STANFORD

Lecture 7

Binary Codes and BICM

January 30, 2024

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

Instructor EE379A – Winter 2024

Announcements & Agenda

- Announcements
  - PS3 due tomorrow
  - PS4 due Feb 6, no late (solutions immediate)
  - Midterm Feb 8 (PS5 the following week)
    - Open book, laptop, internet
    - In class (or other arrangements)
  - Web site is usually best place for latest copy
    - Canvas uses R1, R2, … notation so you can see history
      - Removed by SU after quarter end
  - The Edstem page (responding there also)
    - Just arrived this morning for me, and I responded.
  - Feedback
    - 6-15 Hours
    - Ethan very much appreciated.
    - Homework extends understanding.

- Today
  - Finish L6
  - Binary Codes in GF(2) – Basics
    - Convolutional code tables
    - Block code parametrization (LDPC)
  - Binary Code Use
  - Mappings to M-ary Symbols: BICM

- PS3.1 (1.63)
  - $L(d)$ is an overall gain multiplier applied to each (all) multipaths
  - Equivalently to the entire channel response
  - d_{bp} is a specific model parameter (simplified Wi-Fi) where an extra attenuation factor applies for distances longer than this "break-point" distance

- PS3.2 (1.65)
  - For the last part, the number of samples per try might best be 100k , not 10k
    - This gives a little more accurate match between theory and simulation.

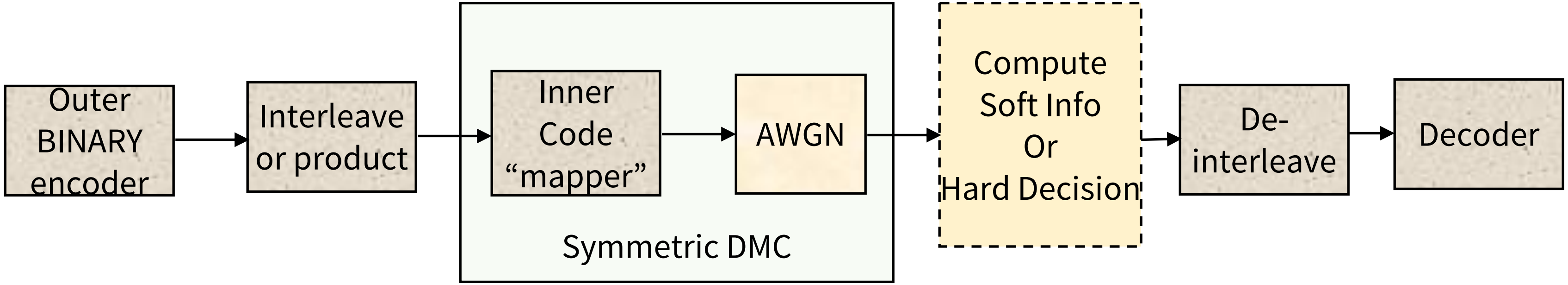
- Problem Set 4 = PS4 due Tuesday February 6 at 17:00, no late
  1. 8.1 A convolutional encoder and code
  2. 8.2 Systematic encoders
  3. 8.4 A fool's code
  4. 8.5 power-bandwidth trade at $\bar{b} < 1$
  5. 8.8 code for satellite transmission

Finish L6

Sections 2.1-2

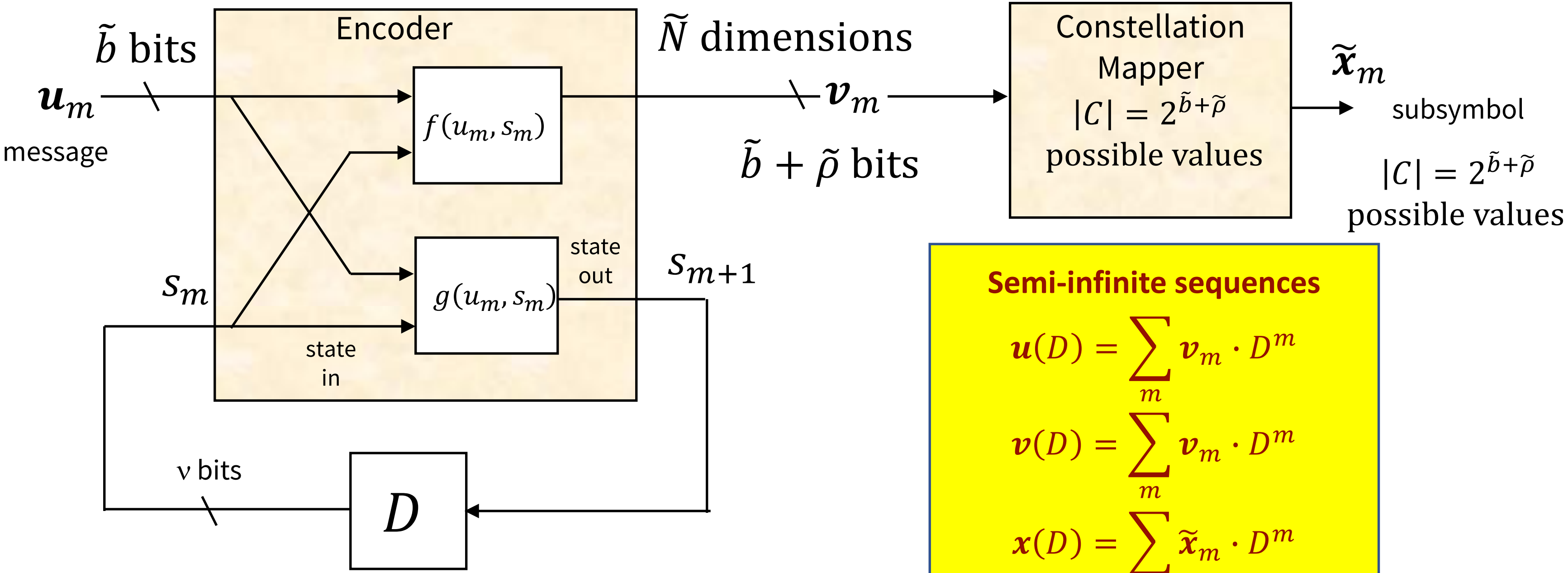
Modern Powerful codes

- γ_f is large, equivalently can be reliably decoded (low Pe).
- γ_f is large with good long-length binary codes:
  - With binary-to-$|C|$ "**mapper**" for larger QAM constellations
  - But leaves shaping (γ_s) to the constellation boundary design (< 1.53 dB).



Generalization: Sequential Encoder & Mapper

- **Trellis** or **Convolutional** Codes (see feedback below) have model:



Semi-infinite sequences

$$\boldsymbol{u}(D) = \sum_m \boldsymbol{v}_m \cdot D^m$$

$$\boldsymbol{v}(D) = \sum_m \boldsymbol{v}_m \cdot D^m$$

$$\boldsymbol{x}(D) = \sum_m \tilde{\boldsymbol{x}}_m \cdot D^m$$

- Inputs have $\tilde{b}$ – usually bits.
- Outputs are $\tilde{N}$ –dimensional.
  - When $\tilde{\boldsymbol{x}} \in \mathbb{C}^{\tilde{N}}$ → Trellis Code.
  - When $\tilde{\boldsymbol{x}} = \boldsymbol{v} \in GF(2)^{\tilde{N}}$ → Binary convolutional code.

