ERROR CONTROL

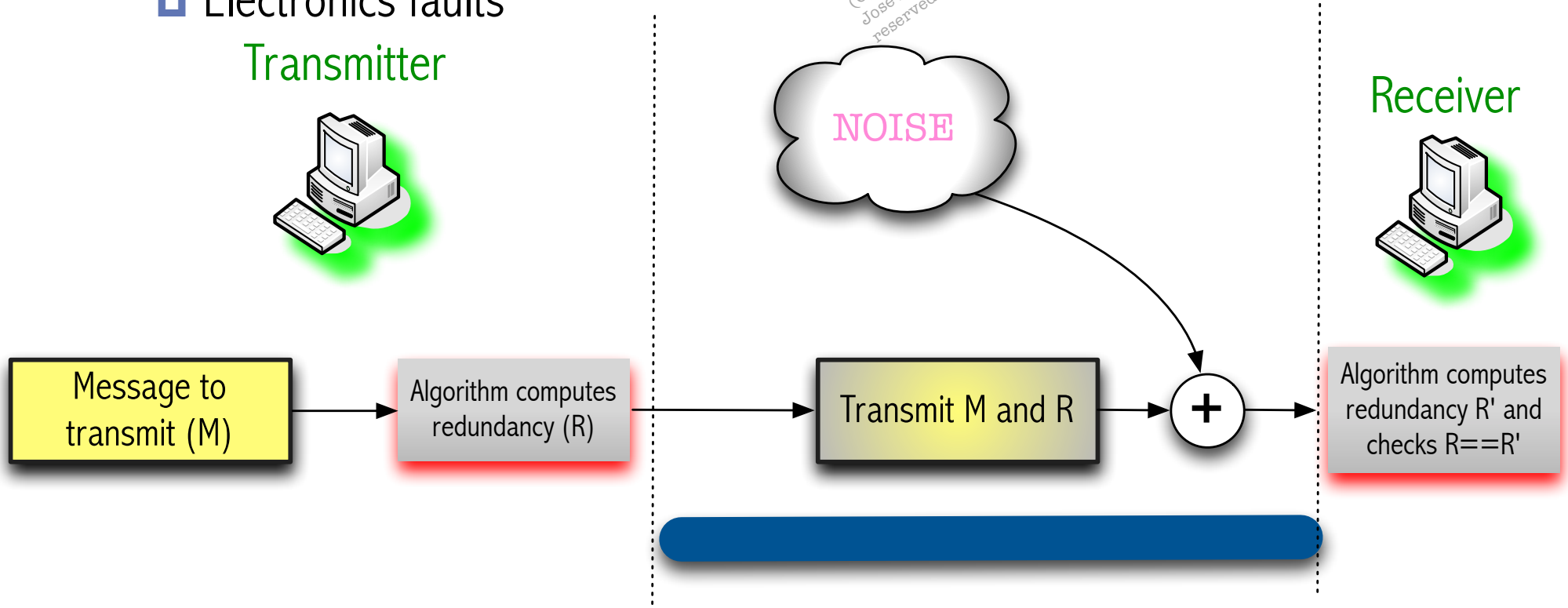
v 2.4 27/March/2020

Based on textbook *Conceptual Computer Networks* by:

Error control

Error Control

- Bit errors are introduced into frames
  - Electromagnetic interference
  - Noise
  - Electronics faults



Error Control

- Error detection
- Error correction

- If recipient detects an error, two options:
  1. Notify the sender
     - Retransmit the frame
     - If the probability of error is limited, the frame will be delivered without errors
  2. Receiver reconstructs message by using an error correcting code

Error Detection

- Common techniques
  - Parity
  - Two Dimensional Parity (BISYNC)
  - Internet Checksum (IP)
    - IP header, UDP, TCP
  - CRC (Cyclic Redundancy Check)
    - Used in HDLC, DDCMP, CSMA/CD, Token Ring

