ERROR CORRECTION AND DETECTION

Lecture Notes 2B

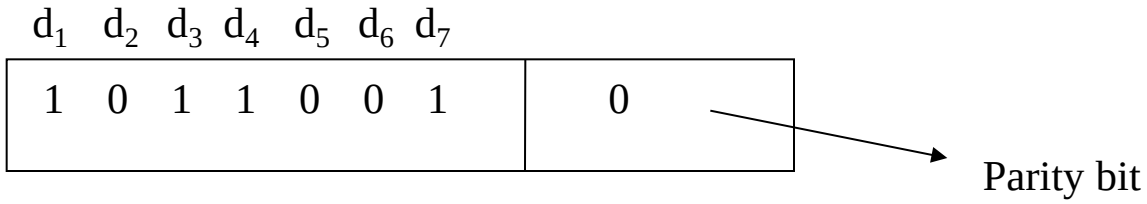
Error control: detection and correction

- Some sources of errors
  - Electromagnetic distortion of signal: “noise” on the line
  - Error in sampling pulse relative to neighbor pulse: “inter-symbol interference”
  - Energy coupling due to nearby links: “cross-talk”
  - Improperly terminated circuits and receiver transmitter problems: “echoes”
- Typical bit error rates
  - Optical fiber : less than 1 in 10^9
  - Wireless: 10^{-3}
  - Copper such as telephone lines: between 10^{-6} and 10^{-5}
- Issues in determining errors
  - Characterization of typical error occurrence: bit versus burst errors
  - Probabilistic assumptions: independence of certain types of errors (for example bit errors)

Types and use of error control

- Error control algorithms were initially used to ensure the reliability of a "link." The idea was to ensure that bits on a line were transmitted correctly
- Error control is also used end-to-end such as in TCP and also in specialized cases such as long distance satellite links, etc.
- Two basic approaches are:
  - Transmitter adds bits to aid in *error detection*; the receiver determines if an error was found, and requests a retransmission. This is termed **ARQ or automatic transmission request**
  - Transmitter adds bits to aid in *error correction*; the receiver attempts to correct the error. This is termed **FEC or forward error correction.** Of course, to correct an error you first need to detect it.
- Where are ARQ and FEC typically used
  - **ARQ** is used in data transfer over links and end-to-end over moderate distances
  - **FEC** is used in situations where retransmission is difficult or infeasible: deep space communication, audio and video, CDs, etc. FEC results in greater overhead.

Parity bit: a basic scheme



Add a bit to form a code word. The bit added should make, for example, the resulting sum of the bits modulo 2 even.

Thus, for even parity, $\varphi_{even} = \Sigma_{i=1,8} d_i \bmod 2 = 0$

Note that we are doing arithmetic modulo 2. That is:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

The operator is the same as exclusive or. Note that multiplication can be similarly defined.

The one bit parity check code can detect a single error or any odd number of errors in the bits. It cannot detect an even number of errors.

Note that odd parity has the obvious definition.

the probability of not detecting error

Suppose the probability of an error during transmission of a bit is *p*, where the errors are independent of each other and *p* is small. We thus have the random bit error model. Suppose *n* bits are transmitted.

Then, the probability of *k* errors in *n* bits is simply:

$P(k \text{ errors}) = (n;k)\, p^k\, (1-p)^{n-k}$ *the binomial distribution*

(n; k) = (n choose k) = n!/ (k!(n-k)!)

For *p* small the following is a good approximation: $(1-p)^j \approx (1-jp)$

The probability of not detecting an error is the same as that of having an even number of errors. If p is small the first term dominates.

This is P(2 errors or 4 errors or …)

$\approx (n;2)\, p^2\, (1-p)^{n-2} \approx (n;2)\, p^2\, (1-(n-2)p) \approx (n;2)\, p^2$

For $n = 8$, $p = 10^{-6}$, $(8;2) \times 10^{-12} = 28 \times 10^{-12}$.

Probability of an undetected error is 2.8×10^{-11}.

Two dimensional parity checks

$d_{1,1}$	…	$d_{1,n}$	φ_{even}[row 1]
$d_{2,1}$	…	$d_{2,n}$	φ_{even}[row 2]
.	…	.	
.	…	.	
.	…	.	
$d_{m,1}$	…	$d_{m,n}$	φ_{even}[row m]
φ_{even}[col 1]	. . .	φ_{even}[col n]	φ_{even}

Can detect up to three errors and correct single error.

