Making Error Correcting Codes Work for Flash Memory
Part I: Primer on ECC, basics of BCH and LDPC codes

Lara Dolecek

Laboratory for Robust Information Systems (LORIS)
Center on Development of Emerging Storage Systems (CoDESS)
Department of Electrical Engineering, UCLA
ECC is a must for Flash!

Ariel Maislos, “A New Era in Embedded Flash Memory”, Flash Summit 2011 (Anobit)
ECC is a must for Flash!

Ariel Maislos, “A New Era in Embedded Flash Memory”, Flash Summit 2011 (Anobit)
ECC is a must for Flash!

Ariel Maislos, “A New Era in Embedded Flash Memory”, Flash Summit 2011 (Anobit)
Today we will

- Learn the basics of ECC operations
- Learn about fundamental coding approaches (BCH, LDPC)
- Learn about the system-level perspective on ECC
- Learn about recent advanced coding-oriented approaches to Flash
Errors in Flash are modeled as transmission a noisy communication channel

The simplest example is binary symmetric channel (BSC).
A Simple Channel/Storage Model

• Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
• While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
• In this example $P_e = 0.01$, $Pr(\text{success}) = 1 - P_e = 0.99$.
• The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, $RBER = 0.01$.

Errors in Flash are modeled as transmission a noisy communication channel.
The simplest example is binary symmetric channel (BSC).
This example:
1. Raw bit error rate (RBER) is 0.01.
2. Undetected bit error rate (UBER) is 0.01.
A Simple Channel/Storage Model

- Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example, $P_e = 0.01$, $Pr(\text{success}) = 1 - P_e = 0.99$.
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, RBER = 0.01.

Suppose we now use repetition coding:

0 → 000, 1 → 111

Decoding rule:

- Receive {000, 001, 010, 100} → Decide 0 was sent
- Receive {111, 110, 101, 011} → Decide 1 was sent

RBER is still 0.01... What is UBER?

UBER is 0.01

This is now 33 times better!...

Any downsides?
A Simple Channel/Storage Model

- Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example $P_e = 0.01$, $Pr(\text{success}) = 0.99$.
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, RBER = 0.01.

Suppose we now use repetition coding

$0 \rightarrow 000, 1 \rightarrow 111$

Decoding rule:
- Receive $\{000, 001, 010, 100\} \rightarrow$ Decide 0 was sent
- Receive $\{111, 110, 101, 011\} \rightarrow$ Decide 1 was sent

RBER is still 0.01... What is UBER?

UBER is 0.

This is now 33 times better!... Any downsides?
A Simple Channel/Storage Model

- Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example, \( P_e = 0.01 \), \( P\text{r}(\text{success}) = 0.99 \).
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, RBER = 0.01.

Suppose we now use repetition coding

- Suppose we now use repetition coding
  \[ 0 \rightarrow 000, 1 \rightarrow 111 \]

Decoding rule:
- Receive \{000, 001, 010, 100\} \(\rightarrow\) Decide 0 was sent
- Receive \{111, 110, 101, 011\} \(\rightarrow\) Decide 1 was sent
- RBER is still 0.01...What is UBER?
A Simple Channel/Storage Model

- Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example $P_e = 0.01$, $Pr(\text{success}) = 1 - P_e = 0.99$.
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, $RBER = 0.01$.

Suppose we now use repetition coding $0 \rightarrow 000, 1 \rightarrow 111$

Decoding rule:
- Receive $\{000, 001, 010, 100\} \rightarrow \text{Decide 0 was sent}$
- Receive $\{111, 110, 101, 011\} \rightarrow \text{Decide 1 was sent}$

RBER is still 0.01...What is UBER?

UBER is $0.01^3 + 3 \times 0.01^2 \times 0.99 = 0.000298$
A Simple Channel/Storage Model

- Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example, $P_{e} = 0.01$, $Pr(\text{success}) = 1 - P_{e} = 0.99$.
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, RBER = 0.01.

Suppose we now use repetition coding

$0 \rightarrow 000$, $1 \rightarrow 111$

- Decoding rule:
  - Receive $\{000, 001, 010, 100\} \rightarrow$ Decide 0 was sent
  - Receive $\{111, 110, 101, 011\} \rightarrow$ Decide 1 was sent

RBER is still 0.01...What is UBER?
UBER is $0.01^3 + 3 \times 0.01^2 \times 0.99 = 0.000298$
This is now 33 times better!...
A Simple Channel/Storage Model

- Example: binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1.
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example, \( P_e = 0.01 \), \( \text{Pr}(\text{success}) = 0.99 \).
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, \( \text{RBER} = 0.01 \).

