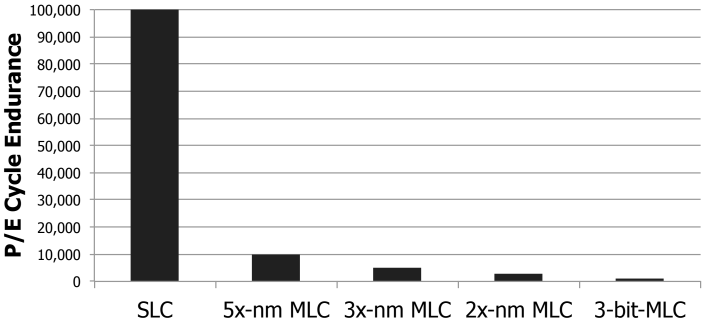
Making Error Correcting Codes Work for Flash Memory

Part I: Primer on ECC, basics of BCH and LDPC codes

Lara Dolecek

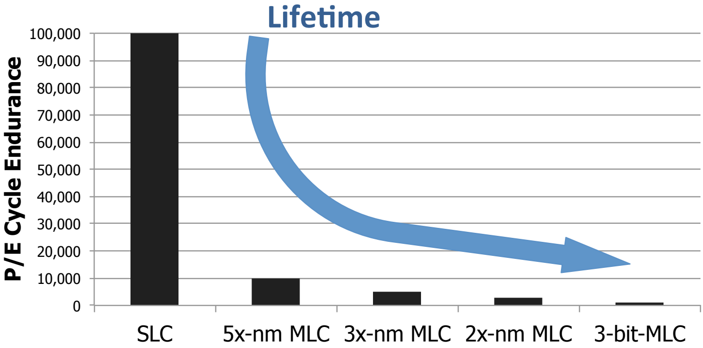
Laboratory for Robust Information Systems (LORIS)
Center on Development of Emerging Storage Systems (CoDESS)
Department of Electrical Engineering, UCLA

ECC is a must for Flash!



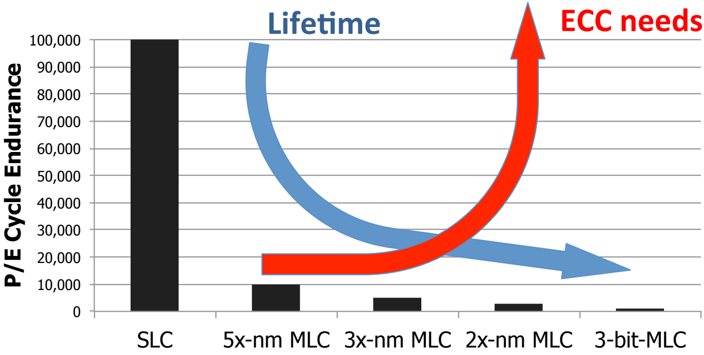
Ariel Maislos, "A New Era in Embedded Flash Memory", Flash Summit 2011 (Anobit)

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!



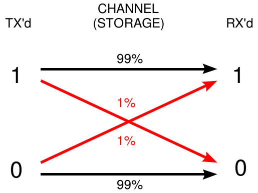
Ariel Maislos, "A New Era in Embedded Flash Memory", Flash Summit 2011 (Anobit)

What is this tutorial about

Today we will

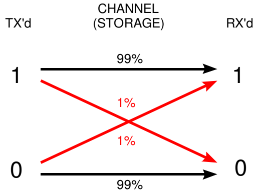
- Learn the basics of ECC operations
- Learn about fundamental coding approaches (BCH, LDPC)
- Learn about the system-level perspective on ECC
- Learn about recent advanced coding-oriented approaches to Flash

A simple example

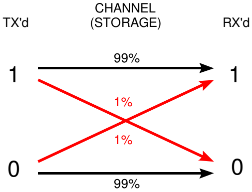


- Errors in Flash are modeled as transmission a noisy communication channel
- The simplest example is binary symmetric channel (BSC).

A simple example

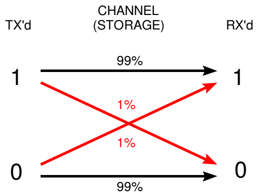


- Errors in Flash are modeled as transmission a noisy communication channel
- The simplest example is binary symmetric channel (BSC).
- This example:
  1. Raw bit error rate (RBER) is 0.01.
  2. Undetected bit error rate (UBER) is 0.01



- Suppose we now use repetition coding

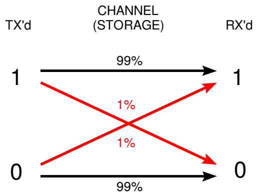
$$0 \rightarrow 000, 1 \rightarrow 111$$



- Suppose we now use repetition coding

$$0 \to 000, 1 \to 111$$

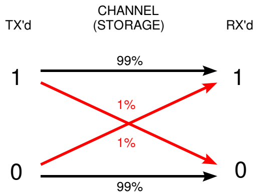
- Decoding rule:
  - Receive $\{000, 001, 010, 100\}$ $\to$ Decide 0 was sent
  - Receive $\{111, 110, 101, 011\}$ $\to$ Decide 1 was sent



- Suppose we now use repetition coding

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

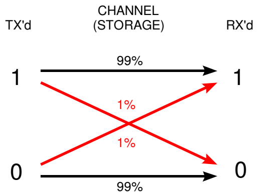
- Decoding rule:
  - Receive $\{000, 001, 010, 100\} \rightarrow$ Decide 0 was sent
  - Receive $\{111, 110, 101, 011\} \rightarrow$ Decide 1 was sent
- RBER is still 0.01...What is UBER?