Internet checksum

- Assume header consists of *n* 16-bit words. Add these words up modulo 2^{16}-1. That is:
- $S = d_1 + d_2 + \ldots + d_n \bmod 2^{16}-1$
- -S is the checksum.
- Note that the resulting codeword has the following feature: it is $S + (-S) = 0 \bmod 2^{16}-1$.
- Note that 1's complement arithmetic is assumed and thus there are two representations of 0, all 1's and all 0's.
- Example with 4-bit code words instead of 16 bit.

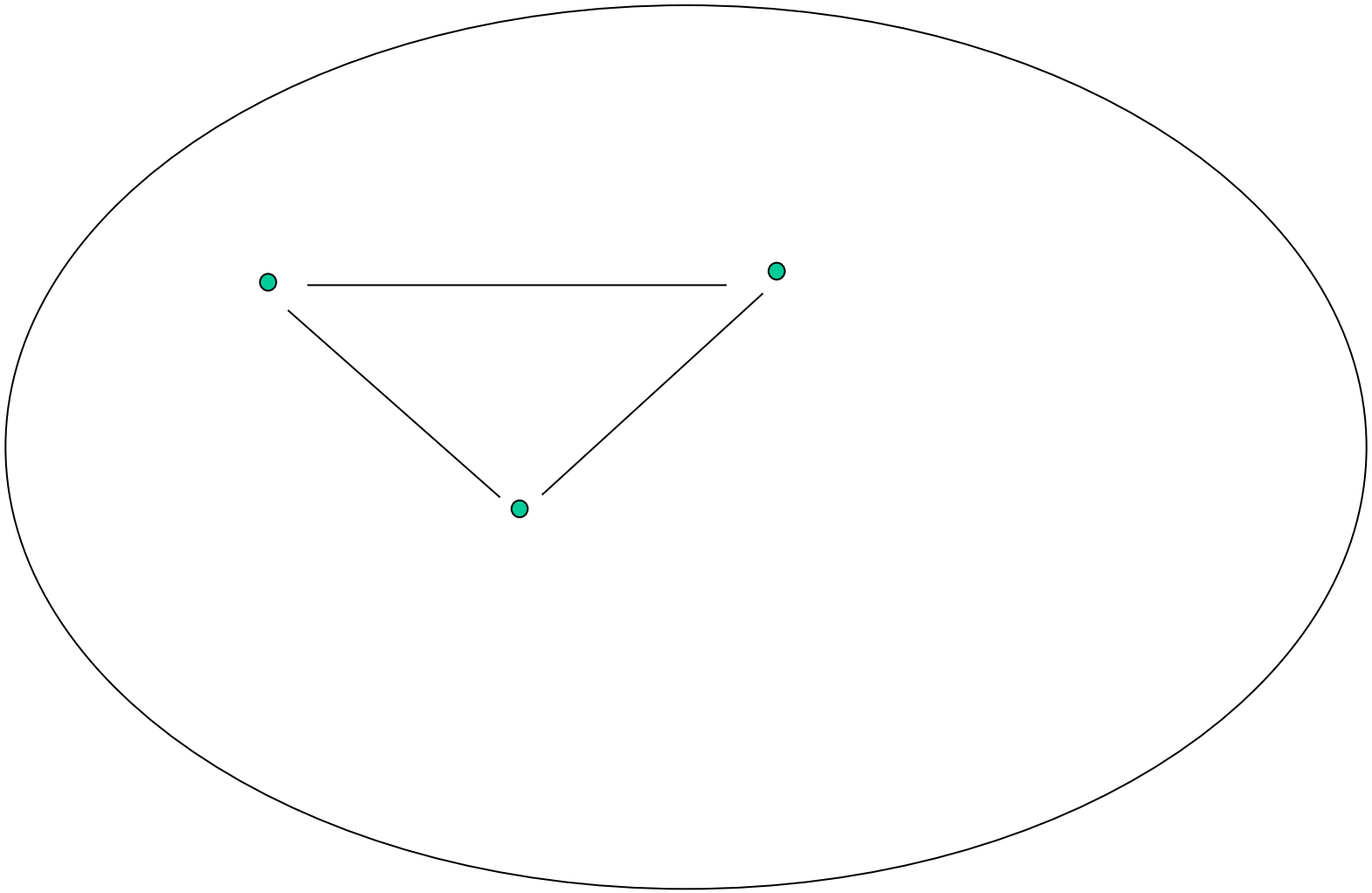
  1011 + 0001 + 0101 + 1001 = 00011010 (11 + 1 + 5 + 9 = 26)

