

# Error Detection and Correction

CIT 595  
Spring 2008

## Motivation

- Data that is either transmitted over communication channel (e.g. bus) or stored in memory is not completely *error free*
- Error can be caused by:
  1. Transmission Errors
    - Signal distortion or attenuation
    - E.g. sender and receiver out of syn, can happen if clocks are not synchronized – systems distributed over network
  2. Storage Errors
    - DRAM memory cell contents can change spuriously due to some electromagnetic interference
    - In magnetic storage devices such as disks, magnetic flux density increases could cause one or more bits to flip (change that value)

CIT 595

2

## Error Detection and Correction

- Error *detection* is the ability to detect errors
- Error *correction* has an additional feature that enables identification and correction of the errors
- Error detection always precedes error correction
- Both can be achieved by having *extra/redundant/check* bits in addition to data deduce that there is an error
  - Original Data is encoded with the redundant bit(s)
  - New data formed is known as *code word*

CIT 595

3

## Parity

- The simplest and oldest error detection method
- A binary digit called *parity* is used to indicate whether the number of bits with "1" in a given set of bits is even or odd
  - The parity bit is then appended to original data
- Usually used to detect transmission error
  - *Sender* adds the parity bit to existing data bits before transmission
  - *Receiver* checks for the expected parity, If wrong parity found, the received data is discarded and retransmission is requested

CIT 595

4

### Parity type: Even

- *Forced* an even number of one's on total data sent
  - 000 0001 → 1000 0001
  - 000 0011 → 0001 0001
- Generating even parity bit is just an XOR function
- Data Received Examples:
  - 0111 1111 - incorrect
  - 1000 0000 - incorrect
  - 1000 0001 - valid
- Note that error could be in data or parity
- Not entirely fool proof

CIT 595

5

### Parity type: Odd

- *Forced* an odd number of one's
  - 000 0001 → 0 000 0001
  - 000 0011 → 1 000 0011
- Odd parity is generated using a XNOR function

CIT 595

6

### Limitations of Parity

- Cannot determine which bit position has a problem
- If 001 is encountered, it is not a valid code-word and hence error is detected
- The correct code-word could either be 101 or 011 but we cannot tell

Data Word	Parity Code (Even)	Code Word
00	0	000
01	1	011
10	1	101
11	0	110

000	100
001	101
010	110
011	111

8 possible, only 4 correct code-words

CIT 595

7

### Limitations of Parity (contd..)

- What happens if the code word is subjected to 2-bit error?
  - E.g. 011 became 000 while transmission
  - According to parity scheme, *no error* is detected but error!!
- In general
  - If an *odd number of bits* (including the parity bit) are changed from a set of bits
  - Then parity bit will be incorrect and will thus indicate that an error has occurred

CIT 595

8

## Hamming Distance and Error Detection

- Hamming Distance = # of bit positions in which 2 code words differ
  - E.g. 10001001 and 10110001 have distance of 3
- If distance is  $d$ , then  $d$ -single bit errors are required to convert any one valid code into another
  - Implying that this error would not be detected
- In previous parity example (slide 7)
  - Could detect 1-bit error as 4 code words had hamming distance = 2
  - But could not detect 2-bit error
- In general, to detect  $k$ -single bit error, minimum hamming distance  $D(\min) = k + 1$ 
  - Hence we need code words that have  $D(\min) = 2 + 1 = 3$  to detect 2-bit errors

CIT 595

9

## Hamming Distance and Error Correction

- If there is a *larger hamming distance* between valid code words
  - Then we may be able to determine which valid codeword was intended
- Suppose a code needs just 2 different values, and we use:
  - One valid value = 0000 0000 and the other = 1111 1111
  - Then distance between these is 8
- Suppose we got 2 bit changes so that:
  - 0000 0000 became 0011 0000
- Can we determine what the transmitted value was?
  - Was it more likely 0000 0000 or 1111 1111?