- Suppose we now use repetition coding

$$0 \to 000, 1 \to 111$$

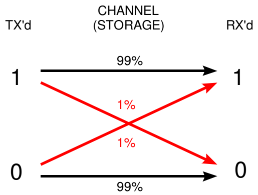
- Decoding rule:
  - Receive $\{000, 001, 010, 100\}$ $\to$ Decide 0 was sent
  - Receive $\{111, 110, 101, 011\}$ $\to$ Decide 1 was sent
- RBER is still 0.01...What is UBER?
- UBER is $0.01^3 + 3 \times 0.01^2 \times 0.99 = 0.000298$



- 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!...

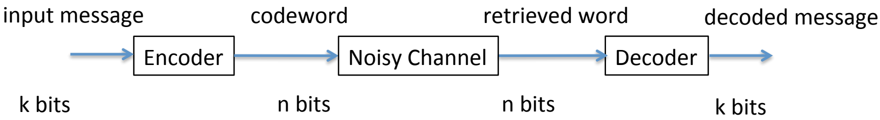


- Suppose we now use repetition coding

$$0 \to 000, 1 \to 111$$

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

Concepts of interest



- 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

input message	codeword
0	000
1	111

- Message length $k = 1$
- Total number of codewords $2^1 = 2$.
- Codeword length $n = 3$.
- Code rate $R = 1/3$.

Concepts of interest -ctd.

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.

Concepts of interest -ctd.

Linear block code C of dimension k and codeword length n can be represented by

- a $k \times n$ generator matrix G $\boxed{mG = c}$
- 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.

Concepts of interest -ctd.

Linear block code C of dimension k and codeword length n can be represented by

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

- G specifies the range space of C and H specifies the null space of C.
- The two representations are mathematically equivalent.

Repetition code example

input message	codeword
0	000
1	111

- 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.

How many errors can you correct ?

- Our toy repetition code corrects 1 error.

In general, $\boxed{k + d \leq n + 1}$, 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 $\boxed{t = \lfloor (d-1)/2 \rfloor}$ errors.

Computing UBER

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

Exact:

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

Good approximation for small error values:

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

- Here, $\binom{n}{j}$ is the binomial coefficient $\frac{n!}{j!(n-j)!}$

Concepts of interest

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

product	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

sum	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Brief review of finite fields

- $GF(q)$ can also be expressed as $\{\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}$, for suitably chosen α.

Example:

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

- Consider an element α 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 α.
- If $s = q - 1$, then α is called a primitive element of $GF(q)$.

$\boxed{GF(q) \text{ is thus generated by powers of a primitive element } \alpha.}$

Brief review of finite fields

- 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 α denotes a primitive element of $GF(q^m)$ and is a root of so-called primitive polynomial.

Example:

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

$$\begin{aligned} \alpha^0 &= 1 \\ \alpha^1 &= \alpha \\ \alpha^2 &= \alpha^2 \\ \alpha^3 &= \alpha + 1 \end{aligned}$$

$$\begin{aligned} \alpha^4 &= \alpha^2 + \alpha \\ \alpha^5 &= \alpha^2 + \alpha + 1 \\ \alpha^6 &= \alpha^2 + 1 \\ \alpha^{-\infty} &= 0 \end{aligned}$$

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 α having order n:

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

- 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 α 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 $(1,0,1,0,1)$.

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 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 $(x-\alpha^b)(x-\alpha^{b+1})\cdots(x-\alpha^{b+d-2})$.

BCH code example

- 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 α that satisfies $\alpha^3 + \alpha + 1 = 0$. Notice that $\alpha^7 = 1$.
- Then,

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \alpha^{10} & \alpha^{12} \end{bmatrix}$$

BCH code example

- 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 α 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

$$1 \to \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \alpha \to \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \alpha^2 \to \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \alpha^3 \to \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$\alpha^4 \to \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad \alpha^5 \to \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \alpha^6 \to \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad 0 \to \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

BCH code example

We can then interpret this parity check matrix in the binary domain as

$$H = \left[\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

Here H is 6×7 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 α^b, α^{b+1},..,α^{b+d-2}.

1. Compute the syndromes of the received polynomial $r(x)$– tells us which of α'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 $c(x)$.

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 ?

Code length	RBER	Strength (t)	Code Rate	UBER
1023	0.002	12	0.8827	2.8017×10^{-10}
1023	0.002	13	0.8729	4.0511×10^{-11}
1023	0.002	14	0.8631	5.4703×10^{-12}
1023	0.002	15	0.8534	6.9272×10^{-13}
1023	0.002	16	0.8436	8.2572×10^{-14}
1023	0.002	17	0.8387	9.2968×10^{-15}
1023	0.002	18	0.8289	9.9314×10^{-16}

BCH code parameter tradeoffs

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

Code length	RBER	Strength (t)	Code Rate	UBER
1023	0.002	12	0.8827	2.8017×10^{-10}
1023	0.002	13	0.8729	4.0511×10^{-11}
1023	0.002	14	0.8631	5.4703×10^{-12}
1023	0.002	15	0.8534	6.9272×10^{-13}
1023	0.002	16	0.8436	8.2572×10^{-14}
1023	0.002	17	0.8387	9.2968×10^{-15}
1023	0.002	18	0.8289	9.9314×10^{-16}

Improve by 6

Improve by > 200,000 times

BCH code parameter tradeoffs

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

Code length	RBER	Strength (t)	Code Rate	UBER
63	0.002	10	0.2857	3.7007×10^{-17}
127	0.002	10	0.5039	7.0119×10^{-17}
255	0.002	10	0.7020	4.4666×10^{-14}
511	0.002	10	0.8239	2.7184×10^{-11}
1023	0.002	10	0.9022	1.0700×10^{-8}
2047	0.002	10	0.9463	1.7231×10^{-6}
4096	0.002	10	0.9707	5.1165×10^{-5}