  26 mod 15 = 11 (right shift 00011010 4 bits to get 0001, add it to original right 4 bits to get 00001011, (until upper 4 bits are 0000), and consider only low order 4 bits = 1011. Complement this to get 0100 as the checkbits.
  When adding headers + checkbits, result should be 1111.

Hamming distance

- How to design codes that have error correction/detection capability?
  - Hamming codes are a family of linear error correcting codes.
  - **Hamming distance** $d_H(i, j)$ between two code words i, j:
    - the number of bit positions in which two code words differ.
    - Example: 010101 and 111000? Hamming distance = 4
    - Note that not all 6 bit values can be code words since code words are a subset of all possible 6 bit words. We sometimes call these legitimate or legal code words. All words are also sometimes termed received words.
  - **Hamming distance** d_H of a complete code: the minimum Hamming distance of any two code words in the code. $d_H = \min_{i,j} d_H(i, j)$
    - E.g 010101, 111000, 000111, 111111
    - Hamming distance d_H = ?
  - Suppose we assume C or fewer errors are possible and the Hamming distance between legitimate code words is 2C + 1.
  - Then, we can correct any error under the above assumption!

Code word space



Error detection and correction with block codes

- Error detection
  - $d_H = 2$, then can detect any 1 bit error
  - $d_H = d + 1$, then can detect any d or fewer bits errors

- Error correction
  - $d_H = 2\,C + 1$, then can correct C or fewer bit errors.

- Weight of a code word: distance from the all 0's code word. Equivalent to the number of 1's in a non zero code word. For example 10110 has weight 3.

- (n,k) block codes, also called Hamming codes
  - k bits of data, (n – k) bits of check bits
  - Check bits are a linear sum mod 2 of the data bits, with binary coefficients

Example of a (7,4) block code

- Let m_4, m_5, m_6, m_7, be the data bits, and c_1, c_2, c_3, be the check bits.
- Let the equations be:
  $c_1 \oplus m_4 \oplus m_6 \oplus m_7 = 0$
  $c_2 \oplus m_4 \oplus m_5 \oplus m_6 = 0$
  $c_3 \oplus m_5 \oplus m_6 \oplus m_7 = 0$

- In this example, it turns out that the minimum weight of any code word is 3 and the minimum distance between two legitimate code words is also 3. Thus single errors can be corrected, and double errors can be detected.
- It can be shown that the minimum weight of a block code is also the minimum distance between any two code words.
- Block codes have the property that the sum of two code words is also a code word.

Matrices associated with (7,4) block code

Hamming codes naturally have two matrices associated with the codes termed the generator matrix **G** and the parity check matrix **H**.

The check matrix **H**.

The check matrix **H** can be derived from the equation of the block code. This 3×7 matrix is as follows:

$$\mathbf{H}_{3,7} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

If we let $[c_1\ c_2\ c_3\ m_4\ m_5\ m_6\ m_7]$ be a code word **v** as a column vector, then $\mathbf{H}\,\mathbf{v} = \mathbf{0} = [0\ 0\ 0]^T$ defines the set of code words.

The next slide shows the full set of 16 code words. Remember that the data that we are actually sending is $m_4\ m_5\ m_6\ m_7$, hence 16 code words. If the code word **v** is sent but the received vector is **r**, we have the following:

0 errors: $\mathbf{H}\,\mathbf{r} = \mathbf{0}$

1 errors: $H\,\mathbf{r} = \mathbf{s} \neq \mathbf{0}$ and the error is detected and corrected.

