Outline

- Galois fields algebra
- Reed-Solomon Codes
- Low-density parity-check codes
Need a Bit of Algebra

- Coding operations are usually done *modulo-2*
  - For circuit designers: an XOR is modulo-2 addition

- Finite fields, GF($2^n$)
  - Sets of numbers generated using generator polynomials
  - Any operation that involves numbers from the field produces results that are also in the field

- Example:
  - $x = 2$
  - $x^3 + x^2 + 1 = 2^3 + 2^2 + 1 = 1101$
  - $(x^3 + x^2 + 1)(x^4 + x^2 + 1) = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$
    $= x^7 + x^6 + x^5 + x^3 + 1$

Generator Polynomials

- Generator polynomials (GP) are primitive (irreducible).
- E.g. $x^4 + 1$ is not primitive - reduces to $x + 1$ and $x^3 + x^2 + x + 1$

- Example of a GP
  - $x^2 + x + 1$
  - $x^3 + x + 1$
  - $x^3 + x^2 + 1$

- GP describes the field of the order $n-1$
- PRBS is an example sequence generated by a GP
Galois Fields

- E.g. GF(2^3)
- GP: \(x^3 + x + 1 = 0\)
- Chose an element \(\alpha\) (primitive root)
  \[\alpha = x = \text{010 (2)}; \quad \alpha^{2^n - 1} = \alpha^0 = 1\]
  \[
  \begin{align*}
  \alpha^1 &= x = \text{010 (2)} \\
  \alpha^2 &= x^2 = \text{100 (4)} \\
  \alpha^3 &= x^3 = x + 1 = \text{011 (3)} \\
  \alpha^4 &= \alpha \alpha^3 = x(x + 1) = \text{110 (6)} \\
  \alpha^5 &= \alpha \alpha^4 = x(x^2 + x) = (x + 1) + x^2 = \text{111 (7)} \\
  \alpha^6 &= \alpha^2 \alpha^4 = x^2(x^2 + x) = x^2 + 1 = \text{101 (5)} \\
  \alpha^7 &= \alpha \alpha^6 = x(x^2 + 1) = (x + 1) + x = \text{001 (1)}
  \end{align*}
\]

Operations in Galois Fields

- Addition is just an XOR
  \[
  \begin{align*}
  \alpha + \alpha^3 &= \text{010 + 011 = 001, or} \\
  \alpha + \alpha^3 &= x + (x + 1) = (x + x) + 1 = 001
  \end{align*}
\]
- Multiplication
  \[
  \begin{align*}
  \alpha^4 \alpha^5 &= \alpha^9 = \alpha^7 \alpha^2 = 1\alpha^2 = \alpha^2 = 100 \text{ or} \\
  \alpha^4 \alpha^5 &= (x^2 + x)(x^2 + x + 1) = x^4 + x^3 + x^2 + x^2 + x = x^4 + x \\
  &= xx^3 + x = x(x + 1) + x = x^2 = 100
  \end{align*}
\]
Reed-Solomon Codes

- Invented ~1960
- A special case of BCH (Bose-Chaudhury-Hocquenghem) codes
- Starts with bits, $d_i$
  \[ \sum_{i=0}^{m-1} d_i \alpha^i = 0 \]
- And replaces with symbols, $p_i$
  \[ \sum_{i=0}^{m-1} p_i \alpha^i = 0 \]
- Symbols are typically 8-bit (don’t have to be)

RS($n$, $k$) code over GF($2^m$)
- $2^m - 1$ symbols
- $k$ user symbols
- Minimum distance $n - k + 1$
- Can correct up to $t = (n - k)/2$ errors
- E.g. RS(255,239) over GF($2^8$) code is used in long-haul optical communications
  - 16 parity bytes added to 239 user bytes (7% overhead)
  - 5.5dB coding gain @ BER = $10^{-12}$
  - This coding gain moves BER from $10^{-4}$ to $10^{-15}$
    or from $10^{-5}$ to $10^{-24}$ (at given SNR)
  - Can correct bursts up to 64b
  - 16-way interleaved code can correct 1024-b bursts
Reed-Solomon Encoding

