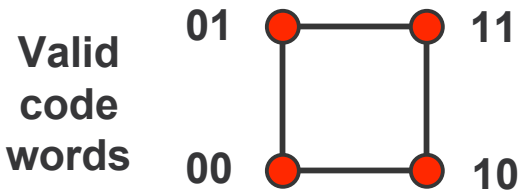
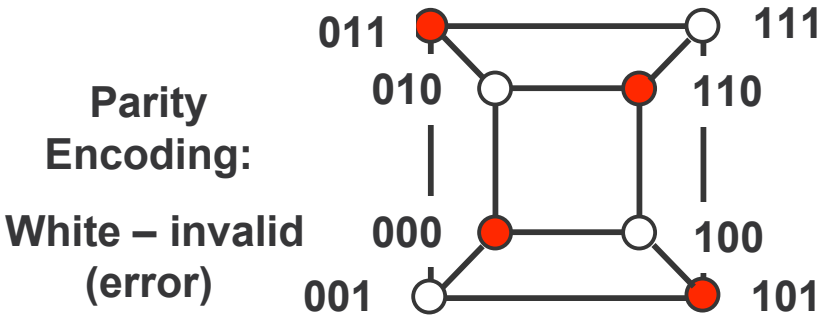
Error Coding

- Transmission process may introduce errors into a message.
  - Single bit errors versus burst errors
- Detection:
  - Requires a convention that some messages are invalid
  - Hence requires extra bits
  - An (n,k) code has codewords of n bits with k data bits and r = (n-k) redundant check bits
- Correction
  - Forward error correction: many related code words map to the same data word
  - Detect errors and retry transmission

Parity

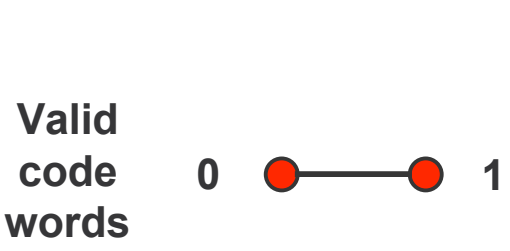
- 1-bit error detection with parity
  - Add an extra bit to a code to ensure an even (odd) number of 1s
  - Every code word has an even (odd) number of 1s





Voting

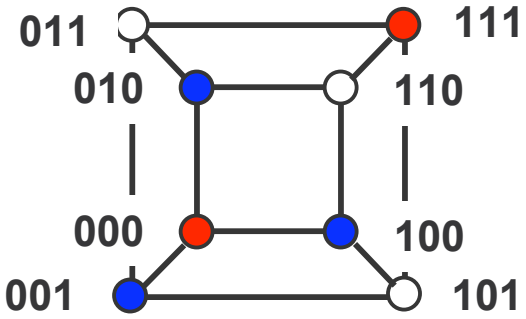
- 1-bit error correction with voting
  - Every codeword is transmitted n times



Voting:

White – correct to 1

Blue - correct to 0



Basic Concept: Hamming Distance

- *Hamming distance* of two bit strings = number of bit positions in which they differ.
- If the valid words of a code have minimum Hamming distance D, then D-1 bit errors can be detected.
- If the valid words of a code have minimum Hamming distance D, then [(D-1)/2] bit errors can be corrected.

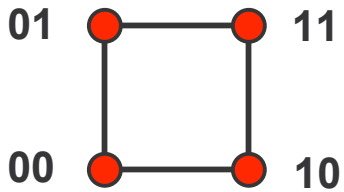
1	0	1	1	0
1	1	0	1	0

HD=2

Examples

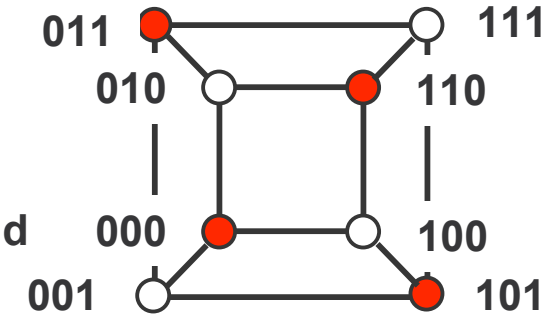
- Parity

Valid code words



Parity Encoding:

