# Error Detection

- Data transmission can contain errors
  - Single-bit
  - Burst errors of length n
    (n: distance between the first and last errors in data block)
- How to detect errors
  - If only data is transmitted, errors cannot be detected
    → Send more information with data that satisfies a special relationship
    → Add redundancy

# Error Detection Methods

- **Vertical Redundancy Check (VRC)**
  - Append a single bit at the end of data block such that the number of ones is even
    → Even Parity (odd parity is similar)
    0110011 → 01100110
    0110001 → 01100011
  - VRC is also known as **Parity Check**
  - Performance:
    - Detects all odd-number errors in a data block

# Error Detection Methods

- **Longitudinal Redundancy Check (LRC)**
  - Organize data into a table and create a parity for each column

11100111 11011101 00111001 10101001

11100111
11011101
00111001
10101001

10101010

11100111 11011101 00111001 10101001 10101010

Original Data LRC

# Error Detection Methods

- Performance:
  - Detects all burst errors up to length n (number of  columns)
  - Misses burst errors of length n+1 if there are n-1 uninverted bits between the first and last bit
  - If the block is badly garbled, the probability of acceptance is $\left(\frac{1}{2}\right)^n$
- **Checksum**
  - Used by upper layer protocols
  - Similar to LRC, uses one's complement arithmetic

# Cyclic Redundancy Check

- Powerful error detection scheme
- Rather than addition, binary division is used → Finite Algebra Theory (Galois Fields)
- Can be easily implemented with small amount of hardware
  - Shift registers
  - XOR (for addition and subtraction)

# Cyclic Redundancy Check

- Let us assume *k* message bits and *n* bits of redundancy

$$\left.\underbrace{\text{xxxxxxxxxxx}}_{\text{k bits}}\ \underbrace{\text{yyyy}}_{\text{n bits}}\right\}\ \text{Block of length k+n}$$

- Associate bits with coefficients of a polynomial

1 0 1 1 0 1 1

$1x^6+0x^5+1x^4+1x^3+0x^2+1x+1$

$= x^6+x^4+x^3+x+1$

# Cyclic Redundancy Check

- Let M(x) be the **message polynomial**
- Let P(x) be the **generator polynomial**
  - P(x) is fixed for a given CRC scheme
  - P(x) is known both by sender and receiver
- Create a block polynomial F(x) based on M(x) and P(x) such that F(x) is divisible by P(x)

$$\frac{F(x)}{P(x)} = Q(x) + \frac{0}{P(x)}$$

# Cyclic Redundancy Check

- Sending
  1. Multiply M(x) by $x^n$
  2. Divide $x^n M(x)$ by P(x)
  3. Ignore the quotient and keep the reminder C(x)
  4. Form and send $F(x) = x^n M(x)+C(x)$

- Receiving
  1. Receive F'(x)
  2. Divide F'(x) by P(x)
  3. Accept if remainder is 0, reject otherwise

# Proof of CRC Generation

Prove that $x^n M(x) + C(x)$ is divisible by $P(x)$

$$P(x)\overline{\big)\, x^n M(x)}^{\;Q(x)}\text{, remainder } C(x)$$

$$\therefore\ x^n M(x) = P(x)Q(x) + C(x)$$

$$\frac{x^n M(x) + C(x)}{P(x)} = \underbrace{\frac{P(x)Q(x)}{P(x)}}_{\text{Remainder 0}} + \underbrace{\frac{C(x)+C(x)}{P(x)}}_{\text{Remainder 0}}$$

**Note: Binary modular addition is equivalent to binary modular subtraction → C(x)+C(x)=0**

# Example

- Send
  - M(x) = 110011 → $x^5+x^4+x+1$ (6 bits)
  - P(x) = 11001 → $x^4+x^3+1$ (5 bits, n = 4)
    → 4 bits of redundancy
  - Form $x^n M(x)$ → 110011 <u>0000</u>
    → $x^9+x^8+x^5+x^4$
  - Divide $x^n M(x)$ by P(x) to find C(x)

```
            100001
11001 ) 1100110000
        11001
        -----
             10000
             11001
             -----
              1001 = C(x)
```

Send the block 110011 <u>1001</u>

- Receive

```
11001 ) 1100111001
        11001
        -----
             11001
             11001
             -----
             00000
```

No remainder
→ Accept

# Properties of CRC

- Sent F(x), but received F'(x) = F(x)+E(x)

  When will E(x)/P(x) have no remainder,
  i.e., when does CRC fail to catch an error?