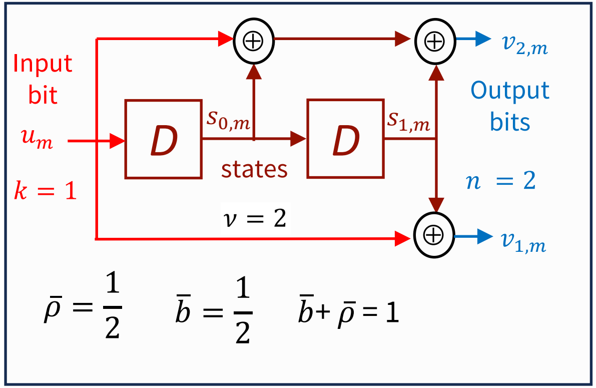
This``fakes'' a larger block length with finite real-time complexity/delay

Binary Codes in GF(2) - Basics

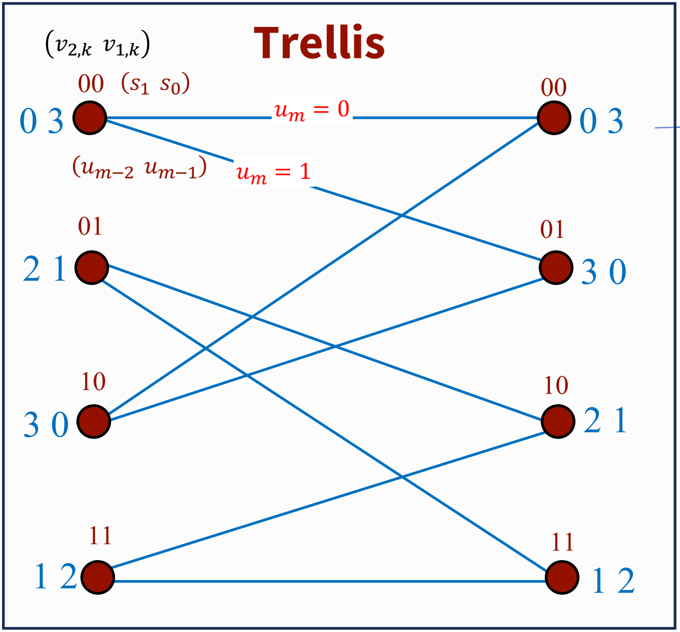
Section 8.1

Example, rate $r = ½$ convolutional code

$(v_{2,m} \;\; v_{1,m})=(0,0) = 0$
$(v_{2,m} \;\; v_{1,m})=(0,1) = 1$
$(v_{2,m} \;\; v_{1,m})=(1,0) = 2$
$(v_{2,m} \;\; v_{1,m})=(0,1) = 3$



$\bar{\rho} = \frac{1}{2}$ $\quad \bar{b} = \frac{1}{2}$ $\quad \bar{b} + \bar{\rho} = 1$



Trellis stage for each time m has

2 possible output Subsymbols, $n = 2$.

Encoder is in 1 of 4 STATES, $\nu = 2$.

$d_{free} = 5$

$G(D) = k \times n$ **generator matrix**, elements/ops in $GF(2) = \mathbb{F}_2$.

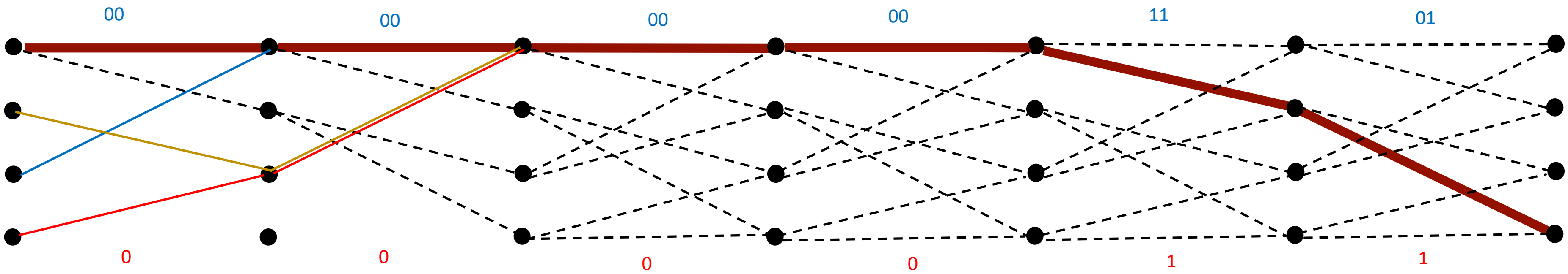
$$\boldsymbol{v}(D) = u(D) \cdot \underbrace{[1 + D + D^2 \quad 1 + D^2]}_{G(D)}$$

$H(D)$ is an $(n - k) \times n$ **parity matrix**, null space of $G(D)$, $G(D) \cdot H^t(D) = 0$,

$$H(D) = [1 + D^2 \quad 1 + D + D^2]$$

Example with 6 bits of input (12 output)

Cardinal is sequence or path corresponding to input bits below trellis (outputs blue).



- Other paths are possible, indeed 63 more of them (if initial state known as shown ---).

- Each path has 12 output bits,
  - and there is 1-to-1 map if we know initial state.
  - The other possible "unknown-initial-state" paths differ only in first $\nu = 2$ stages.

Binary Codewords & Sequences

- **Galois Field 2** → GF(2), or $\mathbb{F}_2$, is the binary field of two elements $\{0\,,1\}$ or bits - See Appendix B.
  - Addition is ``exclusive or,'' $\oplus$.
  - Multiplication is “and,” $\wedge$, which this class writes as “$\cdot$ ”.
  - No complex variables exist in our finite fields (not in this class).
- **Block** codes’ codewords are **finite sequences** of $n \triangleq N$ binary subsymbols.
- **Convolutional** codes’ codewords are **semi-infinite sequences** of $n \triangleq \tilde{N}$-dimensional binary-vector subsymbols.
  - Sequence time index is m, which has **D-Transform** notation $a(D) = \sum_{m=-\infty}^{\infty} a_m \cdot D^m$.

D is dummy variable, $D \oplus D = 0\,; D^l \cdot D^m = D^{l+m}$.

