STANFORD

Supplementary Lecture 9B

BCH Decoders

February 3, 2026

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

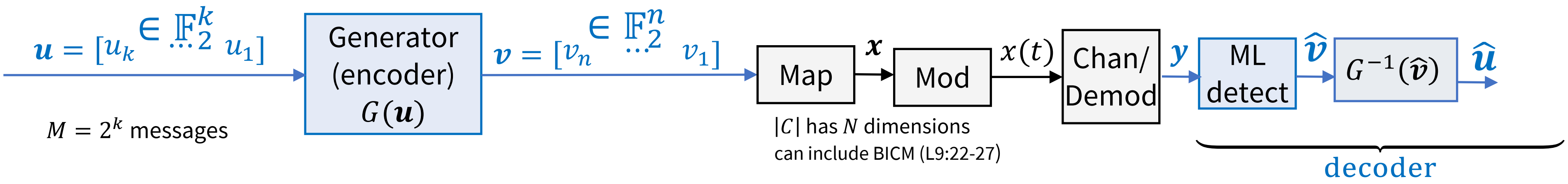
Instructor EE379A – Winter 2026

Announcements & Agenda

- Announcements

- Binary code refresh
- BCH Binary Codes' Decoding

Binary Codes Revisited



- So what are good G and/or H for binary block codes?
  - BCH codes have $g(D)$ as a product of binary primitive polynomials, with these polynomials chosen to have $GF(2^m)$ roots as prime powers $\{\alpha^1, \alpha^3, \alpha^5, \alpha^7, , \alpha^{11}, \ldots\}$ - each prime-power value being the sole root chosen from a conjugacy class.
- The variable D nominally is zeroed, BUT cyclic binary codes reintroduce it within the block
  - Cyclic codes' codewords will all be cyclic shifts of one another. Thus, D is basically (almost) a cyclic shift.
- Very high d_{free} is possible, and ML decoders can have reasonable complexity.
- BCH Codes are cyclic; Reed Muller are not (but these have a trellis so decoder can use Viterbi again)
  - Simplest BCH is a Hamming Code
  - Simplest RM is an extended Hadamaard Code

Bose-Chaudhuri–Hocquenghem (BCH) *Binary* Codes

Section 7.2

S9A on Galois Field Arithmetic can be helpful review (Also Appendix B).

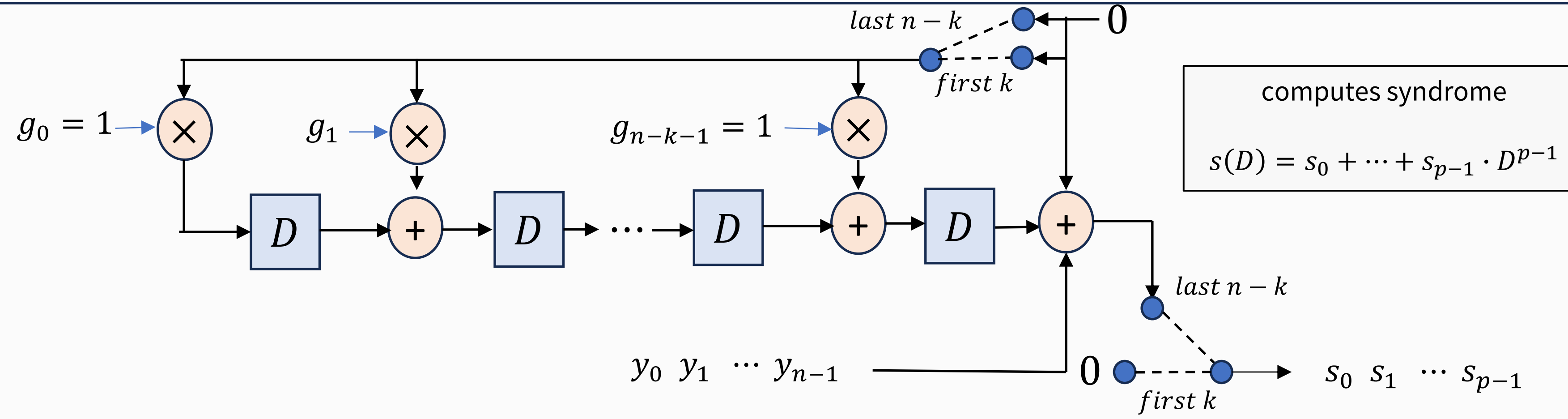
Calculate the syndrome

- $\boldsymbol{s} = \boldsymbol{y} \cdot H^t = \boldsymbol{e} \cdot H^t$
- Where

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-2} \\ 1 & (\alpha^2) & (\alpha^2)^2 & \dots & (\alpha^2)^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & (\alpha^{2t}) & (\alpha^{2t})^2 & \dots & (\alpha^{2t})^{n-1} \end{bmatrix}$$

