

# **CS123: Lecture 6, Error Detection**

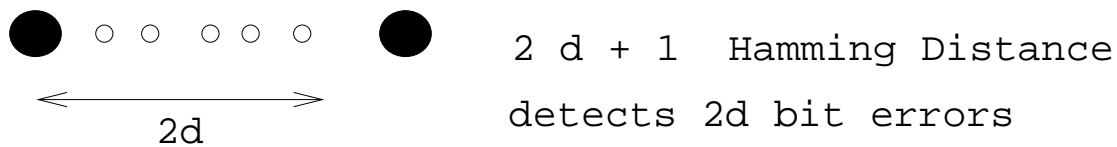
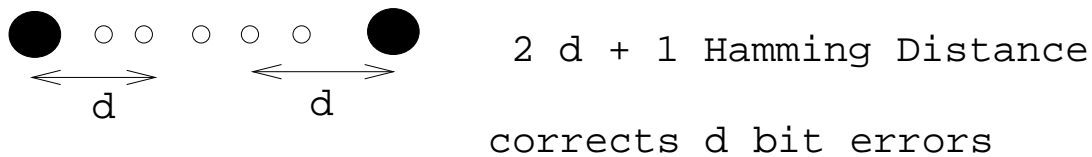
**George Varghese**

October 12, 2006

## Error Detection using Parity

- Random versus Burst Errors. For same error rate, burst is better.
- Parity: ExOR of bits. Can detect all odd bit errors in a frame. Can't detect 2 bit errors. 1011 sent as 10111.
- Would like to do better than parity using so-called checksums for detecting larger number of errors (happens often). A simple concept called Hamming Distance explains why some codes detect and correct more bits.
- Hamming Distance between two strings S and R is the number of bit positions they differ. Thus Hamming Distance of 11011 and 10111 is 2.

# Hamming Distance, Error Detection, and Error Correction



- Can *detect*  $d$  random bit errors if Hamming distance is  $d + 1$  because flipping  $d$  bits cannot move from valid codeword (black) to another.
- Can *correct*  $d$  errors if Hamming distance is  $2d + 1$ . Can draw a “ball” of radius  $d$  around each valid codeword  $C$  and assign invalid codewords within ball to  $C$ . Can also *detect*  $2d$  errors with same code but cannot do both correction and detection at same time!

## ORDINARY DIVISION CHECKSUM

- Consider message  $M$  and generator  $G$  to be binary integers.
- Let  $r$  be number of bits in  $G$ . We find the remainder  $t$  of  $2^r M$  when divided by  $G$ . Why not just  $M$ ? So that we can separate checksum from message at receiver by looking at last  $r$  bits.
- Thus  $2^r M = k.G + t$ . Thus  $2^r M + G - t = (k + 1)G$ . Thus we add a checksum  $c = G - t$  to the shifted message and the result should divide  $G$ .
- Has some reasonable properties. However integer division hard to implement. Prefer to do without carries.

## The Big Idea

- In ordinary division checksums we transmitted a message plus checksum that was divisible by the generator  $G$ . Thus any errors that cause the resulting number to be not divisible by  $G$  (invalid codewords) will be detected.
- In CRCs, we do the same thing except that we use Mod 2 arithmetic instead of ordinary arithmetic.

## MODULO 2 ARITHMETIC

- No carries. Repeated addition does not result in multiplication. e.g.  $1100 + 1100 = 0000$ ;  
 $1100 + 1100 + 1100 = 1100$
- Multiplication is normal except for no carries: e.g.  $1001 * 11 = 10010 + 1001 = 11011$ . Shift and Ex-or instead of Shift and Add as in normal arithmetic.
- Similar algorithm to ordinary division. Again let  $r$  be number of bits in  $G$ . We find the remainder  $c$  of  $2^{r-1}M$  when divided by  $G$ . Why only shift message  $r - 1$  bits this time?
- Thus  $2^{r-1}M = k.G + c$ . Thus  $2^{r-1}M - c = k.G$ . Thus  $2^{r-1}M + c = k.G$  because addition is same as subtraction. Send  $c$  as checksum

## Recall how ordinary division works

$$\begin{array}{r} 118 \\ \hline 62 \overline{) 7344} \\ \underline{62} \phantom{00} \\ 114 \\ \underline{62} \phantom{00} \\ 524 \\ \underline{496} \phantom{00} \\ 28 \end{array}$$

- Can be viewed as repeated subtractions of multiples of 62 (i.e., 6200, 620, 496) until we get a number *less* than 62, which is the remainder.

## How Mod 2 Division Works

**Generator**      **Shifted Message**

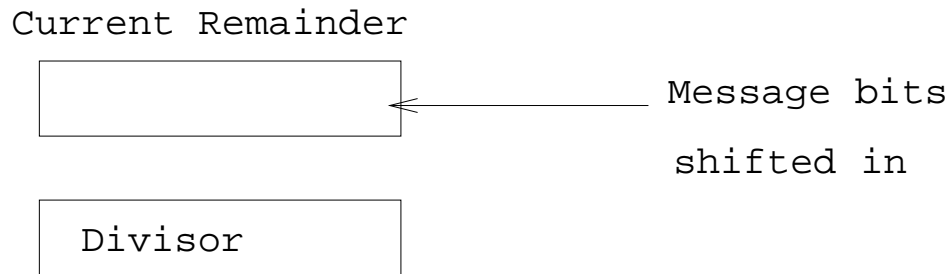
$$\begin{array}{r}
 1 \ 1 \ 1 \\
 \phantom{1 \ 1 \ 1} \xrightarrow{\hspace{1.5cm}} \begin{array}{r}
 1 \ 1 \ 0 \ 0 \ 0 \\
 \underline{1 \ 1 \ 1} \quad \downarrow \\
 0 \ 1 \ 0 \\
 \underline{0 \ 0 \ 0} \quad \downarrow \\
 1 \ 0 \ 0 \\
 \phantom{1 \ 0 \ 0} \xrightarrow{\hspace{1.5cm}} \begin{array}{r}
 1 \ 1 \ 1 \\
 \underline{1 \ 1 \ 1} \\
 0 \ 1 \ 1
 \end{array}
 \end{array}
 \end{array}$$