2 errors: $H\,r \neq \mathbf{0}$ and the error is detected, but not correctly corrected.

c_1	c_2	c_3	m_4	m_5	m_6	m_7
0	0	0	0	0	0	0
1	0	1	0	0	0	1
1	1	1	0	0	1	0
0	1	0	0	0	1	1
0	1	1	0	1	0	0
1	1	0	0	1	0	1
1	0	0	0	1	1	0
0	0	1	0	1	1	1
1	1	0	1	0	0	0
0	1	1	1	0	0	1
0	0	1	1	0	1	0
1	0	0	1	0	1	1
1	0	1	1	1	0	0
0	0	0	1	1	0	1
0	1	0	1	1	1	0
1	1	1	1	1	1	1

Matrices associated with (7,4) block code (continued)

The generator matrix G.

The generator matrix is a matrix that given a message $m = [m_4\ m_5\ m_6\ m_7]$ will generate the code word *v* by $m\,G = v$, where here G (or $G_{4,7}$) is a 4×7 matrix. It can easily be derived from the check matrix H if H is viewed as follows:

$$H_{3,7} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} = [I_{3,3} \mid A]$$

In this example A is a 3×4 matrix and I is the identity matrix.

The syndrome **s**

The syndrome **s** (a column vector) is defined as **s** = H r, where **r** is the received vector. Assuming 1 error, if you compare the syndrome with the columns of **H** and the j^{th} column is the match, then the error is in the j^{th} column of **r**. *Try to prove this. Hint: write* ***r*** *=* ***v*** *+* ***e*** *and determine the effect of* ***e****.*

Polynomial codes

- Also called CRC codes (cyclic redundancy check)
- Binary codes that represent a word as a polynomial over GF(2)

$[v_4\ v_3\ v_2\ v_1\ v_0]$ $\qquad$ $v_4 x^4 + v_3 x^3 + v_2 x^2 + v_1 x^1 + v_0$

$[\ 1\ \ 1\ \ 1\ \ 0\ \ 1\ ]$ $\qquad$ $x^4 + x^3 + x^2 + 1$

- Note that addition, multiplication, etc. is mod 2.
- Polynomials over GF(2) can be multiplied, added, subtracted, divided etc., just like ordinary polynomials.

- Addition example:

$$\begin{array}{r} x^4 + \quad + x^2 \quad + 1 \\ \underline{x^4 + x^3 + \quad + 1} \\ x^3 + x^2 \end{array}$$

- Multiplication example

  $(x^2 + 1) * (x^3 + x^2) = x^5 + x^4 + x^3 + x^2$

- Division example

$$
\begin{array}{r|l}
 & x^3 + x \longrightarrow \text{Quotient Q(x)} \\
\cline{2-2}
x^2 + 1 & x^5 + x + 1 \\
 & x^5 + x^3 \\
\cline{2-2}
 & x^3 + x + 1 \\
 & x^3 + x \\
\cline{2-2}
 & 1 \longrightarrow \text{Remainder R(x)}
\end{array}
$$

- Note that the number of bits of the remainder will be the order of the polynomial (that is 2 in the example, ie remainder is 0x + 1.

The generator polynomial

- Define the generator polynomial G(x) of degree r to be a polynomial of degree r. We will assume it has lowest term 1.

  That is: $x^r + \ldots + 1$.

  For example, $x^{16} + x^{15} + x^2 + 1$, $r = 16$

- Note that the remainder of any polynomial divided by G(x) would be of order ≤ 15, and can be defined by 16 bits.

- Let M(x) be the message and suppose it is k bits long. Then $M(x) = m_{k-1}\, x^{k-1} + \ldots + m_1\, x + m_0$

Encoding message and adding the CRC check bits

1. $M(x) * x^r$ this gives r 0's in the lower order positions.

2. Divide this result by G(x)

   $(M(x) * x^r) / G(x) = Q(x) + R(x) / G(x)$

   ie. $M(x) * x^r = Q(x)\, G(x) + R(x)$

3. Add the remainder to $M(x) * x^r$. That is:

   $T(x) = M(x)\, x^r + R(x)$ *Note there are r low order parity bits*

4. Note that $G(x) \mid T(x)$ *G(x) divides T(x)*
5. When generator polynomial is order 16, we have 16 bits of CRC that is added to the message.