White – invalid (error)



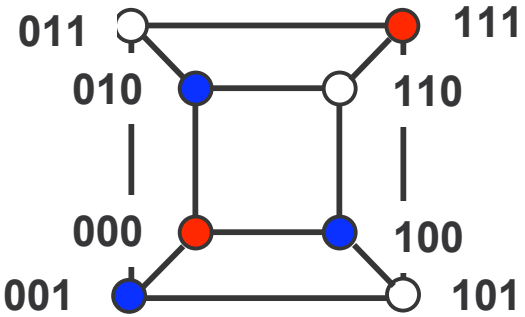
- Voting

Valid code words

Voting:

White – correct to 1

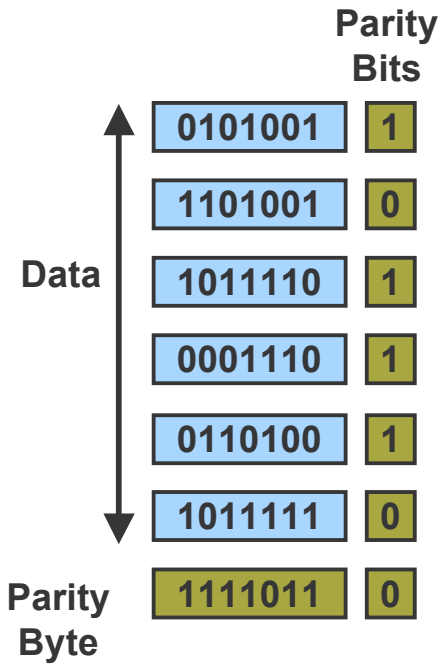
Blue - correct to 0



Digital Error Detection Techniques

- Two-dimensional parity
  - Detects up to 3-bit errors
  - Good for burst errors
- IP checksum
  - Simple addition
  - Simple in software
  - Used as backup to CRC
- Cyclic Redundancy Check (CRC)
  - Powerful mathematics
  - Tricky in software, simple in hardware
  - Used in network adapter

Two-Dimensional Parity



- Use 1-dimensional parity
  - Add one bit to a 7-bit code to ensure an even/odd number of 1s
- Add 2nd dimension
  - Add an extra byte to frame
    - Bits are set to ensure even/odd number of 1s in that position across all bytes in frame
- Comments
  - Catches all 1-, 2- and 3-bit and most 4-bit errors

Internet Checksum

- Idea
  - Add up all the words
  - Transmit the sum
- Internet Checksum
  - Use 1's complement addition on 16bit codewords
  - Example
    - Codewords: -5 -3
    - 1's complement binary: 1010 1100
    - 1's complement sum 1000
- Comments
  - Small number of redundant bits
  - Easy to implement
  - Not very robust