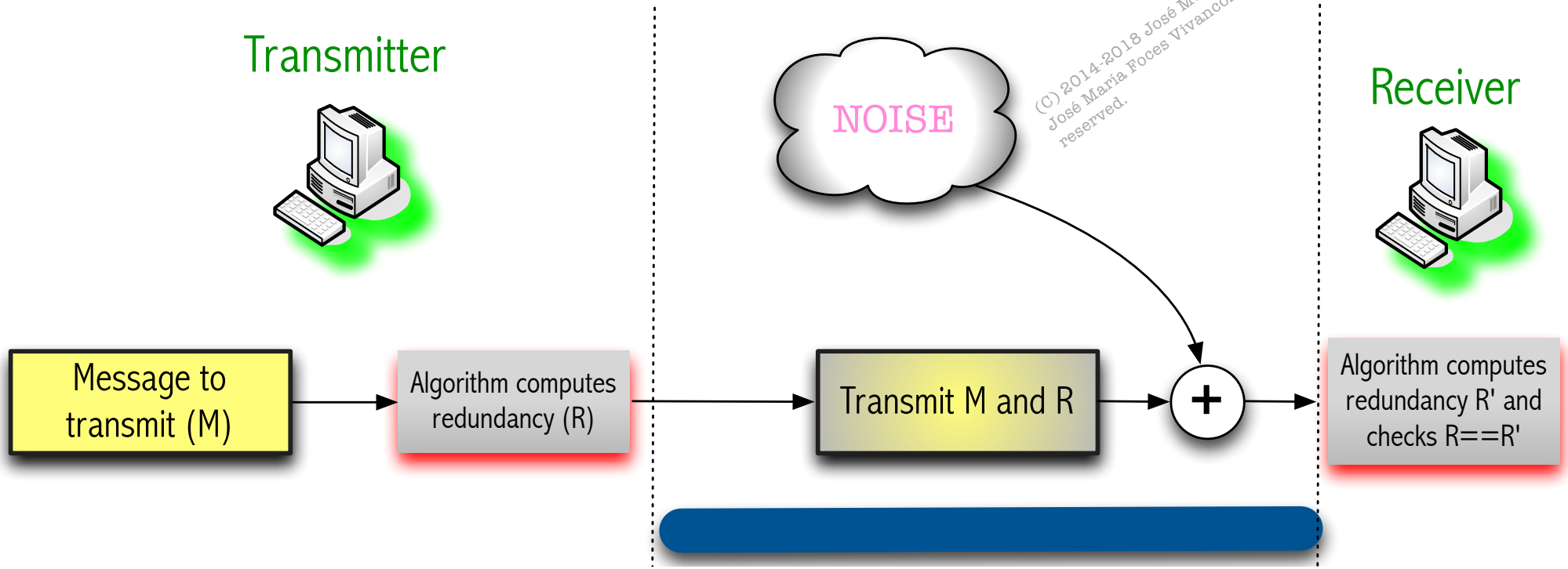
Error Detection

- Basic Idea of Error Detection
  - To add redundant information (REDUNDANCY) to a frame that can be used to determine if errors have been introduced

- Naïve approach: Transmit two complete copies of the data?
    - Identical → No error
      - WRONG!
    - Unequal → Error
    - Extremely inefficient
      - n bit message, n bit redundant information
- Use strong error detection, instead
    - k redundant bits, n bits message, k << n
    - Example: Ethernet CRC-32
      - frame can carry up to 12,000 bits of data
      - requires only 32 bits of redundancy

Error Detection

- Extra bits are redundant
  - They add no new information to the message
  - Computed from the original message by applying *an algorithm* known by transmitter and receiver



Horizontal parity

- 1 bit of redundancy
  - Normally used for 7-bit data (ASCII characters, for example)
  - Add 1-bit parity, transmit the resulting 8 bits
    - **Odd-parity** sets the eighth bit to 1 if needed to give an odd number of 1s in the byte
      - 7-bit data: 0101010, Odd parity: 0, Send 01010100
      - 7-bit data: 0001010, Odd parity: 1, Send 00010101
    - Even parity sets the eighth bit to 1 if needed to yield an even number of 1s in the byte
      - 7-bit data: 1111010, even parity: 1, Send 11110101
      - 7-bit data: 1111010, Odd parity: 0, Send 11110100

