# LE 517
# Data Communications and Networks

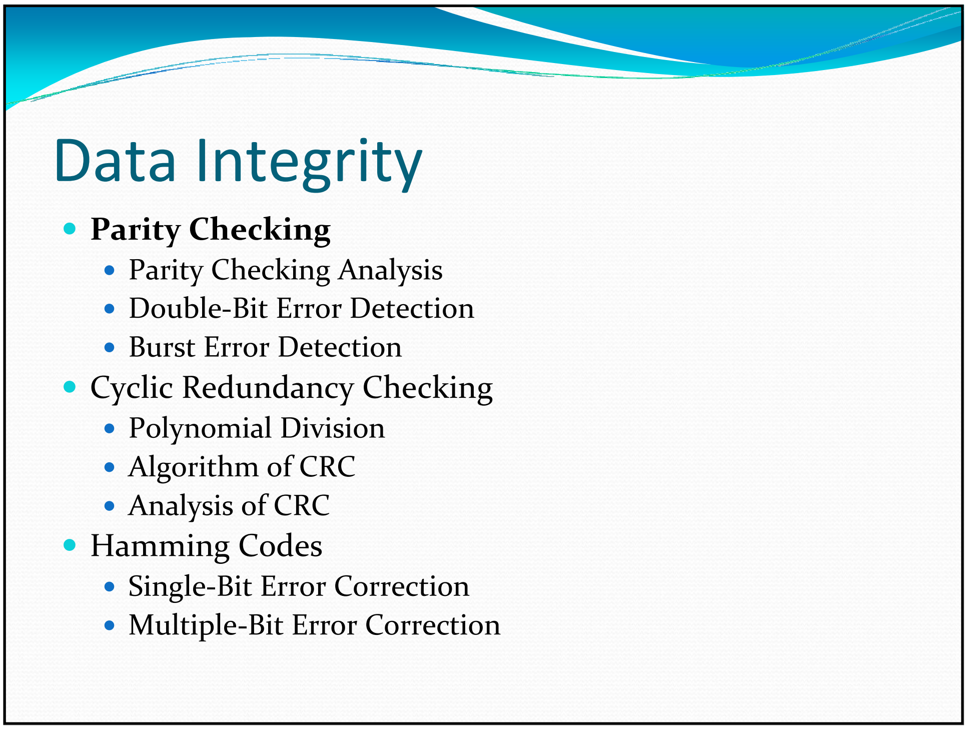
Week 6:- Data Integrity

By

Dr. Piya Techateerawat

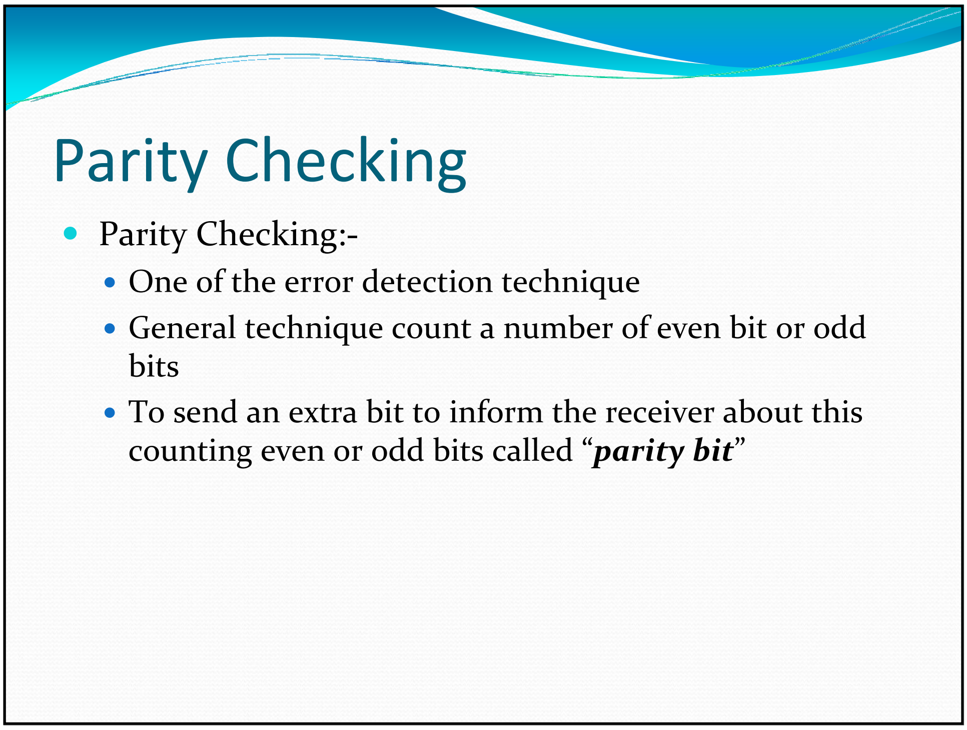
## Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction



# Data Integrity

- **Parity Checking**
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction



# Parity Checking

- Parity Checking:-
  - One of the error detection technique
  - General technique count a number of even bit or odd bits
  - To send an extra bit to inform the receiver about this counting even or odd bits called “***parity bit***”

## Data Integrity

- Parity Checking
  - **Parity Checking Analysis**
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction

## Parity Checking Analysis

- Parity checking initially develop to detect single bit error.