- For CRC, we need to repeatedly add (mod 2) multiples of the generator until we get a number that is  $r - 1$  bits long that is the remainder.
- The only way to reduce number of bits in Mod 2 arithmetic is to remove MSB by adding (mod 2) a number with a 1 in the same position.

## CRC: Polynomial View

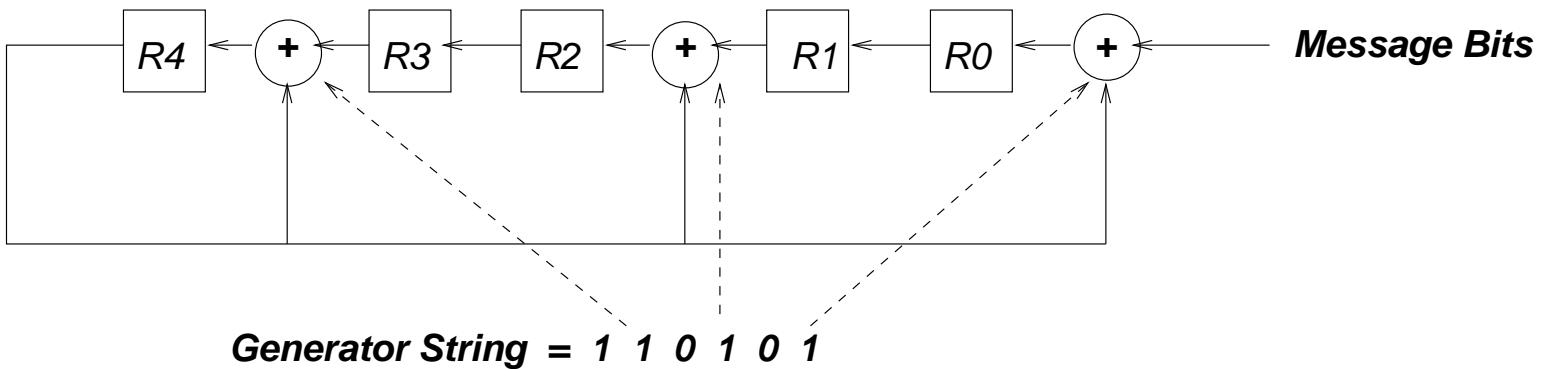
- 101 and 011 can be represented as  $X^2 + 1$  and  $X + 1$ .  $X^i$  term iff the  $i$ -th bit is 1.
- Normal addition:  $X^2 + X + 2$ . No carries between powers.  $2X$  is bad. Fix by using Mod 2 addition (EX-OR) to get:  $X^2 + X$
- Can think of CRC computation as dividing a shifted message polynomial (multiplied by  $x^{r-1}$ ) by CRC divisor polynomial and adding remainder.
- Equivalent to arithmetic view, but poly view is easier to *analyze*.

# Implementing CRCs



- The current remainder is held in a register initialized with first  $r$  bits of the message.
- If MSB of current remainder is 1, then EXOR current remainder with divisor; if the MSB is 0, do nothing.
- Shift the current remainder 1 bit to the left and shift in next message bit.

## CRC in Hardware: LFSR



- Registers  $R0$  through  $R5$  are several single bit registers corresponding to the single multibit register in previous slide.
- Ex-OR placed to right of register  $i$  if bit  $i$  set in generator.
- When a message bit shifts in, all registers send (in parallel) their bit values to left, some through Ex-OR gates. Combines left shift of an iteration with MSB check and Ex-OR of next iteration. Ex-OR during left shift.
- Avoids check for MSB using output of  $R4$  as input to all Ex-ORs.

## CRC PROPERTIES

CRC-16:  $X^{16} + X^{15} + X^2 + 1$ .

Error results in adding in a polynomial. Use normal polynomial division intuition.

Single bit errors: result in addition of  $x^i$ . If  $G(x)$  has at least two terms, any multiple of  $G(x)$  will have two terms.

Two bit errors correspond to adding  $x^i + x^j$ , which will not divide if  $G(x)$  does not divide  $x^k + 1$  for sufficiently large  $k$ .

Odd bit error polynomials are never divisible by  $x + 1$ . So make  $G$  have  $x + 1$  as a factor.

Burst errors of length  $k$  adds  $x^i(x^{k-1} + ..1)$ . Can catch if  $k \leq$  polynomial degree. Any multiple of generator will have a term of  $x^k$  or higher.

## Lessons from Framing and CRCs

- End-to-end argument.
- Sublayering is a powerful tool: bit stuffing implementation, error recovery on top of framing. Sublayers extract their penalty
- Common problems at layers and exploiting solutions at other layers: coding, bit and frame synchronization, getting extra symbols from physical layer.
- Arguing by Analogy: ordinary division and CRC. Helps when trying to do CRC multiple bits at a time.
- Having Multiple Views: Bit string view for CRC computation and polynomial view for analysis.
- General and abstract approaches help: error detection in terms of coding.