EECS 150 - Components and Design Techniques for Digital Systems

Lec 18 – Error Coding

David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http://inst.eecs.berkeley.edu/~cs150

Outline

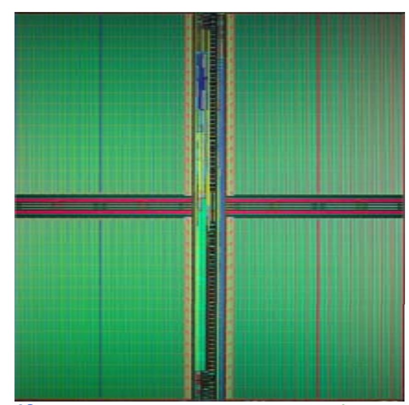
- Errors and error models
- Parity and Hamming Codes (SECDED)
- Errors in Communications
- LFSRs
- Cyclic Redundancy Check (CRC)

Our beautiful digital world….

- The real world has continuous electrical signals
- In the real world, electrons keep flowing
- In the real world, things take time
- We've designed circuits to create logical gates that behave like boolean operators
- We designed storage elements that hold their logical value
- We've developed a synchronous timing methodology so that values appear to change on clock edges
  - Acyclic combinational logic and storage elements
  - Clock cycle > worst propagation delay + setup

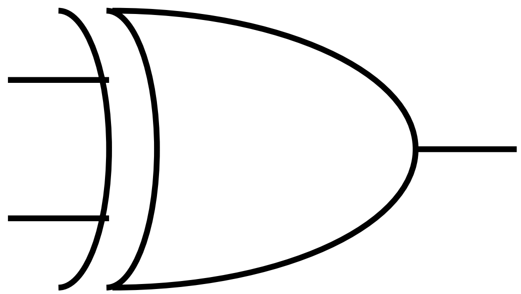
In the real world … α

- _ _ i t happens !
- Alpha particles flip bits in memory
- Electrostatics zap wires
- Electromagnetic interference clobbers communication
- …



The Challenge

- How do we design digital systems that behave correctly even in the presence of errors?



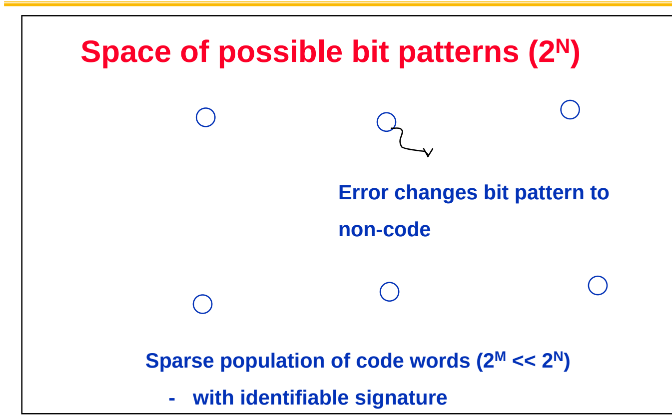
Definitions

- An *error* in a digital system is the corruption of data from its correct value to some other value.
- An error is caused by a physical *failure*.
  - Temporary or permanent
- The effects of failures are predicted by *error models*.
- Example: *independent error model*
  - a single physical failure is assumed to affect only a single bit of data – a single error
  - Multiple failures may cause multiple errors
    - » Much less likely

Error Correction Codes (ECC)

- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - “Soft” errors occur occasionally when cells are struck by alpha particles or other environmental upsets.
  - Less frequently, “hard” errors can occur when chips permanently fail.
  - Problem gets worse as memories get denser and larger
- Where is “perfect” memory required?
  - servers, spacecraft/military computers, ebay, …
- Memories are protected against failures with ECCs
- Extra bits are added to each data-word
  - used to detect and/or correct faults in the memory system
  - in general, each possible data word value is mapped to a unique “code word”. A fault changes a valid code word to an invalid one - which can be detected.