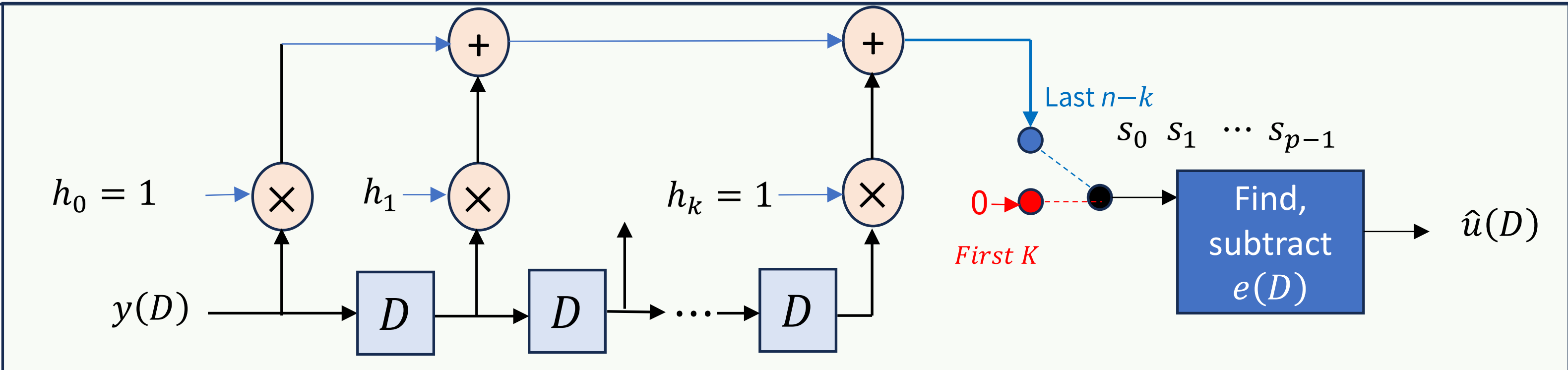
- $\boldsymbol{s}$ will have nonzero contributions only from locations where errors are 1.
- This set of locations will be $L_e = \{n_1 \cdots n_t\}$.
- So the syndrome elements essentially ”pick off” the corresponding powers, α^{n_i} , and add them.

Detector Syndrome Calculation (binary logic) review



- Almost the same as encoder, except it essentially adds the remainder back (if no errors, $s(D) = 0$)

OR



Decoding BCH

- The number of errors $\leq t$ where the parity is $2t$ and $d_{free} \geq 2t+1$.

- The BSC ML decoder finds the (initially unknown) locations of these errors, $L_e = \{n_1 \cdots n_t\}$.
  - Then, it flips the bits in these locations.

- The syndrome polynomial $s(D) = \sum_{i=1}^{n-k} s_i \cdot D^i, = 0$ *with no errors,* at each $g(D)$ root in $GF(2^m)$, but with errors:

$$
\begin{aligned}
S_1 &\triangleq e(\alpha^{n_1}) = & e_{n_1} \cdot \alpha^{n_1} + \ldots + e_{n_t} \cdot \alpha^{n_t} = \sum_{n_i \in L_e} \alpha^i \\
S_2 &\triangleq e(\alpha^{n_2}) = & e_{n_1} \cdot \alpha^{2 \cdot n_1} + \ldots + e_{n_t} \cdot \alpha^{2 \cdot n_t} = \sum_{n_i \in L_e} \alpha^{2i} \\
\vdots & \quad \ldots & \vdots \\
S_{2t} &\triangleq e(\alpha^{n_t}) = & e_{n_1} \cdot \alpha^{2t \cdot n_1} + \ldots + e_{n_t} \cdot \alpha^{2t \cdot n_t} = \sum_{n_i \in L_e} \alpha^{2ti}
\end{aligned}
$$

- More than t of these are linearly dependent, so delete those (only 1 root per conjugacy class).

Reduced Syndrome to linearly independent

- Eliminate all roots but 1 in the same conjugacy class.

$$
\begin{aligned}
S_1 &= Y_{n_1} \cdot X_{n_1} + Y_{n_2} \cdot X_{n_2} + \ldots + Y_{n_t} \cdot X_{n_t} = \sum_{i=1}^{t} Y_{n_i} \cdot X_{n_i} \\
S_2 &= Y_{n_1} \cdot X_{n_1}^2 + Y_{n_2} \cdot X_{n_2}^2 + \ldots + Y_{n_t} \cdot X_{n_t}^2 = \sum_{i=1}^{t} Y_{n_i} \cdot X_{n_i}^2 \\
\vdots &= \qquad \vdots \\
S_t &= Y_{n_1} \cdot X_{n_1}^t + Y_{n_2} \cdot X_{n_2}^t + \ldots + Y_{n_t} \cdot X_{n_t}^t = \sum_{i=1}^{t} Y_{n_i} \cdot X_{n_i}^t \ .
\end{aligned}
$$