Suppose we now use repetition coding

\[ 0 \rightarrow 000, \quad 1 \rightarrow 111 \]

Decoding rule:
- Receive \( \{000, 001, 010, 100\} \) → Decide 0 was sent
- Receive \( \{111, 110, 101, 011\} \) → Decide 1 was sent
- RBER is still 0.01...What is UBER?
- UBER is \( 0.01^3 + 3 \times 0.01^2 \times 0.99 = 0.000298 \)
- This is now 33 times better!...Any downsides?
A channel code $C$ maps a message $m$ of length $k$ into a codeword $c$ of length $n$, with $n > k$ (encoder).

- Total number of codewords: $2^k$.
- Code rate: $R = k/n$.
- Structure of $C$ is used to determine the stored message (decoder).
Repetition code example

<table>
<thead>
<tr>
<th>input message</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
</tr>
</tbody>
</table>

- Message length $k = 1$
- Total number of codewords $2^1 = 2$.
- Codeword length $n = 3$.
- Code rate $R = 1/3$. 
Linear block code $C$ of dimension $k$ and codeword length $n$ can be represented by

- a $k \times n$ generator matrix $G$
- a $(n - k) \times n$ parity check matrix $H$

$G$ specifies the range space of $C$ and $H$ specifies the null space of $C$.

The two representations are mathematically equivalent.
Linear block code $C$ of dimension $k$ and codeword length $n$ can be represented by

- a $k \times n$ generator matrix $G$
- a $(n - k) \times n$ parity check matrix $H$

$G$ specifies the range space of $C$ and $H$ specifies the null space of $C$.

The two representations are mathematically equivalent.
Linear block code $C$ of dimension $k$ and codeword length $n$ can be represented by:

- a $k \times n$ generator matrix $G$ \[ mG = c \]
- a $(n - k) \times n$ parity check matrix $H$ \[ cH^T = 0 \]

- $G$ specifies the range space of $C$ and $H$ specifies the null space of $C$.
- The two representations are mathematically equivalent.
## Repetition code example

<table>
<thead>
<tr>
<th>input message</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
</tr>
</tbody>
</table>

- **Generator matrix**
  
  $$G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

- **Parity check matrix**
  
  $$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$
How many errors can you correct?

- Our toy repetition code corrects 1 error.

In general, $k + d \leq n + 1$, where $k$ is the message length, $n$ is the codeword length, $d$ is the minimum separation between codewords a.k.a. minimum code distance. Code can correct $t = \left\lfloor \frac{(d - 1)}{2} \right\rfloor$ errors.
How many errors can you correct?

- Our toy repetition code corrects 1 error.

In general, $k + d \leq n + 1$, where
- $k$ is the message length, $n$ is the codeword length
- $d$ is the minimum separation between codewords a.k.a. minimum code distance
- Code can correct $t = \lfloor (d - 1)/2 \rfloor$ errors.
Computing UBER

For a code with message length $k$ and codeword length $n$.

Exact:

$$UBER = \frac{\sum_{j=t+1}^{n} \binom{n}{j} \times RBER^j \times (1 - RBER)^{n-j}}{k}$$

Good approximation for small error values:

$$UBER = \frac{\binom{n}{t+1} \times RBER^{t+1} \times (1 - RBER)^{n-t-1}}{k}$$

Here, $\binom{n}{j}$ is the binomial coefficient $\frac{n!}{j!(n-j)!}$.
Linear block codes can be divided in two categories:

- algebraic codes (BCH codes, Hamming codes, Reed-Solomon codes)
- graph-based codes (LDPC codes, Turbo codes)

A good practical channel code should

- be able to correct as many transmission errors as possible with the least overhead
- be equipped with a simple decoding algorithm
Algebraic Codes
Brief review of finite fields

Suppose \( q \) is prime.

- \( GF(q) \) can be viewed as the set \( \{0, 1, \ldots, q - 1\} \).
- Operations are performed modulo \( q \).

**Example:**

- \( GF(5) \) has elements \( \{0, 1, 2, 3, 4\} \) such that

<table>
<thead>
<tr>
<th>product</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sum</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
GF(q) can also be expressed as 
\{\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}, for suitably chosen \alpha.

Example:

- In \( GF(5) \): 0 \(\rightarrow\) \(\alpha^{-\infty}\), 1 \(\rightarrow\) \(\alpha^0\), 2 \(\rightarrow\) \(\alpha\), 3 \(\rightarrow\) \(\alpha^3\) and 4 \(\rightarrow\) \(\alpha^2\)

Consider an element \(\alpha\) of \( GF(q) \) such that \(\alpha \neq 0\) and \(\alpha \neq 1\).

- Let \(s\) be the smallest positive integer such that \(\alpha^s = 1\). Then, \(s\) is the order of \(\alpha\).
- If \(s = q - 1\), then \(\alpha\) is called a primitive element of \( GF(q) \).