Checking Received Message for Correctness

- Let the message that is received be A(x)
- Compute A(x) / G(x) and determine the remainder.
- If remainder is 0, we say there are no errors and received message is correct.
- If remainder is not 0, we detect error.

Error detection capabilities of polynomial codes

- We transmitted T(x). Suppose instead that we receive

  $$T(x) \oplus E(x)$$

  Note that each 1 bit in E(x) represents a transmission error or an inversion in the corresponding transmitted code word bit.

  Now, $\Re[\,(T(x) \oplus E(x)) / G(x)] = \Re[\,E(x) / G(x)]$

  since $G(x) \mid T(x)$ ($\Re$ *is remainder*)

- Therefore, if G(x) does not divide E(x) we detect an error.

- Detecting single bit errors

  $E(x) = x^i$ for some i. Now, assume that our generator polynomial G(x) has more than 1 term. Eg., $x^3 + x + 1$.

  Note that G(x) cannot divide E(x) with a 0 remainder.

  Eg., $x^3 + x + 1 \overline{)\, x^5}$ does not give 0 remainder

Detecting double bit errors

$E(x) = x^j + x^i \qquad j > i$

$= x^i (x^{j-i} + 1)$

We must check that G(x) does not divide $x^{j-i} + 1$

A polynomial is irreducible if it cannot be factored.

$x^4 + x^2 + x + 1$ is not irreducible. Why? Note that $x + 1$ is a factor. Note that $x^2 + x + 1$ is irreducible.

An irreducible polynomial is primitive if it "generates" a finite field in a specific way. We will not consider this further as it is somewhat complicated to explain this notion.

If p(x) is *primitive* and order N, it will not divide $(x^m + 1)$

for $m < 2^N - 1$. For example, $p(x) = x^{15} + x + 1$ is primitive and thus does not divide $x^m + 1$ for $m < 2^{15} - 1$. Therefore,

if $j-i < 2^{15} - 1$, then no double bit error of this "length" is possible, if $G(x) = (x + 1)(x^{15} + x + 1)$.

Detecting an odd number of errors

Suppose E(x) has an odd number of terms

Claim: E(x) does not have x + 1 as a factor

Suppose it did. Then, $E(x) = (x + 1)\, Q(x)$

therefore $E(1) = (1 + 1)\, Q(1) = 0$ (*substituting 1 for x*)

This is a contradiction since E(1) for an odd number of terms cannot be 0 but must be 1.

Therefore, if we make x + 1 a factor of G(x), then G(x) does not divide E(x) for an odd number of terms.

Burst errors

- Polynomial code with r check bits will detect burst errors $\leq$ r.

  Burst length = k, can be represented by

  $x^i [x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x + 1]$ where i is location of burst

  Note: x^i is not a factor of G(x) and if $x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x + 1$ is less in degree than G(x), then remainder cannot be 0.

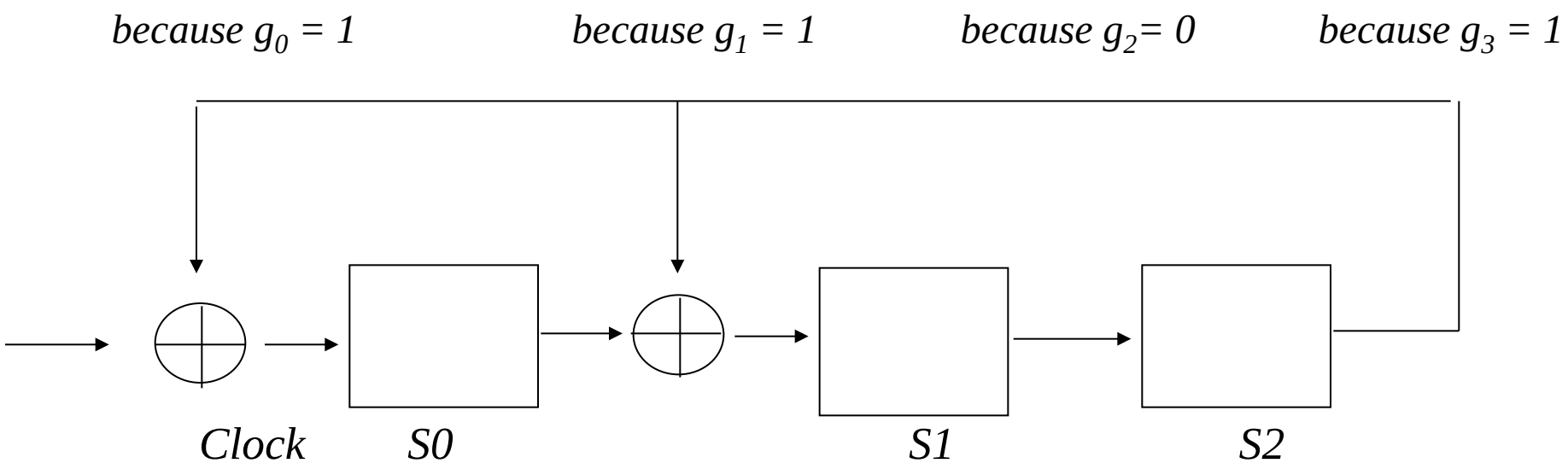
  For example $x^i [x^{r-1} + x^{r-3} + \ldots + x + 1]$ will be detected.

