Chapter 2. 데이터의 표현

01 : 진법과 진법 변환

02 : 정수 표현

03 : 실수 표현

04 : 디지털 코드

05 : 에러 검출 코드

에러검출(error detecting), 에러보정(error correcting)

- 에러검출 : 에러가 발생했음을 알지만 어디에 발생했는지 모름
- 에러보정 : 에러가 어디에 발생했는지 알고 수정가능

- Error detecting : parity bit
- Error correcting : hamming code, crc code

1. 패리티 비트

- parity bit : 가장 간단한 에러검출코드, 주로 주기억장치에서 사용
- 메모리에 저장하거나 전달할 데이터에 패리티 비트를 붙여 전송
- 짝수 패리티(even parity) : 패리티 비트 포함 1의 개수 짝수
- 홀수패리티(odd parity): 패리티 비트 포함 1의 개수 홀수
- 패리티 비트는 전송과정에 에러 발생여부를 검출만 함
- 1비트 패리티 비트 사용시에 1개의 에러 발생여부 검출, 에러가 짝수 개로 발생하면 검출 못함

예제 2-6

짝수 패리티 시스템에서 코드그룹 10110, 11010, 110011, 10101110100, 110001010101011을 수신했다. 에러가 발생한 그룹을 찾아보시오.

패리티 생성기, 검출기

- Parity bit : 1비트의 에러를 검출
- 패리티 생성기

$$parity\ bit = x \oplus y \oplus z$$

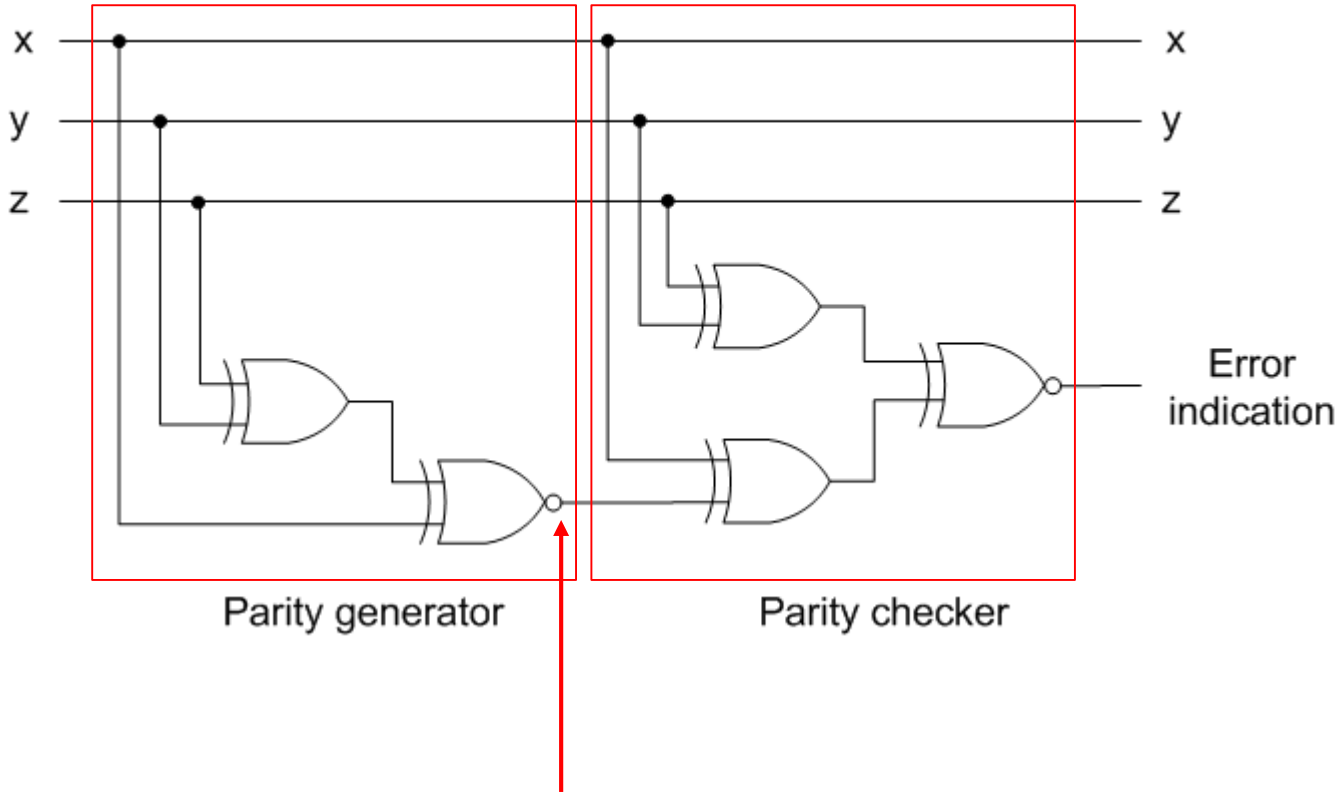
  - x,y,z의 1의 개수가 홀수개일 때 parity bit 출력이 1
  - x,y,z,parity bit의 4비트를 전송하는 시스템(짝수 패리티)

