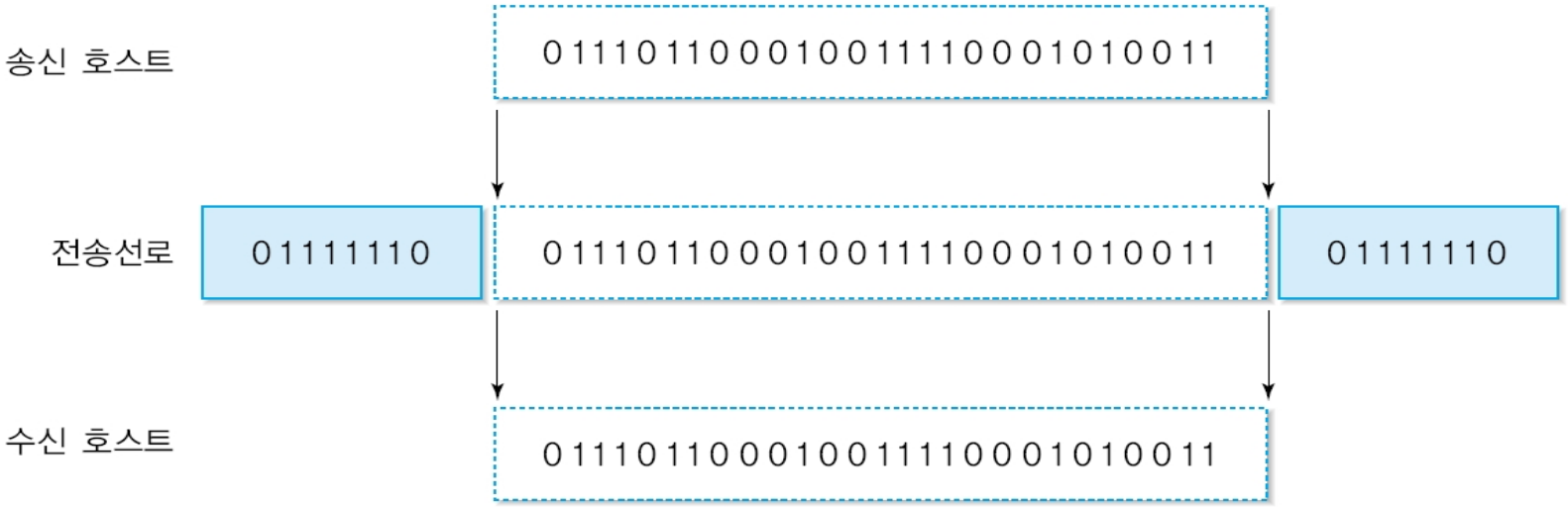


[그림 4-14] 문자 스터핑

□ 비트 프레임(Bit Frame)

- 프레임을 문자 단위로 해석하지 않음
- 프레임의 시작과 끝을 구분하기 위하여 플래그 (01111110) 사용
- IBM의 SDLC 및 ISO의 HDLC/ITU-T의 X.25 프로토콜 사용

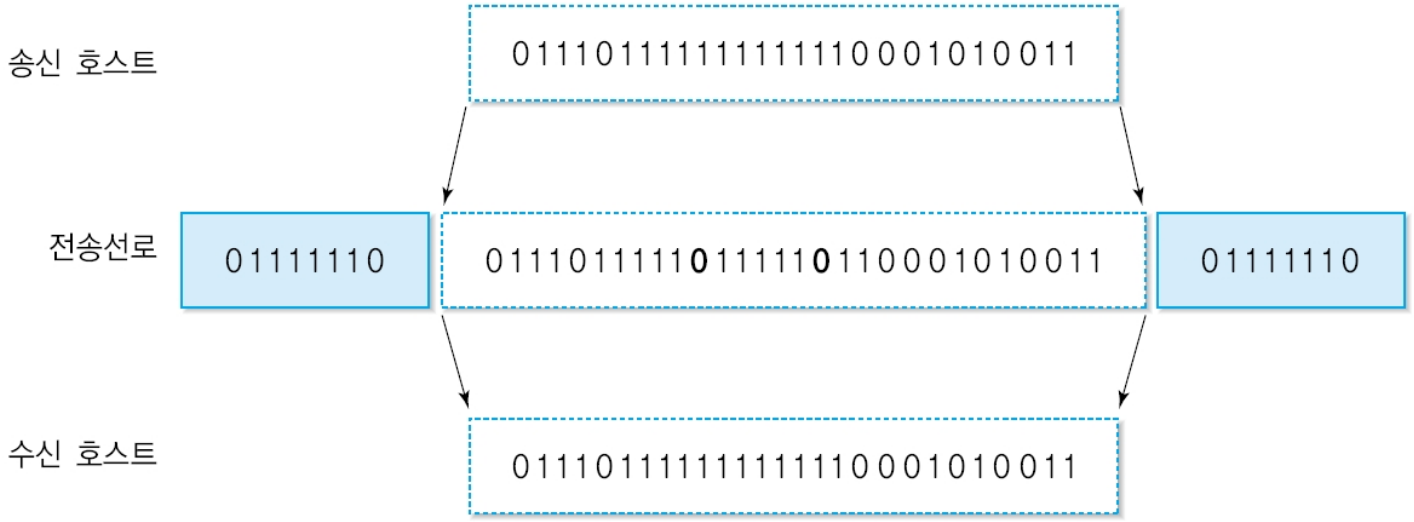
▪ 프레임의 구조 [그림 4-15]