\( GF(q) \) is thus generated by powers of a primitive element \(\alpha\).
We are often interested in the extension field $GF(q^m)$ of $GF(q)$, where $q$ is prime and $m$ is a positive integer.

$GF(q^m)$ is then $\{\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha, \alpha^2, \ldots, \alpha^{q^m-1}\}$, where $\alpha$ denotes a primitive element of $GF(q^m)$ and is a root of so-called primitive polynomial.

Example:

- $GF(8) = GF(2^3)$.
- Here, $\alpha$ is a root of the polynomial $x^3 + x + 1$.
- We then have

\[
\begin{align*}
\alpha^0 &= 1 \\
\alpha^1 &= \alpha \\
\alpha^2 &= \alpha^2 \\
\alpha^3 &= \alpha + 1 \\
\alpha^4 &= \alpha^2 + \alpha \\
\alpha^5 &= \alpha^2 + \alpha + 1 \\
\alpha^6 &= \alpha^2 + 1 \\
\alpha^{-\infty} &= 0
\end{align*}
\]
BCH code construction

BCH code $C$ is a linear, cyclic code described by a $(d - 1) \times n$ parity check matrix $H$ with elements from $GF(q^m)$ with $\alpha$ having order $n$:

$$H = \begin{bmatrix}
1 & \alpha^b & \alpha^{2b} & \ldots & \alpha^{(n-1)b} \\
1 & \alpha^{b+1} & \alpha^{2(b+1)} & \ldots & \alpha^{(n-1)(b+1)} \\
\vdots & \vdots & \vdots & \ldots & \vdots \\
1 & \alpha^{b+d-2} & \alpha^{2(b+d-2)} & \ldots & \alpha^{(n-1)(b+d-2)}
\end{bmatrix}$$

- $b$ is any (positive) integer and $d$ is integer $2 \leq d \leq n$.
- Minimum distance of $C$ is at least $d$. The code corrects at least $t = \lfloor \frac{d-1}{2} \rfloor$ errors.
BCH code construction

- If $\alpha$ is a primitive element, then the blocklength is $n = q^m - 1$ (largest possible).
- If $b = 1$, BCH code is called narrow-sense (simplifies some encoding and decoding operations).
- For $m = 1$, BCH codes are also known as Reed-Solomon codes.
BCH code properties

- A code $C$ is called a **cyclic code** if all cyclic shifts of a codeword in $C$ are also codewords.

Example:

- Suppose $(0,1,0,1,1) \leftrightarrow x^3 + x + 1$ is a codeword in $C$. Then so are $(1,0,1,1,0)$, $(0,1,1,0,1)$, $(1,1,0,1,0)$ and $(1,0,1,0,1)$. 
A code $C$ is called a **cyclic code** if all cyclic shifts of a codeword in $C$ are also codewords.

Example:

Suppose $(0, 1, 0, 1, 1) \leftrightarrow x^3 + x + 1$ is a codeword in $C$. Then so are $(1, 0, 1, 1, 0), (0, 1, 1, 0, 1), (1, 1, 0, 1, 0)$ and $(1, 0, 1, 0, 1)$.

Cyclic code is generated by a generator polynomial $g(x)$, such that each codeword $c$ corresponds to a polynomial $p_c(x) = m(x)g(x)$. All rows of the generator matrix $G$ are cyclic shifts of $g(x)$. 
BCH code properties

- A code $C$ is called a cyclic code if all cyclic shifts of a codeword in $C$ are also codewords.

Example:
- Suppose $(0,1,0,1,1) \leftrightarrow x^3 + x + 1$ is a codeword in $C$. Then so are $(1,0,1,1,0)$, $(0,1,1,0,1)$, $(1,1,0,1,0)$ and $(1,0,1,0,1)$.

- Cyclic code is generated by a generator polynomial $g(x)$, such that each codeword $c$ corresponds to a polynomial $p_c(x) = m(x)g(x)$. All rows of the generator matrix $G$ are cyclic shifts of $g(x)$.

- BCH code: Each codeword $c$ corresponds to a polynomial $p_c(x) = m(x)g(x)$ where $g(x)$ is LCM of $(x - \alpha^b)(x - \alpha^{b+1}) \cdots (x - \alpha^{b+d-2})$. 
Let’s construct a narrow-sense BCH code over $GF(8)$ correcting $t = 1$ error and of length $n = 7$.

We consider a primitive element $\alpha$ that satisfies $\alpha^3 + \alpha + 1 = 0$. Notice that $\alpha^7 = 1$.

Then,

$$H = \begin{bmatrix}
1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\
1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \alpha^{10} & \alpha^{12}
\end{bmatrix}$$
Let’s construct a narrow-sense BCH code over $GF(8)$ correcting $t = 1$ error and of length $n = 7$.