11010100   0   what if error show as 11011000  0

↑

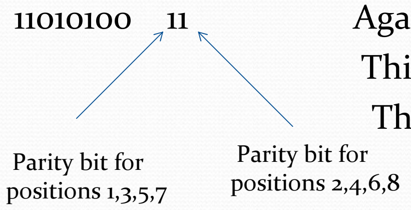
Parity bit is 0 to inform that number of 1 is even.

# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - **Double-Bit Error Detection**
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction

# Double-Bit Error Detection

- An improve version of error detection.
- Adding parity checking to detect multiple errors.



11010100 11

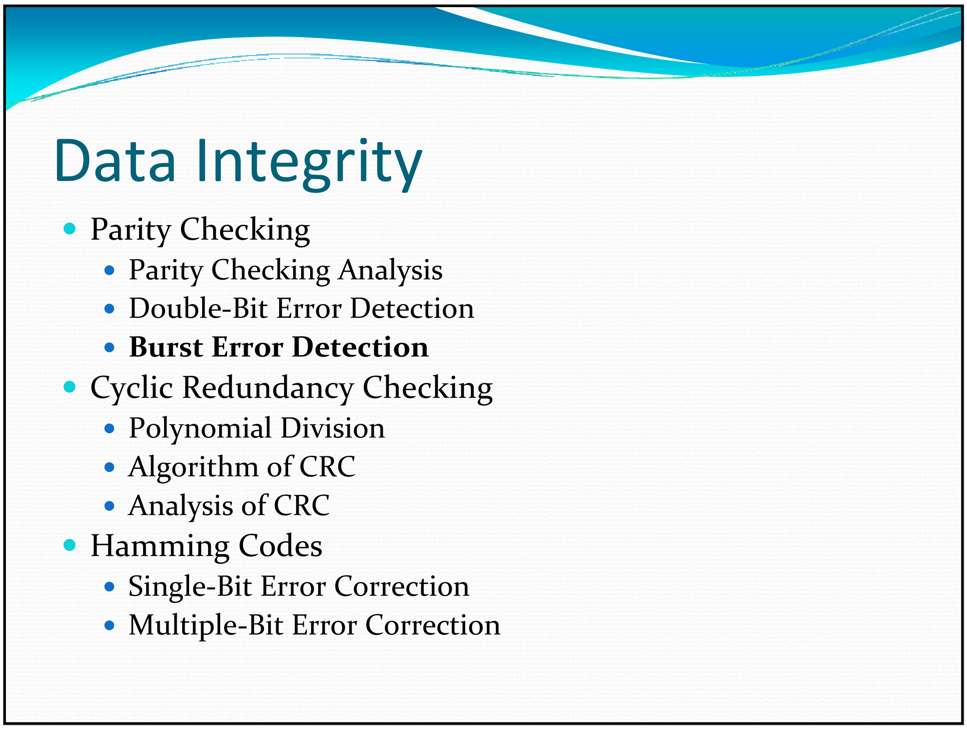
Parity bit for positions 1,3,5,7

Parity bit for positions 2,4,6,8

Again, if it has error 11011000 11