- The **ring** of **finite-length** binary sequences is

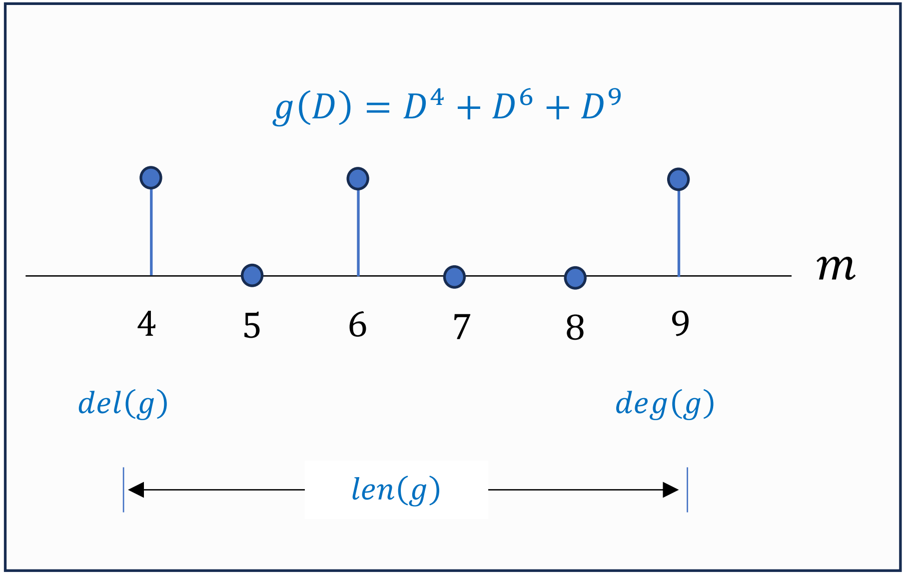
$$F[D] \triangleq \left\{ a(D) \middle| a(D) = \sum_{m=-\infty}^{\infty} a_m \cdot D^m, a_m \in \mathbb{F}_2\,, \nu \in \{0, Z^+\} \right\}.$$

- The **field** of **causal infinite-length binary sequences** is

$$F_r[D] = \triangleq \left\{ c(D) \middle| c(D) = \frac{a(D)}{b(D)},\ a(D), b(D) \in F[D],\ b(D) \neq 0 \wedge b_0 = 1 \right\},$$ ~ long division

Sequence parameters

- **Sequence delay** is
  - $del(a) = \min_m a_m = 1 \quad (a(D) = 0 \rightarrow del = \infty)$
  - Lowest power of D
  - $del(g) = 4$.
- **Sequence degree** is
  - $deg(a) = \max_m a_m = 1 \quad (a(D) = 0 \rightarrow del = -\infty)$
  - Highest power of D
  - $del(g) = 9$.
- **Sequence length** is
  - $len(a) = deg(a) - del(a) + 1 \, ; len(0) = 0$
  - $\text{len}(g) = 6$.

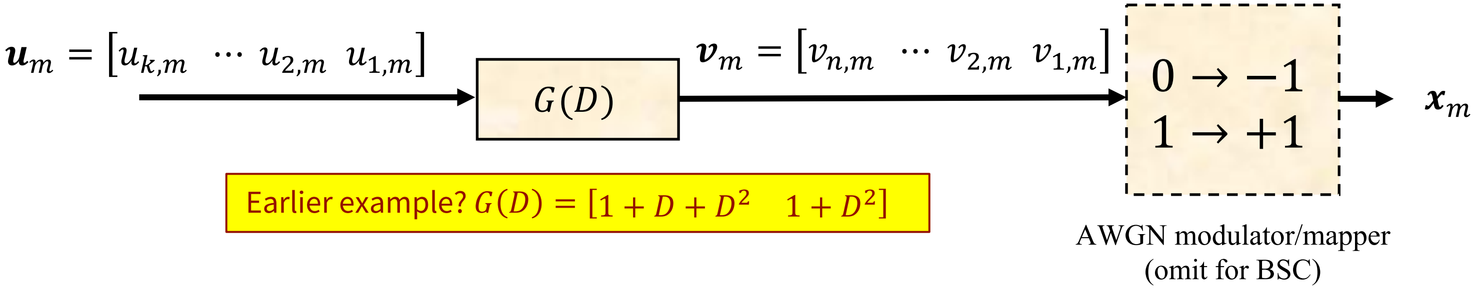


- **Constraint length** is
  - $\nu = len(a) - 1 = deg(a) - del(a)$ or number of delay elements if $a(D)$ has feedback.

LINEAR Binary Code

- **Linear Binary Code** is a set of binary sequences such that
  - $C[G] \triangleq \{v(D) \mid v(D) = u(D)\cdot G(D), u(D) \in F_r(D)\}$.
  - **Rate** $r = k/n$, so conv codes often use m as a time index.
  - **Systematic** if $v_{n-i} = u_{k-i}$, $i = 0, \dots, k-1$ for all times m.
  - **Free Distance** $d_{free} = \min_{\boldsymbol{v} \neq \boldsymbol{v}'} d_H(\boldsymbol{v}, \boldsymbol{v}')$.

Code = outputs



$\boldsymbol{u}_m = [u_{k,m} \ \cdots \ u_{2,m} \ u_{1,m}]$

$G(D)$

$\boldsymbol{v}_m = [v_{n,m} \ \cdots \ v_{2,m} \ v_{1,m}]$

$0 \rightarrow -1$

$1 \rightarrow +1$

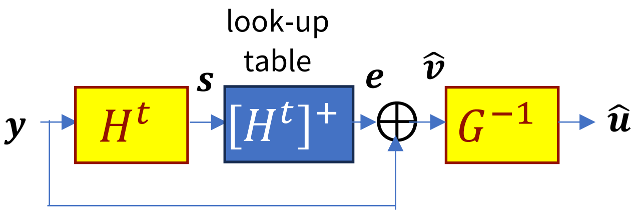
$\boldsymbol{x}_m$

Earlier example? $G(D) = [1 + D + D^2 \quad 1 + D^2]$

AWGN modulator/mapper
(omit for BSC)

When $\mathrm{G}(D) = G(0)$, it is a **block code**, otherwise a **convolutional code .**

Syndrome Decoding for Linear (binary) Block Codes



- **Parity Matrix**
  - H is a $(n-k)\times n$ binary matrix such that $\boldsymbol{v}\cdot H^t = 0\,, \forall\ v \in C$
  - $G\cdot H^t = 0$
  - H is a generator too (dual code, rate $1-r$), and spans the null space of G.
    - The generator and parity matrices together span $[GF(2)]^N = \{\boldsymbol{u}\cdot G\} \cup \{\boldsymbol{u}'\cdot H\}$.
      - This is the same concept as a real/complex matrix' pass and null spaces, but with finite field.
- Binary-channel output $\boldsymbol{y} = \boldsymbol{v} \oplus \boldsymbol{e}$; $\boldsymbol{e}$ is the error sequence.