1. **Single Bit Error →** $E(x) = x^i$
   If P(x) has two or more terms, P(x) will not divide E(x)
2. **2 Isolated Single Bit Errors** (double errors)
   $E(x) = x^i+x^j$, $i>j$
   $E(x) = x^j(x^{i-j}+1)$
   Provided that P(x) is not divisible by x, a sufficient condition to detect all double errors is that P(x) does not divide $(x^t+1)$ for any t up to i-j (i.e., block length)

# Properties of CRC

3. **Odd Number of Bit Errors**
   If x+1 is a factor of P(x), all odd number of bit errors are detected

   **Proof:**
   Assume an odd number of errors has x+1 as a factor.
   Then $E(x) = (x+1)T(x)$.
   Evaluate $E(x)$ for $x = 1$
   $\rightarrow$ $E(x) = E(1) = 1$ since there are odd number of terms
   $(x+1) = (1+1) = 0$
   $(x+1)T(x) = (1+1)T(1) = 0$
   $\therefore E(x) \neq (x+1)T(x)$

# Properties of CRC

4. **Short Burst Errors**
   (Length $t \le n$, number of redundant bits)
   $E(x) = x^j(x^{t-1}+\ldots+1)$ → Length t, starting at bit position j
   If P(x) has an $x^0$ term and $t \le n$, P(x) will not divide E(x)
   ∴All errors up to length n are detected
5. **Long Burst Errors** (Length $t = n+1$)
   Undetectable only if burst error is the same as P(x)
   $P(x) = x^n + \ldots + 1$ n-1 bits between $x^n$ and $x^0$
   $E(x) = 1 + \ldots + 1$ must match
   Probability of not detecting the error is $2^{-(n-1)}$
6. **Longer Burst Errors** (Length $t > n+1$)
   Probability of not detecting the error is $2^{-n}$

# Properties of CRC

- Example:
  - CRC-12 $= x^{12}+x^{11}+x^3+x^2+x+1$
    CRC-16 $= x^{16}+x^{15}+x^2+1$
    CRC-CCITT $= x^{16}+x^{12}+x^5+1$
  - CRC-16 and CRC-CCITT catch all
    - Single and double errors
    - Odd number of bit errors
    - Bursts of length 16 or less
    - 99.997% of 17-bit error bursts
    - 99.998% of 18-bit and longer error bursts

# Hardware Implementation

- Usual practice:
  - After taking k data bits, n 0s are padded to the stream, then divided by the generator
- Aim:
  - Introduce the last n bits of 0s without requiring n extra shifts
  - Eliminate the need to wait for all data to enter the system to start generating CRC
- Approach:
  - Guess the next n bits of message as all 0s
  - Correct the guess as the actual bits arrive

# Hardware Implementation

- Message = 1011011 k = 7
  P(x) = 1101 = $x^3+x^2+x^0$ n = 3

**Conventional Method:**

```
         1100101
1101 )1011011000
      1101
       1100
       1101
        0011
        0000
         0111
         0000
          1110
          1101
           0110
           0000
            1100
            1101
             001
```

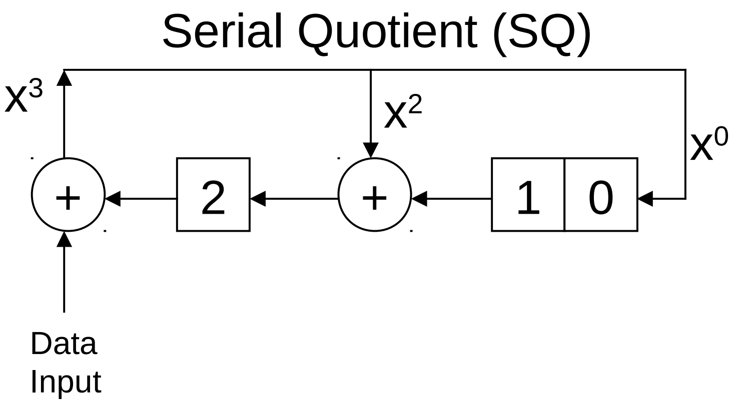
**Circuit:**

```
             1100101
1101 )0000000000
      1000
      1101
1      1010
0      1010
       1101
        1110
1       0110
        0000
         1100
1        0100
         0000
          1000
0         1000
          1101
           1010
1          0010
           0000
            0100
1           1100
            1101
             001
```

Message

# Hardware Implementation

## Transmit:




MSB

Message

LSB

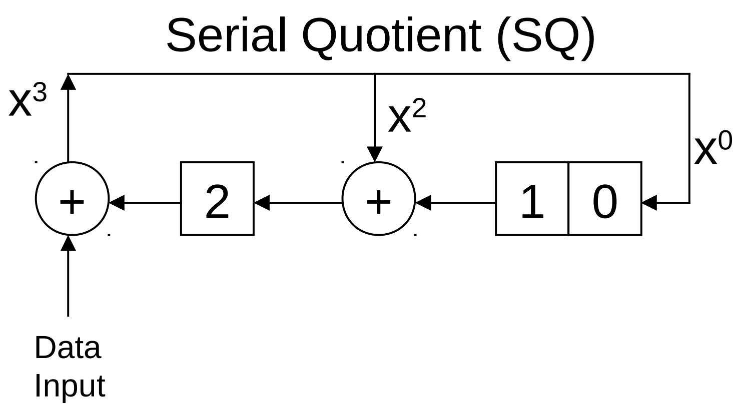
| Data | SQ | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| | | 0 | 0 | 1 |