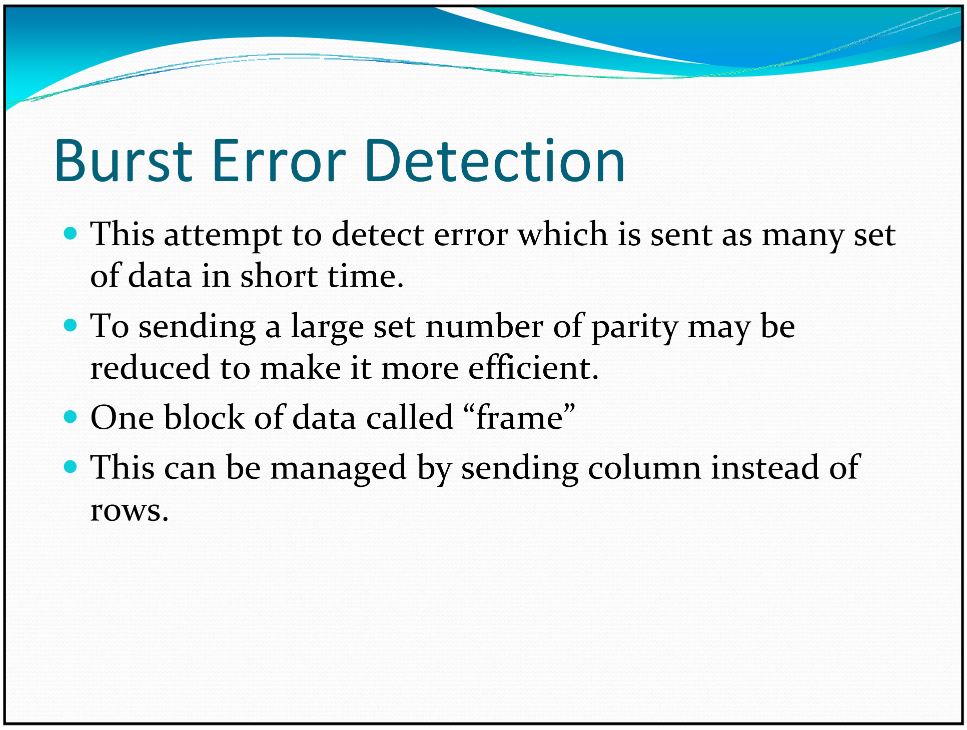
This will be detected.

This is detected all case ?



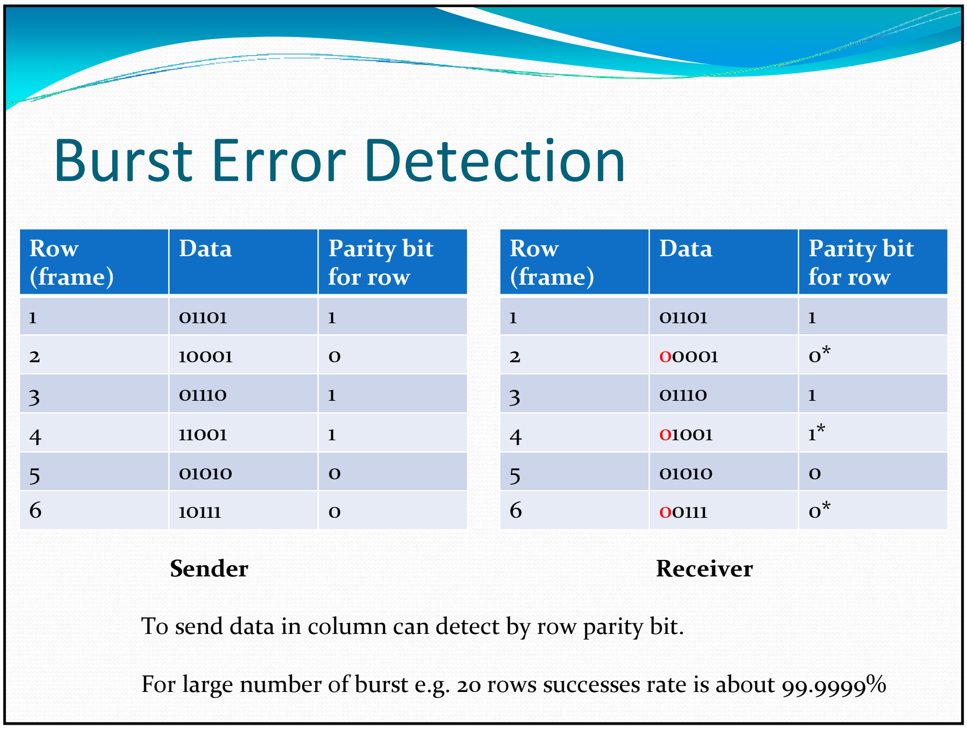
# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - **Burst Error Detection**
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction



# Burst Error Detection

- This attempt to detect error which is sent as many set of data in short time.
- To sending a large set number of parity may be reduced to make it more efficient.
- One block of data called “frame”
- This can be managed by sending column instead of rows.