- Assume: E.g. GF(2^3), GP: \( x^3 + x + 1 = 0 \), 3-bit symbols
- Message: 3, 4, 6, 0, 1, 1, and add \( p_0 \) as a check symbol
- \( 3, 4, 6, 0, 1, 1, p_0 = \alpha^3, \alpha^2, \alpha^4, 0, \alpha^0, \alpha^0, p_0 \)
- The encoded word must satisfy: \( \sum_{i=0}^{m-1} p_i \alpha^i = 0 \)
- \( \alpha^3 \alpha^6 + \alpha^2 \alpha^5 + \alpha^4 \alpha^4 + 0 \alpha^3 + \alpha^0 \alpha^2 + \alpha^0 \alpha^1 + p_0 \alpha^0 = 0 \)
- \( \alpha^2 + \alpha^0 + \alpha^2 + 0 + \alpha^2 + \alpha^1 + p_0 = 0 \)
- Therefore \( p_0 = \alpha^5 \)

Reed-Solomon Decoding

- If no errors – the polynomial evaluates to 0
- If there is an error at e.g. position \( \alpha^2 \):
- \( \alpha^3, \alpha^2, \alpha^4, 0, \alpha^0, (\alpha^0 + e), \alpha^5 \)
- Syndrome is
- \( \alpha^3 \alpha^6 + \alpha^2 \alpha^5 + \alpha^4 \alpha^4 + 0 \alpha^3 + (\alpha^0 + e) \alpha^2 + \alpha^0 \alpha^1 + p_0 \alpha^0 = 0 + e \alpha^2 \)
- The error can be detected, but can’t be corrected
  - Need to add another symbol
Reed-Solomon Decoding

- New constraint \( \sum_{i=0}^{m-1} p_i \alpha^{ij} = 0 \) for \( j = 0, 1 \)

- Our example (with one symbol less):
  - \( j = 0: \alpha^3 + \alpha^2 + \alpha^4 + 0 + \alpha^0 + p_1 + p_0 = 0 \)
  - \( j = 1: \alpha^3 \alpha^6 + \alpha^2 \alpha^5 + \alpha^4 \alpha^4 + 0 \alpha^3 + \alpha^0 \alpha^2 + p_1 \alpha^1 + p_0 \alpha^0 = 0 \)
  - \( p_1 + p_0 = 0; p_1 \alpha + p_0 = 0 \), therefore, \( p_1 = p_0 = 1 = \alpha^0 \)
- The receiver calculates two syndromes, for \( j = 0 \) and \( j = 1 \):
  \[ S_j = \sum_{i=0}^{m-1} p_i \alpha^{ij} \]

---

Reed-Solomon Decoding

- Adding an error \( e = \alpha \)
- Our message \( \alpha^3, \alpha^2, \alpha^4, 0, \alpha^0, (\alpha^0 + e), \alpha^5 \)
- Becomes \( \alpha^3, \alpha^2, \alpha^4, 0, \alpha^0, \alpha^3, \alpha^5 \)
- \( S_0 = \alpha \) - error magnitude
- \( S_1 = \alpha^2 \) - \( e\alpha^k \), \( k \) is the position
- \( \alpha^2 = S_1/S_0 \)
- In this example \( S_1/S_0 = \alpha^1, k = 1 \)
- The error is corrected by adding \( S_0 \) to the received symbol at position \( k \).
- This can be extended to correct for multiple errors
Correcting Multiple Errors

- To correct for 2 symbol errors, need 4 checks:
  2 positions + 2 error magnitudes
- Decoder evaluates 4 syndromes
- But it is difficult to work out the error magnitudes and error locations from them
  - Simultaneous equations in GF (so called “key equations”)
- Alternatives are trial-and-error solutions, known as Berlekamp-Massey algorithm and Euclidean algorithm

Practical RS Decoding

1. Syndrome computation
   \[ S_j = \sum_{i=0}^{m-1} p_i \alpha^{ixj} \]

2. Key equation
   \[ S_j = \Lambda_j \cdot \Omega_j \mod(\alpha^m) \]
   \( \Lambda_j \) – error locator, \( \Omega_j \) – error evaluator polynomials

3. Compute error locations and error values from \( \Lambda_j, \Omega_j \)
   - Error locations – Chien’s search
   - Error values – Forney’s algorithm
Decoder Design