CRC
Send MSB first

k shifts later, CRC is in register
Shift out (without any XOR) in n shifts

# Hardware Implementation

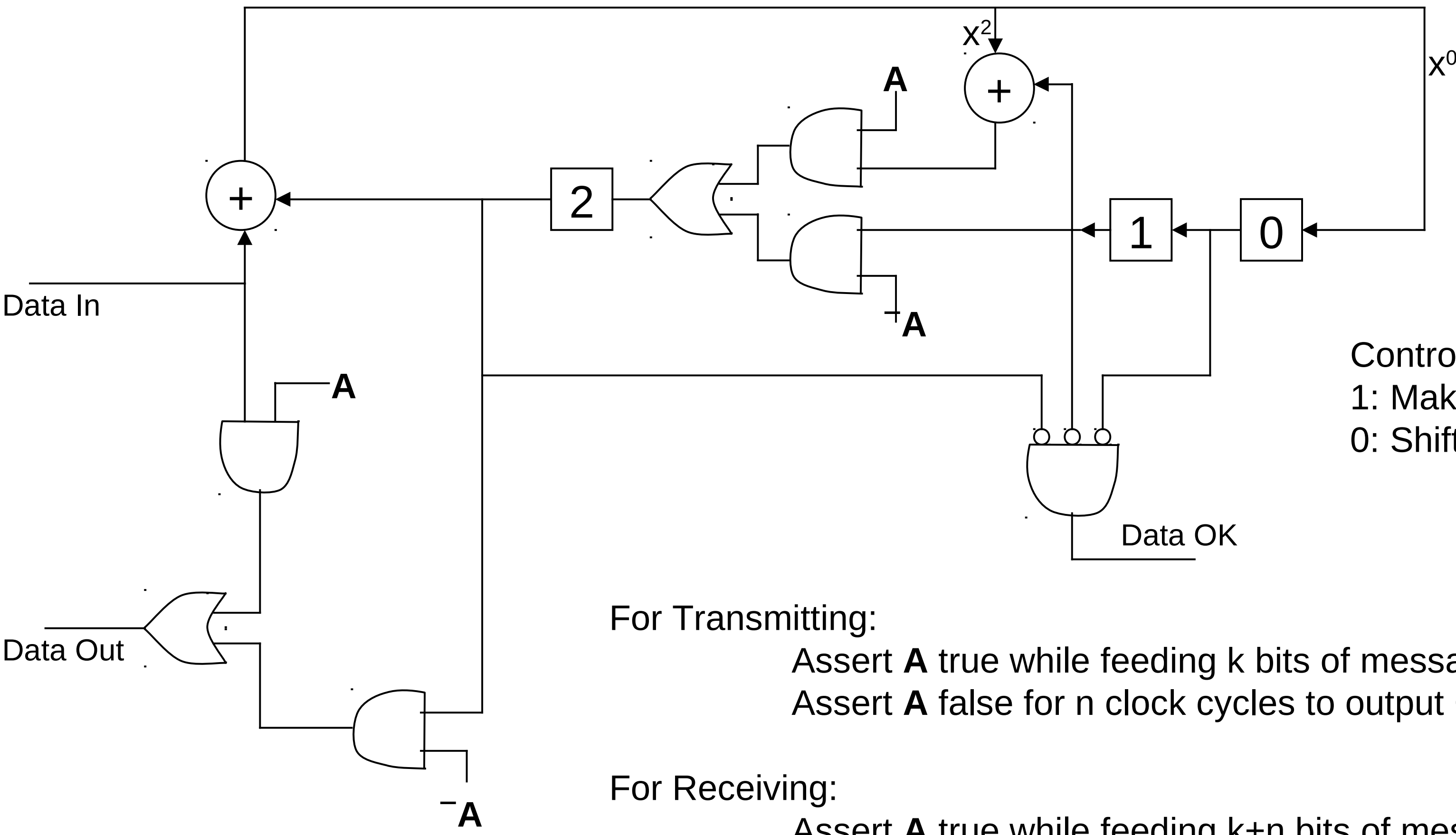
## Receive:




n+k shifts later, remainder is 0
Data accepted

| | Data | SQ | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|
| Message (MSB) | 1 | 1 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 0 | 1 |
| | 1 | 0 | 1 | 1 | 1 |
| | 1 | 0 | 1 | 1 | 0 |
| | 0 | 1 | 1 | 0 | 0 |
| | 1 | 0 | 1 | 0 | 1 |
| Message (LSB) | 1 | 1 | 0 | 1 | 0 |
| CRC (MSB) | 0 | 0 | 0 | 0 | 1 |
| | 0 | 0 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 0 | 0 |
| CRC (LSB) | | | 0 | 0 | 0 |

# Hardware Implementation




Control Line A:
1: Make/Test CRC
0: Shift Out CRC

For Transmitting:

Assert **A** true while feeding k bits of message
Assert **A** false for n clock cycles to output CRC

For Receiving:

Assert **A** true while feeding k+n bits of message and CRC
Ignore **Data Out**, check **Data OK** for correctness