CIT 595

10

## Hamming Distance and Error Correction (contd..)

- The greater the distance between valid code words, the easier it is to figure what the correct codeword was
- Requires additional redundant bits (> 1 parity bit) to chose code words that are far apart
- $D(\min) = 2k + 1$  is required for correcting  $k$ -errors

CIT 595

11

## Hamming Code (HC)

- Hamming Code is type of Error Correcting Code (ECC)
  - Provides error detection and correction mechanism
- Adopt parity concept, but have more than one parity bit
- In general hamming code is code word of  $n$  bits with  $m$  data bits and  $r$  parity (or check bits)
  - i.e.  $n = m + r$
- Can detect  $D(\min) - 1$  errors
- Can correct  $\left\lfloor \frac{D(\min) - 1}{2} \right\rfloor$  errors
- Hence to correct  $k$  errors, need  $D(\min) = 2k + 1$ 
  - Need a least a distance of 3 to correct a single bit error

CIT 595

12

### Determining # of Parity bits for single-bit correction

- Hamming Code for single-bit error correction is the most commonly used
  - Experiments (IBM study) show 98% time there are single-bit errors
- Need determine  $r$  for  $m$ -data bits that provides code words of  $n$ -bits that has single-bit correction capabilities

CIT 595

13

### Determining # of Parity bits for single-bit correction

- An error could occur in any of the  $n$  bits of the code word, so each code word can be associated with  $n$  erroneous words (at a hamming distance of 1)
  - E.g. Previous Parity Example (slide 7)
  - 000 can become 001 or 010 or 100 due to single bit error
- Therefore, we have  $n + 1$  bit patterns for each code word: one valid code word, and  $n$  erroneous words
- This gives us the inequality:  $(n + 1) \times 2^m \leq 2^n$ 
  - $2^m$  is the number of legal code

CIT 595

14

### Hamming Code: Determining Parity bits for single-bit correction

- Because  $n = m + r$ , we can rewrite the inequality as:  
 $(m + r + 1) \times 2^m \leq 2^{m+r}$  or  $(m + r + 1) \leq 2^r$
- This inequality gives us a *lower limit* on the number of parity bits that we need in our code words
- Example: Suppose we have data words of length  $m = 4$   
 $(4 + r + 1) \leq 2^r$ 
  - Implies that  $r$  must be greater than or equal to 3
  - To build a code with 4-bit data words that will correct single-bit errors, we must add 3 check bits

CIT 595

15

### Hamming Algorithm for Code Generation

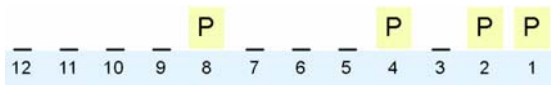
- Hamming Algorithm provides a technique to design codes for single-bit error correction
- We will look at an example generating code word
  - Steps/rules are provided in the book section 2.7.2

CIT 595

16

### Example: Generation of 12-bit Code word

- 8-bit data needs 4 parity bits, total of 12-bit code word
- Using our code words of length 12, number each bit position starting with 1 in the low-order bit
- Each bit position corresponding to power of 2 will be occupied by a parity or check bit
- All other bit positions are for the data to be encoded
- Each parity bit calculates the parity for some of the bits in the code word



CIT 595

17

### Example: 12-bit Code word

- We then write each bit position, 1 through 12, in powers of 2

$$\begin{array}{lll}
 1 = 2^0 & 5 = 2^2 + 2^0 & 9 = 2^3 + 2^0 \\
 2 = 2^1 & 6 = 2^2 + 2^1 & 10 = 2^3 + 2^1 \\
 3 = 2^1 + 2^0 & 7 = 2^2 + 2^1 + 2^0 & 11 = 2^3 + 2^1 + 2^0 \\
 4 = 2^2 & 8 = 2^3 & 12 = 2^3 + 2^2
 \end{array}$$

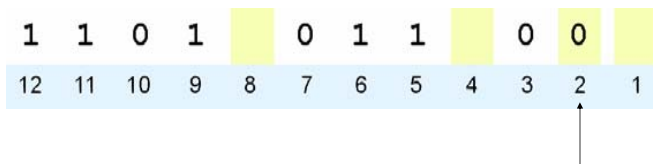
- 1 ( $= 2^0$ ) contributes to all of the odd-numbered digits
  - Hence parity at position 1 will be based on bits in position 3, 5, 7, 9, 11
- And so on...

