에러 검출 코드; Error Correction Code,
부호화; Encoding,
복호화: Decoding

05 에러 검출 코드

1 패리티 비트

- 짝수패리티(even parity) : 데이터에서 1의 개수를 짝수 개로 맞춤
- 홀수패리티(odd parity) : 1의 개수를 홀수 개로 맞춤
- 패리티 비트는 데이터 전송과정에서 에러 검사를 위한 추가비트
- 패리티는 단지 에러 검출만 가능하며, 여러 비트에 에러가 발생할 경우에는 검출이 안될 수도 있음

표 2-15 7비트 ASCII 코드에 패리티 비트를 추가한 코드

ASCII코드	짝수 패리티	홀수 패리티
⋮	⋮	⋮
A	0 1000001	1 1000001
B	0 1000010	1 1000010
C	1 1000011	0 1000011
D	0 1000100	1 1000100
⋮	⋮	⋮

2 해밍 코드

- 에러를 정정할 수 있는 코드
- 추가적으로 많은 비트가 필요하므로 많은 양의 데이터 전달이 필요
- 데이터 비트와 추가되는 패리티 비트와의 관계

$$2^p \geq d + p + 1$$

- 예를 들어 d = 8이면 $2^p \geq 8+p+1$을 만족하는 p를 계산하면 4가 된다.
  (즉, 8비트 데이터에 4비트 해밍 패리티 비트가 추가되어 12비트 코드가 됨)
- 해밍코드에서는 **짝수 패리티를** 사용

P1은 한칸씩 띄워서 계산
P2는 두칸씩 띄워서 계산
P4는 네칸씩 띄워서 계산
P8는 여덟칸씩 띄워서 계산

표 2-16 해밍 코드에서 패리티 비트의 위치와 패리티 생성 영역

MSB

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
P_1 영역	✓		✓		✓		✓		✓		✓	
P_2 영역		✓	✓			✓	✓			✓	✓	
P_4 영역				✓	✓	✓	✓					✓
P_8 영역								✓	✓	✓	✓	✓

❑ 8비트 데이터의 에러 정정 코드

$$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11}$$
$$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11}$$
$$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12}$$
$$P_8 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12}$$

MSB

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
		0		0	1	0		1	1	1	0

$$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11} = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0$$
$$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11} = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12} = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$
$$P_8 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12} = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

XOR의 진리표

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

표 2-17 원본 데이터가 00101110일 경우 해밍 코드 생성 예

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
원본 데이터			0		0	1	0		1	1	1	0
생성된 코드	0	1	0	1	0	1	0	1	1	1	1	0

❑ 해밍 코드에서 패리티 비트 검사 과정

전송된 데이터 : 010111011110

MSB

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
0	1	0	1	1	1	0	1	1	1	1	0

☞ 패리티들을 포함하여 검사

$$P_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11} = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

$$P_2 = P_2 \oplus D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11} = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$$

$$P_4 = P_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_{12} = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1$$

$$P_8 = P_8 \oplus D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12} = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

- 검사된 패리티를 $P_8\,P_4\,P_2\,P_1$ 순서대로 정렬
- 모든 패리티가 0이면 에러 없음
- 하나라도 1이 있으면 에러 발생: 결과가 **0101**이므로 에러 있음
- 0101을 10진수로 바꾸면 **5**이며, 수신된 데이터에서 앞에서 5번째 비트 010**1**1011111에 에러가 발생한 것이므로 010**1**01011110으로 바꾸어 주면 에러가 정정된다.

표 2-17 원본 데이터가 00101110일 경우 해밍 코드 생성 예

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
원본 데이터			0		0	1	0		1	1	1	0
생성된 코드	0	1	0	1	0	1	0	1	1	1	1	0

송신 ↘

수신

표 2-18 해밍 코드에서 에러가 발생한 경우 정정

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
해밍 코드	0	1	0	1	1	1	0	1	1	1	1	0
패리티 검사	1	0		1				0				
$P_8P_4P_2P_1 = 0101_{(2)} = 5_{(10)}$: 5번째 비트에 에러가 발생. 1 → 0으로 정정												
해밍 코드 수정	0	1	0	1	0	1	0	1	1	1	1	0

3 순환 중복 검사(CRC)

- 높은 신뢰도를 확보하며 에러 검출을 위한 오버헤드가 적고, 랜덤 에러나 버스트 에러를 포함한 에러 검출에 매우 좋은 성능을 갖는다.