  Note: a large fraction of bursts $\geq$ r are also detected.

Some standard generator polynomials

- CRC-12: $x^{12} + x^{11} + x^3 + x^2 + x + 1$ Bisync
- CRC-16: $x^{16} + x^{15} + x^2 + x + 1$ Bisync
- CCITT-16 $x^{16} + x^{12} + x^5 + 1$ HDLC, XMODEM
- CCITT-32 $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11}$
  $+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ IEEE 802

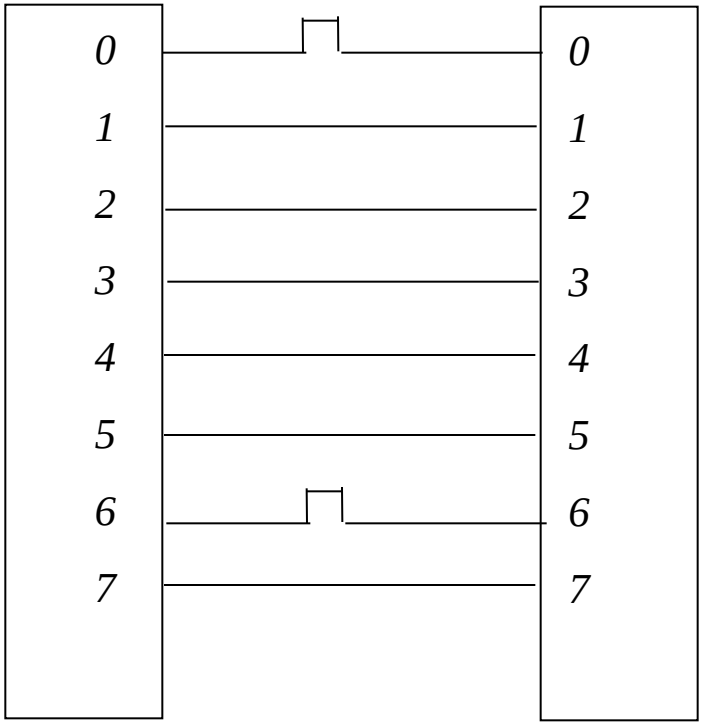
Implementation using shift registers division by $x^3 + x + 1$



Clock	S0	S1	S2
0	0	0	0
3	a_2	a_3	a_4
4	$a_1 + a_4$	$a_2 + a_4$	a_3
5	$a_0 + a_3$	$a_1 + a_4 + a_3$	$a_2 + a_4$

Suppose you want to divide $< a_4 , a_3 , a_2 , a_1 , a_0 >$ then shift the input in over 5 clock cycles. The remainder appears in <S2, S1, S0>