Correcting Code Concept



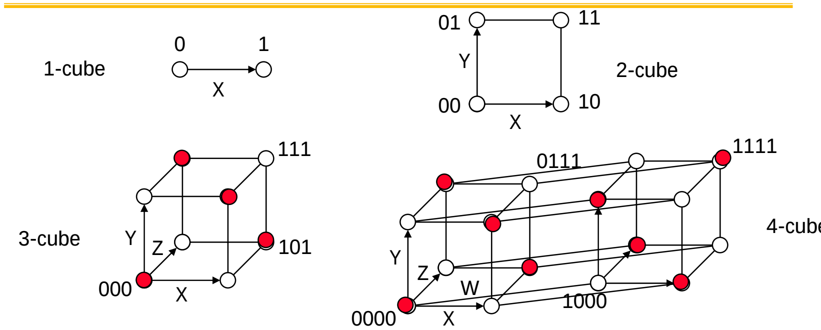
- Detection: bit pattern fails codeword check
- Correction: map to nearest valid code word

Simple Error Detection Coding: Parity

- Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have *even parity*:
- Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).

- A non-zero parity indicates an error occurred:
  - two errors (on different bits) is not detected (nor any even number of errors)
  - odd numbers of errors are detected.
- What is the probability of multiple simultaneous errors?

Recall: Boolean cubes



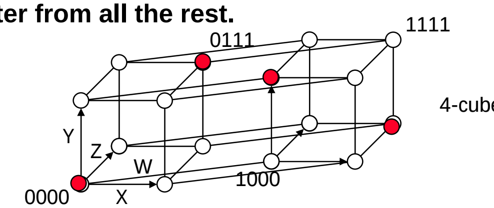
- Neighbors differs by one bit
- The *Hamming Distance* between two values is the number of bits that must be changed to convert one into the other.
- Parity – code words have minimum distance > 1

Single Error Detection

- N information bits + 1 parity bit
  - 2^N code words with minimum distance 2.
- What if we added another parity bit on the N+1 bits?
  - min-distance-3 code => detects double bit errors
- What do you do if an error is detected?
- What would you need to know to correct the error?

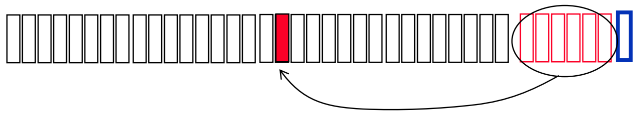
Error correction

- When we receive an non code word, we correct the error by locating the *nearest* code word
  - Extremely likely to have been the one that was transmitted
- Example: distance 3 code => single error will produce a value at distance 1 from the original and distance 2 or greater from all the rest.



- 2c+1 code can correct errors up to c bits
- 2c+d+1 code can correct errors up to c bits and detect errors in up to d additional bits
- SECDED most common

SECDED idea



- Add enough parity bits that with a single error the parity sequence gives the "address" of the bit that flipped!
- Add one more bit for parity of the whole thing
- How many bits does it take

Hamming Error Correcting Code

- Use more parity bits to pinpoint bit(s) in error, so they can be corrected.
- Example: Single error correction (SEC) on 4-bit data
  - use 3 parity bits, with 4-data bits results in 7-bit code word
  - 3 parity bits sufficient to identify any one of 7 code word bits
  - overlap the assignment of parity bits so that a single error in the 7-bit work can be corrected
- Procedure: group parity bits so they correspond to subsets of the 7 bits:
  - p_1 protects bits 1,3,5,7 (bit 1 is on)
  - p_2 protects bits 2,3,6,7 (bit 2 is on)
  - p_3 protects bits 4,5,6,7 (bit 3 is on)

1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Note: number bits from left to right.