# Burst Error Detection

| Row (frame) | Data | Parity bit for row |
|---|---|---|
| 1 | 01101 | 1 |
| 2 | 10001 | 0 |
| 3 | 01110 | 1 |
| 4 | 11001 | 1 |
| 5 | 01010 | 0 |
| 6 | 10111 | 0 |

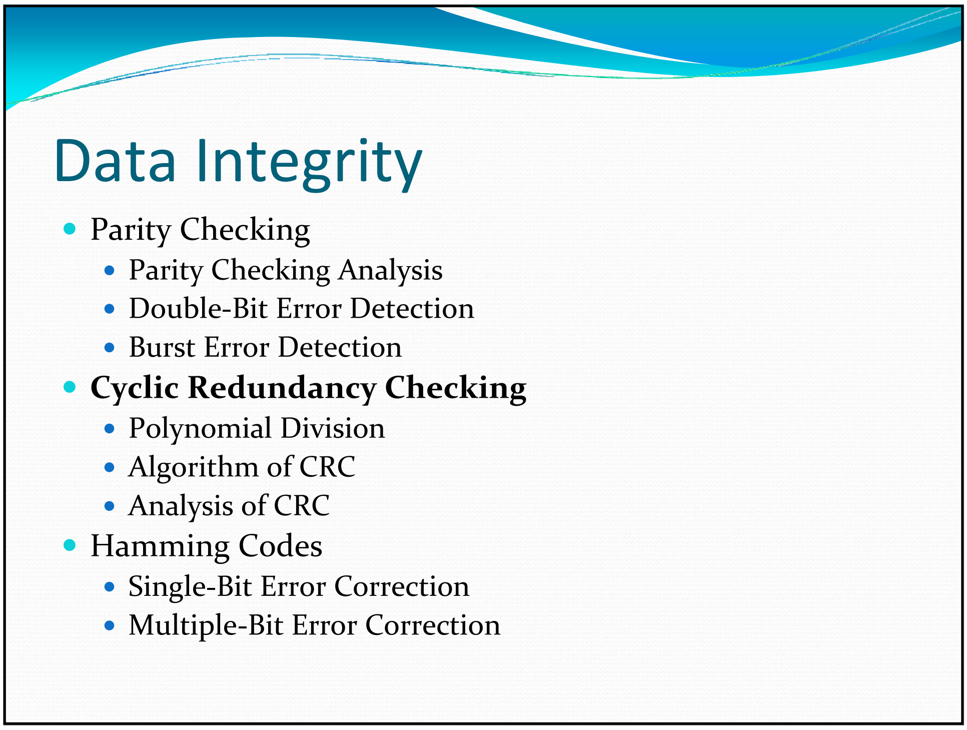
**Sender**

| Row (frame) | Data | Parity bit for row |
|---|---|---|
| 1 | 01101 | 1 |
| 2 | 00001 | 0* |
| 3 | 01110 | 1 |
| 4 | 01001 | 1* |
| 5 | 01010 | 0 |
| 6 | 00111 | 0* |

**Receiver**

To send data in column can detect by row parity bit.

For large number of burst e.g. 20 rows successes rate is about 99.9999%



# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- **Cyclic Redundancy Checking**
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction

# Cyclic Redundancy Checking

- $b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots + b_1x + b_0$

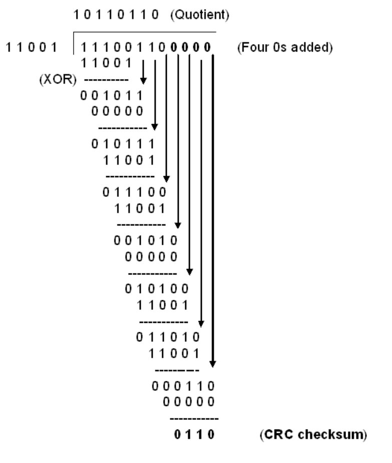
- For example, bit string 10010101110 can be interpreted as

$$x^{10} + x^7 + x^5 + x^3 + x^2 + x^1$$

# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - **Polynomial Division**
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction

# Polynomial Division



```
              1 0 1 1 0 1 1 0   (Quotient)
           ____________________
  1 1 0 0 1 | 1 1 1 0 0 1 1 0 0 0 0 0     (Four 0s added)
              1 1 0 0 1
       (XOR)  ----------
              0 0 1 0 1 1
                0 0 0 0 0
                -----------
                0 1 0 1 1 1
                  1 1 0 0 1
                  -----------
                  0 1 1 1 0 0
                    1 1 0 0 1
                    -----------
                    0 0 1 0 1 0
                      0 0 0 0 0
                      -----------
                      0 1 0 1 0 0
                        1 1 0 0 1
                        -----------
                        0 1 1 0 1 0
                          1 1 0 0 1
                          ---------
                          0 0 0 1 1 0
                            0 0 0 0 0
                            -----------
                            0 1 1 0        (CRC checksum)
```