DVD RS Code

Product RS code

Chang, JSSC 02/01
Decoding Sequence

The data stream of N-th codeword

Syndrome Calculator

the data stream of (N+1)-th codeword

Key Equation Solver

σ(x) Ω(x)

Chien Search

Error Value Evaluator

Syndrome Calculation

Syndrome calculation is straightforward

 Syndrome cell

 Syndrome calculator cell

\[ S_j = \sum_{i=0}^{m-1} p_i \alpha^{ixj} \]

Chang, JSSC 02/01
Solving the Key Equation

- Iterative solutions using Berlekamp-Massey or Euclid's algorithm
  - Both require GF multiplication and division in each iteration
- Euclidean algorithm calculates the greatest common divisor among two polynomials ($S_j$ and $\alpha^2$)
  - Solves the key equation by iterative polynomial division and multiplication
- Division-free Euclidean algorithm [Shao, CICC’85]
- Inversionless Berlekamp-Massey algorithm [Shayan, TCom’93]
  - Replaces polynomial division with cross-multiplication

Berlekamp-Massey Computation

$$\Delta_{r+1} = \sum_{j=0}^{r} \Lambda_j^{(r)} \cdot S_{r+1-j}, \quad \Lambda^{(r+1)}(x) = R^{(r)} \cdot \Lambda^{(r)}(x) - \Delta_{r+1} \cdot x \cdot B^{(r)}(x)$$

Song, ISSCC’02
Modified Euclidean Computation

\[ R(x) = q_{msb} \cdot R(x) - r_{msb} \cdot Q(x) \cdot x^1 \]

Coefficients of \( R(x) \)

Coefficients of \( Q(x) \)

Song, ISSCC'02

Modified Euclidean Algorithm

Song, JSSC 11/02
Chien Search

- Chien search evaluates polynomials

---

Why Use FEC?

- Relax component requirements
- Improve distance
  - 3dB can double the reach of DSL
- Increase capacity (rate)
  - 3dB can improve the rate by 50% in QPSK
- Reduce system cost
- Improve system quality

- Only total system evaluation can determine if it is worth to implement FEC in high-speed links
Example RS Decoders

- Song, JSSC 11/02, 10Gb/s, 4 x 2.5Gb/s 83MHz, 0.16µm CMOS, 340mW
- Song, JSSC 11/02, 40Gb/s, 112MHz, 0.16µm CMOS, 360mW
- Seki, CICC’01 10Gb/s, 0.18µm CMOS, 350mW
- RS decoders are typically a part of bigger chips (10mm x 10mm) that include framing, Mux/Demux, 1-10W

OC-192 RS CoDec

Seki, CICC’01
Performance of RS Decoders

Parity Checks

- (7, 4) code, 4 user bits and 3 parity checks
  - \( d_0, d_5, d_4, d_3, p_2, p_1, p_0 \)

Parity checks:
  - \( p_0 = d_3 + d_5 + d_6 \)
  - \( p_1 = d_3 + d_4 + d_5 \)
  - \( p_2 = d_3 + d_4 + d_6 \)

The code has a distance of 3, therefore can correct 1 error
Parity Checks

- Parity check equations
  \[ d_6 + d_5 + d_3 + p_0 = 0 \]
  \[ d_5 + d_4 + d_9 + p_1 = 0 \]
  \[ d_6 + d_5 + d_3 + p_2 = 0 \]

- Parity-check matrix
  \[
  \begin{array}{ccccccc}
  1 & 1 & 0 & 1 & 0 & 0 & 1 \\
  0 & 1 & 1 & 1 & 0 & 1 & 0 \\
  1 & 1 & 0 & 1 & 1 & 0 & 0 \\
  \end{array}
  \]

Low-Density Parity-Check Code

- Parity codes based on sparse H matrices
- Discovered in 1960, rediscovered in 1990s
- Exceptional error correcting capability
  - Within 0.01dB of Shannon bound
- Efficient decoding based on message passing
- Part of almost every new communications standard, like ADSL2+, 10GBase-T
Message Passing Analogy (Trellis)

How many people are there?