BCH code parameter tradeoffs

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

Code length	RBER	Strength (t)	Code Rate	UBER
63	0.002	10	0.2857	3.7007×10^{-17}
127	0.002	10	0.5039	7.0119×10^{-17}
255	0.002	10	0.7020	4.4666×10^{-14}
511	0.002	10	0.8239	2.7184×10^{-11}
1023	0.002	10	0.9022	1.0700×10^{-8}
2047	0.002	10	0.9463	1.7231×10^{-6}
4095	0.002	10	0.9707	5.1165×10^{-5}

Increase by 64 times

Decrease by ~
1 000 000 000 000 times

BCH code parameter tradeoffs

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

Code length	RBER	Strength (t)	Code Rate	UBER
1023	0.002	15	0.8534	6.9272×10^{-13}
1023	0.004	15	0.8534	6.9350×10^{-9}
1023	0.006	15	0.8534	7.0667×10^{-7}
1023	0.008	15	0.8534	1.1161×10^{-5}
1023	0.010	15	0.8534	6.4448×10^{-5}
1023	0.012	15	0.8534	2.0042×10^{-4}
1023	0.014	15	0.8534	4.1505×10^{-4}

BCH code parameter tradeoffs

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

Code length	RBER	Strength (t)	Code Rate	UBER
1023	0.002	15	0.8534	6.9272×10^{-13}
1023	0.004	15	0.8534	6.9350×10^{-9}
1023	0.006	15	0.8534	7.0667×10^{-7}
1023	0.008	15	0.8534	1.1161×10^{-5}
1023	0.010	15	0.8534	6.4448×10^{-5}
1023	0.012	15	0.8534	2.0042×10^{-4}
1023	0.014	15	0.8534	4.1505×10^{-4}

Increase by
7 times

Decrease by ~
1 000 000 000 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 ℓ=2 parity-check equations and each parity-check equation contains $r = 3$ coded bits.

LDPC Preliminaries

Definition 3: Tanner graph

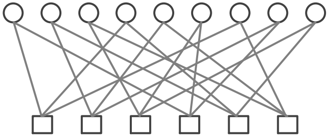
A Tanner graph of a code C with a parity check matrix H is the bipartite graph such that:

- 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

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

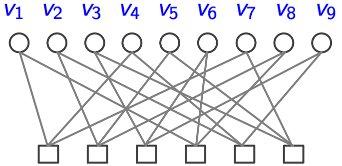


Parity check c_3: $v_3 + v_6 + v_9 \equiv 0$ over $GF(2)$.

An Example

LDPC code: parity check matrix H and its Tanner graph

$$H = \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \\ \begin{array}{ccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{array} \end{array}$$

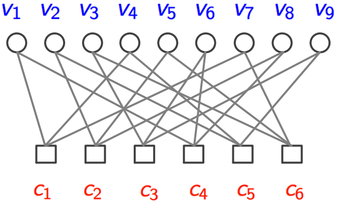


Parity check c_3: $v_3 + v_6 + v_9 \equiv 0$ over $GF(2)$.

An Example

LDPC code: parity check matrix H and its Tanner graph

$$
\begin{array}{c}
H = \left[\begin{array}{ccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}\right]
\begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{array} \\
\begin{array}{ccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{array}
\end{array}
$$



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

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:

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,

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,

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 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:

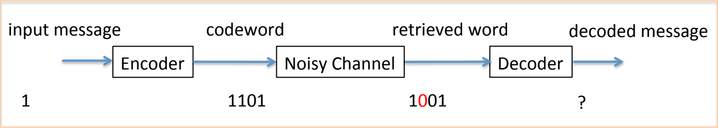
1. (bit-to-check) Each variable node sends a message to each check node it is connected to,
2. (check processing) Each check node then computes the consistency of incoming messages,
3. (check-to-bit) Each check node then sends a message to each variable node it is connected to,
4. (bit processing) Each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

Message Passing Decoding

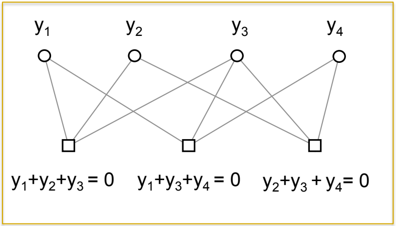
Passed messages can be either

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

An Example



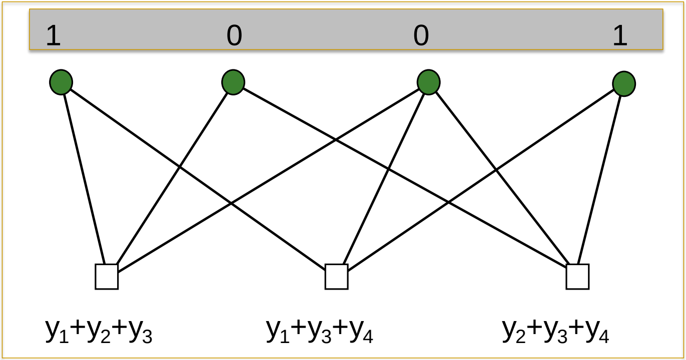
Message $\underline{m}$		Codeword $\underline{y}$
m_1		$y_1y_2y_3y_4$
0	$\rightarrow$	0 0 0 0
1	$\rightarrow$	1 1 0 1