We consider a primitive element $\alpha$ that satisfies $\alpha^3 + \alpha + 1 = 0$. Notice that $\alpha^7 = 1$.

Then,

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \end{bmatrix}$$
BCH code example

We can interpret this code in the binary domain by substituting

\[
\begin{align*}
1 & \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \alpha & \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} & \alpha^2 & \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} & \alpha^3 & \rightarrow \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\
\alpha^4 & \rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} & \alpha^5 & \rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} & \alpha^6 & \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} & 0 & \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\end{align*}
\]
We can then interpret this parity check matrix in the binary domain as

\[
H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1
\end{bmatrix}
\]

Here \( H \) is \( 6 \times 7 \) and has rank 3. This code can correct 1 error.
Decoding BCH codes

Decoding algorithm heavily relies on the algebraic structure of the code: recall that each codeword polynomial $c(x)$ must have as roots $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+d-2}$.

1. Compute the syndromes of the received polynomial $r(x)$—tells us which of $\alpha$’s are not the roots.
2. Based on the syndromes, compute the locations of the errors (system of linear equations).
3. Compute the error values at these location (system of non-linear equations that are in the Vandermode form).
4. Based on steps 2 and 3, build error polynomial $e(x)$.
5. Add $e(x)$ to $r(x)$ to produce the estimate of $c(x)$. 

Flash Memory Summit 2014, Santa Clara, CA
Decoding BCH codes

- If the system of equations cannot be solved, declare a decoding failure. This is a hard limit on the number of correctable errors.
- Implementation can be greatly reduced using the shift-registers viewpoint in the Berlekamp-Massey algorithm.
### BCH code parameter tradeoffs

For fixed code length and RBER, how does UBER depend on $t$?

<table>
<thead>
<tr>
<th>Code length</th>
<th>RBER</th>
<th>Strength ($t$)</th>
<th>Code Rate</th>
<th>UBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023</td>
<td>0.002</td>
<td>12</td>
<td>0.8827</td>
<td>2.8017 x 10^-10</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>13</td>
<td>0.8729</td>
<td>4.0511 x 10^-11</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>14</td>
<td>0.8631</td>
<td>5.4703 x 10^-12</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>15</td>
<td>0.8534</td>
<td>6.9272 x 10^-13</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>16</td>
<td>0.8436</td>
<td>8.2572 x 10^-14</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>17</td>
<td>0.8387</td>
<td>9.2968 x 10^-15</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>18</td>
<td>0.8289</td>
<td>9.9314 x 10^-16</td>
</tr>
</tbody>
</table>
**BCH code parameter tradeoffs**

For **fixed** code length and RBER, how does UBER depend on $t$?

<table>
<thead>
<tr>
<th>Code length</th>
<th>RBER</th>
<th>Strength (t)</th>
<th>Code Rate</th>
<th>UBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023</td>
<td>0.002</td>
<td>12</td>
<td>0.8827</td>
<td>2.8017 x 10^{-10}</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>13</td>
<td>0.8729</td>
<td>4.0511 x 10^{-11}</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>14</td>
<td>0.8631</td>
<td>5.4703 x 10^{-12}</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>15</td>
<td>0.8534</td>
<td>6.9272 x 10^{-13}</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>16</td>
<td>0.8436</td>
<td>8.2572 x 10^{-14}</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>17</td>
<td>0.8387</td>
<td>9.2968 x 10^{-15}</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>18</td>
<td>0.8289</td>
<td>9.9314 x 10^{-16}</td>
</tr>
</tbody>
</table>

**Improve by 6**

**Improve by > 200,000 times**
BCH code parameter tradeoffs

For **fixed** RBER and $t$, how does UBER depend on codelength?

<table>
<thead>
<tr>
<th>Code length</th>
<th>RBER</th>
<th>Strength (t)</th>
<th>Code Rate</th>
<th>UBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>0.002</td>
<td>10</td>
<td>0.2857</td>
<td>$3.7007 \times 10^{-17}$</td>
</tr>
<tr>
<td>127</td>
<td>0.002</td>
<td>10</td>
<td>0.5039</td>
<td>$7.0119 \times 10^{-17}$</td>
</tr>
<tr>
<td>255</td>
<td>0.002</td>
<td>10</td>
<td>0.7020</td>
<td>$4.4666 \times 10^{-14}$</td>
</tr>
<tr>
<td>511</td>
<td>0.002</td>
<td>10</td>
<td>0.8239</td>
<td>$2.7184 \times 10^{-11}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>10</td>
<td>0.9022</td>
<td>$1.0700 \times 10^{-8}$</td>
</tr>
<tr>
<td>2047</td>
<td>0.002</td>
<td>10</td>
<td>0.9463</td>
<td>$1.7231 \times 10^{-6}$</td>
</tr>
<tr>
<td>4096</td>
<td>0.002</td>
<td>10</td>
<td>0.9707</td>
<td>$5.1165 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
**BCH code parameter tradeoffs**

For fixed RBER and $t$, how does UBER depend on codementh? 