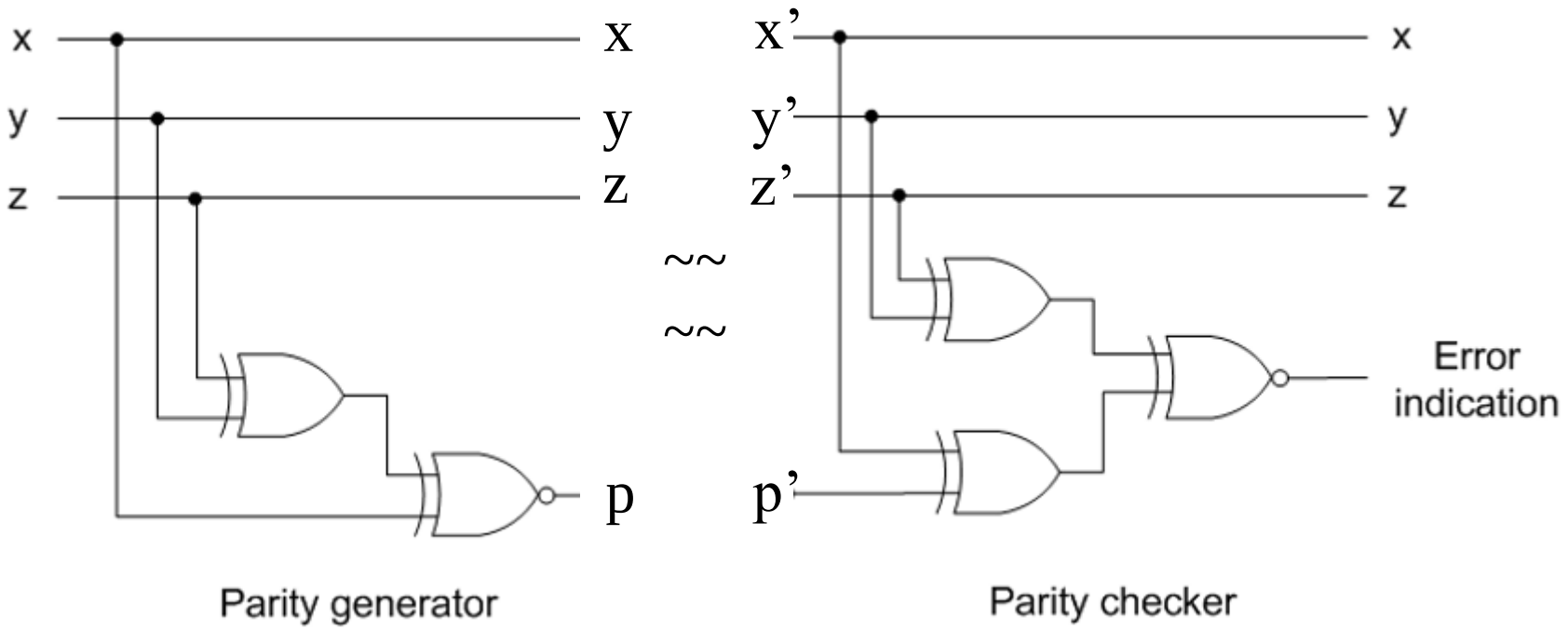
- 패리티 검출기

$$parity\ check = x \oplus y \oplus z \oplus parity\ bit$$

  - 1의 개수가 홀수개 입력되면 출력은 1 : 전송시 에러를 나타냄
  - 1의 개수가 짝수개 입력되면 출력은 0 : 에러없이 전송됨

패리티 발생기 및 검출기

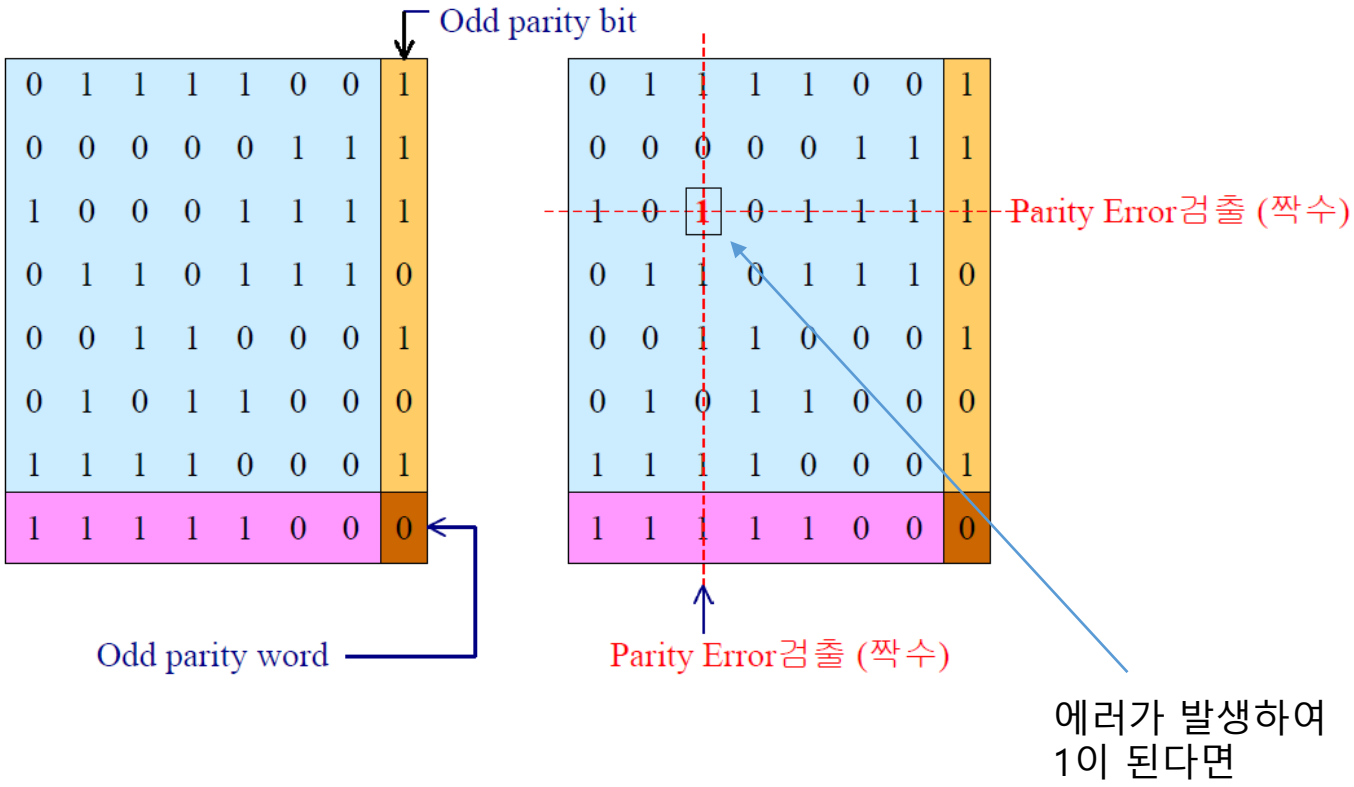

x
y
z
x
y
z
Error
indication
Parity generator
Parity checker
패리티 비트 발생



xyzp를 전송 x'y'z'p'의 패리티 검사

Word parity(1비트 에러 복원 가능)

코드를 한 묶음 단위로 하는 Block단위, 에러 검출은 물론 정정 까지도 가능 하다



2 해밍 코드(Hamming Code)

- 벨(Bell)연구소의 해밍(Hamming)이 개발
- 한 비트의 에러 발생시 위치를 찾아 정정할 수 있는 코드
- 원본 데이터 외에 추가적으로 많은 비트가 필요