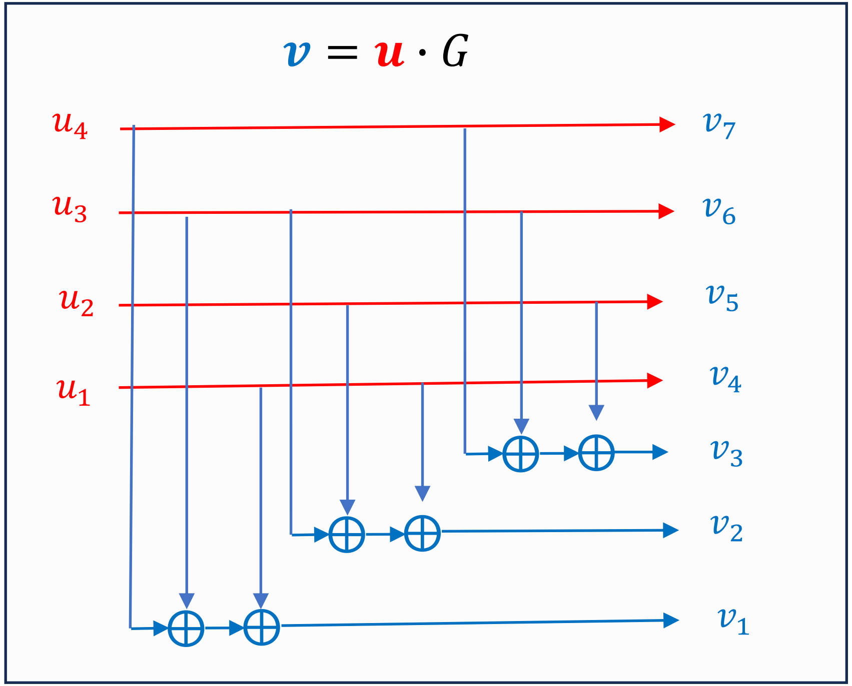
$\boldsymbol{s} = \boldsymbol{y}\cdot H^t = \boldsymbol{e}\cdot H^t$ is the $1\times(n-k)$ **syndrome vector.**

- ML Decoder finds the smallest Hamming weight $\boldsymbol{e}$ that solves this equation for the given $\boldsymbol{s}$.
  - There are fancy algorithms that find this finite-field pseudoinverse efficiently for certain linear codes.

- For present discussion, store 2^{n-k} values of $\boldsymbol{e}$ in a look-up table.

- $\widehat{\boldsymbol{v}} = \boldsymbol{y} \oplus \boldsymbol{e} \rightarrow \widehat{\boldsymbol{u}} = G^{-1}\cdot\widehat{\boldsymbol{v}}$ (for systematic codes, this is simply $\hat{u}_{k-i} = \hat{v}_{n-i}$, $i = 0, \ldots, k-1$).

LINEAR Block-Code example Hamming (7,4) code



only corrects a single error, but
$\bar{b}$ = 4/7 > 1/3 majority- vote

- $k = 4\,; n = 7$
- Systematic $G = \begin{bmatrix} 1&0&0&0&1&0&1 \\ 0&1&0&0&1&1&1 \\ 0&0&1&0&1&1&0 \\ 0&0&0&1&0&1&1 \end{bmatrix} = [I\ h^t]$
- Parity $H = \underbrace{\begin{bmatrix} 1&1&1&0&1&0&0 \\ 0&1&1&1&0&1&0 \\ 1&1&0&1&0&0&1 \end{bmatrix}}_{[h \qquad I]}\,; G \cdot H^t = 0$
- Rate/redundancy
  - $r = \bar{b} = \frac{4}{7}$
  - $\bar{\rho} = \frac{3}{7}$
  - $\bar{b} + \bar{\rho} = 1$
- Performance $d_{free} = 3$
  - $\nu = 0$

```
>> G(end:-1:1,end:-1:1) =
    1   0   0   0   1   0   1
    0   1   0   0   1   1   1
    0   0   1   0   1   1   0
    0   0   0   1   0   1   1

>> H(end:-1:1,end:-1:1) =
    1   1   1   0   1   0   0
    0   1   1   1   0   1   0
    1   1   0   1   0   0   1

>> gf(G)*gf(H')=
       0   0   0
       0   0   0
       0   0   0
       0   0   0
```

General Hamming Code

Integer parameter $p \geq 2$

$n = 2^p - 1\,;\ \ k = n - p$

$r = k/n$ & $d_{free} = 3$

```
>> p=3;
>> [H,G]=hammgen(p);
>> G(end:-1:1,end:-1:1)
>> H(end:-1:1, end:-1:1)
```

Column/row permutaitons (reindexing) does not change code.

General Hamming (higher SNR)

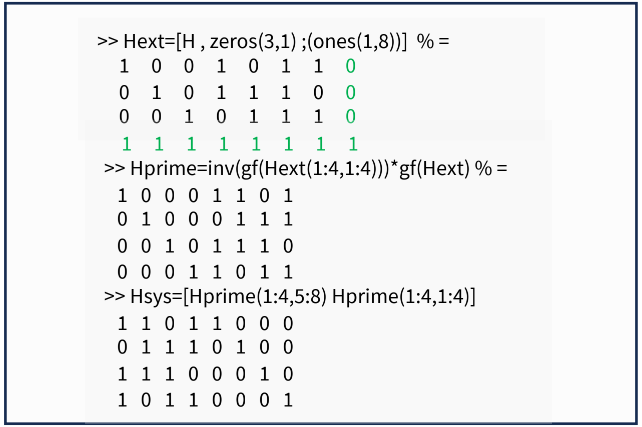
- **General Hamming** Codes – choose number of **parity bits** $p \geq 2$.
  - so $n = 2^p - 1$; $k = n - p$, $d_{free} = 3$, rate $r \to 1$ as $n \to \infty$.
  - Enumerate indices $i = 1, \ldots, 2^p - 1$ as binary p-digit values for H (rearrange to systematic):

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

  - The last p bits (last p columns) appear only once in $\boldsymbol{v} \cdot H^t = 0$ and sum other 1-positions' bits.
  - It is easily possible to add 3 H columns to zero, confirming $d_{free} = 3$.
  - Clever rearranging of H's columns can cause the syndromes 3-bit value to be the position of a single bit error (more than 1 bit error cannot be corrected).