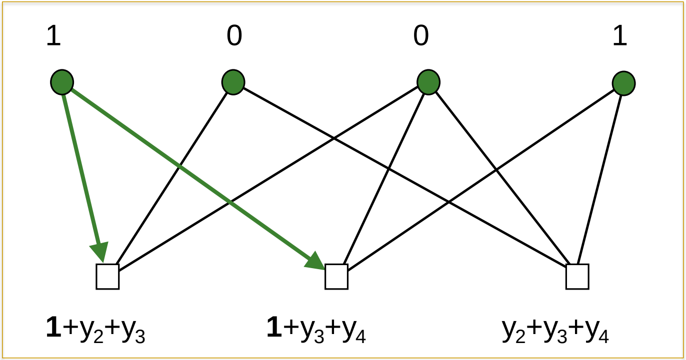
Message Passing Decoding

Bit-flipping algorithm

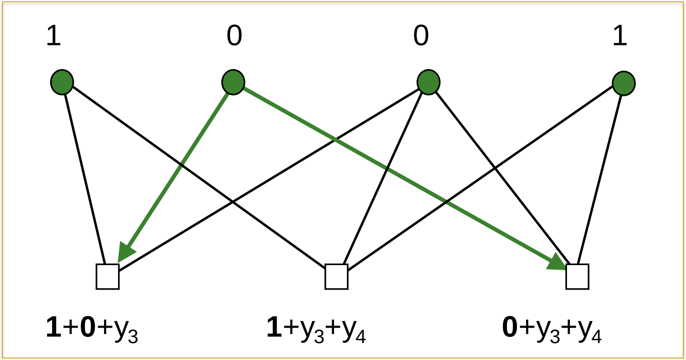
Received Codeword



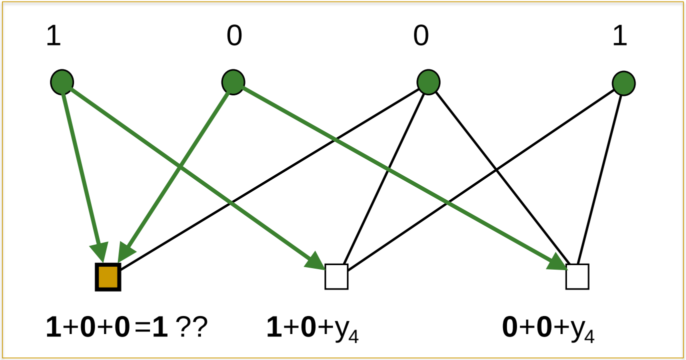
Bit-to-Check Messages



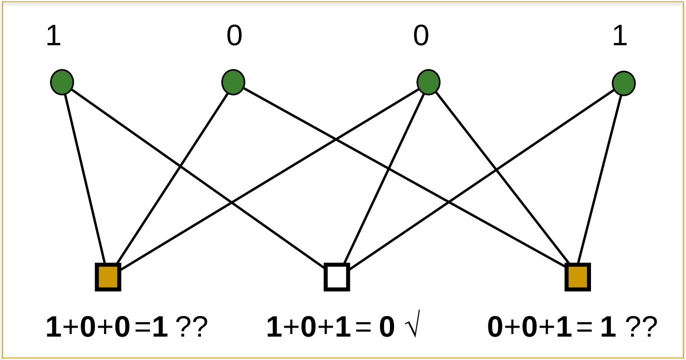
Bit-to-Check Messages



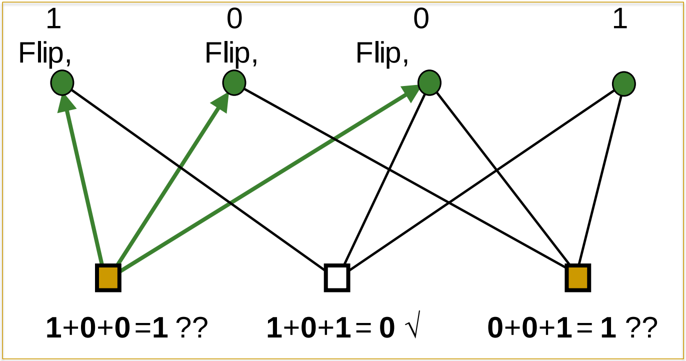
Check Processing



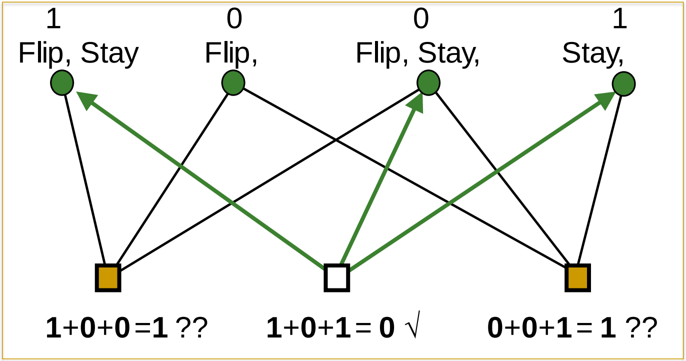
Check Processing



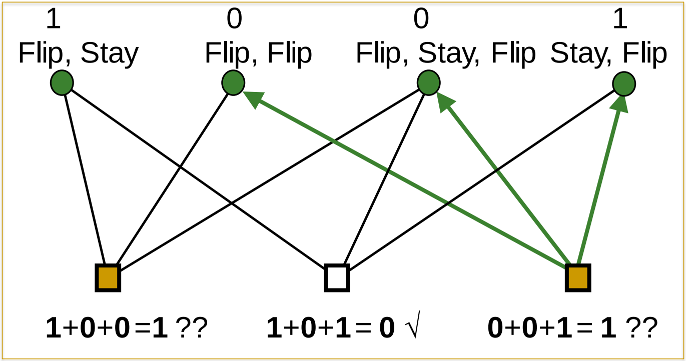
Check-to-Bit Messages



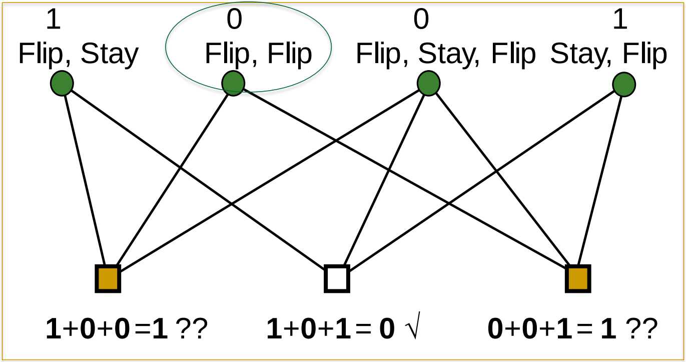
Check-to-Bit Messages



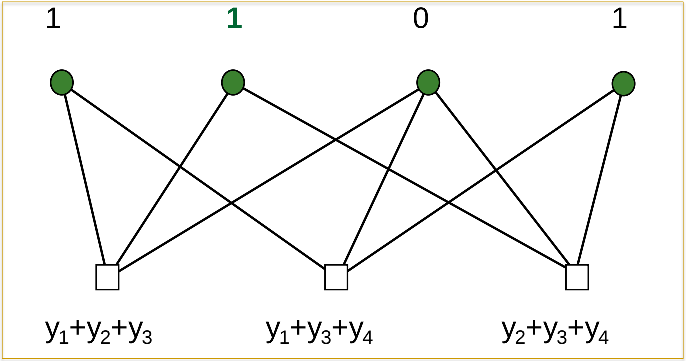
Check-to-Bit Messages



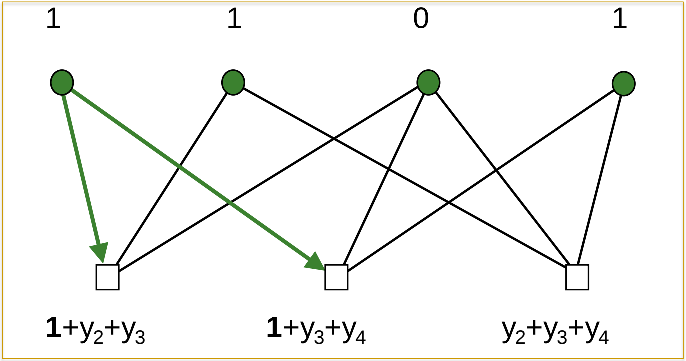
Bit Processing



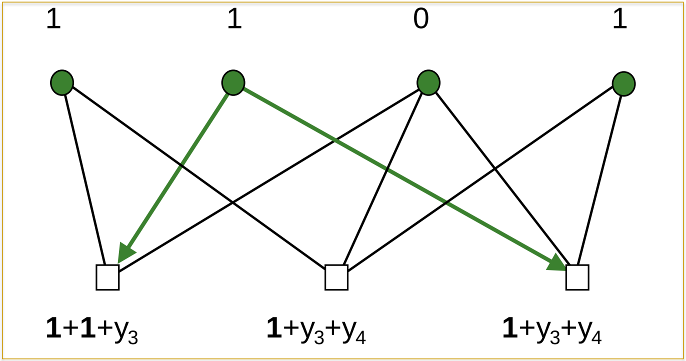
Bit Processing



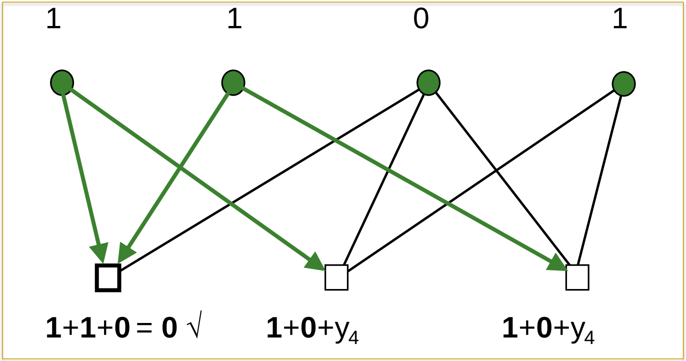
Bit-to-Check Messages



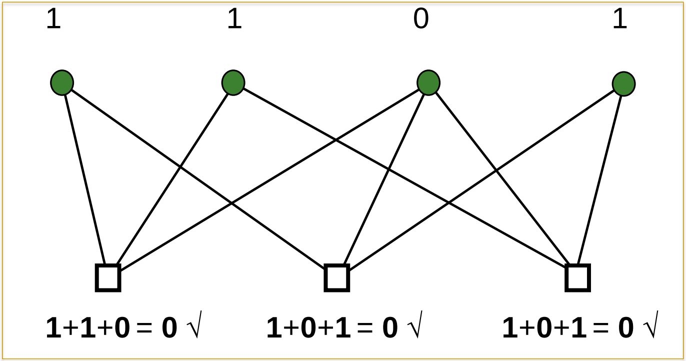