<table>
<thead>
<tr>
<th>Code length</th>
<th>RBER</th>
<th>Strength (t)</th>
<th>Code Rate</th>
<th>UBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>0.002</td>
<td>10</td>
<td>0.2857</td>
<td>$3.7007 \times 10^{-17}$</td>
</tr>
<tr>
<td>127</td>
<td>0.002</td>
<td>10</td>
<td>0.5039</td>
<td>$7.0119 \times 10^{-17}$</td>
</tr>
<tr>
<td>255</td>
<td>0.002</td>
<td>10</td>
<td>0.7020</td>
<td>$4.4666 \times 10^{-14}$</td>
</tr>
<tr>
<td>511</td>
<td>0.002</td>
<td>10</td>
<td>0.8239</td>
<td>$2.7184 \times 10^{-11}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.002</td>
<td>10</td>
<td>0.9022</td>
<td>$1.0700 \times 10^{-8}$</td>
</tr>
<tr>
<td>2047</td>
<td>0.002</td>
<td>10</td>
<td>0.9463</td>
<td>$1.7231 \times 10^{-6}$</td>
</tr>
<tr>
<td>4095</td>
<td>0.002</td>
<td>10</td>
<td>0.9707</td>
<td>$5.1165 \times 10^{-5}$</td>
</tr>
</tbody>
</table>

Increase by 64 times  
Decrease by $\sim 1 \ 000 \ 000 \ 000 \ 000 \ 000$ times
For fixed codelength and $t$, how does UBER depend on RBER?

<table>
<thead>
<tr>
<th>Code length</th>
<th>RBER</th>
<th>Strength (t)</th>
<th>Code Rate</th>
<th>UBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023</td>
<td>0.002</td>
<td>15</td>
<td>0.8534</td>
<td>6.9272 x 10^{-13}</td>
</tr>
<tr>
<td>1023</td>
<td>0.004</td>
<td>15</td>
<td>0.8534</td>
<td>6.9350 x 10^{-9}</td>
</tr>
<tr>
<td>1023</td>
<td>0.006</td>
<td>15</td>
<td>0.8534</td>
<td>7.0667 x 10^{-7}</td>
</tr>
<tr>
<td>1023</td>
<td>0.008</td>
<td>15</td>
<td>0.8534</td>
<td>1.1161 x 10^{-5}</td>
</tr>
<tr>
<td>1023</td>
<td>0.010</td>
<td>15</td>
<td>0.8534</td>
<td>6.4448 x 10^{-5}</td>
</tr>
<tr>
<td>1023</td>
<td>0.012</td>
<td>15</td>
<td>0.8534</td>
<td>2.0042 x 10^{-4}</td>
</tr>
<tr>
<td>1023</td>
<td>0.014</td>
<td>15</td>
<td>0.8534</td>
<td>4.1505 x 10^{-4}</td>
</tr>
</tbody>
</table>
BCH code parameter tradeoffs

For **fixed** codelength and $t$, how does UBER depend on RBER?

<table>
<thead>
<tr>
<th>Code length</th>
<th>RBER</th>
<th>Strength (t)</th>
<th>Code Rate</th>
<th>UBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023</td>
<td>0.002</td>
<td>15</td>
<td>0.8534</td>
<td>$6.9272 \times 10^{-13}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.004</td>
<td>15</td>
<td>0.8534</td>
<td>$6.9350 \times 10^{-9}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.006</td>
<td>15</td>
<td>0.8534</td>
<td>$7.0667 \times 10^{-7}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.008</td>
<td>15</td>
<td>0.8534</td>
<td>$1.1161 \times 10^{-5}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.010</td>
<td>15</td>
<td>0.8534</td>
<td>$6.4448 \times 10^{-5}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.012</td>
<td>15</td>
<td>0.8534</td>
<td>$2.0042 \times 10^{-4}$</td>
</tr>
<tr>
<td>1023</td>
<td>0.014</td>
<td>15</td>
<td>0.8534</td>
<td>$4.1505 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

*Increase by 7 times*

* Decrease by $\sim 1000000000$ times
Graph-Based Codes
Low Density Parity Check (LDPC) Codes

Definition 1: LDPC code

An LDPC block code $C$ is a linear block code whose parity-check matrix $H$ has a small number of ones in each row and column.