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

- **Expanded Hamming** Codes (expansion applies all odd length linear-binary codes)
  - Expand codeword length by 1 redundant bit, so $n = 2^p$.
    - $k = n - p - 1$.
  - First add column of all zeros to (previous Hamming parity matrix) H
  - Then add row of all ones (overall parity check, which increases distance by 1 if all-zeros column was first added) $d_{free} = 4$.



```
>> Hext=[H , zeros(3,1) ;(ones(1,8))]  % =
    1    0    0    1    0    1    1    0
    0    1    0    1    1    1    0    0
    0    0    1    0    1    1    1    0
    1    1    1    1    1    1    1    1
 >> Hprime=inv(gf(Hext(1:4,1:4)))*gf(Hext) % =
   1  0  0  0  1  1  0  1
   0  1  0  0  0  1  1  1
   0  0  1  0  1  1  1  0
   0  0  0  1  1  0  1  1
 >> Hsys=[Hprime(1:4,5:8) Hprime(1:4,1:4)]
   1  1  0  1  1  0  0  0
   0  1  1  1  0  1  0  0
   1  1  1  0  0  0  1  0
   1  0  1  1  0  0  0  1
```

Use for SNRs > 3 dB

Hadamard Codes (low SNR)

- Hadamard is low-SNR binary code and has:
  - Large $d_{free} = n/2$,
  - Small rate $r \ll 1$
  - All codewords are mutually orthogonal (in $GF(2)$),
    - SO KIND OF LIKE BINARY ORTHOGONAL.
  - All codewords have weight $n/2$.
- Hadamard generator forms from $0:2^k - 1$ in binary
  - Each of its n rows/columns are orthogonal to one another in $GF(2)$.
  - All zeros is a codeword, but all other codewords have at least $n/2$ 1's.
  - For parity, note the k systematic columns are there, so group them.
    - The rest is then h^t for systematic parity matrix.
- Augmented Hadamard code:
  - Has $n = 2^m$; $k = m + 1$; $d_{free} = n/2$,
  - Takes only columns of $G(m+1)$ that start with 1.
  - Is dual code of Expanded Hamming with codeword length $n/2$.

There is a matlab Hadamard command that generates the unitary Walsh-Hadamard Transform matrix of +/- 1's. This is related and used in multiuser systems, but easier to create generator as shown above.

General Hadamard Code

$n = 2^k$; $k = \log_2 n$

$r = k/2^k$ for $d_{free} = n/2$

```
n=16;
k=log2(n);
Gtemp=dec2bin(0:2^k-1)';
   G=zeros(k,n);
 for i=1:k for j=1:n
   G(i,j)=bin2dec(Gtemp(i,j));
 end ; end
>> G % =
  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1

4 x 16 (n=16, k=4, dfree is 8)
```

```
gf(G)*gf(G)' % =
 0  0  0  0
 0  0  0  0
 0  0  0  0
 0  0  0  0
```

```
GA=G(:,8:16) =
  0   1   1   1   1   1   1   1   1
  1   0   0   0   0   1   1   1   1
  1   0   0   1   1   0   0   1   1
  1   0   1   0   1   0   1   0   1

4 x 8   (n=8, k=4, m=3), dfree is now 4
```

Matlab binary block codes

- **encode.m** – handles Hamming or general linear (binary).

  codeword = encode(inbits, n, k, 'hamming’)
      % don’t need generator nor parity matrices

  codeword = encode(inbits, n,k, ‘linear’, G)
      % if not Hamming, then input generator
      % can also have ‘cyclic’ for cyclic binary codes (eBCH)

- **decode.m** – handles Hamming or general linear (binary).

  msgbits = decode(y, n, k, 'hamming’)
      don’t need generator nor parity matrices

  msgbits = decode(y, n,k, ‘linear’, G)
      use this for Hadamard G or any other linear G

No time to present specific decoder simplifications for Hamming nor Hadamard – however, see L12 GRAND.

```
>> y = encode([ 1 0 1 1 0 1 0 0 0 1 1],15,11,'hamming') =
    0   1   0   1   1   0   1   1   0   1   0   0   0   1   1

>> error=[zeros(1,7) 1 zeros(1,7)];

>> decode(xor(y,error),15,11,'hamming')
                 1 0 1 1 0 1 0 0 0 1 1
```

These functions for small codes – could have long run time for arbitrary G , which may have to test all codewords.

Other Binary Block Codes

- Cylic Binary (BCH)
- Reed Muller
- Polar
- eBCH
- Golay
- "product codes" of the above

- See EE387
  - May have a little more on this in 379B
  - Product codes after midterm, L12.

(General) Linear Code Equivalence and Parity

- Code Equivalence
  - $G'(D) = \underbrace{A(D)}_{k \times k} \cdot G(D)$, $|A(D)| = 1$ (invertible)
  - $G'(D)$ and $G(D)$ generate the same codewords (sequences).
- Alternate code description is
  - $C[G] \triangleq \{v(D) \mid v(D) \cdot H^t(D) = 0 \,, v(D) \in F_r(D)\}$
- $H(D)$ is a generator for "dual code."
  - Every codeword in dual code is orthogonal to codeword in original code (G).
  - High-rate codes are often specified more compactly by $H(D)$.
- Complexity $\mu = \min_{\{G(D) \text{ for } C[G]\}} (v)$.
- When $\mu = \nu$, $G(D)$ is a **Minimal Encoder.**
- There is always a minimal encoder, and with feedback possible, a minimal systematic encoder. (See Appendix B – this is non-trivial and not covered.)

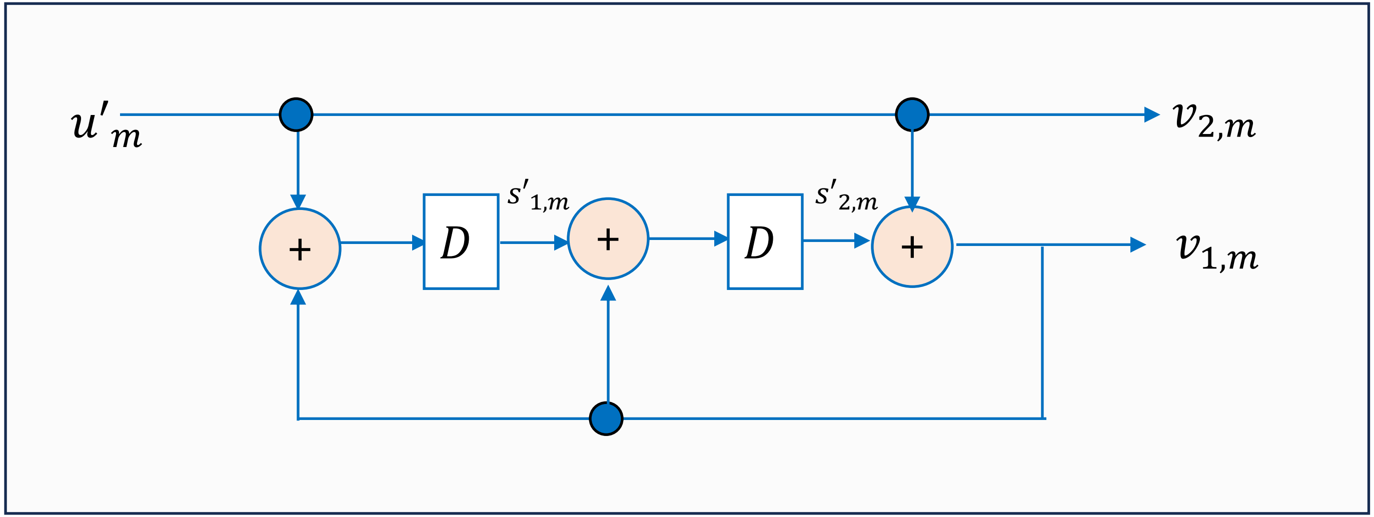
The codes we use here are always always be minimal

Sections 8.1.2 & B.7

4-state example (same code)

- Premultiply $G(D) = [1 + D + D^2 \quad 1 + D^2]$ by feedback $A(D) = \frac{1}{1+D+D^2}$ to get systematic equivalent:
  - $G_{sys} = \left[1 \quad \frac{1+D^2}{1+D+D^2}\right]$

- G_{sys} produces the same output bit sequences, but with different input-to-output mapping.
  - G_{sys} has a different trellis input-bit mapping, but otherwise has all the same paths (infinite length).
  - G_{sys} has the same free distance and the same number of states.