Bit-to-Check Messages



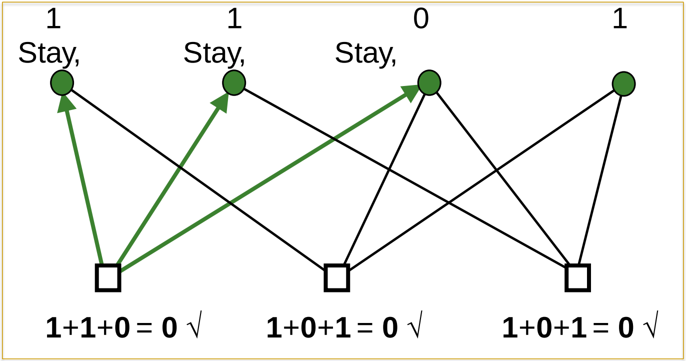
Bit-to-Check Messages



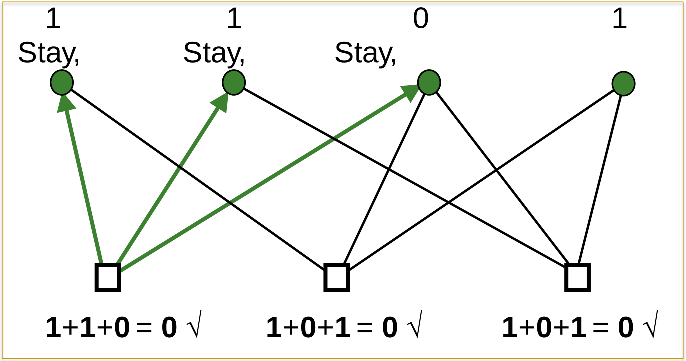
Check Processing



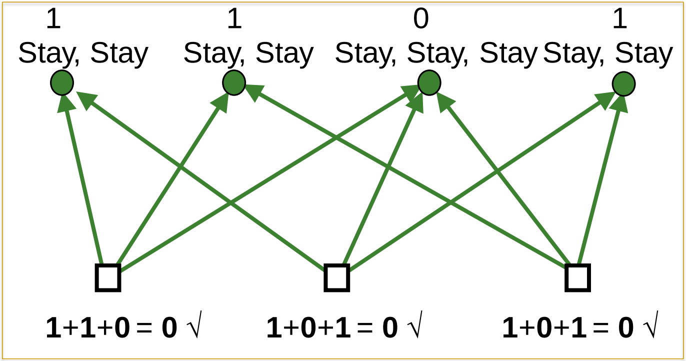
Check-to-Bit Messages



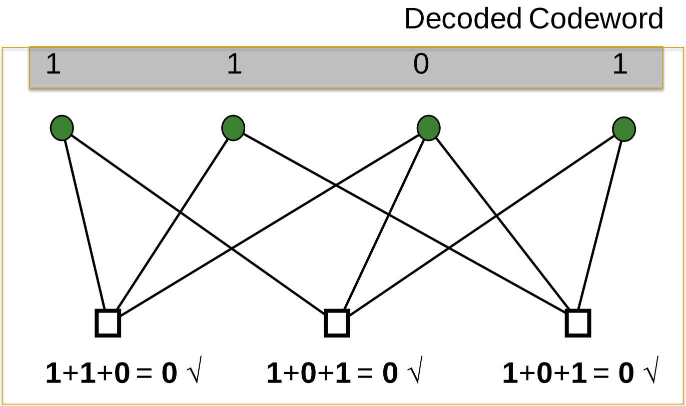
Check-to-Bit Messages



Check-to-Bit Messages



Bit Processing



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]

Min-sum algorithm (MSA) [3]

[1] R. Gallager, *MIT Press*, 1963.
[2] T. Richardson and R. Urbanke, *IEEE Trans. on Info. Theory*, 2001.
[3] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, *IEEE Trans. on Comm.*, 1999.

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 $L(v_i \to c_j) = \sum_{j' \in N(i) \backslash j} L(c_j' \to v_i) + L^{int}(v_i)$
- check-to-bit $L(c_j \to v_i) = \Phi^{-1}\left(\sum_{i' \in N(j) \backslash i} \Phi(|L(v_i' \to c_j)|) \sum_{i' \in N(j) \backslash i} sgn(L(v_i' \to c_j))\right)$ where $\Phi(x) = -\log(\tanh(x/2))$