- Invented by Gallager in 1963 but were all but forgotten until late 1990’s.
- In the limit of very large block-lengths LDPC codes are known to approach the Shannon limit (i.e., the highest rate at which the code can be designed that guarantees reliable communication)
- LDPC codes are amenable to low-complexity iterative decoding.
An Example

LDPC code described by the sparse parity check matrix $H$:

$$H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{bmatrix}$$

Matrix $H$ has 9 columns and 6 rows.
An Example

LDPC code described by the sparse parity check matrix $H$:

$$H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 
\end{bmatrix}$$

Matrix $H$ has 9 columns and 6 rows. There are 9 coded bits and 6 parity-check equations. Each coded bit participates $\ell = 2$ parity-check equations and each parity-check equation contains $r = 3$ coded bits.
Definition 3: Tanner graph

A Tanner graph of a code $C$ with a parity check matrix $H$ is the bipartite graph such that:

- each coded symbol $i$ is represented by a variable node $v_i$,
- each parity-check equation $j$ is represented by a check node $c_j$,
- there exists an edge between a variable node and a check node if and only if $H(j, i) = 1$. 
An Example

LDPC code: parity check matrix $H$ and its Tanner graph

$$H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{bmatrix}$$

Parity check $c_3$: $v_3 + v_6 + v_9 \equiv 0$ over $GF(2)$. 
An Example

LDPC code: parity check matrix $H$ and its Tanner graph

$$H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{bmatrix}$$

Parity check $c_3$: $v_3 + v_6 + v_9 \equiv 0$ over $GF(2)$. 
An Example

LDPC code: parity check matrix $H$ and its Tanner graph

$$H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{bmatrix}$$

Parity check $c_3$: $v_3 + v_6 + v_9 \equiv 0$ over $GF(2)$. 
Message-passing (belief propagation) is an iterative decoding algorithm that operates on the Tanner graph of the code. In each iteration of the algorithm:
Message-passing (belief propagation) is an iterative decoding algorithm that operates on the Tanner graph of the code. In each iteration of the algorithm:

1. (bit-to-check) Each variable node sends a message to each check node it is connected to,
Message-passing (belief propagation) is an iterative decoding algorithm that operates on the Tanner graph of the code. In each iteration of the algorithm:

1. (bit-to-check) Each variable node sends a message to each check node it is connected to,

2. (check processing) Each check node then computes the consistency of incoming messages,
Message Passing Decoding

Message-passing (belief propagation) is an iterative decoding algorithm that operates on the Tanner graph of the code. In each iteration of the algorithm:

1. (bit-to-check) Each variable node sends a message to each check node it is connected to,

2. (check processing) Each check node then computes the consistency of incoming messages,

3. (check-to-bit) Each check node then sends a message to each variable node it is connected to,
Message-passing (belief propagation) is an iterative decoding algorithm that operates on the Tanner graph of the code. In each iteration of the algorithm:

1. (bit-to-check) Each variable node sends a message to each check node it is connected to,
2. (check processing) Each check node then computes the consistency of incoming messages,
3. (check-to-bit) Each check node then sends a message to each variable node it is connected to,
4. (bit processing) Each variable node (coded symbol) updates its value.
Passed messages can be either

- Hard decisions: 0 or 1
- Soft decisions/likelihoods: real numbers
An Example

<table>
<thead>
<tr>
<th>Message $m$</th>
<th>Codeword $y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>$y_1y_2y_3y_4$</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 1</td>
</tr>
</tbody>
</table>

Input message $m$, encoded to codeword $y$. After passing through a noisy channel, the retrieved word is $y' = 1001$. The decoder is used to decode the message, resulting in $m_1$.

Graph representation of the LDPC code:

- $y_1 + y_2 + y_3 = 0$
- $y_1 + y_3 + y_4 = 0$
- $y_2 + y_3 + y_4 = 0$
Message Passing Decoding

Bit-flipping algorithm
Received Codeword

\[ y_1 + y_2 + y_3 \quad y_1 + y_3 + y_4 \quad y_2 + y_3 + y_4 \]
Bit-to-Check Messages

1 + y_2 + y_3

1 + y_3 + y_4

y_2 + y_3 + y_4
Bit-to-Check Messages

1 + 0 + y_3  
1 + y_3 + y_4  
0 + y_3 + y_4
Check Processing

1 + 0 + 0 = 1
1 + 0 + y_4
0 + 0 + y_4
Check Processing

\[
\begin{align*}
1 + 0 + 0 &= 1 \quad \text{??} \\
1 + 0 + 1 &= 0 \quad \sqrt{} \\
0 + 0 + 1 &= 1 \quad \text{??}
\end{align*}
\]
Check-to-Bit Messages

1 + 0 + 0 = 1 ??
1 + 0 + 1 = 0 \sqrt{}
0 + 0 + 1 = 1 ??

Flip, Stay
Flip, Stay
Flip, Stay,
Stay,
Check-to-Bit Messages

1 + 0 + 0 = 1 ??

1 + 0 + 1 = 0 √

0 + 0 + 1 = 1 ??

