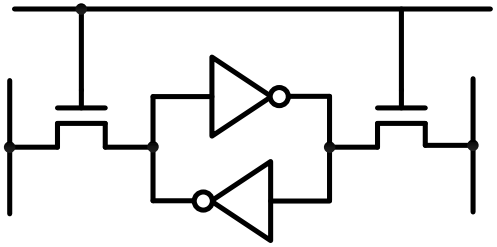
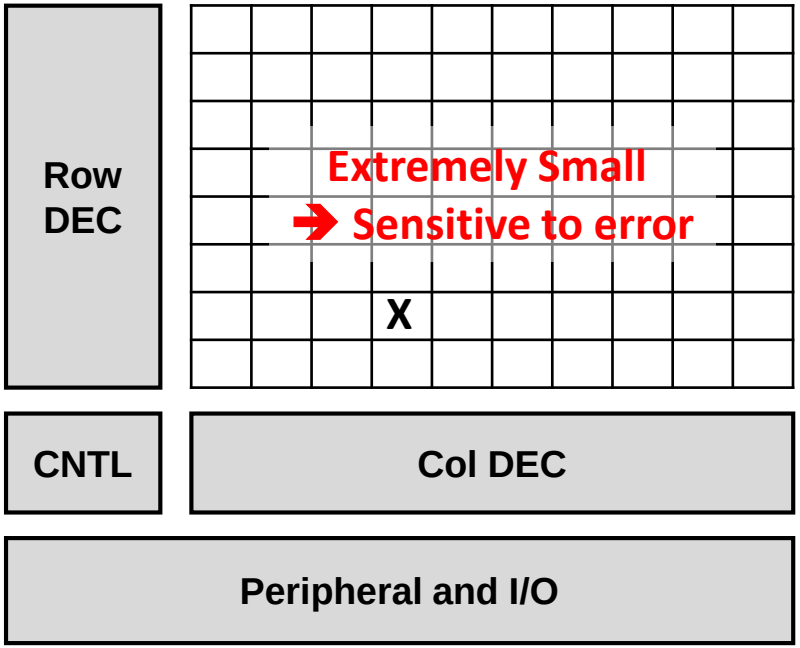
Memory Yield



Memory BC



- There are numerous techniques for enhancing yield in process/device/circuit/architecture/system/algorithm level

Chap 7 (2)
Introduction to Error Detection and Correction Codes

Hanwool Jeong

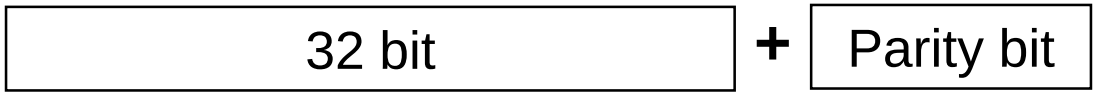
hwjeong@kw.ac.kr

Revisit: Parity Bit for Error Detection

- A parity bit is an extra bit included with a message to make the total number of 1's either even or odd. Consider the following two characters and their even and odd parity:

	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

For 32 bit processor, we can store 32 + parity bit into memory



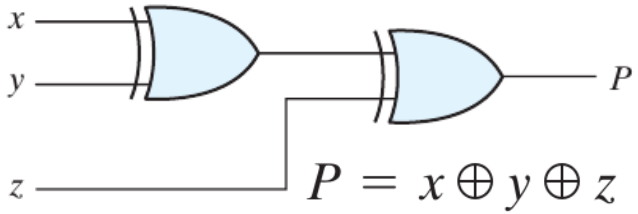
Revisit: Parity Generator & Checker

Even-Parity-Generator Truth Table

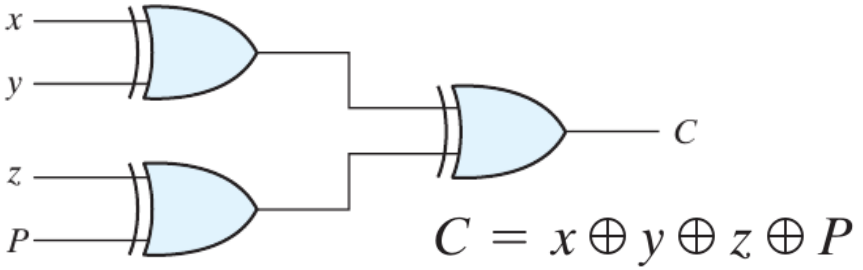
Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