[그림 4-15] 비트 프레임의 구조

비트 프레임

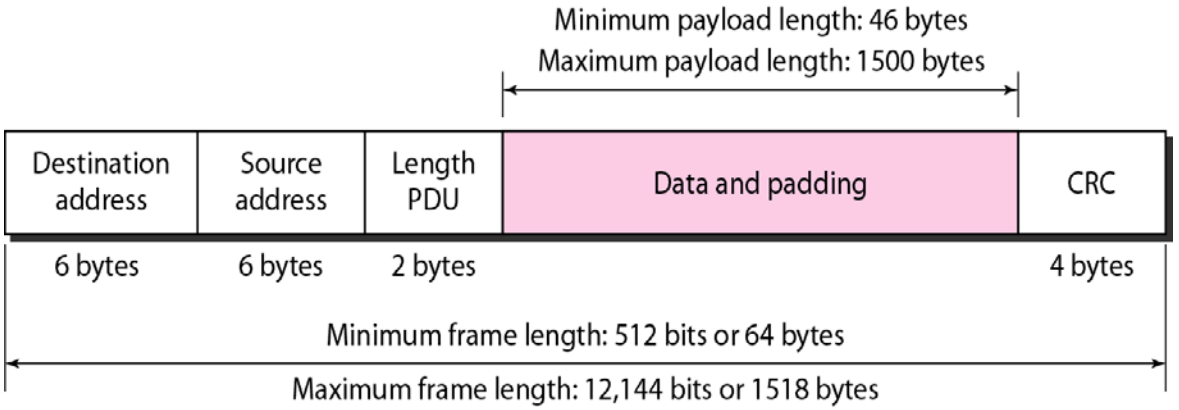
- 비트 스터핑 (Bit Stuffing)[그림 4-16]
  - 전송 데이터에 플래그 패턴이 포함되면 혼선이 발생
  - 송신 호스트: 데이터에 1 이 연속해서 5번 발생하면 강제로 0을 추가
  - 수신 호스트: 데이터에 1 이 연속해서 5번 발생하면 이어진 0을 제거



[그림 4-16] 비트 스터핑

LAN에서 MAC 계층의 프레임(참고)