Flip, Stay
Flip, Flip
Flip, Stay, Flip
Stay, Flip
Bit Processing

```
1 0 0 1
1+0+0=1 ?? 1+0+1= 0 √ 0+0+1= 1 ?? 
Flip, Stay Flip, Flip Flip, Stay, Flip Stay, Flip
```
Bit Processing

\[ y_1 + y_2 + y_3 \quad y_1 + y_3 + y_4 \quad y_2 + y_3 + y_4 \]
Bit-to-Check Messages

\[
\begin{align*}
1 + y_2 + y_3 & \\
1 + y_3 + y_4 & \\
y_2 + y_3 + y_4 & 
\end{align*}
\]
Bit-to-Check Messages
Bit-to-Check Messages

\[ 1 + 1 + 0 = 0 \quad \sqrt{1 + 0 + y_4} \quad 1 + 0 + y_4 \]
Check Processing

\[1 + 1 + 0 = 0 \sqrt{}\]
\[1 + 0 + 1 = 0 \sqrt{}\]
\[1 + 0 + 1 = 0 \sqrt{}\]
Check-to-Bit Messages

\[
\begin{align*}
1 + 1 + 0 &= 0 \\
1 + 0 + 1 &= 0 \\
1 + 0 + 1 &= 0
\end{align*}
\]
Check-to-Bit Messages
Check-to-Bit Messages

Stay, Stay, Stay

1 + 1 + 0 = 0

Stay, Stay

1 + 0 + 1 = 0

Stay, Stay, Stay

1 + 0 + 1 = 0
Bit Processing

Decoded Codeword

1 + 1 + 0 = 0  √  
1 + 0 + 1 = 0  √  
1 + 0 + 1 = 0  √  

1 1 0 1

Improved variants of message passing algorithm use soft information as messages, i.e., log-likelihood ratio $L = \log \frac{P(x_i=0|y_i)}{P(x_i=1|y_i)}$.

**Sum-product algorithm (SPA)** [1,2]

**Min-sum algorithm (MSA)** [3]

Soft Iterative Decoding

Improved variants of message passing algorithm use soft information as messages, i.e., log-likelihood ratio \( L = \log \frac{P(x_i=0|y_i)}{P(x_i=1|y_i)} \).

**Sum-product algorithm (SPA)** [1,2]
- bit-to-check \( L(v_i \rightarrow c_j) = \sum_{j' \in N(i) \setminus j} L(c'_j \rightarrow v_i) + L^{\text{int}}(v_i) \)
- check-to-bit \( L(c_j \rightarrow v_i) = \Phi^{-1} \left( \sum_{i' \in N(j) \setminus i} \Phi(|L(v'_i \rightarrow c_j)|) \sum_{i'' \in N(j) \setminus i} \text{sgn}(L(v''_i \rightarrow c_j)) \right) \)
  where \( \Phi(x) = -\log(\tanh(x/2)) \)

**Min-sum algorithm (MSA)** [3]
- check-to-bit \( L(c_j \rightarrow v_i) = \min_{i' \in N(j) \setminus i} |L(v'_i \rightarrow c_j)| \prod_{i'' \in N(j) \setminus i} \text{sgn}(L(v''_i \rightarrow c_j)) \)

Soft Decoding

Bit values    1   1   0   1

[Diagram showing a graph with nodes connected by edges, not transcribed here]
### Soft Decoding

<table>
<thead>
<tr>
<th>Bit values</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values using BPSK</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
</tbody>
</table>

![Graph](image-url)
Soft Decoding

<table>
<thead>
<tr>
<th>Bit values</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values using BPSK</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>Values from channel</td>
<td>-1.1</td>
<td>0.1</td>
<td>1.2</td>
<td>-0.9</td>
</tr>
</tbody>
</table>
**Soft Decoding**

Bit values: 1 1 0 1

Values using BPSK: -1 -1 +1 -1

Values from channel: -1.1 0.1 1.2 -0.9

Beliefs ($L_{vi}^{(int)}$): -2.2 0.2 2.4 -1.8

$L_{vi}^{(int)} = \log \left( \frac{e^{-(y_i-1)^2/2\sigma_n^2}}{e^{-(y_i+1)^2/2\sigma_n^2}} \right) = \frac{2}{\sigma_n^2} y_i$

We assume $\sigma_n = 1$. 

\[ L_{vi}^{(int)} = \log \left( \frac{e^{-(y_i-1)^2/2\sigma_n^2}}{e^{-(y_i+1)^2/2\sigma_n^2}} \right) = \frac{2}{\sigma_n^2} y_i \]

We assume $\sigma_n = 1$. 