IP Checksum

```
u_short cksum(u_short *buf, int count) {
    register u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xFFFF0000) {
        /* carry occurred, so wrap around */
                sum &= 0xFFFF;
                sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

Cyclic Redundancy Check (CRC)

- Goal
  - Maximize protection, Minimize extra bits
- Idea
  - Add k bits of redundant data to an n-bit message
  - N-bit message is represented as a n-degree polynomial with each bit in the message being the corresponding coefficient in the polynomial
  - Example
    - Message = 10011010
    - Polynomial

$$= 1 * x^7 + 0 * x^6 + 0 * x^5 + 1 * x^4 + 1 * x^3 + 0 * x^2 + 1 * x + 0$$

$$= x^7 + x^4 + x^3 + x$$

CRC

- Select a divisor polynomial C(x) with degree k
  - Example with k = 3:
    - $C(x) = x^3 + x^2 + 1$
- Transmit a polynomial P(x) that is evenly divisible by C(x)
  - P(x) = M(x) + k bits

CRC - Sender

- Steps
  - T(x) = M(x) by x^k (zero extending)
  - Find remainder, R(x), from T(x)/C(x)
  - P(x) = T(x) – R(x) $\Rightarrow$ M(x) followed by R(x)
- Example
  - M(x) = 10011010 = $x^7 + x^4 + x^3 + x$
  - C(x) = 1101 = $x^3 + x^2 + 1$
  - T(x) = 10011010000
  - R(x) = 101
  - P(x) = 10011010101

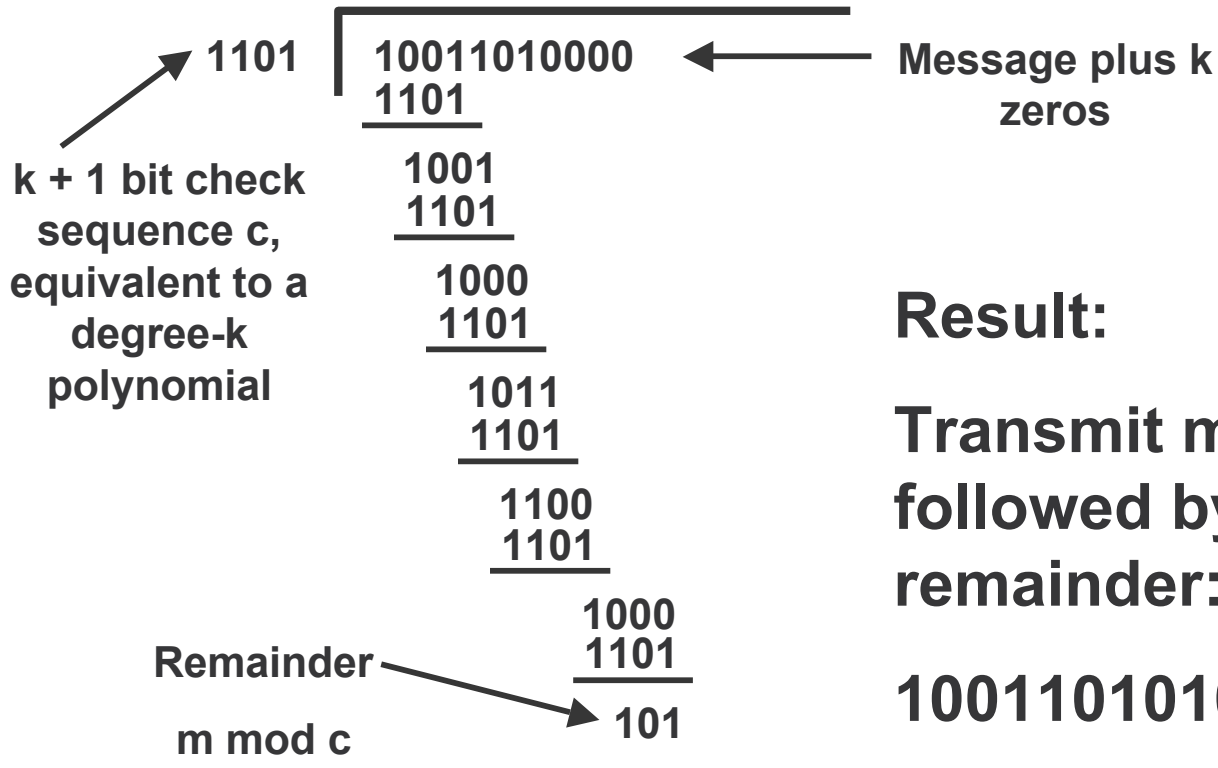
CRC - Receiver

- Receive Polynomial P(x) + E(x)
  - E(x) represents errors
  - E(x) = 0, implies no errors
- Divide (P(x) + E(x)) by C(x)
  - If result = 0, either
    - No errors (E(x) = 0, and P(x) is evenly divisible by C(x))