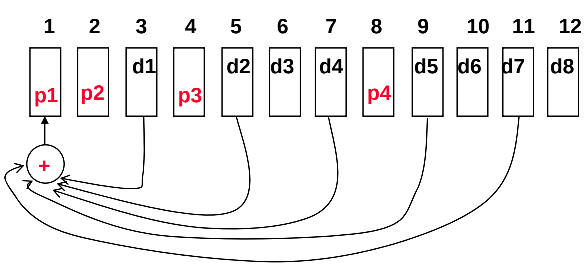
Bit position number

$001 = 1_{10}$
$011 = 3_{10}$
$101 = 5_{10}$
$111 = 7_{10}$ } p_1

$010 = 2_{10}$
$011 = 3_{10}$
$110 = 6_{10}$
$111 = 7_{10}$ } p_2

$100 = 4_{10}$
$101 = 5_{10}$
$110 = 6_{10}$
$111 = 7_{10}$ } p_3

Example: 8 bit SEC



- Takes four parity bits
  - In power of 2 positions
- Rest are the data bits
- Bits with i in their address feed into parity calculation for pi
- What to do with bit 0?

Hamming Code Example

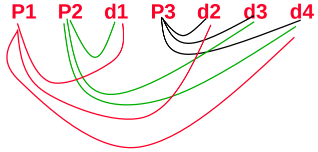
1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

- Note: parity bits occupy power-of-two bit positions in code-word.
- On writing to memory:
  - » parity bits are assigned to force even parity over their respective groups.
- On reading from memory:
  - » check bits (c_3,c_2,c_1) are generated by finding the parity of the group and its parity bit. If an error occurred in a group, the corresponding check bit will be 1, if no error the check bit will be 0.
  - » check bits (c_3,c_2,c_1) form the position of the bit in error.

- Example: $c = c_3c_2c_1 = 101$
  - error in 4,5,6, or 7 (by c_3=1)
  - error in 1,3,5, or 7 (by c_1=1)
  - no error in 2, 3, 6, or 7 (by c_2=0)
- Therefore error must be in bit 5.
- Note the check bits point to 5
- By our clever positioning and assignment of parity bits, the check bits always address the position of the error!
- c=000 indicates no error
  - *eight possibilities*

Interactive Quiz

1	2	3	4	5	6	7	positions
001	010	011	100	101	110	111	
P1	P2	d1	P3	d2	d3	d4	role



Position of error = $C_3C_2C_1$

Where C_i is parity of group i

- You receive:
  - 1111110
  - 0000010
  - 1010010
- What is the correct value?

Hamming Error Correcting Code

- Overhead involved in single error correction code:
  - let *p* be the total number of parity bits and *d* the number of data bits in a *p + d* bit word.
  - If p error correction bits are to point to the error bit (*p + d* cases) plus indicate that no error exists (1 case), we need:

    $2^p >= p + d + 1$,

    thus $p >= \log(p + d + 1)$

    for large *d*, *p* approaches $\log(d)$

  8 data => 4 parity

  16 data => 5 parity

  32 data => 6 parity

  64 data => 7 parity

- Adding on extra parity bit covering the entire word can provide double error detection

  1 2 3 4 5 6 7 8

  p_1 p_2 d_1 p_3 d_2 d_3 d_4 p_4

- On reading the C bits are computed (as usual) plus the parity over the entire word, P:

  C=0 P=0, no error

  C!=0 P=1, correctable single error

  C!=0 P=0, a double error occurred

  C=0 P=1, an error occurred in p_4 bit

Typical modern codes in DRAM memory systems:

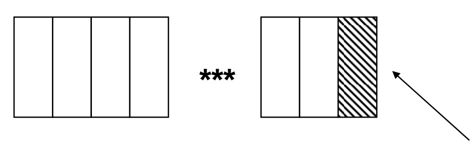
64-bit data blocks (8 bytes) with 72-bit code words (9 bytes).