- PDU의 길이(데이터 필드의 크기)를 표시



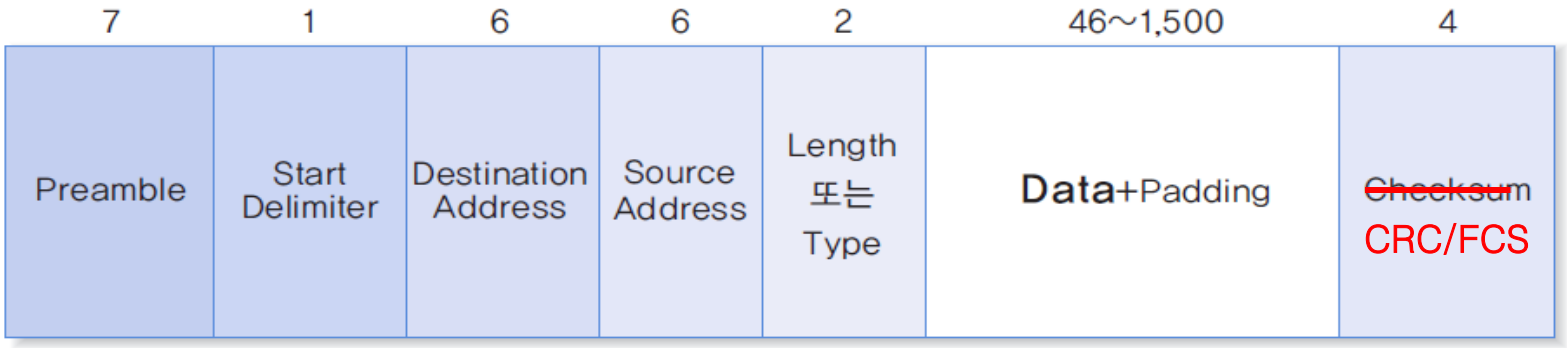


그림 5-7 이더넷 프레임의 구조

- Preamble: 수신 호스트가 송신 호스트의 클록 동기를 맞추는 용도,10101010
- Start Delimiter: 프레임의 시작 위치 구분, 10101011

4절. 다항 코드

- 순방향 오류 복구(FEC, Forward Error Correction)
  : 오류 복구 코드를 이용해 수신 호스트 스스로 오류를 복구
- 역방향 오류 복구 (BEC, Backward Error Correction)
  ARQ(automatic Repeat reQuest)
  : 오류 검출 코드를 이용해 수신 호스트가 송신 호스트에게 오류를 통지

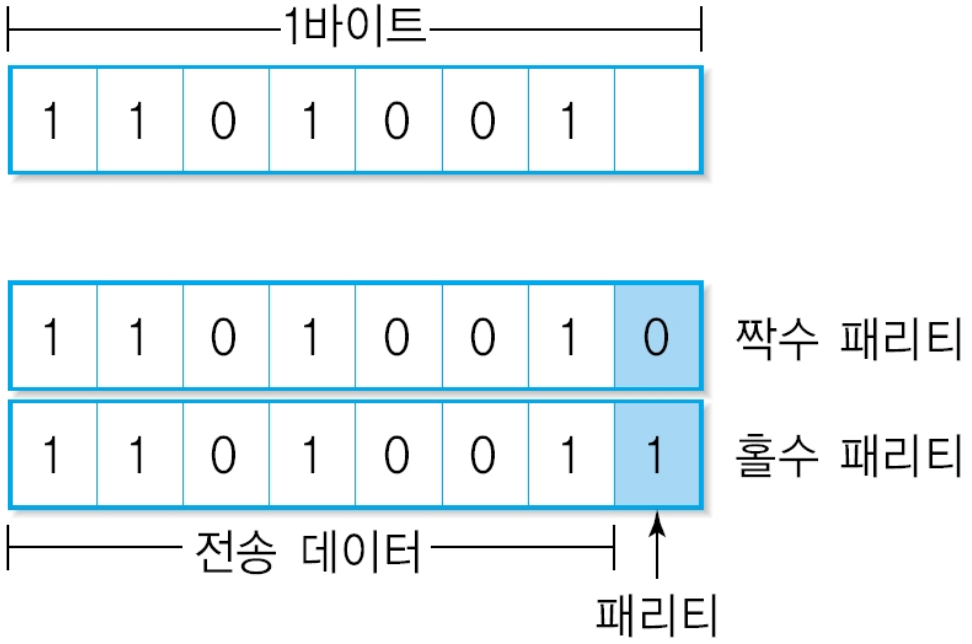
□오류 검출

▪ 패리티 (Parity)[그림 4-17]

- 1 바이트 = 7 비트 ASCII 코드 + 1 비트 패리티
- 짝수 패리티: 1의 개수가 짝수가 되도록 패리티를 지정
- 홀수 패리티: 1의 개수가 홀수가 되도록 패리티를 지정
- 송신 호스트와 수신 호스트는 동일한 패리티 방식을 사용해야 함

오류 검출

- **패리티 [그림 4-17]**
  - 전송 과정에서 홀수개의 비트가 깨지면 오류 검출 가능
  - 전송 과정에서 **짝수개의 비트가 깨지면 오류 검출 불가능**



[그림 4-17] 패리티 비트

오류 검출

- **블록 검사(Block Sum Check) [그림 4-18]**
  - 짝수개의 비트가 깨지는 오류를 검출
  - 수평, 수직 방향 모두에 패리티 비트를 지정