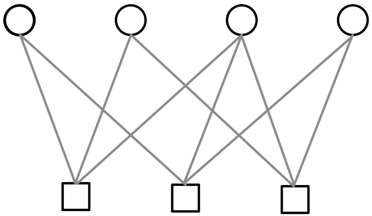
Min-sum algorithm (MSA) [3]

- check-to-bit $L(c_j \to v_i) = \min_{i' \in N(j) \backslash i} |L(v_i' \to c_j)| \prod_{i' \in N(j) \backslash i} sgn(L(v_i' \to c_j))$

[1] R. Gallager, *MIT Press*, 1963.
[2] T. Richardson and R. Urbanke, *IEEE Trans. on Info. Theory*, 2001.
[3] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, *IEEE Trans. on Comm.*, 1999.

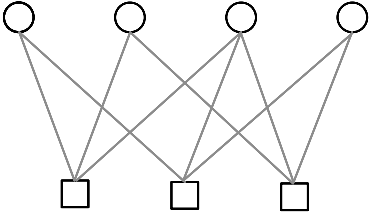
Soft Decoding

Bit values 1 1 0 1



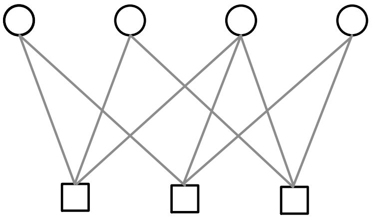
Soft Decoding

Bit values	1	1	0	1
Values using BPSK	-1	-1	+1	-1



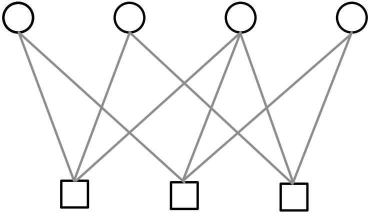
Soft Decoding

Bit values	1	1	0	1
Values using BPSK	-1	-1	+1	-1
Values from channel	-1.1	0.1	1.2	-0.9