Announcements

- Reading
  - http://en.wikipedia.org/wiki/Hamming_code
  - XILINX IEEE 802.3 Cyclic Redundancy Check (pages 1-3)
- Optional
  - http://www.ross.net/crc/download/crc_v3.txt

Concept: Redundant Check

- Send a message M and a "check" word C
- Simple function on <M,C> to determine if both received correctly (with high probability)
- Example: XOR all the bytes in M and append the "checksum" byte, C, at the end
  - Receiver XORs <M,C>
  - What should result be?
  - What errors are caught?



bit i is XOR of ith bit of each byte

Example: TCP Checksum

7 Application (HTTP,FTP, DNS)

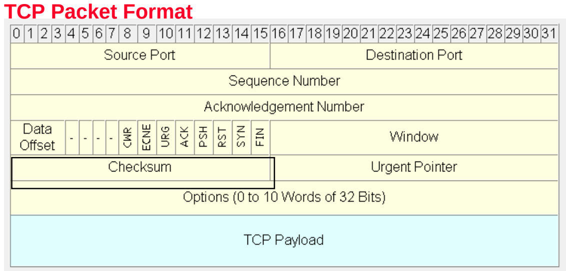
4 Transport (TCP, UDP)

3 Network (IP)

2 Data Link (Ethernet, 802.11b)

1 Physical

TCP Packet Format



- TCP Checksum a 16-bit checksum, consisting of the one's complement of the one's complement sum of the contents of the TCP segment header and data, is computed by a sender, and included in a segment transmission. (note end-around carry)
- Summing all the words, including the checksum word, should yield zero

Detecting burst errors

- In a network link or a magnetic disk, the failure that causes and errors often causes a burst of errors
  - Wipes a sequence of bytes
- What can we do to detect such burst errors?

Example: Ethernet CRC-32

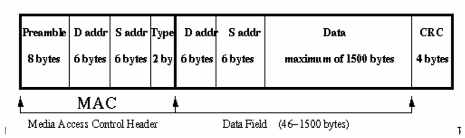
7 Application (HTTP,FTP, DNS)

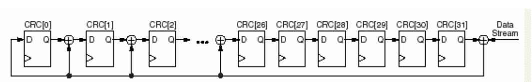
4 Transport (TCP, UDP)

3 Network (IP)

2 Data Link (Ethernet, 802.11b)

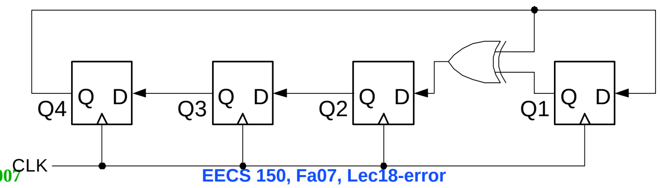
1 Physical



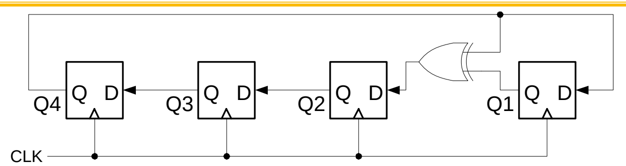


Linear Feedback Shift Registers (LFSRs)

- These are n-bit counters exhibiting *pseudo-random* behavior.
- Built from simple shift-registers with a small number of xor gates.
- Used for:
  - random number generation
  - counters
  - error checking and correction
- Advantages:
  - very little hardware
  - high speed operation
- Example 4-bit LFSR:



4-bit LFSR




```
        0 0 0 1 0
    xor 0 0 0 0 0
          0 0 0 1 0 0
      xor 0 0 0 0 0
            0 0 1 0 0 0
        xor 0 0 0 0 0
              0 1 0 0 0 0
          xor 1 0 0 1 1
                0 0 0 1 1 0
            xor 0 0 0 0 0
                  0 0 1 1 0 0
              xor 0 0 0 0 0
                    0 1 1 0 0 0
                xor 1 0 0 1 1
                      0 1 0 1 1
```

Q4 Q3 Q2 Q1

0001
0010
0100
1000
0011
0110
1100
1011
0101
1010
0111
1110
1111
1101
1001
0001

- Circuit counts through 2^4-1 different non-zero bit patterns.
- Left most bit determines shiftl or more complex operation
- Can build a similar circuit with any number of FFs, may need more xor gates.
- In general, with n flip-flops, 2^n-1 different non-zero bit patterns.
- (Intuitively, this is a counter that *wraps around* many times and in a strange way.)