0	1	0	0	1	1	0	1
1	1	0	1	0	0	1	0
0	0	1	1	0	0	1	1
1	1	0	1	1	1	1	0
1	1	1	0	1	0	1	1
0	1	1	1	0	0	1	0
1	1	1	0	1	0	1	1

블록 검사

[그림 4-18] 블록 검사

4절. 다항 코드

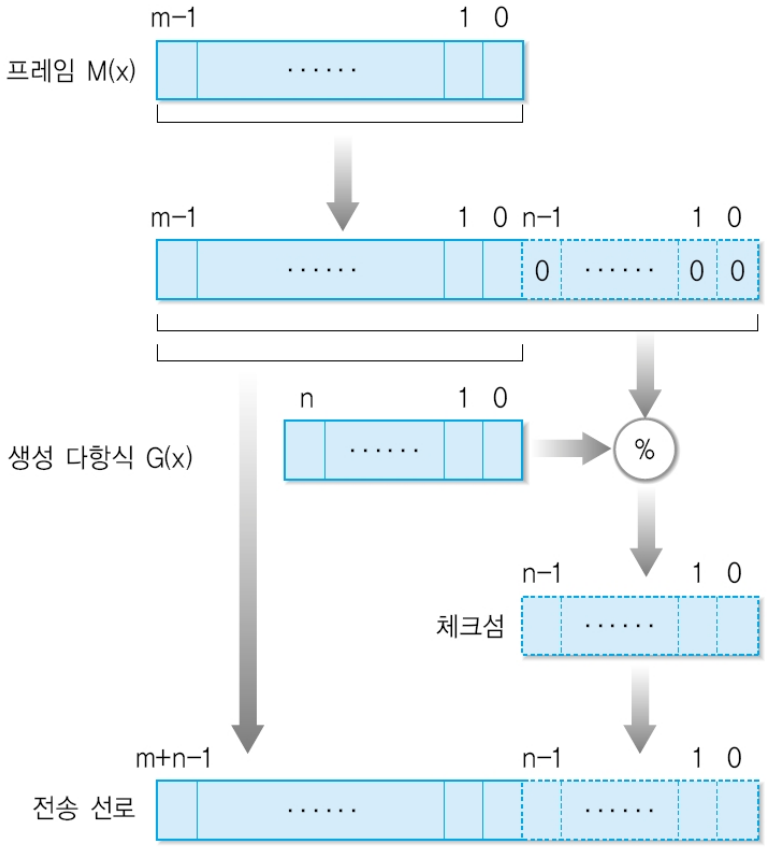
□ 다항 코드(Polynomial Code) 방식 혹은 CRC(Cyclic Redundancy Code)코드
➔ FCS(Frame Check sequence)라고도 함

- 생성 다항식
  - 다항코드 100101 = 생성 다항식 $x^5 + x^2 + 1$
  - 전송 데이터: m 비트 크기의 M(x)
  - 생성 다항식: n+1 비트 크기의 G(x)
  - 체크섬
    - 전송 데이터와 생성 다항식을 이용하여 계산 [그림 4-19]
    - n 비트 크기
  - 송신 호스트: "전송 데이터 + FCS"을 수신 호스트에게 전송
  - 수신 호스트: "전송 데이터 + FCS"을 생성 다항식으로 나누어 결과를 확인
    - 나머지가 0 이면 전송 오류가 없는 경우
    - 나머지가 0 이 아니면 전송 오류가 있는 경우

다항 코드

생성 다항식

- **나머지 값** 계산 원리 [그림 4-19]



[그림 4-19] 생성 다항식

다항 코드

생성 다항식

- **나머지 값** 계산 과정에서 뺄셈 연산 방법
  - 모듈로-2 방식을 사용
  - 두 수가 같으면 0
  - 두 수가 다르면 1

- 모듈로-2 방식
  - 덧셈의 자리 올림이나 뺄셈의 자리 빌림이 생략됨
  - 덧셈과 뺄셈 모두 배타적 논리합 (Exclusive OR) 연산과 동일