Soft Decoding

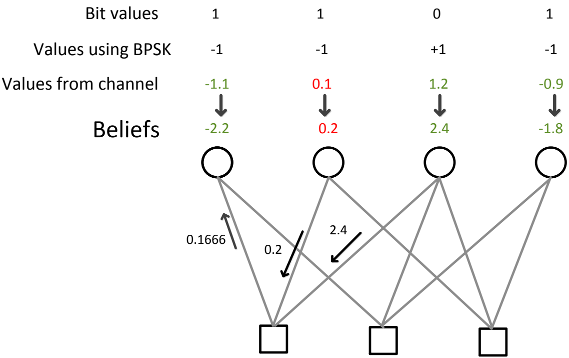
Bit values	1	1	0	1
Values using BPSK	-1	-1	+1	-1
Values from channel	-1.1	0.1	1.2	-0.9
Beliefs ($L_{v_i}^{(int)}$)	-2.2	0.2	2.4	-1.8



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

We assume $\sigma_n = 1$.

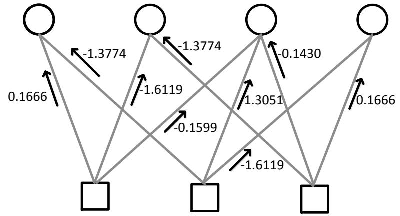
Soft Decoding



$$L_{c_j \to v_j} = 2\tanh^{-1}\left(\prod_{\substack{l \neq i \\ v_l \to c_j}} \tanh \frac{1}{2} L_{v_l \to c_j}\right)$$

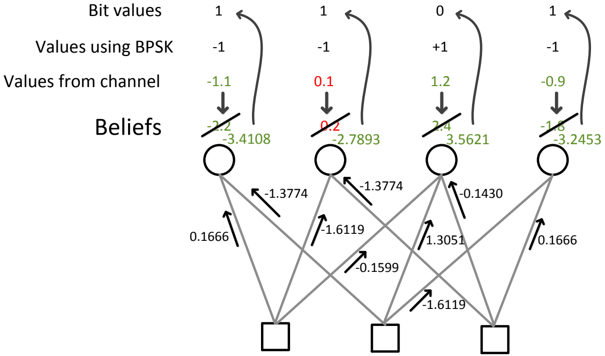
Soft Decoding

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



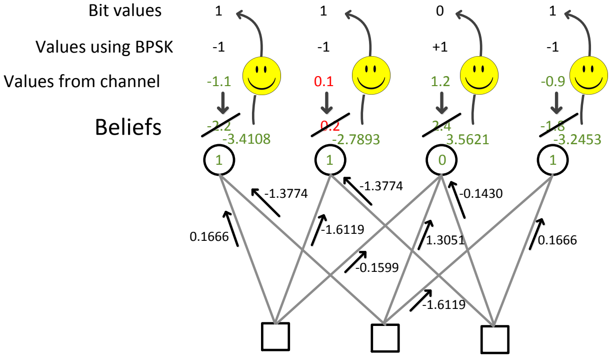
$$L_{c_j \to v_j} = 2\tanh^{-1}\left(\prod_{\substack{l \neq i \\ v_l \to c_j}} \tanh \frac{1}{2} L_{v_l \to c_j}\right)$$

Soft Decoding



$$L_{v_i} = L_{v_i}^{(int)} + \sum_{c_j \to v_i} L_{c_j \to v_j}$$

Soft Decoding



All variable nodes are decoded to correct bit value.

Performance with multi read

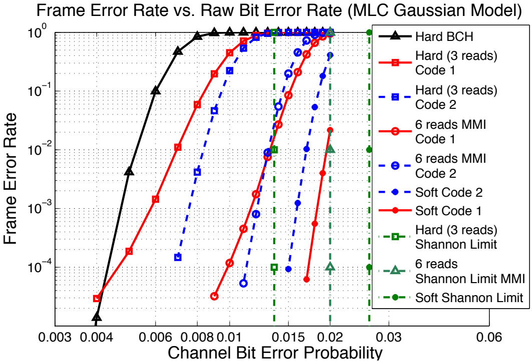
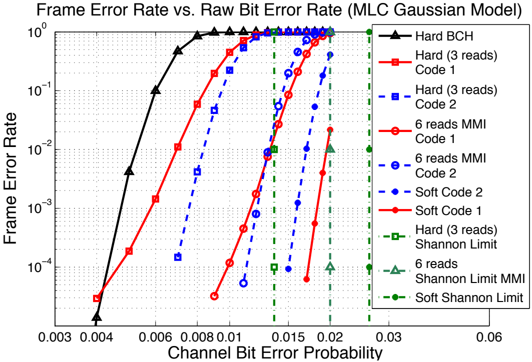


Figure: Rate 0.9 LDPC and BCH codes of length $n = 9100$.

Performance with multi read

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



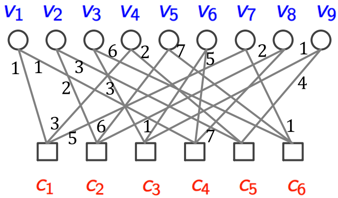
Caution:

- 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 $\alpha^k \to k+1$ for $0 \leq k \leq 6$)

$$
\begin{array}{cc}
H = \begin{bmatrix}
1 & 0 & 0 & 3 & 0 & 0 & 5 & 0 & 0 \\
0 & 2 & 0 & 0 & 6 & 0 & 0 & 2 & 0 \\
0 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 5 & 0 & 7 & 0 \\
0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 6 & 0 & 7 & 0 & 1 & 0 & 0
\end{bmatrix} & \begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{array} \\
\begin{array}{ccccccccc} \quad\quad v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{array} &
\end{array}
$$

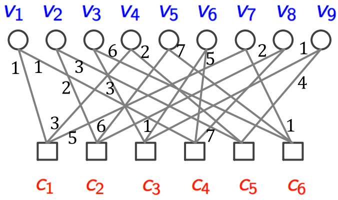


Parity check c_3: $3v_3 + v_6 + v_9 \equiv 0$ over $GF(8)$.