- Mod 2 Arithmetic

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$

$0 - 0 = 0$

$0 - 1 = 1$

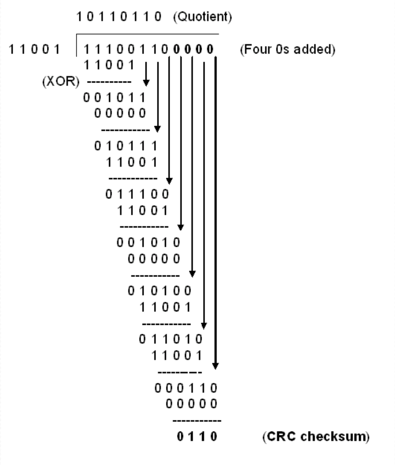
$1 - 0 = 1$

$1 - 1 = 0$

# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - **Algorithm of CRC**
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction

# Algorithm of CRC



- Sender:
  - Original Data: 11100110
  - Transmitted Data: 111001100110
- Receiver:
  - Find mod 2 division of 111001100110 and 11001
  - Remainder must be “0”

# Data Integrity

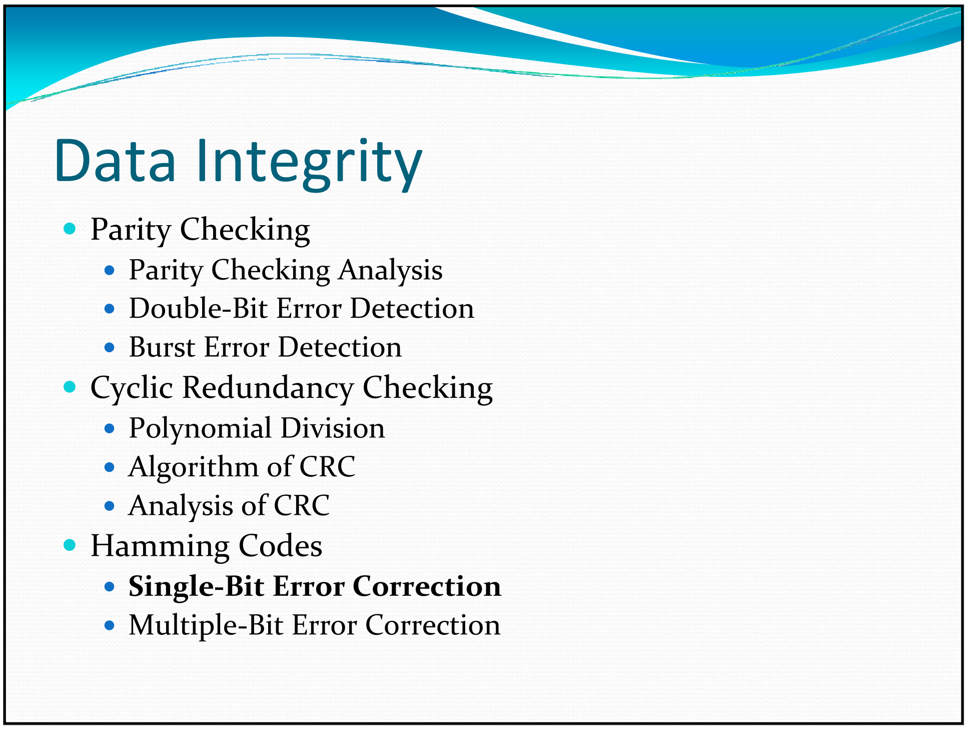
- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - **Analysis of CRC**
- Hamming Codes
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction

# Analysis of CRC

- Is that possible that damage frame cannot be detected?
  - If x is not a factor of G(x), then all burst errors having length smaller than or equal to the degree of G(x) are detected.
  - If x +1 is a factor of G(x), then all burst errors damaging an odd number of bits are detected.
  - All burst errors for length > Degree of G(x) + 1 the probability is better than 7th 9 or 99.9999998%