Applications of LFSRs

- Performance:
  - In general, xors are only ever 2-input and never connect in series.
  - Therefore the minimum clock period for these circuits is:

    $T > T_{2\text{-input-xor}}$ + clock overhead
  - Very little latency, and independent of n!
- This can be used as a fast counter, if the particular sequence of count values is not important.
  - Example: micro-code micro-pc
- Can be used as a random number generator.
  - Sequence is a pseudo-random sequence:
    - » numbers appear in a random sequence
    - » repeats every 2^n-1 patterns
  - Random numbers useful in:
    - » computer graphics
    - » cryptography
    - » automatic testing
- Used for error detection and correction
  - » CRC (cyclic redundancy codes)
  - » ethernet uses them

CRC concept

- I have a msg polynomial M(x) of degree m
- We both have a generator poly G(x) of degree m
- Let r(x) = remainder of M(x) $\boxed{x^n}$ / G(x)
  - $M(x)\,x^n = G(x)p(x) + r(x)$
  - r(x) is of degree n
- What is $(M(x)\,x^n - r(x)) / G(x)$?

n bits of zero at the end

- So I send you M(x) x^n $\boxed{- r(x)}$
  - m+n degree polynomial
  - You divide by G(x) to check
  - M(x) is just the m most signficant coefficients, r(x) the lower m

tack on n bits of remainder

Instead of the zeros

- n-bit Message is viewed as coefficients of n-degree polynomial over binary numbers

Galois Fields - the theory behind LFSRs

- LFSR circuits performs multiplication on a *field*.
- A field is defined as a *set* with the following:
  - two operations defined on it:
    - » "addition" and "multiplication"
  - closed under these operations
  - associative and distributive laws hold
  - additive and multiplicative identity elements
  - additive inverse for every element
  - multiplicative inverse for every non-zero element
- Example fields:
  - set of rational numbers
  - set of real numbers
  - set of integers is *not* a field (why?)
- Finite fields are called *Galois* fields.
- Example:
  - Binary numbers 0,1 with XOR as "addition" and AND as "multiplication".
  - Called GF(2).

  - 0+1 = 1
  - 1+1 = 0
  - 0-1 = ?
  - 1-1 = ?

Galois Fields - The theory behind LFSRs

- Consider *polynomials* whose coefficients come from GF(2).
- Each term of the form x^n is either present or absent.
- Examples: $0, 1, x, x^2$, and $x^7 + x^6 + 1$

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

- With addition and multiplication these form a field:
- "Add": XOR each element individually with no carry:

$$\begin{array}{r} x^4 + x^3 + x + 1 \\ +\; x^4 + x^2 + x \\ \hline x^3 + x^2 + 1 \end{array}$$

- "Multiply": multiplying by x^n is like shifting to the left.

$$\begin{array}{r} x^2 + x + 1 \\ \times \qquad x + 1 \\ \hline x^2 + x + 1 \\ x^3 + x^2 + x \\ \hline x^3 \qquad\quad + 1 \end{array}$$

So what about division (mod)

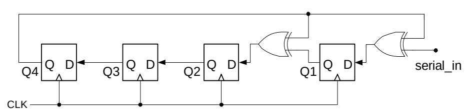
$$\frac{x^4 + x^2}{x} = x^3 + x \text{ with remainder 0}$$

$$\frac{x^4 + x^2 + 1}{x + 1} = x^3 + x^2 \text{ with remainder 1}$$

$$\begin{array}{r|l} & x^3 + x^2 + 0x \;\; + 0 \\ \hline x + 1 & x^4 + 0x^3 + x^2 + 0x + 1 \\ & x^4 + x^3 \\ \hline & \qquad x^3 + x^2 \\ & \qquad x^3 + x^2 \\ \hline & \qquad\qquad 0x^2 + 0x \\ & \qquad\qquad\qquad 0x + 1 \end{array}$$

