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.
- Where "perfect" memory is required
  - servers, spacecraft/military computers, …
- Memories are protected against failures with ECCs
- Extra bits are added to each data-word
  - extra bits are 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.

Simple Error Detection Coding

Parity Bit

- 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.
### 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 resulting 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
  - Group parity bits so they correspond to subsets of the 7 bits:
    - $p_1$ protects bits 1, 3, 5, 7
    - $p_2$ protects bits 2, 3, 6, 7
    - $p_3$ protects bits 4, 5, 6, 7

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
p_1 & p_2 & d_1 & p_3 & d_2 & d_3 & d_4 \\
\end{array}
\]

### Hamming Code Example

- **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!

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & p & \\
001 & 011 & 101 & 111 & 010 & 011 & 110 & 111 & \\
\end{array}
\]

### 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:
  \[2^p \geq p + d + 1,\]
  thus $p \geq \log(p + d + 1)$
for large $d$, $p$ approaches $\log(d)$

### Typical modern codes in DRAM memory systems:

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

### 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:
  - pseudo-random number generation
  - counters
  - error checking and correction
- Advantages:
  - very little hardware
  - high speed operation
- Example 4-bit LFSR:

\[
\begin{array}{cccc}
Q_4 & Q_3 & Q_2 & Q_1 \\
\end{array}
\]
4-bit LFSR

- Circuit counts through \(2^4-1\) different non-zero bit patterns.
- Leftmost bit decides whether the "10011" xor pattern is used to compute the next value or if the register just shifts left.
- 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:
    \[ T > T_{2\text{-input-xor}} + \text{clock overhead} \]
  - 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

### 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
- Finite fields are called Galois fields.
- Example:
  - Binary numbers 0,1 with XOR as "addition" and AND as "multiplication".
  - Called GF(2).

- Consider polynomials whose coefficients come from GF(2).
- Each term of the form \(x^i\) is either present or absent.
- Examples: \(0, 1, x, x^2, x^5 + x\)
- With addition and multiplication these form a field:
  - "Add": XOR each element individually with no carry:
    \[
    x^i + x^j + x + 1
    \]
  - "Multiply": multiplying by \(x^i\) is like shifting to the left.
    \[
    x^i + x^j + x + 1
    \]
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, \( \alpha \), such that all non-zero elements of the field can be expressed as a power of \( \alpha \). By raising \( \alpha \) 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 \( \alpha \) will generate all non-zero elements of \( GF(16) \).
    - Example on next slide.

**Primitive Polynomials**

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

Galois Field

- Multiplication by \( x \) ⇔ shift left
- Taking the result mod \( p(x) \) ⇔ XOR-ing with the coefficients of \( p(x) \)
- Obtaining all \( 2^n - 1 \) non-zero elements by evaluating \( x^k \) for \( k = 1, \ldots, 2^n - 1 \)

Galois Field Hardware

- Note this pattern of coefficients matches the bits from our 4-bit LFSR example.
- In general finding primitive polynomials is difficult. Most people just look them up in a table, such as:

```plaintext
ν0 = \( x^4 \mod x^4 + x + 1 \)
ν1 = \( x^4 \oplus x^4 + x + 1 \)
ν2 = \( x \)
```

Building an LFSR from a Primitive Polynomial

- 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 \( x^{n+1} + 1 \).
- The \( x^i \) term corresponds to connecting the feedback directly to the D input of FF1.
- Each term of the form \( x^i \) corresponds to connecting an xor between FF \( n \) and \( n+i \).
- 4-bit example, uses \( x^4 + x + 1 \)
  - \( x^4 \) ⇔ FF4’s Q output
  - \( x \) ⇔ xor between FF1 and FF2
  - \( i \) ⇔ 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.
Error Correction with LFSRs

11 message bits  4 check bits

bit sequence: 1 1 0 0 1 0 0 0 1 1 1 0 0 0

• XOR Q4 with incoming bit sequence. Now values of shift-register don't follow a
  fixed pattern. Dependent on input sequence.
• Look at the value of the register after 15 cycles: “1010”
• Note the length of the input sequence is $2^4 - 1 = 15$ (same as the number of
different nonzero patterns for the original LFSR)
• Binary message occupies only 11 bits, the remaining 4 bits are “0000”.
  – They would be replaced by the final result of our LFSR: “1010”
  – If we run the sequence back through the LFSR with the replaced bits, we would get
    “0000” for the final result.
  – 4 parity bits, “neutralize” the sequence with respect to the LFSR.
• If parity bits not all zero, an error occurred in transmission.
• If number of parity bits = log total number of bits, then single bit errors can be
corrected.
• Using more parity bits allows more errors to be detected.
• Ethernet uses 32 parity bits per frame (packet) with 16-bit LFSR.