Output @ each node = (Marginalized sum of inputs + 1 )
**LDPC: Overview**

- **LDPC representation by either parity check matrix or the bi-partite graph.**
- **Message passing: Bit-to-Check / Check-to-Bit**
- **Total Number of Edges:** 18432

\[
\begin{bmatrix}
1 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 1 \\
0 & 1 & 1 & \cdots & 0 \\
1 & 0 & 0 & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 1 & \cdots & 1 \\
\end{bmatrix}
\]

- Column Weight = 4

---

**Parallel LDPC Decoder Architecture**

- Fully parallel structure
  - e.g. satellite receiver:
  - 64,000 variable node processing elements, PE_v
  - Variable number check node processing elements, PE_c
- High throughput; low power
- Routing complexity

\[w_c = 512 \quad w_r = 4608\]

\[\text{Column Weight} = 4\]

\[\text{Row Wt.} = 36\]

\[\text{PE}_{v1} \quad \text{PE}_{v2} \quad \text{PE}_{v3} \quad \text{PE}_{v4} \quad \cdots \quad \text{PE}_{vN}\]

\[\text{PE}_{c1} \quad \text{PE}_{c2} \quad \cdots \quad \text{PE}_{cM}\]

---


[A. Blanksby and C. J. Howland, JSSC 2002]
Parallel LDPC Decoder Architecture

- 1Gb/s, 690mW, 1024-bit blocks, rate -1/2
- Routing density 50%, 50mm² chip

Bit-to-Check Message Computation

\[ Q_{4} = R_{i,1} + R_{i,2} + R_{i,3} \]

Parallel Tree-Adder Structure
Check-to-Bit Computations

\[ R_{ij} = f(Q_{2j}, Q_{3j}, \ldots, Q_{36j}) \]

\[ f(Q_1, Q_2, \ldots, Q_N) \approx \prod_i \text{sgn}(Q_i) \cdot \text{Min}\{Q_i\} \]

Recursive-Serial Adder Structure

LDPC Code Performance

- BER performance of LDPC depends on
  - Graph girth
  - Code expansion + block size
  - Hamming distance
- “Random” parity-check matrices usually achieve good performance
  - The interconnect network is random, too
  - A 1024-bit LDPC decoder in 0.15µm occupies 7mm x 7mm with 50% density [Blanksby, Howland, JSSC’02]
- Structured codes achieve good performance with structured interconnect
  - Permutation codes, Ramanujan and Cayley graphs
Structured LDPCs

- Construction based on Ramanujan graphs allows for hierarchical decomposition and good performance

Serial LDPC Decoder Architecture

- Latency dependent on total number of nodes
- Messages are stored in SRAM
  - Large memory requirement
- Natural structure for microprocessors, DSPs, etc.
- Parallelizing computation with limited PEs

[Al-Rawi, J., Cioffi, and M. Horowitz, ITCC 2001]
Staggered Serial LDPC Decoder

- Increase number of variable node PEs
- Staggered message updating
  - reduced complexity of PE
- Messages stored in variable node PEs
  - reduced memory requirement
- Improved BER @ reduced iteration counts

[E. Yeo, et. al. Globecom2001]

Pipelined Architecture of LDPC Decoder

1. Randomness of connectivity in bi-partite graph inhibits any kind of memory reuse.
2. Two banks of memory alternating between read and write required.
3. Total memory requirement: 72k words
LDPC Codes Based on Galois Fields

- Codes based on GF projections are low rate.
  - No cycles of length 4 (short loop)
  - Cyclic rows
  - e.g. (1023 x 1023) code has rate of 0.68

- **Column splitting**
  - Each column in original matrix is split into four
  - Non-zero entries in original column are cycled through the 4 new columns
  - e.g. (1023 x 4092) code has rate of 0.75
  - Partial loss of regularity (cyclic structure)
  - Complex $O(N^2)$ encoding

- **Puncturing**
  - Truncate height of PC matrix
  - Columns in the maximum zero runlength region correspond to parity bit locations
  - Cyclic encoding using direct application of PC matrix now possible

Shift Register-Based Implementation

- Staggered decoding.
- Regularity of codes based on Finite Field geometries.

[E. Yeo, et. al. Globecom2001]
LDPC Performance

LDPC Error Floors

Richardson, Allerton’03