추가되는 패리티 비트의 수 계산

$2^p \geq d + p + 1$, p는 추가할 패리티 비트 수, d는 데이터 비트 수

전송하고자 하는 데이터가 8비트라면 $d = 8$

$2^p \geq 8 + p + 1$을 만족하는 p를 찾는다

$p = 1$인 경우, $2^1 \geq 8 + 1 + 1$는 거짓

$p = 2$인 경우, $2^2 \geq 8 + 2 + 1$는 거짓

$p = 3$인 경우, $2^3 \geq 8 + 3 + 1$는 거짓

$p = 4$인 경우, $2^4 \geq 8 + 4 + 1$는 참

따라서 8비트의 데이터에는 4비트의 해밍코드가 필요

패리티 비트의 위치는 1,2,4,8,16,32 등의 2의 거듭제곱 위치에 들어감

비트위치	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	✓		✓		✓		✓		✓		✓	
P_2		✓	✓			✓	✓			✓	✓	
P_4				✓	✓	✓	✓					✓
P_8								✓	✓	✓	✓	✓

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

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

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

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

원본 데이터가 00101110일 때 해밍코드를 생성하는 과정

비트위치	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

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

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

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

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

수신된 데이터가 010111011110일 때 에러 체크하는 과정

비트위치	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}
해밍코드	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 $1 \rightarrow 0$으로 정정												
해밍코드 수정	0	1	0	1	0	1	0	1	1	1	1	0

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

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

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

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

8421로 읽음

예제 2-7 다음 해밍 코드 중 에러가 있는지 검사하여라

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

풀이)

비트위치		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}
해밍코드		1	1	1	1	0	1	0	0	1	0	1	0
P_1	0	1		1		0		0		1		1	
P_2	0		1	1			1	0			0	1	
P_4	0				1	0	1	0					0
P_8	0								0	1	0	1	0

$P_8P_4P_2P_1 = 0000_2$ 이므로 에러없음

3 순환 중복 검사(Cyclic Redundancy Check)

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

❖ CRC 발생기 및 검출기

- 수신 측에서는 수신된 $d + k$비트의 데이터를 키 값으로 나누었을 때 나머지가 0이면 에러가 없는 것이지만, 0이 아니면 에러가 발생한 것으로 판단한다.