\[ L_{vi}^{(int)} = \log \left( \frac{e^{-(y_i-1)^2/2\sigma_n^2}}{e^{-(y_i+1)^2/2\sigma_n^2}} \right) = \frac{2}{\sigma_n^2} y_i \]
Soft Decoding

Bit values

1 1 0 1

Values using BPSK

-1 -1 +1 -1

Values from channel

-1.1 0.1 1.2 -0.9

Beliefs

-2.2 0.2 2.4 -1.8

\[ L_{c_j \rightarrow v_j} = 2 \tanh^{-1} \left( \prod_{l \neq i} \tanh \frac{1}{2} L_{v_l \rightarrow c_j} \right) \]
Soft Decoding

<table>
<thead>
<tr>
<th>Bit values</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values using BPSK</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>Values from channel</td>
<td>-1.1</td>
<td>0.1</td>
<td>1.2</td>
<td>-0.9</td>
</tr>
<tr>
<td>Beliefs</td>
<td>-2.2</td>
<td>0.2</td>
<td>2.4</td>
<td>-1.8</td>
</tr>
</tbody>
</table>

\[
L_{c_j \rightarrow v_j} = 2 \tanh^{-1} \left( \prod_{i \neq j} \tanh \frac{1}{2} L_{v_i \rightarrow c_j} \right)
\]
Soft Decoding

\[ L_{v_i} = L_{v_i}^{(\text{int})} + \sum_{c_j \rightarrow v_i} L_{c_j \rightarrow v_j} \]
Soft Decoding

Bit values

Values using BPSK

Values from channel

Beliefs

All variable nodes are decoded to correct bit value.
Figure: Rate 0.9 LDPC and BCH codes of length \( n = 9100 \).
Performance with multi read

**Figure:** Rate 0.9 LDPC and BCH codes of length $n = 9100$.

Frame Error Rate vs. Raw Bit Error Rate (MLC Gaussian Model)

Caution:
- Optimal code design in the error floor region depends on the chosen quantization.
- AWGN-optimized LDPC codes may not be the best for the quantized (and asymmetric) Flash channel!
Non-binary LDPC codes

Entries in the parity check matrix $H$ are taken from $GF(q)$. Example: $GF(8) = 0, 1, 2, ..., 7$. (with $\alpha^k \rightarrow k + 1$ for $0 \leq k \leq 6$)

$$H = \begin{bmatrix}
1 & 0 & 0 & 3 & 0 & 0 & 5 & 0 & 0 \\
0 & 2 & 0 & 0 & 6 & 0 & 0 & 2 & 0 \\
0 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 5 & 0 & 7 & 0 \\
0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 6 & 0 & 7 & 0 & 1 & 0 & 0 \\
\end{bmatrix}$$

Parity check $c_3$: $3v_3 + v_6 + v_9 \equiv 0$ over $GF(8)$. 
Non-binary LDPC codes

Entries in the parity check matrix $H$ are taken from $GF(q)$.
Example: $GF(8) = 0, 1, 2, ..., 7$. (with $\alpha^k \rightarrow k + 1$ for $0 \leq k \leq 6$)

$$H = \begin{bmatrix}
1 & 0 & 0 & 3 & 0 & 0 & 5 & 0 & 0 \\
0 & 2 & 0 & 0 & 6 & 0 & 0 & 2 & 0 \\
0 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 5 & 0 & 7 & 0 \\
0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 6 & 0 & 7 & 0 & 1 & 0 & 0 \\
\end{bmatrix}$$

Parity check $c_3$: $3v_3 + v_6 + v_9 \equiv 0$ over $GF(8)$.

See talk on Thursday: Flash Controller Design (8:30 – 10:50)
Figure: Non-binary LDPC codes vs. BCH codes performance comparison for AWGN channel. Code rate is 0.9, block length is 1000 bits. BCH code corrects 13 errors.
Non-binary LDPC decoding

- Decoding is more complex than in the binary case. Keep track of $q - 1$ likelihoods on each edge.
- Popular approaches:
  - Direct implementation has complexity on the order of $O(q^2)$
  - FFT-based SPA has complexity on the order of $O(q \log q)$
  - Min-sum and its variants can further reduce the complexity
Algebraic codes (BCH)
- Performance is acceptable
+ Guaranteed error correction capability
+ Structure allows for efficient decoder implementation
- Not amenable for soft decoding

Graph-based codes (LDPC)
+ Performance is excellent
- No guaranteed error correction capability (but we have ideas)
- Decoder complexity is acceptable but now low
+ Amenable for soft decoding

With the move to MLC/TLC technologies, advanced coding schemes will need to be considered!
Further information, papers, references etc. available at http://loris.ee.ucla.edu

Selected list:


We would like to invite you to explore CoDESS:

http://www.uclacodess.org

For more information, please contact

Prof. Lara Dolecek
dolecek@ee.ucla.edu