Remainder 1

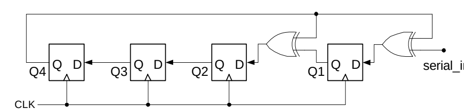
Polynomial division




```
         0 0 0 0 1 0 1
       ---------------------
1 0 0 1 1 | 1 0 1 1 0 0 1 0 0 0 0
            1 0 0 1 1
            0 0 1 0 1
              0 1 0 1 0
                1 0 1 0 1
                1 0 0 1 1
                0 0 1 0 0
```

- When MSB is zero, just shift left, bringing in next bit
- When MSB is 1, XOR with divisor and shiftl

CRC encoding



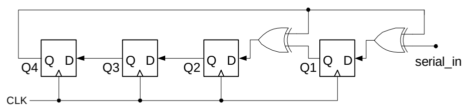
1 0 1 1 0 0 1 0 0 0 0

0	0	0	0	
0	0	0	1	0 1 1 0 0 1 0 0 0 0
0	0	1	0	1 1 0 0 1 0 0 0 0
0	1	0	1	1 0 0 1 0 0 0 0
1	0	1	1	0 0 1 0 0 0 0
0	1	0	1	0 1 0 0 0 0
1	0	1	0	1 0 0 0 0
0	1	1	0	0 0 0 0
1	1	0	0	0 0 0
1	0	1	1	0 0
0	1	0	1	0
1	0	1	0	

Message sent:

1 0 1 1 0 0 1 1 0 1 0

CRC decoding



1 0 1 1 0 0 1 1 0 1 0

Q4	Q3	Q2	Q1	
0	0	0	0	
0	0	0	1	0 1 1 0 0 1 1 0 1 0
0	0	1	0	1 1 0 0 1 1 0 1 0
0	1	0	1	1 0 0 1 1 0 1 0
1	0	1	1	0 0 1 1 0 1 0
0	1	0	1	0 1 1 0 1 0
1	0	1	0	1 1 0 1 0
0	1	1	0	1 0 1 0
1	1	0	1	0 1 0
1	0	0	1	1 0
0	0	0	0	0
0	0	0	0	

Galois Fields - The theory behind LFSRs

- **These polynomials form a *Galois (finite)* field if we take the results of this multiplication modulo a prime polynomial $p(x)$.**
  - **A prime polynomial is one that cannot be written as the product of two non-trivial polynomials $q(x)r(x)$**
  - **Perform modulo operation by subtracting a (polynomial) multiple of $p(x)$ from the result. If the multiple is 1, this corresponds to XOR-ing the result with $p(x)$.**
- **For any degree, there exists at least one prime polynomial.**
- **With it we can form $GF(2^n)$**
- **Additionally, …**
- **Every Galois field has a primitive element, α, such that all non-zero elements of the field can be expressed as a power of α. By raising α to powers (modulo $p(x)$), all non-zero field elements can be formed.**
- **Certain choices of $p(x)$ make the simple polynomial x the primitive element. These polynomials are called *primitive*, and one exists for every degree.**
- **For example, $x^4 + x + 1$ is primitive. So $\alpha = x$ is a primitive element and successive powers of α will generate all non-zero elements of GF(16). *Example on next slide.***

Galois Fields – Primitives

$\alpha^0 = 1$
$\alpha^1 = x$
$\alpha^2 = x^2$
$\alpha^3 = x^3$
$\alpha^4 = x + 1$
$\alpha^5 = x^2 + x$
$\alpha^6 = x^3 + x^2$
$\alpha^7 = x^3 + x + 1$
$\alpha^8 = x^2 + 1$
$\alpha^9 = x^3 + x$
$\alpha^{10} = x^2 + x + 1$
$\alpha^{11} = x^3 + x^2 + x$
$\alpha^{12} = x^3 + x^2 + x + 1$
$\alpha^{13} = x^3 + x^2 + 1$
$\alpha^{14} = x^3 + 1$
$\alpha^{15} = 1$

- **Note this pattern of coefficients matches the bits from our 4-bit LFSR example.**

$$\begin{aligned}\alpha^4 &= x^4 \bmod x^4 + x + 1\\ &= x^4 \text{ xor } x^4 + x + 1\\ &= x + 1\end{aligned}$$

- **In general finding primitive polynomials is difficult. Most people just look them up in a table, such as:**