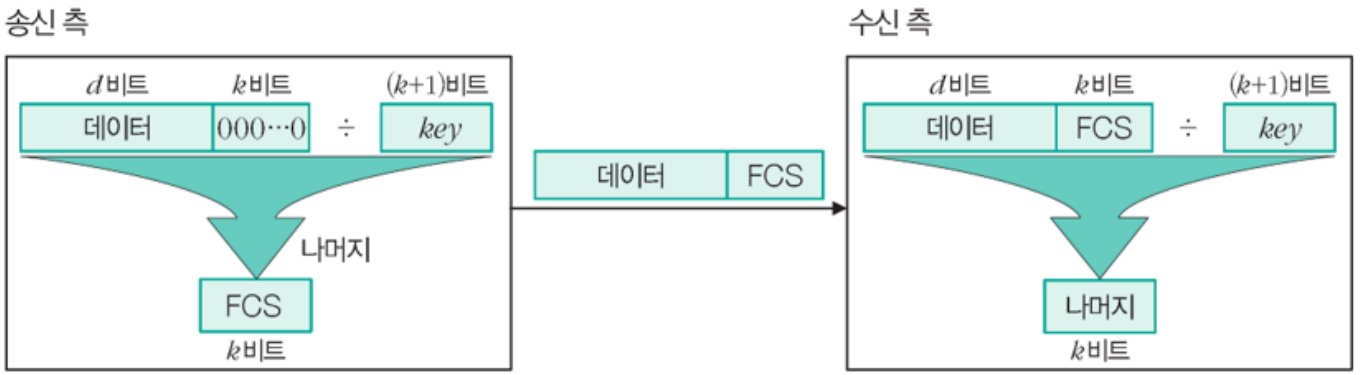


그림 2-8 CRC 발생기와 검사기

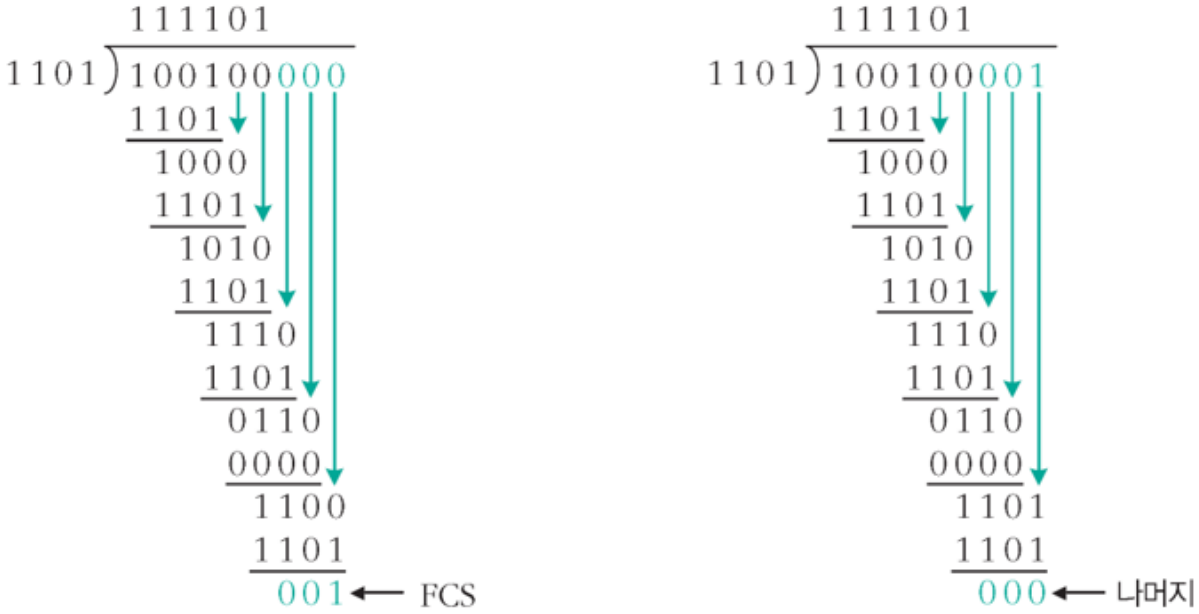
FCS (Frame Check Sequence)

❖ CRC 계산에 사용되는 모듈로-2 연산

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

  - 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 1001000이고, 키 값이 1101인 경우 FCS를 계산하는 예 FCS (Frame Check Sequence)



(a) CRC 발생기에서 계산 과정 (b) 검사기에서 계산 과정

그림 2-9 CRC 발생기와 검사기에서 2진 나눗셈

❖ CRC 하드웨어

- 시프트 레지스터와 XOR 게이트(⊕)를 사용하여 구성할 수 있다.

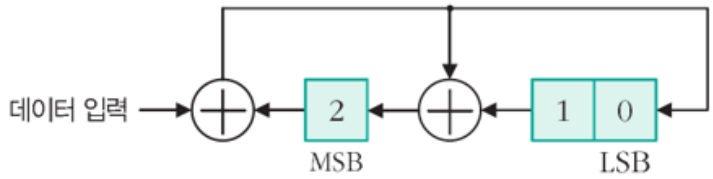


그림 2-10 FCS 생성 회로의 예

FCS (Frame Check Sequence)

- 데이터 100100을 MSB부터 차례로 입력하면, FCS = 001이 된다.

데이터 입력	2	1	0	
	0	0	0	← 초깃값
1	1	0	1	
1 0	1	1	1	
1 0 0	0	1	1	
1 0 0 1	0	1	1	
1 0 0 1 0	1	1	0	
1 0 0 1 0 0	0	0	1	

0번(LSB) =데이터입력 XOR 2(MSB)

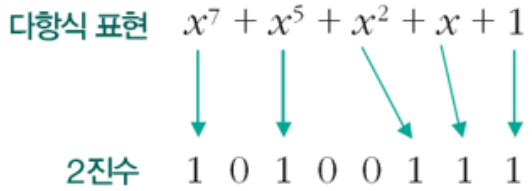
1번 = 0번(LSB) 이전 값

2번(MSB) = 0번들어가는값 XOR 1번 이전값

❖ CRC 테이블 기법

- [그림 2-10]과 같이 비트 단위로 CRC를 계산하면 회로는 간단하지만 처리 속도에 문제가 있다.
- 이를 위해 미리 CRC를 계산하여 메모리에 저장해 놓고 사용한다.

❖ 생성 다항식 (key값은 생성다항식으로 표현)



- 생성 다항식은 에러 검출 능력을 고려하여 결정되는데, 현재 다음 유형이 사용된다
  - CRC-8: $x^8 + x^2 + x + 1$
  - CRC-16: $x^{16} + x^{12} + x^5 + 1$
  - CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

수고하셨습니다!