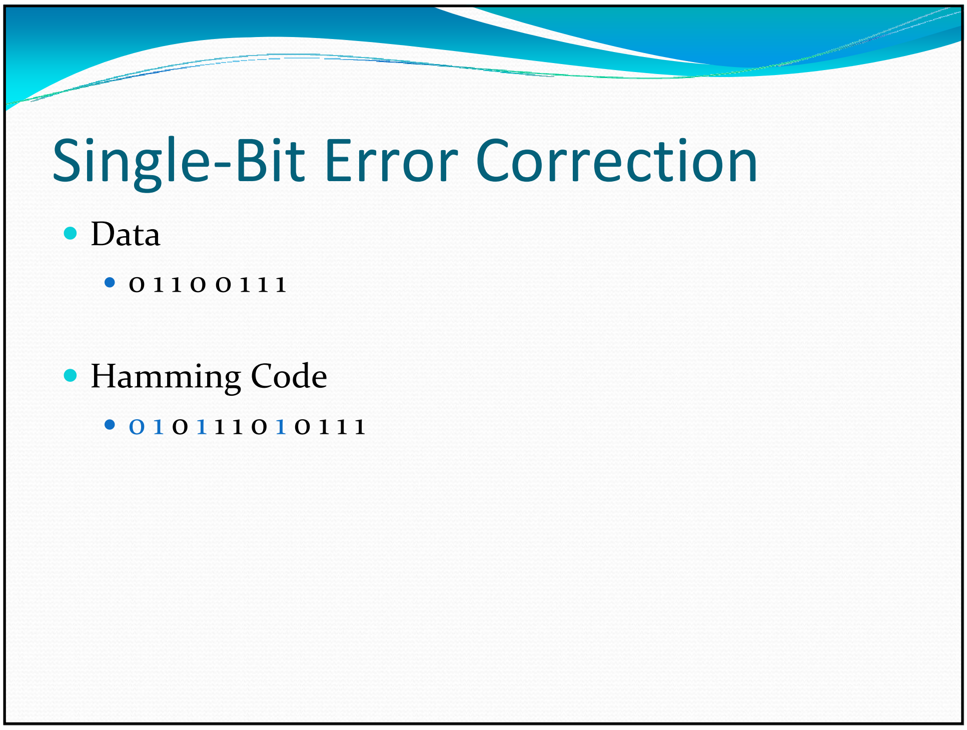
# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- **Hamming Codes**
  - Single-Bit Error Correction
  - Multiple-Bit Error Correction



# Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - **Single-Bit Error Correction**
  - Multiple-Bit Error Correction