    - (P(x) + E(x)) is exactly divisible by C(x), error will not be detected

CRC – Example Encoding

C(x) = $x^3 + x^2 + 1$ = 1101 Generator

M(x) = $x^7 + x^4 + x^3 + x$ = 10011010 Message



Result:

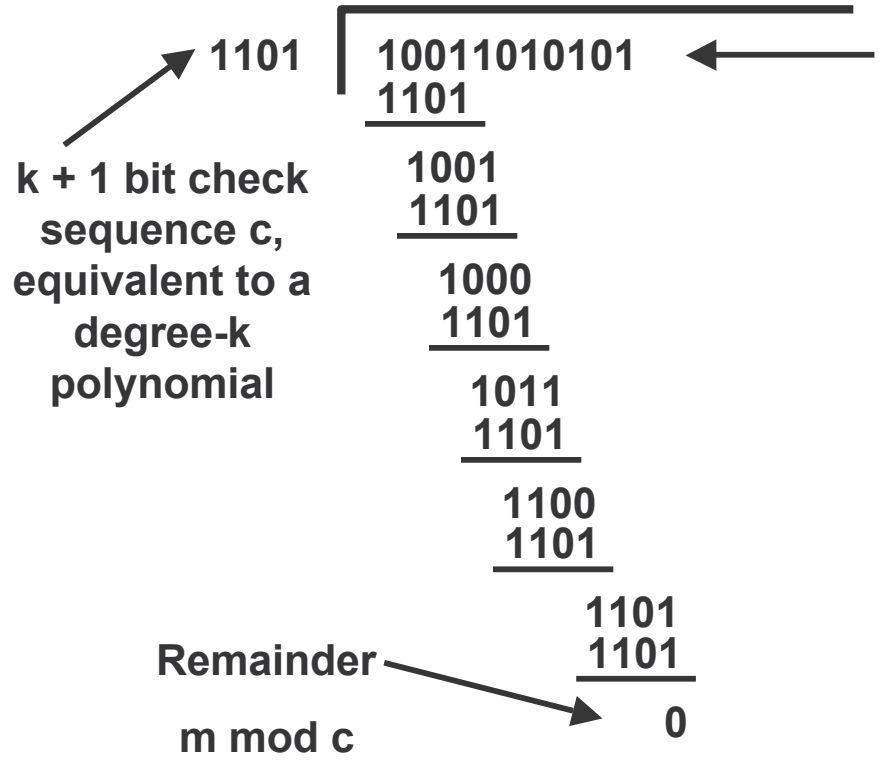
Transmit message followed by remainder:

10011010101

CRC – Example Decoding – No Errors

C(x) = $x^3 + x^2 + 1$ = 1101 Generator

P(x) = $x^{10} + x^7 + x^6 + x^4 + x^2 + 1$ = 10011010101 Received Message



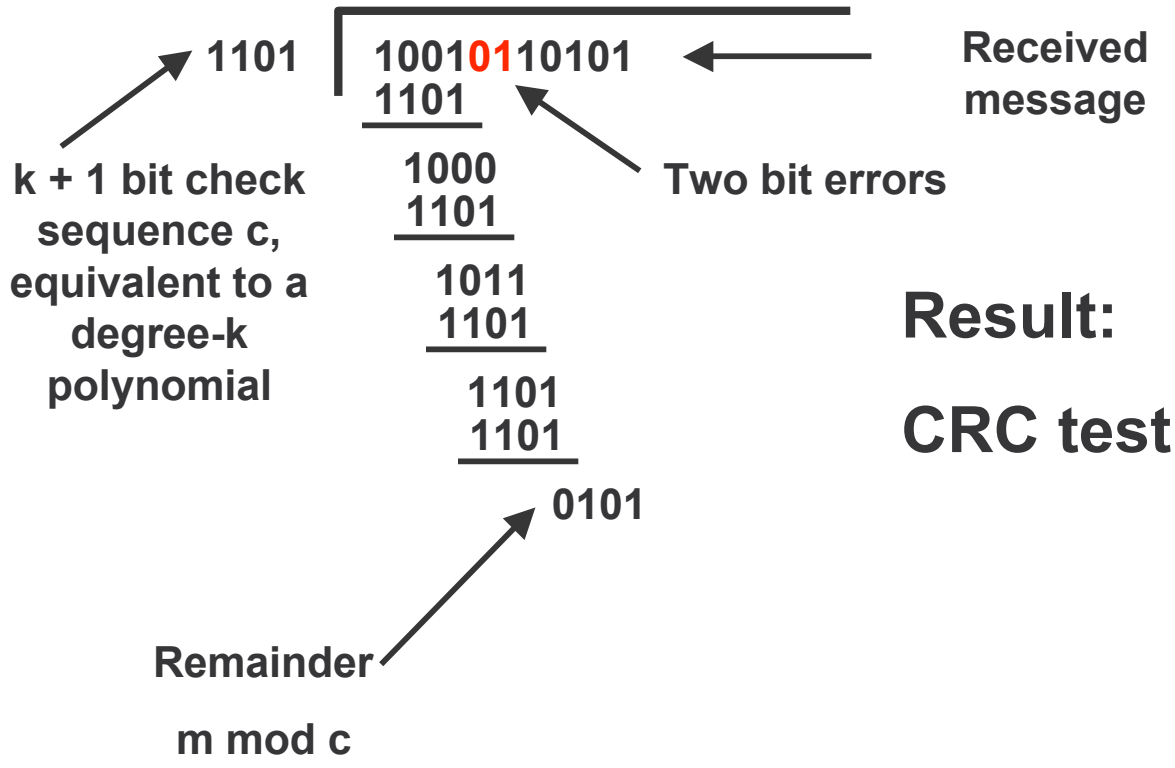
Result:

CRC test is passed

CRC – Example Decoding – with Errors

C(x) = $x^3 + x^2 + 1$ = 1101 Generator

P(x) = $x^{10} + x^7 + x^5 + x^4 + x^2 + 1$ = 10010110101 Received Message



Result:

CRC test failed

CRC Error Detection

- Properties
  - Characterize error as E(x)
  - Error detected unless C(x) divides E(x)
    - (*i.e.*, E(x) is a multiple of C(x))
- What errors can we detect?
  - All single-bit errors, if x^k and x^0 have non-zero coefficients
  - All double-bit errors, if C(x) has at least three terms
  - All odd bit errors, if C(x) contains the factor (x + 1)
  - Any bursts of length < k, if C(x) includes a constant term
  - Most bursts of length ≥ k

CRC Error Detection

- Odd number of bit errors can be detected if C(x) contains the factor (x + 1)

Proof:

- C(x) = (x + 1) C'(x)

=> C(1) = 0 (⇔ C(x) has an even number of terms)

- P(x) = C(x) f(x) = (x + 1) C'(x) f(x)
  => P(1) = 0 (⇔ P(x) has an even number of terms)
- E(x) has an odd number of terms (odd number of error bits)
  ⇔ E(1) = 1

=> P(1) + E(1) = 0 + 1 = 1 …….. (1)