(a) 3-bit even parity generator

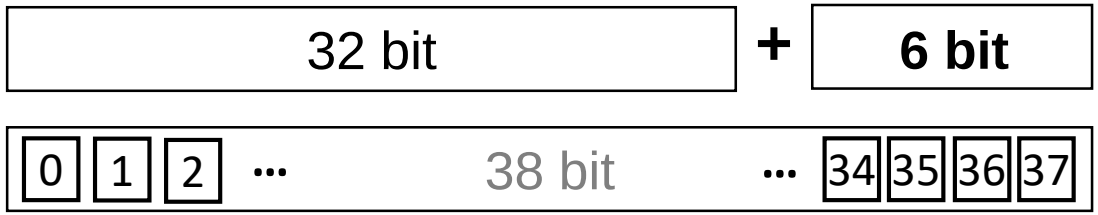


(b) 4-bit even parity checker

Error Detection vs. Error Detection-Correction

- Using even/odd parity bit,
  - The data word is accepted if the parity of the bits read out is correct.
  - If the parity checked results in an inversion, an error is detected, but it cannot be corrected.
- An error-correcting code (ECC) generates multiple parity check bits that are stored with the data word in memory.

➔ If error occurs, the error can be corrected with ECC



How Can You Find Error Bit? Provided 1-bit Error Only at Most

- We can extend even parity (or odd if you want)
- With using multiple parity bits, not only one bit, we can specify the error bit and correct it by flipping it.
- Anyway, an error probability is low

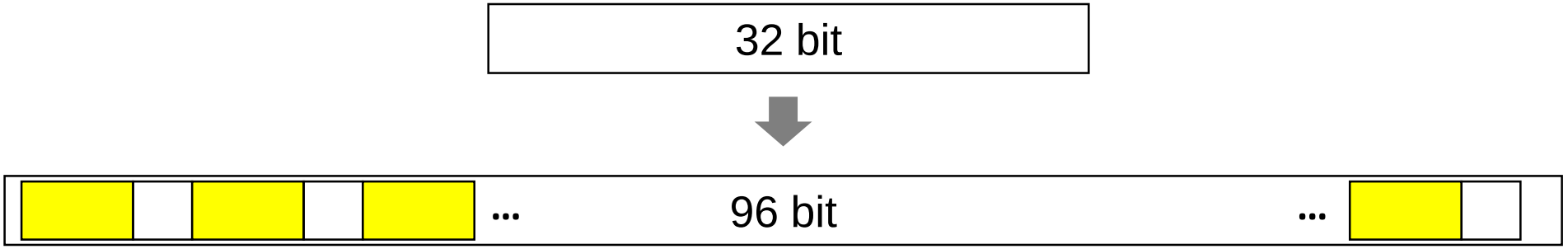
➔How should we send **one-bit** data to enable to be **detected** and **corrected** at the receiving stage?

- Start from simple example and see the possibility

Coding 1		One bit error cases	**Coding 2**		One bit error cases
0	**➔ 00**	**01 10**	**0**	**➔ 000**	**001 010 100**
1	**➔ 11**	**01 10**	**1**	**➔ 111**	**110 101 011**

How About 32 bit?

- Should we extend 32 bit to 96 bit?



- No. We focus on the case of only one bit error.

Why Do We Need Additional 2 bits for One-bit Data; Information Approach

- Revisit this:

Coding 1		One bit error cases	**Coding 2**		One bit error cases
0	**➔ 00**	**01 10**	**0**	**➔ 000**	**001 010 100**
1	**➔ 11**	**01 10**	**1**	**➔ 111**	**110 101 011**

- Suppose that we use coding 1,

B_1	B_0

- We should represent **6 cases**.
  - No error for 0 / No error for 1
  - B_0 is wrong for 0 / B_0 is wrong for 1
  - B_1 is wrong for 0 / B_1 is wrong for 1

➔ **We cannot do this w/ 2 bits**

How About Coding 2?

- That is,



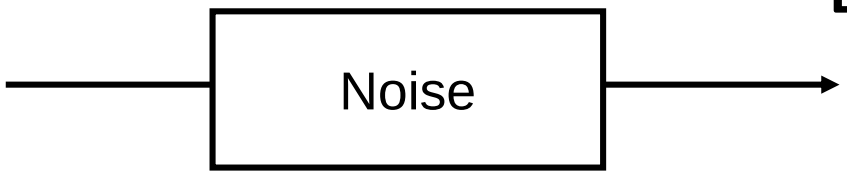
- We can represent **8 cases**.
  - No error for 0 / No error for 1
  - B_2 is wrong for 0 / B_2 is wrong for 1
  - B_1 is wrong for 0 / B_1 is wrong for 1
  - B_0 is wrong for 0 / B_0 is wrong for 1

➔ **Now, it is possible**

How Can We Establish Error Correction System?