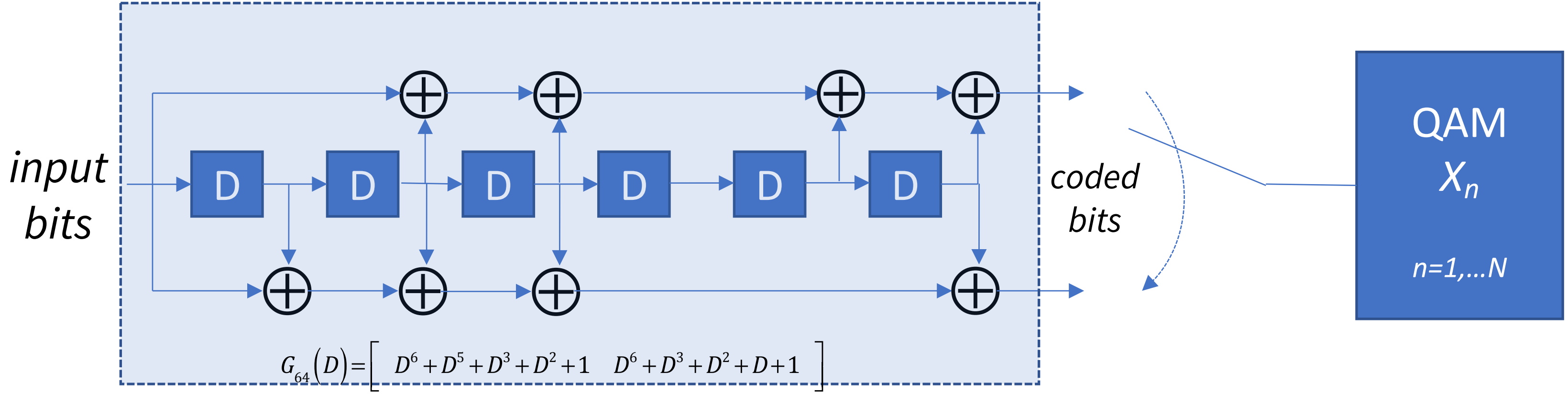
Puncturing:

- Uses same base code, but delete some encoder-output bits.
- Increases rate $r = {}^{k}/_{n} \rightarrow {}^{k}/_{n-i} \;\; i < n-k \;\in Z^{+}$.
- Simplifies encoder/decoder implementation (but changes codewords and can lower minimum distance).
- Example $r = {}^{1}/_{2} \rightarrow {}^{3}/_{4}$
  - 3 input bits: punctures 2 output bits from 6 output bits: 1 1 ~~0~~ 1 1 ~~0~~ 3 in / 4 out
  - Often just 1 1 0 1 1 0
- Example $r = {}^{1}/_{2} \rightarrow {}^{2}/_{3}$
  - 4 input bits punctures 2 bits from 8: 1 1 1 ~~0~~ 1 ~~0~~ 1 1 4 in/6 out

Regular/periodic puncturing retains linear code.

- If the pattern is regular (so occurs in same way over L subsymbol outputs), then
  - Define a puncturing matrix $G_{punc} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ for above 2/3 example ; then $G(D) \rightarrow \begin{bmatrix} G(D) & 0 & 0 \\ 0 & G(D) & 0 \\ 0 & 0 & G(D) \end{bmatrix} \cdot G_{punc}$.

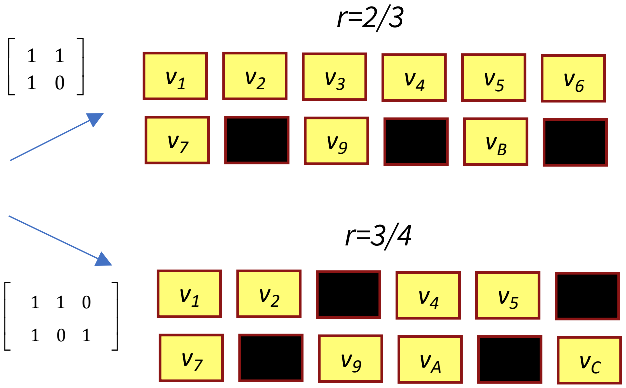
Wi-Fi Puncturing – IEEE 802.11 standards (a,g,ac,ax,be)



$$G_{64}(D)=\left[\begin{array}{cc} D^6+D^5+D^3+D^2+1 & D^6+D^3+D^2+D+1 \end{array}\right]$$

block of 6 input bits

block of 12 coded bits



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

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

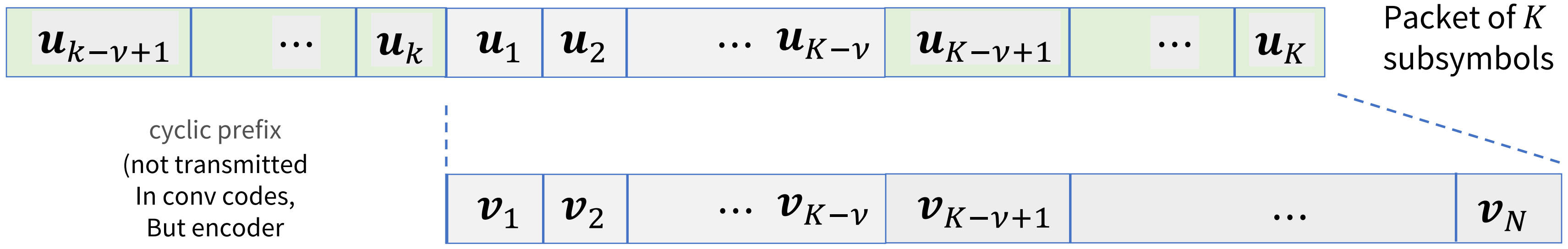
- Mapper can pick $|C|$ = 4, 16, 64, 256 QAM ("MCS" mod-code-scheme),
- And $r = {}^1\!/_2\,, {}^2\!/_3\,, {}^3\!/_4$.
- 1024 QAM and 4096 QAM may be allowed in some advanced WiFi

Puncturing G(D) example

- $G_{punc}(D) = G_{punc}$; max of one 1 in each row/col, rest are 0's.

$$G_{3/4}(D) = G_{64}(D) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Tail Biting ~ converts to block code