- Assume E(x) cannot be detected by CRC with C(x)
  ⇔ P(x) + E(x) = C(x)g(x)

=> P(1) + E(1) = C(1)g(1) = 0 ……(2)

(1) contradicts (2) => E(x) must be detected by C(x)

CRC Error Detection

- Any error bursts of length < k will be detected if C(x) includes a constant term

Proof:

- $E(x) = x^i (x^{k-1} + ... + 1)$, where i >= 0
- $C(x) = x f(x) + 1$

- No power of x can be factored out of C(x)

=> C(x) is not a factor of x^i .....(1)

- C(x) has a degree of k: it cannot be a factor of polynomial with smaller degree (up to k-1)

=> C(x) is not a factor of $x^{k-1} + ... + 1$ .....(2)

(1) (2) => C(x) cannot be a factor of E(x)

Common Polynomials for C(x)

CRC	C(x)
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

Cyclic Redundancy Codes (CRC)

- Commonly used codes that have good error detection properties.
  - Can catch many error combinations with a small number or redundant bits
- Based on division of polynomials.
  - Errors can be viewed as adding terms to the polynomial
  - Should be unlikely that the division will still work
- Can be implemented very efficiently in hardware.
- Examples:
  - CRC-32: Ethernet
  - CRC-8, CRC-10, CRC-32: ATM

Error Detection vs. Error Correction

- Detection
  - Pro: Overhead only on messages with errors
  - Con: Cost in bandwidth and latency for retransmissions
- Correction
  - Pro: Quick recovery
  - Con: Overhead on all messages
- What should we use?
  - Correction if retransmission is too expensive
  - Correction if probability of errors is high