❖CRC 발생기 및 검출기

k가 16이면 CRC-16

- 송신 측에서는 'd비트 데이터를 k비트만큼 올리고' '(k+1)비트로 나눠' 'k비트의 나머지를 합쳐' d + k 비트를 전송
- 수신 측에서는 수신된 d + k 비트의 데이터를 키 값으로 나누었을 때 나머지가 0이면 에러가 없는 것이지만, 0이 아니면 에러가 발생한 것으로 판단한다.

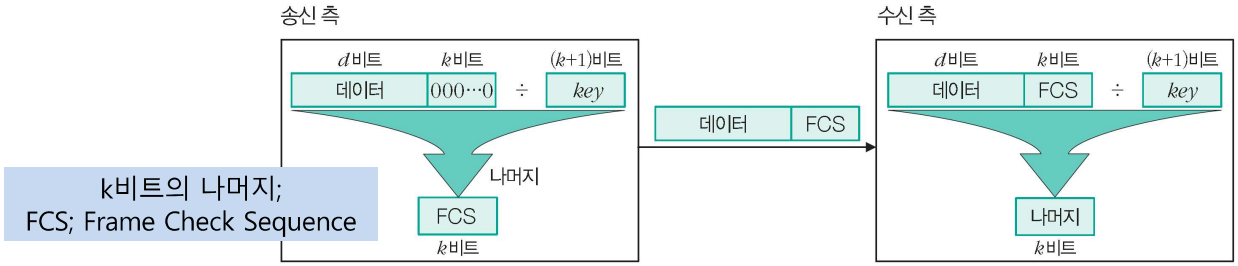


그림 2-8 CRC 발생기와 검사기

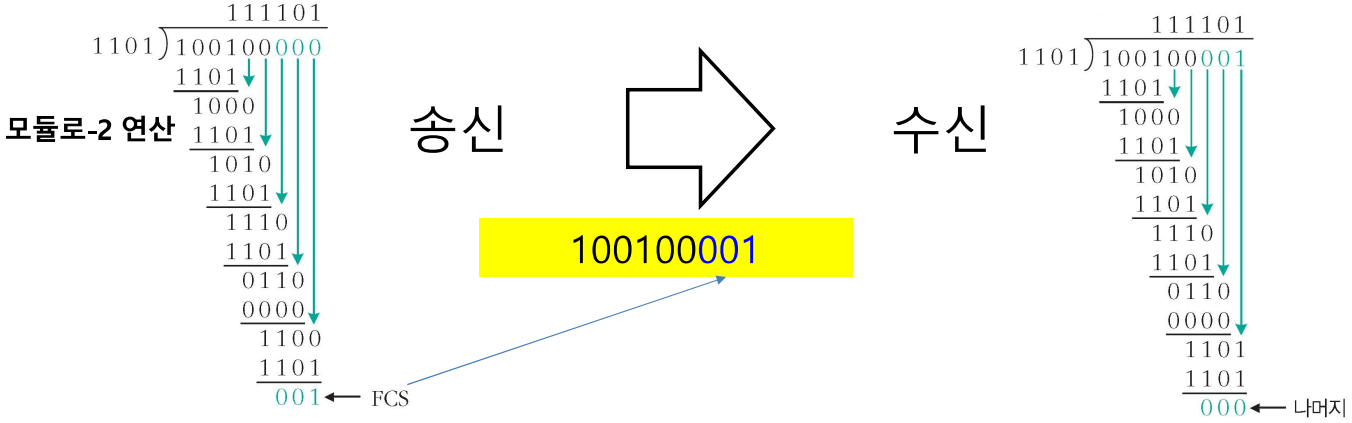
❖ CRC 계산에 사용되는 모듈로-2 연산

- 사칙 연산에서 캐리는 고려하지 않는다.
- 덧셈 연산은 뺄셈 연산과 결과가 같으며 XOR 연산과도 같다.

모듈로-2 연산

- 덧셈 연산: $0+0=0,\ 0+1=1,\ 1+0=1,\ 1+1=0$
- 뺄셈 연산: $0-0=0,\ 0-1=1,\ 1-0=1,\ 1-1=0$

- 데이터가 100100이고, 키 값이 1101인(k+1 = 4 비트) 경우 FCS를 계산하는 예



(a) CRC 발생기에서 계산 과정

(b) 검사기에서 계산 과정

그림 2-9 CRC 발생기와 검사기에서 2진 나눗셈

Byte Ordering

- Byte Odering: Byte 단위로 메모리 저장 또는 통신 전송 되는 순서
  - 2 바이트 이상인 경우부터 차이가 남
- Big-Endian
  - 상위 바이트가 하위 번지에 저장
  - RISC 계열의 CPU
  - 네트웍 통신(TCP/IP 등)
- Little-Endian
  - 하위 바이트가 하위 번지에 저장
  - Intel 계열의 CPU