$\boldsymbol{u}_{k-\nu+1}$ $\cdots$ $\boldsymbol{u}_k$ $\boldsymbol{u}_1$ $\boldsymbol{u}_2$ $\cdots$ $\boldsymbol{u}_{K-\nu}$ $\boldsymbol{u}_{K-\nu+1}$ $\cdots$ $\boldsymbol{u}_K$

Packet of K subsymbols

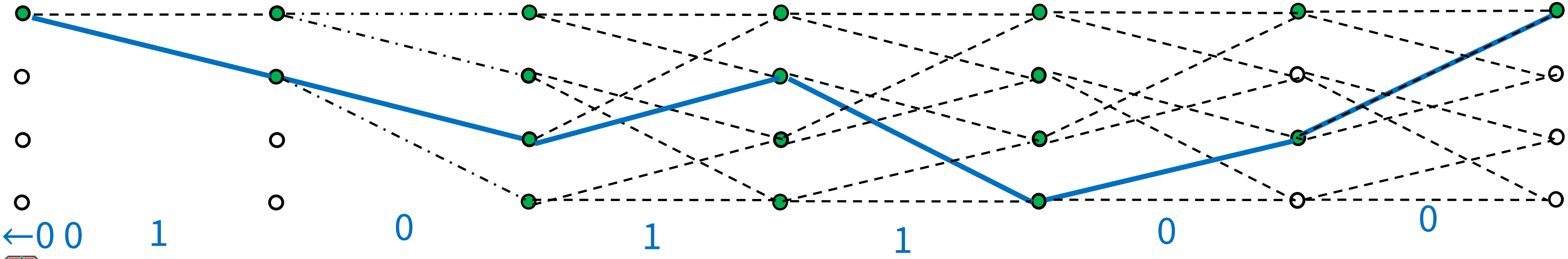
cyclic prefix
(not transmitted
In conv codes,
But encoder
processes to
set the state)

$\boldsymbol{v}_1$ $\boldsymbol{v}_2$ $\cdots$ $\boldsymbol{v}_{K-\nu}$ $\boldsymbol{v}_{K-\nu+1}$ $\cdots$ $\boldsymbol{v}_N$

Packet starts/finishes in same state

Output packet
$N = {}^{K}/_{r}$ subsymbols

- With 2^ν possible states, the current state is a function only of most recent ν input bits.
  - This is a mild nonlinearity in the encoding process that becomes neglible with large K , but which can reduce distance (better to terminate).
- The last ν bits repeat.
  - The packet must be at least ν bits long, but in practice this should be small percentage of k (8 input bits below, but repeat the last 2. These are 00 in example below and not shown; they force a start in state 0). The rate reduction factor is $N/(N+\nu)$.



Binary Code Use

Section 2.2

Matlab Trellis and Encoding Functions

- Convert $G(D) = \left[\underbrace{D^2+D+1}_{111} \quad \underbrace{D^2+1}_{101}\right] = \left[\underbrace{7}_{octal} \quad \underbrace{5}_{octal}\right]$ to design. (Code tables appear in octal later.)

```
>> t=poly2trellis( 3 , [7 5] )

t =  struct with fields:
     numInputSymbols: 2
     numOutputSymbols: 4
     numStates: 4
     nextStates: [4 × 2 double]
     outputs: [4 × 2 double]
```

$\underbrace{3}_{\nu+1}$, $\underbrace{[7\ 5]}_{G(D)}$

```
>> t.nextStates =
   0   2
   0   2
   1   3
   1   3
>> t.outputs =
   0   3
   3   0
   2   1
   1   2
```

Matlab's trellis is equivalent and:
- has same set of paths
- but uses different state-labels.

(We'll translate shortly.)

- Encode bit stream

```
>> convenc([0 0 0 0 1 1],t)   =   00   00   00   00   11   0 1
```

$\underbrace{[0\ 0\ 0\ 0\ 1\ 1]}_{input\ bits}$ $\underbrace{00\ 00\ 00\ 00\ 11\ 01}_{output\ bits}$

convenc(bits, trellis, **init-state**)
(default is zero)

- This is same code as earlier on slide L7:7.
- ploy2trellis works with any (nofeedback) $G(D)$
  - Constraint length+1 → value for each row of $k \times n$ $G(D)$
  - t.numStates = 2^ν
  - t.numinputSymbols = 2^k
  - t.numOutputSymbols = 2^n

TRELLIS = poly2trellis(nu+1, Gnum, Gdenom)
  is the same as the first syntax, but for a feedback convolutional encoder.

  - Gdenom is a 1-by-k vector of octal numbers specifying the
    feedback connection for each of the k inputs. It will be GCM of denoms in each row.
    only works when k=1 unfortunately (matlab bug).

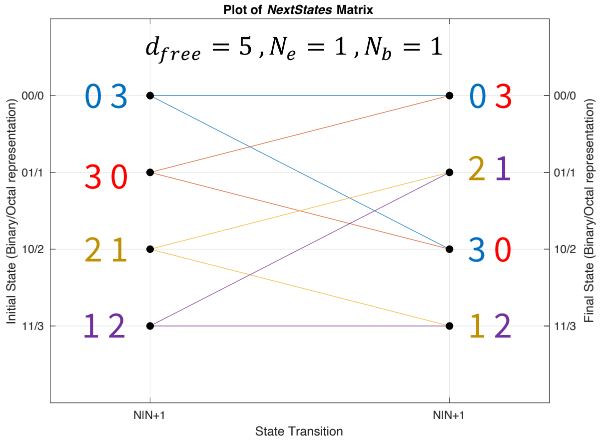
Sec 8.1.5, See PS4.1

Translating Trellises

(see https://www.mathworks.com/help/comm/ref/convenc.html)

- Trellis program plotnextstates(t.nextStates)
- I superimposed t.outputs on figure.
- Looks reversed w.r.t. Slide L7:7 ??
  - Matlab reversed the state-label bits

```
>> t.nextStates =
   0   2
   0   2
   1   3
   1   3
>> t.outputs =
   0   3
   3   0
   2   1
   1   2
```



```
numbits=2;
for i=1:4    for j=1:2
nst(i,j)=uint16( bin2dec( fliplr( dec2bin …
( oct2dec(t.nextStates(i,j)),numbits) ) ) );
end end
```

```
>>cst=bin2dec(fliplr…
(dec2base(0:3,2)))=
    0
    2
    1
    3
```

```
>> nst(cst+1,:)
=
    0    1
    2    3
    0    1
    2    3
```