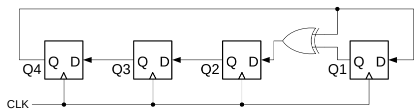
Primitive Polynomials

$x^2 + x + 1$
$x^3 + x + 1$
$x^4 + x + 1$
$x^5 + x^2 + 1$
$x^6 + x + 1$
$x^7 + x^3 + 1$
$x^8 + x^4 + x^3 + x^2 + 1$
$x^9 + x^4 + 1$
$x^{10} + x^3 + 1$
$x^{11} + x^2 + 1$
$x^{12} + x^6 + x^4 + x + 1$
$x^{13} + x^4 + x^3 + x + 1$
$x^{14} + x^{10} + x^6 + x + 1$
$x^{15} + x + 1$
$x^{16} + x^{12} + x^3 + x + 1$
$x^{17} + x^3 + 1$
$x^{18} + x^7 + 1$
$x^{19} + x^5 + x^2 + x + 1$
$x^{20} + x^3 + 1$
$x^{21} + x^2 + 1$
$x^{22} + x + 1$
$x^{23} + x^5 + 1$
$x^{24} + x^7 + x^2 + x + 1$
$x^{25} + x^3 + 1$
$x^{26} + x^6 + x^2 + x + 1$
$x^{27} + x^5 + x^2 + x + 1$
$x^{28} + x^3 + 1$
$x^{29} + x + 1$
$x^{30} + x^6 + x^4 + x + 1$
$x^{31} + x^3 + 1$
$x^{32} + x^7 + x^6 + x^2 + 1$

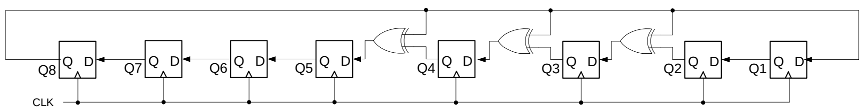
Galois Field		Hardware
Multiplication by x	⇔	**shift left**
Taking the result mod $p(x)$	⇔	**XOR-ing with the coefficients of $p(x)$ when the most significant coefficient is 1.**
Obtaining all 2^n-1 non-zero elements by evaluating x^k for $k = 1, \ldots, 2^n-1$	⇔	**Shifting and XOR-ing 2^n-1 times.**

Building an LFSR from a Primitive Poly

- For *k-bit* LFSR number the flip-flops with FF1 on the right.
- The feedback path comes from the Q output of the leftmost FF.
- Find the primitive polynomial of the form $x^k + \ldots + 1$.
- The $x^0 = 1$ term corresponds to connecting the feedback directly to the D input of FF 1.
- Each term of the form x^n corresponds to connecting an xor between FF n and $n+1$.
- 4-bit example, uses $x^4 + x + 1$
  - $x^4 \Leftrightarrow$ FF4's Q output
  - $x \Leftrightarrow$ xor between FF1 and FF2
  - $1 \Leftrightarrow$ FF1's D input



- To build an 8-bit LFSR, use the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ and connect xors between FF2 and FF3, FF3 and FF4, and FF4 and FF5.



Generating Polynomials

- CRC-16: G(x) = $x^{16} + x^{15} + x^2 + 1$
  - detects single and double bit errors
  - All errors with an odd number of bits
  - Burst errors of length 16 or less
  - Most errors for longer bursts
- CRC-32: G(x) = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
  - Used in ethernet
  - Also 32 bits of 1 added on front of the message
    - » Initialize the LFSR to all 1s

Summary

- Concept of error coding
  - Add a few extra bits (enlarges the space of values) that carry information about all the bits
  - Detect: Simple function to check of entire data+check received correctly
    - » Small subset of the space of possible values
  - Correct: Algorithm for locating nearest valid symbol
- Hamming codes
  - Selective use of parity functions
  - Distance + # bit flips
  - Parity: XOR of the bits => single error detection
  - SECDED
    - » databits+p+1 < 2^p
- Cyclic Redundancy Checks
  - Detect burst errors