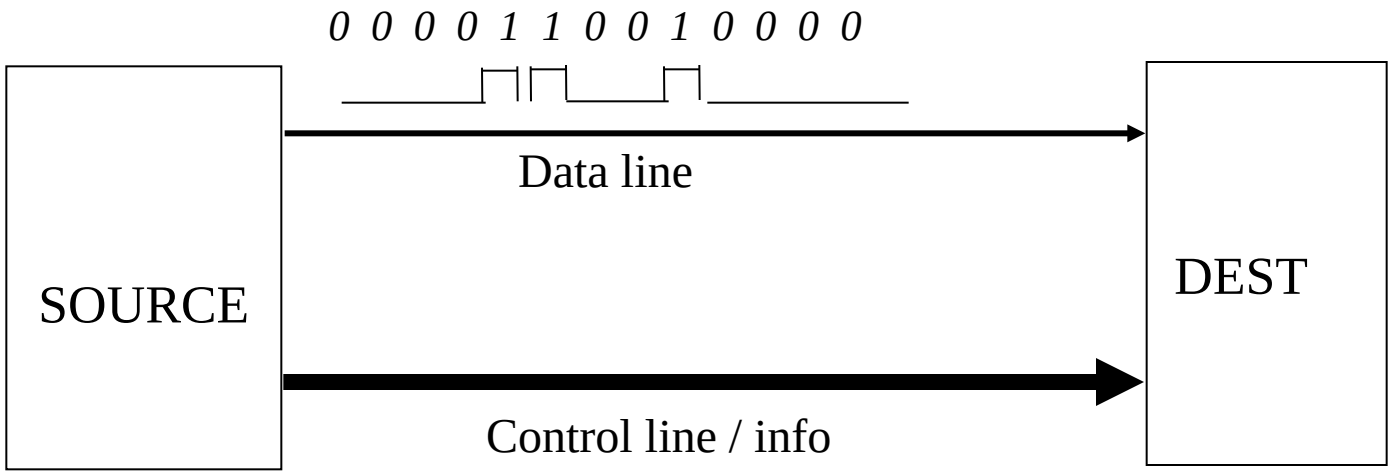
Parallel transfer of information (bits)



Parallel Wires

8 bits, 16 bits, etc.

Serial transfer (bit serial)



Transfer, for example, an octet a single bit at a time using a fixed time interval for each bit. A clock signal helps determine when to sample the bit.

Why use a serial line? As distances increase, multiple wire costs increase, complexity of line drivers and receivers increase, etc.

Thus, to send a character, first serialize the bits, send over the line, and then receive the bits, and convert back to character.

Communication Modes

- Simplex: Data is transmitted in one direction only
- Half duplex: alternating exchange of data between two devices. Need to switch between sender and receiver mode
- Full duplex: data can be exchanged in both directions simultaneously

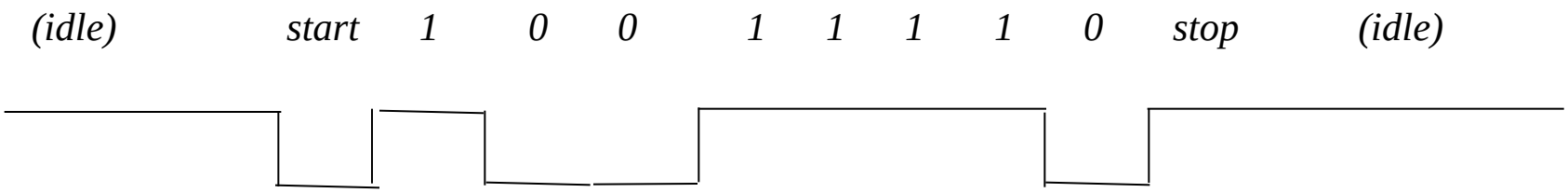
Determining a received bit pattern

Determine

- Start of each bit (or center of bit). *Bit or clock synchronization*
- Start and end of the unit, a character, a byte, etc. *Byte synchronization*
- Start and end of message unit or frame. *Frame synchronization*

Asynchronous transmission

- Receiver and transmitter clocks are not synchronized. They are independent. Receiver resynchronizes at the start of each unit, say a character.
- To determine beginning of character, have a “start” bit
- To be sure about end of character, have a stop bit.
- Know how many bits are in a character, say 8.
- Clock frequency for sampling is usually 16 times bit rate. Receiver attempts to sample in the center of the bit.
- Start bit is usually 1 bit long and stop bits are often 1, 1.5, or 2 bits



Synchronous transmission

- Sender and receiver clocks are synchronized (often through the sending of a clock signal on a control line)

- Start and stop bits are not used. Characters can be sent one after the other without the overhead of start and stop bits.

- Framing is still important.