Non-binary LDPC codes

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

$$
H = \begin{array}{cc}
\left[\begin{array}{ccccccccc}
1 & 0 & 0 & 3 & 0 & 0 & 5 & 0 & 0 \\
0 & 2 & 0 & 0 & 6 & 0 & 0 & 2 & 0 \\
0 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 5 & 0 & 7 & 0 \\
0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 6 & 0 & 7 & 0 & 1 & 0 & 0
\end{array}\right] &
\begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{array} \\
\begin{array}{ccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{array} &
\end{array}
$$

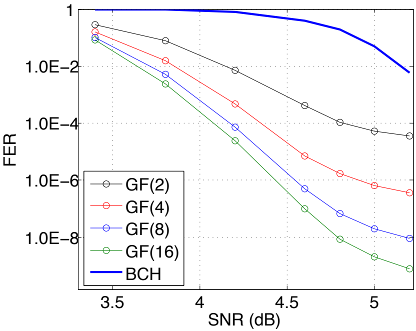


Parity check c_3: $3v_3 + v_6 + v_9 \equiv 0$ over $GF(8)$.

See talk on Thursday: Flash Controller Design (8:30 – 10:50)

Performance evaluation

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:

L. Dolecek, D. Divsalar, Y. Sun and B. Amiri, "Non-Binary Protograph-Based LDPC Codes: Enumerators, Analysis, and Designs," IEEE Transactions on Information Theory, vol. 60 (7), pp. 3913 – 3941, July 2014

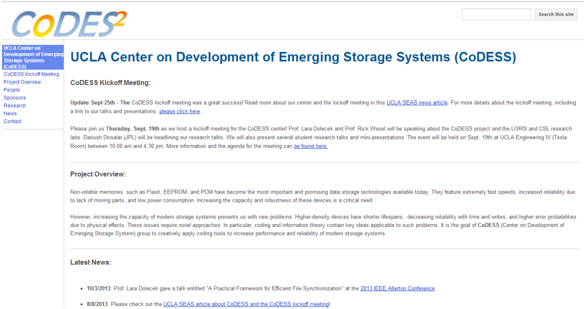
R. Gabrys, E. Yaakobi and L. Dolecek, "Graded bit error correcting codes with applications to Flash memory," *IEEE Transactions on Information Theory*, vol. 59(4), pp. 2315 – 2327, Apr. 2013.

J. Wang, L. Dolecek and R. Wesel, "The Cycle Consistency Matrix Approach to Absorbing Sets in Separable Circulant-Based LDPC Codes," IEEE Transactions on Information Theory, vol. 59(4), pp. 2293 – 2314, Apr. 2013.

B. Amiri, J. Kliewer, and L. Dolecek, "Analysis and Enumeration of Absorbing Sets for Non-Binary Graph-Based Codes," IEEE Transactions on Communications, vol. 62 (2), pp. 398 – 409, Feb. 2014.

We would like to invite you to explore CoDESS:

http://www.uclacodess.org



CoDES2

Search this site

UCLA Center on Development of Emerging Storage Systems (CoDESS)
CoDESS Kickoff Meeting
Project Overview
People
Sponsors
Research
News
Contact

UCLA Center on Development of Emerging Storage Systems (CoDESS)

CoDESS Kickoff Meeting:

Update Sept 25th - **The** CoDESS kickoff meeting was a great success! Read more about our center and the kickoff meeting in this UCLA SEAS news article. For more details about the kickoff meeting, including a link to our talks and presentations, please click here.

Please join us **Thursday, Sept. 19th** as we host a kickoff meeting for the CoDESS center! Prof. Lara Dolecek and Prof. Rick Wesel will be speaking about the CoDESS project and the LORIS and CSL research labs. Dariush Divsalar (JPL) will be headlining our research talks. We will also present several student research talks and mini-presentations. The event will be held on Sept. 19th at UCLA Engineering IV (Tesla Room) between 10:00 am and 4:30 pm. More information and the agenda for the meeting can be found here.

Project Overview:

Non-volatile memories, such as Flash, EEPROM, and PCM have become the most important and promising data storage technologies available today. They feature extremely fast speeds, increased reliability due to lack of moving parts, and low power consumption. Increasing the capacity and robustness of these devices is a critical need.

However, increasing the capacity of modern storage systems presents us with new problems. Higher-density devices have shorter lifespans, decreasing reliability with time and writes, and higher error probabilities due to physical effects. These issues require novel approaches. In particular, coding and information theory contain key ideas applicable to such problems. It is the goal of **CoDESS** (Center on Development of Emerging Storage System) group to creatively apply coding tools to increase performance and reliability of modern storage systems.

Latest News:

- **10/3/2013**: Prof. Lara Dolecek gave a talk entitled "A Practical Framework for Efficient File Synchronization" at the 2013 IEEE Allerton Conference.
- **8/8/2013**: Please check out the UCLA SEAS article about CoDESS and the CoDESS kickoff meeting!

For more information, please contact

Prof. Lara Dolecek
dolecek@ee.ucla.edu