So far, we know $S_{1:t}$ BUT not yet →

$$X_{n_i} \triangleq \alpha^{n_i}$$

$$Y_{n_i} \triangleq e_{n_i}$$

- Define error-locator polynomial

$$
\begin{aligned}
\Lambda(x) &\triangleq (1 - X_{n_1} \cdot x) \cdot (1 - X_{n_2} \cdot x) \ldots (1 - X_{n_t} \cdot x) \\
&= 1 + \Lambda_1 \cdot x + \Lambda_2 \cdot x^2 + \ldots + \Lambda_t \cdot x^t \ ,
\end{aligned}
$$

- Evaluate error-locator

and multiplied by $Y_{n_i} \cdot X_{n_i}^{j+t}$ for each of $j = [1:t]$ and $i = [1:t]$, becomes (with $\Lambda_0 = 1$):

$$0 = Y_{n_i} \cdot X_{n_i}^{j+t} \cdot (1 + \Lambda_1 \cdot X_{n_i}^{-1} + \Lambda_2 \cdot X_{n_i}^{-2} + \ldots + \Lambda_t \cdot X_{n_i}^{-t}) = Y_{n_i} \cdot X_{n_i}^{j+t} \cdot \sum_{\ell=0}^{t} \Lambda_\ell \cdot X_{n_i}^{-\ell} \ .$$

Using (7.11) and summing all instances of (7.14) over $n_{i \in [1:t]}$, for each $j \in [1:t]$, yields

$$0 = \sum_{\ell=0}^{t} \Lambda_\ell \cdot \sum_{i=1}^{t} Y_{n_i} \cdot X_{n_i}^{j+n_t-\ell} = \sum_{\ell=0}^{n_t} \Lambda_\ell \cdot S_{j+t-\ell} \ .$$

Results in Linear set of t equations

- Solve for t:-1:1
- If M is singular, reduce t

$$\underbrace{\begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ \vdots \\ -S_{2t} \end{bmatrix}}_{\boldsymbol{s}} = \underbrace{\begin{bmatrix} S_t & S_{t-1} & \dots & S_1 \\ S_{t+1} & S_t & \dots & S_2 \\ \vdots & \ddots & \ddots & \vdots \\ S_{2t-1} & S_{2t-2} & \dots & S_t \end{bmatrix}}_{M} \cdot \underbrace{\begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix}}_{\boldsymbol{\Lambda}},$$

- If no solution for t, then more than t errors are detected

Example: $g(D) = 1 + D + D^2 + D^4 + D^5 + D^8 + D^{10}$

```
>> Gbch % = GF(2) array. Array elements =
 1 1 1 0 1 1 0 0 1 0 1 0 0 0 0
 0 1 1 1 0 1 1 0 0 1 0 1 0 0 0
 0 0 1 1 1 0 1 1 0 0 1 0 1 0 0
 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 1 1 1 0 1 1 0 0 1 0 1
```

5x15

All circular shifts, and the parity is for roots of $g(D)$

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \\ 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} \end{bmatrix}$$

```
>> FIELD = gftuple([-1 : 2^4-2]', 4, 2) %=
 0 0 0 0
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
 1 1 0 0
 0 1 1 0
 0 0 1 1
 1 1 0 1
 1 0 1 0
 0 1 0 1
 1 1 1 0
 0 1 1 1
 1 1 1 1
 1 0 1 1
 1 0 0 1
```

Gftuple generates Field elements

flip →

1	x	x^2	x^3	α^i	$GF(2^4)$
0	0	0	0	–	0
1	0	0	0	α^0	1
0	1	0	0	α^1	2
0	0	1	0	α^2	4
0	0	0	1	α^3	8
1	1	0	0	α^4	3
0	1	1	0	α^5	6
0	0	1	1	α^6	12
1	1	0	1	α^7	11
1	0	1	0	α^8	5
0	1	0	1	α^9	10
1	1	1	0	α^{10}	7
0	1	1	1	α^{11}	14
1	1	1	1	α^{12}	15
1	0	1	1	α^{13}	13
1	0	0	1	α^{11}	9

```
>> Hbch % = GF(2) array. Array elements =

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

Alpha5 and alpha10 have same two MSBs

Example Encoder and Decoder

```
bits=gf([1 0 1 1 0]); % generate 5 input bits
 v=bits*Gbch;        % form codeword
 v*Hbch'             % check codeword
% ans = GF(2) array.  Array elements =
%   0   0   0   0   0   0   0   0   0   0   (checks)
%
error = gf([0   0   0   0   1   0   0   0   0   1   0   0   0   0   0]); % form 2 errors
y=v+error;  % form received vector with errors
s=y*Hbch'      % form syndrome
% ans = GF(2) array. Array elements =
%  1   0   0   1   0   0   0   0   1   1