Original Data

B_2	B_1	X_0



Data w/ Error Possibility

B_2	B_1	B_0

We should **check** which bit should be changed

B_2	B_1	B_0	C_1	C_0
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**1**	**0**
0	1	1	**1**	**1**
1	0	0	**1**	**1**
1	0	1	**1**	**0**
1	1	0	**0**	**1**
1	1	1	**0**	**0**

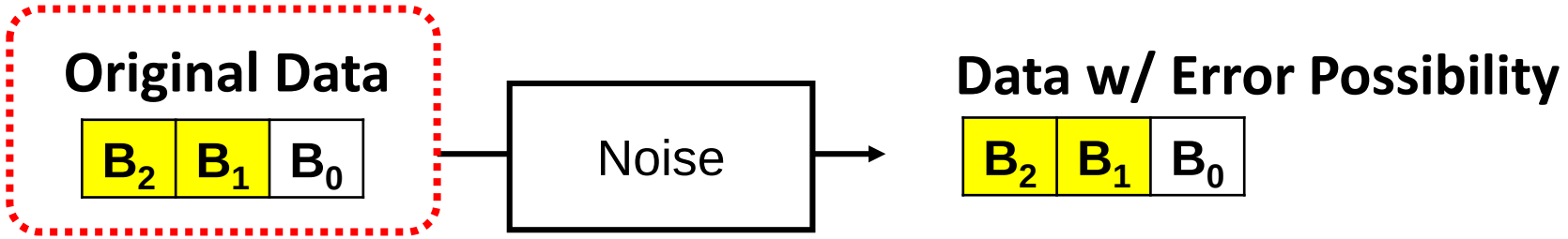
	C_1C_0
✓ No error	00
✓ B_0 is wrong	01
✓ B_1 is wrong	10
✓ B_2 is wrong	11

$C_0 = XOR(B_0, B_2)$ $C_1 = XOR(B_1, B_2)$ ➔ Called check bits or Syndrome signals

Extend to Larger Bits

	$C_2C_1C_0$
✓ No error	000
✓ B_0 is wrong	001
✓ B_1 is wrong	010
✓ B_2 is wrong	011
✓ B_3 is wrong	100
✓ B_4 is wrong	101
✓ B_5 is wrong	110
✓ B_6 is wrong	111

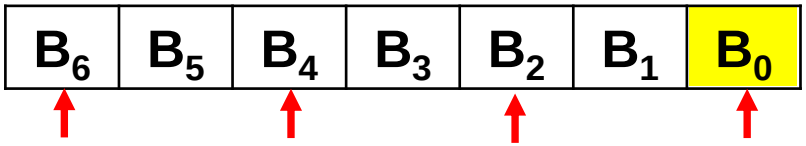
- What if we can generate $C_2C_1C_0$?
- There will exist possible config. for 7bits, 15bits, 31bits, …
  - For 7 bit, 4bit data + 3 bit parity bit. Why?
- How should we make parity bits? What is the rule?



Parity Rule?

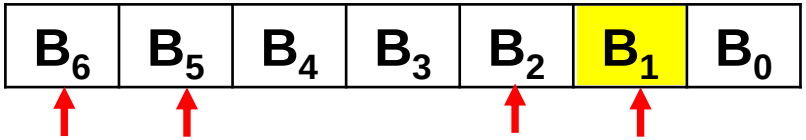
- Focus on C_0,

How about generating B_0 as the parity bit to be XOR(B_6,B_4,B_2,B_0) = 0?



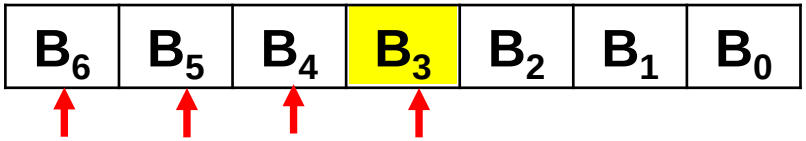
- Focus on C_1,

How about generating B_1 as the parity bit to be XOR(B_6,B_5,B_2,B_1) = 0?