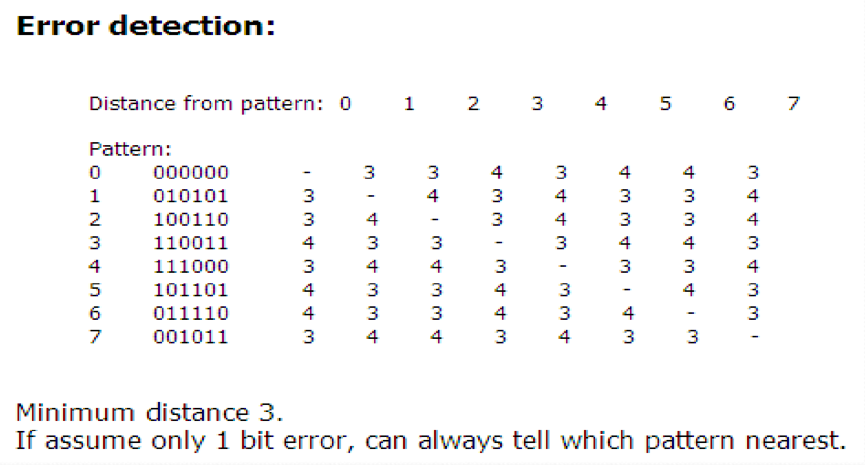
# Single-Bit Error Correction

- Data
  - 0 1 1 0 0 1 1 1
- Hamming Code
  - 0 1 0 1 1 1 0 1 0 1 1 1

# Single-Bit Error Correction

- Position 1 (n=1): skip 0 bits (0=n–1), check 1 bit (n), skip 1 bit (n), check 1 bit (n), skip 1 bit (n), etc. (1,3,5,7,9,11,13,15,…)
- Position 2 (n=2): skip 1 bit (1=n–1), check 2 bits (n), skip 2 bits (n), check 2 bits (n), skip 2 bits (n), etc. (2,3,6,7,10,11,14,15,…)
- Position 4 (n=4): skip 3 bits (3=n–1), check 4 bits (n), skip 4 bits (n), check 4 bits (n), skip 4 bits (n), etc. (4,5,6,7,12,13,14,15,20,21,22,23,…)
- Position 8 (n=8): skip 7 bits (7=n–1), check 8 bits (n), skip 8 bits (n), check 8 bits (n), skip 8 bits (n), etc. (8-15,24-31,40-47,…)
- Position 16 (n=16): skip 15 bits (15=n–1), check 16 bits (n), skip 16 bits (n), check 16 bits (n), skip 16 bits (n), etc. (16-31,48-63,80-95,…)
- Position 32 (n=32): skip 31 bits (31=n–1), check 32 bits (n), skip 32 bits (n), check 32 bits (n), skip 32 bits (n), etc. (32-63,96-127,160-191,…)
- General rule for position *n*: skip *n*–1 bits, check *n* bits, skip *n* bits, check *n* bits… Position 1 (n=1): skip 0 bits (0=n–1), check 1 bit (n), skip 1 bit (n), check 1 bit (n), skip 1 bit (n), etc. (1,3,5,7,9,11,13,15,…)
- Position 2 (n=2): skip 1 bit (1=n–1), check 2 bits (n), skip 2 bits (n), check 2 bits (n), skip 2 bits (n), etc. (2,3,6,7,10,11,14,15,...)
- Position 4 (n=4): skip 3 bits (3=n–1), check 4 bits (n), skip 4 bits (n), check 4 bits (n), skip 4 bits (n), etc. (4,5,6,7,12,13,14,15,20,21,22,23,…)
- Position 8 (n=8): skip 7 bits (7=n–1), check 8 bits (n), skip 8 bits (n), check 8 bits (n), skip 8 bits (n), etc. (8-15,24-31,40-47,...)
- Position 16 (n=16): skip 15 bits (15=n–1), check 16 bits (n), skip 16 bits (n), check 16 bits (n), skip 16 bits (n), etc. (16-31,48-63,80-95,…)
- Position 32 (n=32): skip 31 bits (31=n–1), check 32 bits (n), skip 32 bits (n), check 32 bits (n), skip 32 bits (n), etc. (32-63,96-127,160-191,...)
- General rule for position *n*: skip *n*–1 bits, check *n* bits, skip *n* bits, check *n* bits…

# Single-Bit Error Correction



**Error detection:**

| Distance from pattern: | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| Pattern: | | | | | | | | | |
| 0 | 000000 | - | 3 | 3 | 4 | 3 | 4 | 4 | 3 |
| 1 | 010101 | 3 | - | 4 | 3 | 4 | 3 | 3 | 4 |
| 2 | 100110 | 3 | 4 | - | 3 | 4 | 3 | 3 | 4 |
| 3 | 110011 | 4 | 3 | 3 | - | 3 | 4 | 4 | 3 |
| 4 | 111000 | 3 | 4 | 4 | 3 | - | 3 | 3 | 4 |
| 5 | 101101 | 4 | 3 | 3 | 4 | 3 | - | 4 | 3 |
| 6 | 011110 | 4 | 3 | 3 | 4 | 3 | 4 | - | 3 |
| 7 | 001011 | 3 | 4 | 4 | 3 | 4 | 3 | 3 | - |

Minimum distance 3.
If assume only 1 bit error, can always tell which pattern nearest.

## Single-Bit Error Correction

- If received
  - 0 1 0 1 0 1 0 1 0 1 1 1
  - $P_1$ and $P_3$ fails….
    - Sums error position 0101 = 5

## Data Integrity

- Parity Checking
  - Parity Checking Analysis
  - Double-Bit Error Detection
  - Burst Error Detection
- Cyclic Redundancy Checking
  - Polynomial Division
  - Algorithm of CRC
  - Analysis of CRC
- Hamming Codes
  - Single-Bit Error Correction
  - **Multiple-Bit Error Correction**

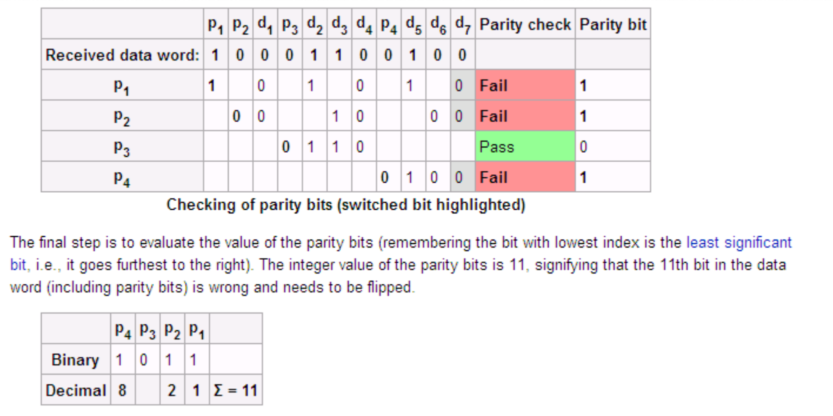
# Multiple-Bit Error Correction

- **General algorithm**
- Although any number of algorithms can be created, the following general algorithm positions the parity bits at powers of two to ease calculation of which bit was flipped upon detection of incorrect parity.
- All bit positions that are powers of two are used as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)
- All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc

# Multiple-Bit Error Correction

- Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
  - Position 1 (n=1): skip 0 bit (0=n−1), check 1 bit (n), skip 1 bit (n), check 1 bit (n), skip 1 bit (n), etc. (1,3,5,7,9,11,13,15,…)
  - Position 2 (n=2): skip 1 bit (1=n−1), check 2 bits (n), skip 2 bits (n), check 2 bits (n), skip 2 bits (n), etc. (2,3,6,7,10,11,14,15,…)
  - Position 4 (n=4): skip 3 bits (3=n−1), check 4 bits (n), skip 4 bits (n), check 4 bits (n), skip 4 bits (n), etc. (4,5,6,7,12,13,14,15,20,21,22,23,…)
  - Position 8 (n=8): skip 7 bits (7=n−1), check 8 bits (n), skip 8 bits (n), check 8 bits (n), skip 8 bits (n), etc. (8-15,24-31,40-47,…)

# Multiple-Bit Error Correction



| | $p_1$ | $p_2$ | $d_1$ | $p_3$ | $d_2$ | $d_3$ | $d_4$ | $p_4$ | $d_5$ | $d_6$ | $d_7$ | Parity check | Parity bit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Received data word:** | **1** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** | | |
| $p_1$ | 1 | | 0 | | 1 | | 0 | | 1 | | 0 | Fail | **1** |
| $p_2$ | | 0 | 0 | | | 1 | 0 | | | 0 | 0 | Fail | **1** |
| $p_3$ | | | | 0 | 1 | 1 | 0 | | | | | Pass | 0 |
| $p_4$ | | | | | | | | 0 | 1 | 0 | 0 | Fail | **1** |

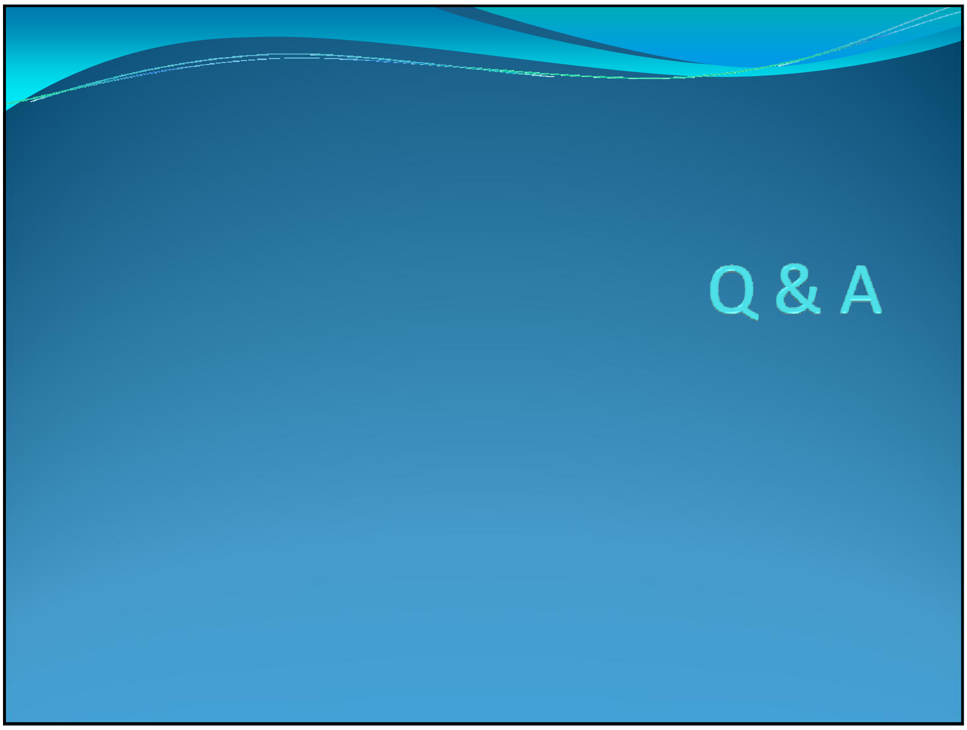
**Checking of parity bits (switched bit highlighted)**

The final step is to evaluate the value of the parity bits (remembering the bit with lowest index is the least significant bit, i.e., it goes furthest to the right). The integer value of the parity bits is 11, signifying that the 11th bit in the data word (including parity bits) is wrong and needs to be flipped.

| | $p_4$ | $p_3$ | $p_2$ | $p_1$ | |
|---|---|---|---|---|---|
| **Binary** | 1 | 0 | 1 | 1 | |
| **Decimal** | 8 | | 2 | 1 | Σ = 11 |

# Multiple-Bit Error Correction

- If you want to play, how does this protocol work.
- http://www.ee.unb.ca/cgi-bin/tervo/hamming.pl?L=6&D=4&X=+Receive+&T=10 0000



# Q & A