>> one16=gf(2*ones(1,15),4); % GF(16) vector of all twos = all alpha^1
% one16 = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)  Array elements
=
%   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2

>> alpha=one16.^[0:14] % GF(16) vector of powers of 2 = alpha  from 0 to 14,
% alpha = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)  Array elements =
%    1    2    4    8    3    6   12   11    5   10    7   14   15   13    9
 >> v16*alpha' % check it is root
% ans = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)   Array elements =
%   0
>> v16*(alpha.^(2*ones(1,15)))'
% ans = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)  Array elements =
%   0    also checks
```

```
>> S1=y16*alpha'                          %     9
>> S2=y16*(alpha.^(2*ones(1,15)))'  %   13
>> S3=y16*(alpha.^(3*ones(1,15)))'  %     0
>> S4=y16*(alpha.^(4*ones(1,15)))'  %   14
>> S5=y16*(alpha.^(5*ones(1,15)))'  %     7
>> S6=y16*(alpha.^(6*ones(1,15)))'  %     0
%
% Form the 3x3 matrix to check if 3 errors
>> M=[S3 S2 S1 ; S4 S3 S2 ; S5 S4 S3]
%M = GF(2^4) array. Primitive polynomial = D^4+D+1 (19
decimal)  Array elements =
%   0   13    9
%  14    0   13
%   7   14    0
%
>> det(M) % =     0 (So 2 errors or less)

%--------------- try size 2 matrix M -----------
M=[S2 S1 ; S3 S2]
%   13    9
 %   0   13
>> Svec=-[S3 ; S4] % form other side of equation
 %    0
 %  14
>> Lambda = inv(M)*Svec  %=
%    9
%   13
```

- Now compute syndromes and form matrix.

Continued

```
roots([1, Lambda']) % find GF(16) roots by recalling first element is 1
% ans = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)  Array
elements =
%   3
%  10
% NOTE THIS IS REVERSE ORDER ALREADY IN MATLAB, so we get error positions
directly
%
alpha % alpha = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)
   1   2   4   8   3   6  12  11   5  10   7  14  15  13   9
 error % error = GF(2) array. Array elements =
   0   0   0   0   1   0   0    0   0   1   0   0   0   0   0  (checks)
 ----  3 errors --------------
error3=error;
error3(1)=gf(1);
y=v+error3  % y = GF(2) array.  Array elements =
  0  1  0  0  0  0  1  0  0  1  0  1  1  1  0
y16 = [0  1  0  0  0  0  1  0  0  1  0  1  1  1  0]; % change back to integer
y16=gf(y16,4); % now change from integer to GF(16)
S1=y16*alpha'                        %     9
S2=y16*(alpha.^(2*ones(1,15)))'  %   13
S3=y16*(alpha.^(3*ones(1,15)))'  %     0
S4=y16*(alpha.^(4*ones(1,15)))'  %   14
S5=y16*(alpha.^(5*ones(1,15)))'  %     7
S6=y16*(alpha.^(6*ones(1,15)))'  %     0
```

```
 % find the inverse
M=[S3 S2 S1 ; S4 S3 S2 ; S5 S4 S3];
>> det(M) % = 15, so non zero and 3 errors
>> Svec=-[S4 ; S5 ; S6]
  15  ; 6 ;   1
>> Lambda = inv(M)*Svec
   8  ;  4  ;  13
>> lambda=roots([1 Lambda'])
   1  ; 3   ;  10
error4 =gf([1 0 1 0 1 0 1 0 0 0 0 0 0 0 0]);
y16 = [0  1  1  0  0  0  0  0  0  0  0  1  1  1  0]; %
convert to integer
y16=gf(y16,4); % convert to GF(16)
S1=y16*alpha'                        %     9
S2=y16*(alpha.^(2*ones(1,15)))'  %   13
S3=y16*(alpha.^(3*ones(1,15)))'  %     0
S4=y16*(alpha.^(4*ones(1,15)))'  %   14
S5=y16*(alpha.^(5*ones(1,15)))'  %     7
S6=y16*(alpha.^(6*ones(1,15)))'  %     0
M=[S3 S2 S1 ; S4 S3 S2 ; S5 S4 S3];
Svec=-[S4 ; S5 ; S6]
Lambda = inv(M)*Svec
  10  ; 8   ;  10
>> lambda=roots([1 Lambda'])
lambda = GF(2^4) array, no output (fails)
```

- See text for 4 errors corrected, can get lucky sometimes

STANFORD

End Lecture S9B