CIT 595

18

### Example: 12-bit Code word

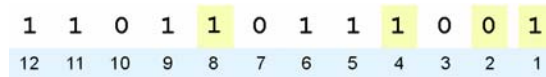
- 2 ( $= 2^1$ ) contributes to positions 2, 3, 6, 7, 10, and 11
  - Position 2 will contain the parity for bits 3, 6, 7, 10, and 11
- For the bit values shown, we have a parity value of 0 in the second bit position using even parity
  - Even parity = XOR i.e.  $0 \wedge 1 \wedge 0 \wedge 0 \wedge 1 = 0$



CIT 595

19

### Example: 12-bit Code word

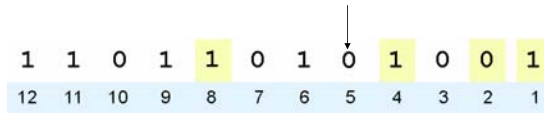


- The completed code word is shown above:
  - Bit 1 checks the positions 3, 5, 7, 9, and 11, so its value is 1
  - Bit 2 checks the positions 3, 6, 7, 10, and 11, so its value is 0
  - Bit 4 checks the positions 5, 6, 7, and 12, so its value is 1
  - Bit 8 checks the positions 9, 10, 11, and 12, so its value is also 1

CIT 595

20

### Example: 12-bit Code word

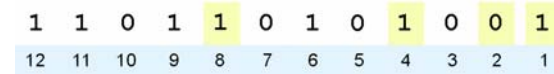


- Suppose an error occurs in bit 5, as shown above
- Our parity bit values are:
  - Bit 1 checks positions, 3, 5, 7, 9, and 11. Its value is 1, *but should be 0*.
  - Bit 2 checks positions 3, 6, 7, 10, and 11. The zero is correct.
  - Bit 4 checks positions 5, 6, 7, and 12. Its value is 1, *but should be 0*
  - Bit 8 checks positions 9, 10, 11, and 12. This bit is correct.

CIT 595

21

### Example: 12-bit Code word



- Parity bits 1 and 4 both check position 5 and 7
- Since parity bit 2 checks bit 7 and indicates no error occurred in the subset of bits it checked that means that error occurred in bit 5
- If we change bit 5 to a 1, all parity bits check and our data is restored

CIT 595

22

### Some notes on Hamming Code

- Require  $D_{min} = 3$  to detect and correct 1-bit error
- If any-one parity bit has error, the error is still correctable
- If two errors occur, Hamming code still produces an error
- This is because the Hamming distance between two code words is enough that double bit errors can be detected
  - $D_{min}$  for 1-bit correction =  $3(2k + 1)$
  - $D_{min}$  for 2-bit error detection =  $k + 1 = 2 + 1 = 3$

CIT 595

23

### Some Notes on Hamming Code (contd..)

- But sometimes cannot tell if 1 or 2 bit errors occurred
  - With 2-bit errors, hamming code ends up correcting wrong single-bit position (which actually is innocent)
- To determine whether we have 1-bit or 2-bit error we add need to add an *extra check bit*
  - The extra check bit is to determine parity across all data and existing parity bits
  - If this extra check bit is calculated to match the received value, then we either have two bit errors, or no errors
  - *No errors* can be confirmed by further determining whether parity at bit position 1, 2, 4, 8, etc are same as received parity at those positions

CIT 595

24

## Some notes on Hamming Code (contd..)

- Overhead in terms of *bits*
  - Small number of bits data -> overhead is high
    - E.g. 4 bits, need 3 parity + 1 for 2-bit detection, overhead is 50%
    - E.g. 1 byte transfers, need 4 parity + 1, overhead is 38 %
  - For larger transfer -> overhead decreases
    - E.g. 6 parity bits for 57 bits (~7 bytes). If add another bit for 2-bit error detection, we need 7 parity bits – still less than a byte
    - Need to devote **1 byte** out of each **8 bytes** i.e. **12.5%** to error correction/detection
- Overhead in terms of *hardware*
  - Some part of storage will be utilized for the check bits
  - Need hardware that will encode and decode code words on the fly – added circuitry
    - If done in software, will be slow – need processor's involvement

CIT 595

25

## Error Correcting Codes (ECC) in General

- Hamming Code is type of ECC
  - Others include Reed-Solomon, Convolution Code
- Requires extra bits for maintaining information integrity
  - E.g. in hamming code: 3 bits are added to a 4-bit data
- The overhead of extra bits does pay off
  - Single-bit *correction* often costs less than sending the entire data twice
  - If the storage is the only source of data (e.g. disk or DRAM) then we want a error-correction to avoid crashing of programs

CIT 595

26