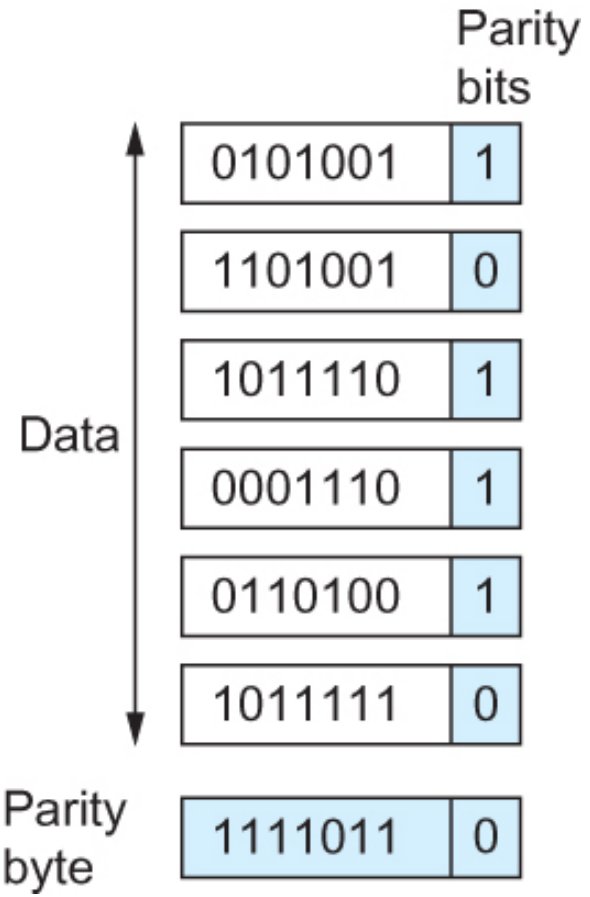
Computing the even parity

Example:

- M = 7 bits of data
- R = 1 bit of redundancy (Even parity)
- Computation:
  - Parity bit = Data0 XOR Data1 XOR Data2 XOR Data 3 XOR Data4 XOR Data5 XOR Data6
  - DATA = 0110110
  - PARITY BIT= 0 xor 1 xor 1 xor 0 xor 1 xor 1 xor 0 = 0
  - Send 01101100

Two-dimensional parity, an example

- Apply parity to each of the 7-bit bytes contained in the frame
- For every 6-BYTE block, compute its block-parity BYTE by computing the parity of each column of bits.
  - Two-dimensional parity catches all 1-, 2-, and 3-bit errors and most 4-bit errors
  - It's capable of correcting 1 error



Internet Checksum Algorithm

- Not used at the datalink level
- Used by IP, UDP and TCP
- Transmitter
  - R = Add up all the words using 1-complement
  - Transmit the 1-complement of R (checksum)
- Receiver
  - Same calculation on the received data
  - Compares the result with the received checksum
- If data|checksum gets corrupted
  - Results will not match
  - Receiver knows that an error occurred

Internet Checksum Algorithm

- Consider the data being checksummed as a sequence of 16-bit integers.
- Add them together using 16-bit ones complement arithmetic (explained next slide) and then take the ones complement of the result.
- That 16-bit number is the checksum

Cyclic Redundancy Check (CRC)

- A few extra bits will **maximize** protection

- Given a (message), a bit string 110001 we can associate a polynomial on a single variable *x* for it
  - $M(x) = 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$
    $= x^5 + x^4 + 1$
  - degree is 5, number of bits is 6
  - A *k*-bit frame has a maximum degree of *k*-1