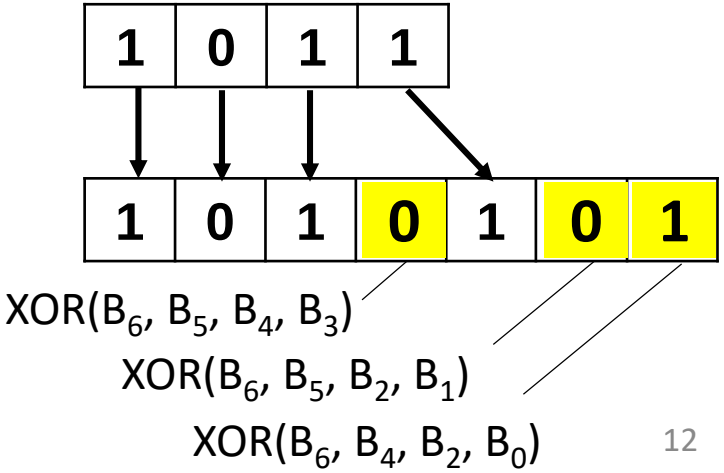
- Focus on C_2,

How about generating B_3 as the parity bit to be XOR(B_6,B_5,B_4,B_3) = 0?



- ✓ No error
- ✓ B_0 is wrong
- ✓ B_1 is wrong
- ✓ B_2 is wrong
- ✓ B_3 is wrong
- ✓ B_4 is wrong
- ✓ B_5 is wrong
- ✓ B_6 is wrong

C_2	C_1	C_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



Parity Generation Rule; = Also Check Rule! XOR Grouping

Bit Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Group 0	■		■		■		■		■		■		■		■
Group 1		■	■			■	■			■	■			■	■
Group 2				■	■	■	■					■	■	■	■
Group 3								■	■	■	■	■	■	■	■

- To transfer 1 bit data error-free ➔ 3 bits are required
- To transfer 4 bit data 1 bit error-free ➔ 7 bits are required
- To transfer 11 bit data 1 bit error-free ➔ 15 bits are required

Hamming Code

- Devised by R. W. Hamming
- In the Hamming code, k parity bits are added to an n -bit data word, forming a new word of n + k bits.
- Consider for example,
  - 8 bit data word 11000100
  - Adding 4 parity bits to the word, forming 12bits as follows

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

- The 4 parity bits, P_1 , P_2 , P_4 , and P_8 , are in positions 1, 2, 4, and 8, respectively.

Parity Bit Calculation

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

$$P_1 = \text{XOR of bits } (3, 5, 7, 9, 11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$P_2 = \text{XOR of bits } (3, 6, 7, 10, 11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4 = \text{XOR of bits } (5, 6, 7, 12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$P_8 = \text{XOR of bits } (9, 10, 11, 12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

	0	0	1	1	1	0	0	1	0	1	0	0
Bit position:	1	2	3	4	5	6	7	8	9	10	11	12

Checking Procedure

$$C_1 = \text{XOR of bits } (1, 3, 5, 7, 9, 11)$$

$$C_2 = \text{XOR of bits } (2, 3, 6, 7, 10, 11)$$

$$C_4 = \text{XOR of bits } (4, 5, 6, 7, 12)$$

$$C_8 = \text{XOR of bits } (8, 9, 10, 11, 12)$$

- Please compare below

$$P_1 = \text{XOR of bits } (3, 5, 7, 9, 11)$$

$$P_2 = \text{XOR of bits } (3, 6, 7, 10, 11)$$

$$P_4 = \text{XOR of bits } (5, 6, 7, 12)$$

$$P_8 = \text{XOR of bits } (9, 10, 11, 12)$$

Error Case Example

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

Hamming Code Bit Range

- In general, the Hamming code consists of k check bits and n data bits, for a total of n + k bits.
- The syndrome value C consists of k bits and 2^k - 1 values can be used to uniquely describe a bit in error.
- Thus, the range of k must be equal to or greater than n + k

$$2^k - 1 \geq n + k$$

Range of Data Bits for k Check Bits

Number of Check Bits, *k*	Range of Data Bits, *n*
3	2–4
4	5–11
5	12–26
6	27–57
7	58–120

Single-Error Correction w/ Double-Error Detection

- The Hamming code can detect and correct only a single error.
- By adding another parity bit to the coded word, the Hamming code can be used to correct a single error and detect double errors.
- The previous 12-bit coded word becomes 001110010100P_{13}, while P_{13} is just even parity bits.
- When 13-bit word is read from memory, the check bits are evaluated, as is the parity P over the entire 13 bits Then,

If $C = 0$ and $P = 0$, no error occurred.
If $C \neq 0$ and $P = 1$, a single error occurred that can be corrected.
If $C \neq 0$ and $P = 0$, a double error occurred that is detected, but that cannot be corrected.
If $C = 0$ and $P = 1$, an error occurred in the P_{13} bit.

Checkpoints

- ✓Error correction & Detection
- ✓How to generate Parity bits & Check bits (syndrome)
- ✓Hamming code
- ✓Single error correction + double error detection