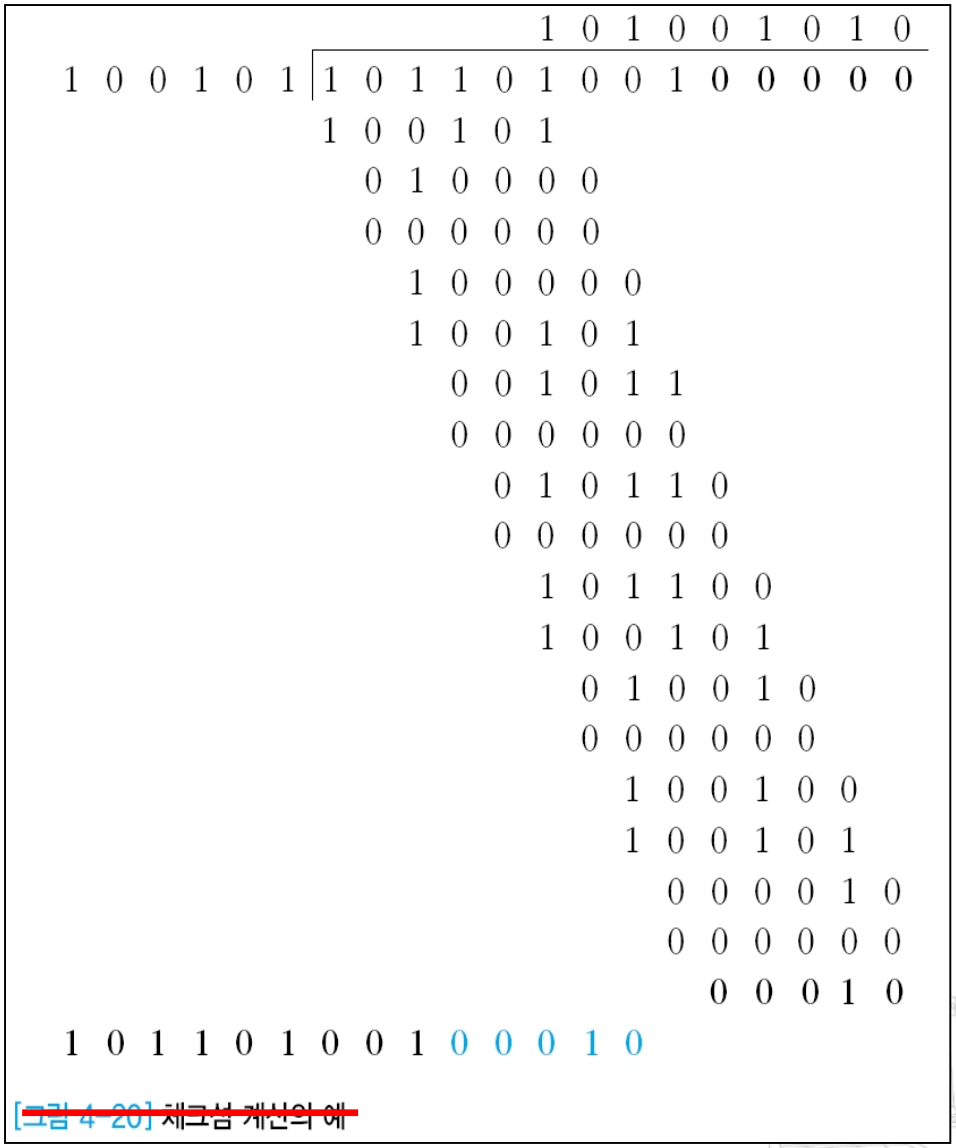
```
  100111101101          100111101101
 -110001110110         +110001110110
 ============          ============
  010110011011          010110011011
```

다항 코드

- **CRC 코드 작성의 예**
  - 생성 다항식 $G(x) = x^5 + x^2 + 1$
  - 전송 데이터: 101101001
  - 나머지 값: 00010

  - 나머지 값 =CRC 코드 값
    =FCS 값

나머지 값을 체크섬(checksum)이라고 부르는 것은 적절치 않음



[그림 4-20] 체크섬 계산의 예

다항 코드

- CRC 코드(혹은 FCS)의 예
  - CRC-12: $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
  - CRC-16: $x^{16} + x^{15} + x^2 + 1$
  - CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$ (HDLC)
  - CRC-32 : $x^{32} + x^{26} + \cdots\cdots + x + 1$ (LAN) - 4 바이트
    (데이터 링크 계층 및 SCTP 프로토콜에서 사용하도록 권고)

그러나 ICMP, TCP, UDP 프로토콜 등에서는 검사합(checksum) 기법을 사용함