- Let M(x) be a message polynomial and
- Let C(x) be a generator polynomial

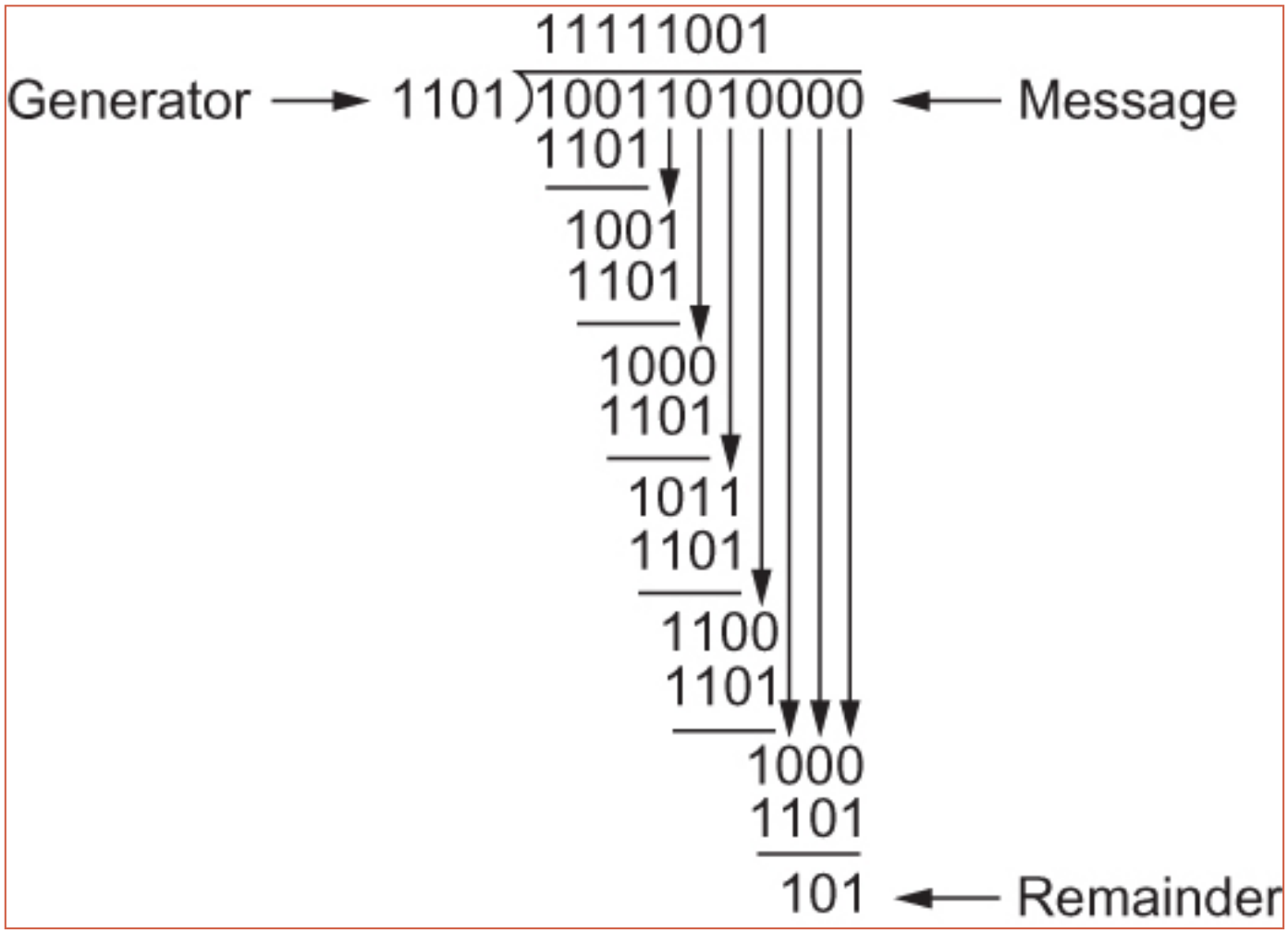
Cyclic Redundancy Check (CRC)

- Assume M(x)/C(x) leaves a remainder of 0
- M(x) is sent but, in fact, M’ (x) is received
- The receiver will compute M’ (x)/C(x)
  - Remainder is not 0: Error has been detected
- Sender and receiver must use the same C(x)
  - Examples: Ethernet (CRC-32)

Cyclic Redundancy Check (CRC)

- Polynomial Arithmetic **Modulo 2**
  - Any polynomial B(x) can be divided by a divisor polynomial C(x) if B(x) is of higher degree than C(x).
  - Any polynomial B(x) can be divided once by a divisor polynomial C(x) if B(x) is of the same degree as C(x).
  - The remainder obtained when B(x) is divided by C(x) is obtained by subtracting C(x) from B(x).
  - To subtract C(x) from B(x), we simply perform the **exclusive-OR (XOR)** operation on each pair of matching coefficients.

Cyclic Redundancy Check (CRC)



CRC Calculation using Polynomial Long Division

Cyclic Redundancy Check (CRC)

- **Properties** of Generator Polynomial
  - In general, it is possible to prove that the following types of errors can be detected by a C(x) with the stated properties
    - All single-bit errors, as long as the xk and x0 terms have nonzero coefficients.
    - All double-bit errors, as long as C(x) has a factor with at least three terms.
    - Any odd number of errors, as long as C(x) contains the factor (x+1).
    - Any "burst" error (i.e., sequence of consecutive error bits) for which the length of the burst is **less than k bits**. (Most burst errors of larger than k bits can also be detected.)

Cyclic Redundancy Check (CRC)

- Six generator polynomials that have become international **standards** are:
  - CRC-8 = x^8+x^2+x+1
  - CRC-10 = $x^{10}+x^9+x^5+x^4+x+1$
  - 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-32 = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Steps to compute the CRC by shift registers 1/4

(Example from pg 101/102 of P&D textbook)

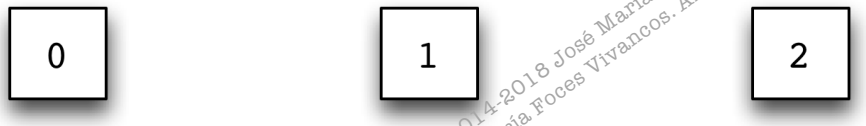
Complementary notes to CRC computing by shift registers

1. Generator polynomial of CRC example in pg. 101 and 102 of P&D textbook.

$$C(x) = 1 + x^2 + x^3 = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$$

2. One-bit shift registers

   The order of C(x) is k = 3, the circuit will have k registers, each assigned to powers of x 0 through k-1, in this case powers are: 0, 1 and 2 and the number of shift registers is 3



Computing CRC by shift registers 2/4

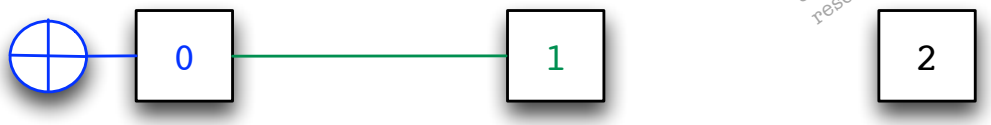
(Example from pg 101/102 of P&D textbook)

$$C(x) = 1 + x^2 + x^3 = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$$

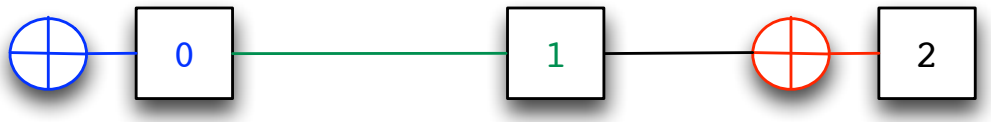
3. XOR gate at the input of each register that does belong in C(x)

Since the term 1 (0 x power) does belong in C(x), we add an XOR gate at the input of register 0

Since the term x1 (1 x power) does NOT belong in C(x), we connect its input directly to the output of its former shift register (0)



Since the term x2 (2 x power) does belong in C(x), we add an XOR gate at the input of register 2

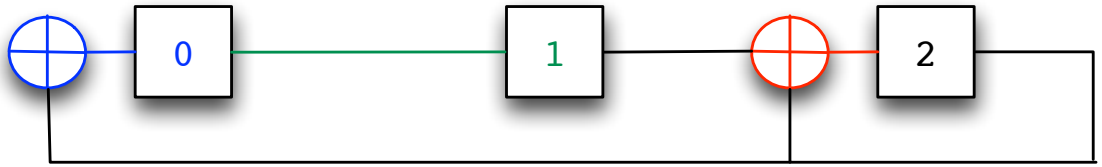


Computing CRC by shift registers 3/4

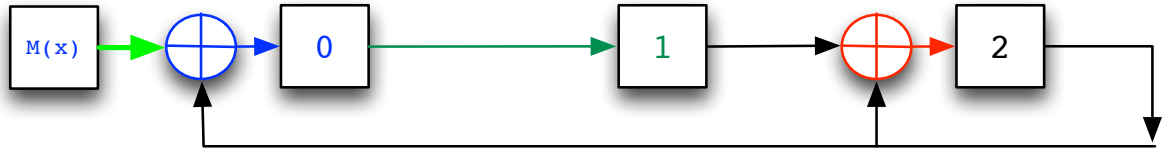
(Example from pg 101/102 of P&D textbook)

$$C(x) = 1 + x^2 + x^3 = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$$

4. Connect the output of the Most Significant Register (2) to all the XOR gates



5. Add one more shift register which will hold *each present value* from the message polynomial M(x). The arrows represent the information flow of the circuit:



Computing CRC by shift registers 4/4

(Example from pg 101/102 of P&D textbook)

$$C(x) = 1 + x^2 + x^3 = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$$

6. The CRC circuit is ready for computing the CRC. This type of circuit computes its next state by using its present state. Initially, the present state of the circuit will be represented by a 0 in each register, except for M(x) which will contain the bits from the message polynomial, starting with its Most Significant Bit – make sure you add k 0's to the least significant bits of M(x), in this case 3. We will represent this computation in tabular form. We proceed by starting with the first row (initial state) and compute the next x^0 bit (next row) according to the circuit and do the same for x^1 and x^2. When a new row has been computed, we have a new present state and proceed by computing the next state(next row) using the explained procedure. The algorithm finishes when there remain no more bits in M(x), at that state, bits x^0, x^1, x^2 contain the CRC.

M(x)	x^0	x^1	x^2
1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	1
1	0	0	1
0	0	0	1
1	1	0	1
0	0	1	1
0	1	0	0
0	0	1	0
0	0	0	1
EMPTY. CRC:	1	0	1