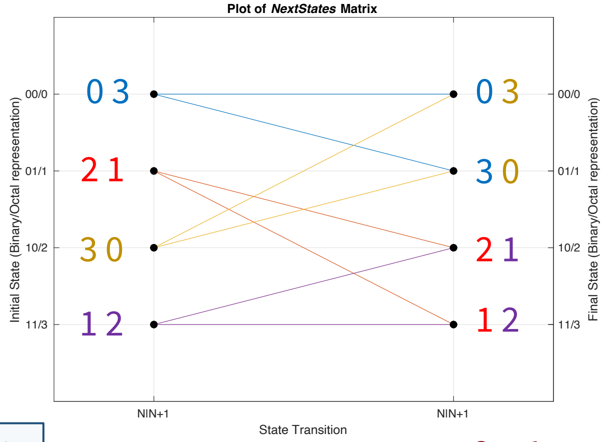
```
 >> t.outputs(cst+1,:)
 =
    0    3
    2    1
    3    0
    1    2
```

```
>> t2=t; t2.nextStates=nst(cst+1,:);
>> t2.nextStates =
    0    1
    2    3
    0    1
    2    3
```

```
>> t2.outputs=t.outputs(cst+1,:);
>> t2.outputs =
    0    3
    2    1
    3    0
    1    2
```

plotnextstates(**t2**.nextStates)

Now it's same as L7:7



```
>> convenc([0 0 0 0 1 1],t2) =    00    00    00    00    11    01
```

Sec 8.1.5

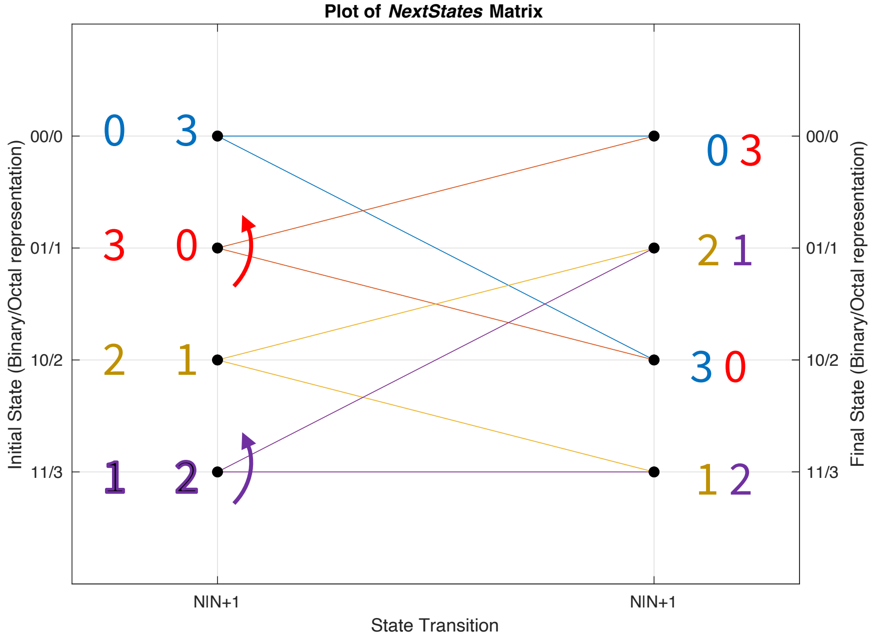
With Feedback

- Matlab’s nextStates & outputs don’t always obey inputs clockwise 0 to 11…1 on branches.

```
>> t3=poly2trellis(3,[7 5],7) =
 struct with fields:
   numInputSymbols: 2
   numOutputSymbols: 4
       numStates: 4

>> t3.nextStates =
   0   2
   2   0  → counter clockwise
   3   1
   1   3  → counter clockwise
>> t3.outputs =
   0   3
   0   3 , so outputs reversed
   1   2
   1   2 , so outputs reversed
>> distspec(t3,1) =
  dfree: 5
  weight: 3 (input bit errors, Nb)
   event: 1
```

```
>> t.nextStates =
    0   2
    0   2
    1   3
    1   3
>> t.outputs =
    0   3
    3   0
    2   1
    1   2
```



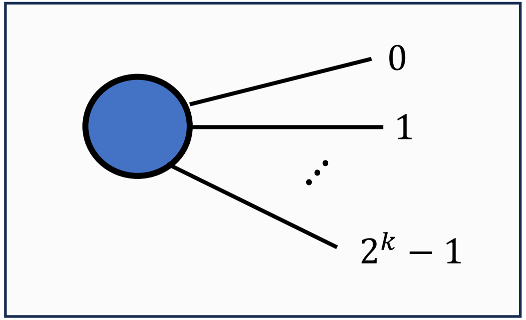
- Trellis is same as non-feedback (non-systematic) code, with a lot of labelling care!
- Mapping to input bits is **different**:

```
>> convenc([0 0 0 0 1 1],t3,0)=   00   00   00   00   11   10
```

Sec 8.1.5, See PS4.2

A rule to avoid (e.g. Matlab's) ugly trellises

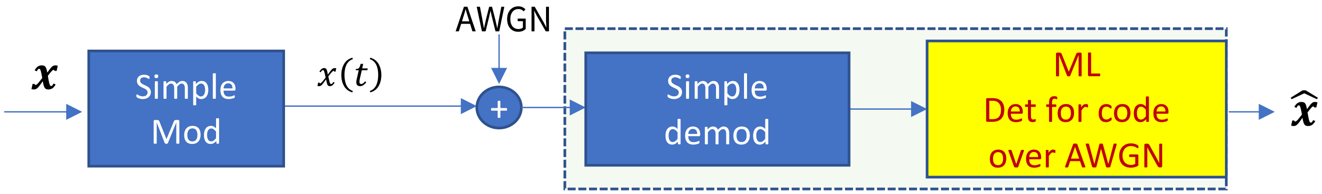
- Always use the nonsystematic (minimal) encoder, like $G(D) = \left[\underbrace{1+D+D^2}_{111} \quad \underbrace{1+D^2}_{101}\right] = \left[\underbrace{7}_{octal} \quad \underbrace{5}_{octal}\right]$.
  - See Slide L7:18.
- These have clockwise input-bit branch assignments with Matlab's nextStates, so 0,1,2,3, … 2^k.



- The systematic (minimal) encoder, like $G_{sys}(D) = \left[1 \quad \frac{1+D^2}{1+D+D^2}\right]$, produces the same code.
  - This has different input-sequence assignments to codewords, but all inputs map 1-to-1 to one another anyway.
- Take the inputs $u(D)$ corresponding to any $G(D)$ path and transform them by $u'(D) = \frac{u(D)}{1+D+D^2}$.
- $u'(D)$ into $G(D)$ produces the same output as $u(D)$ into $G_{sys}(D)$. (so map 1-to-1 on side $u'(D)$ <-> $u(D)$).
  - u = conv(gf([1 1 1], gf(u')) ; u' = deconv(gf([1 1 1], gf(u))

Soft Decoder – decode the symbol

$$\underbrace{p_{y/x} = (2\pi\sigma^2)^{-N/2} \cdot e^{-\|\boldsymbol{y}-\boldsymbol{x}\|^2/2\sigma^2}}$$



AWGN

$\boldsymbol{x}$

Simple Mod

$x(t)$

$+$

Simple demod

ML Det for code over AWGN

$\hat{\boldsymbol{x}}$

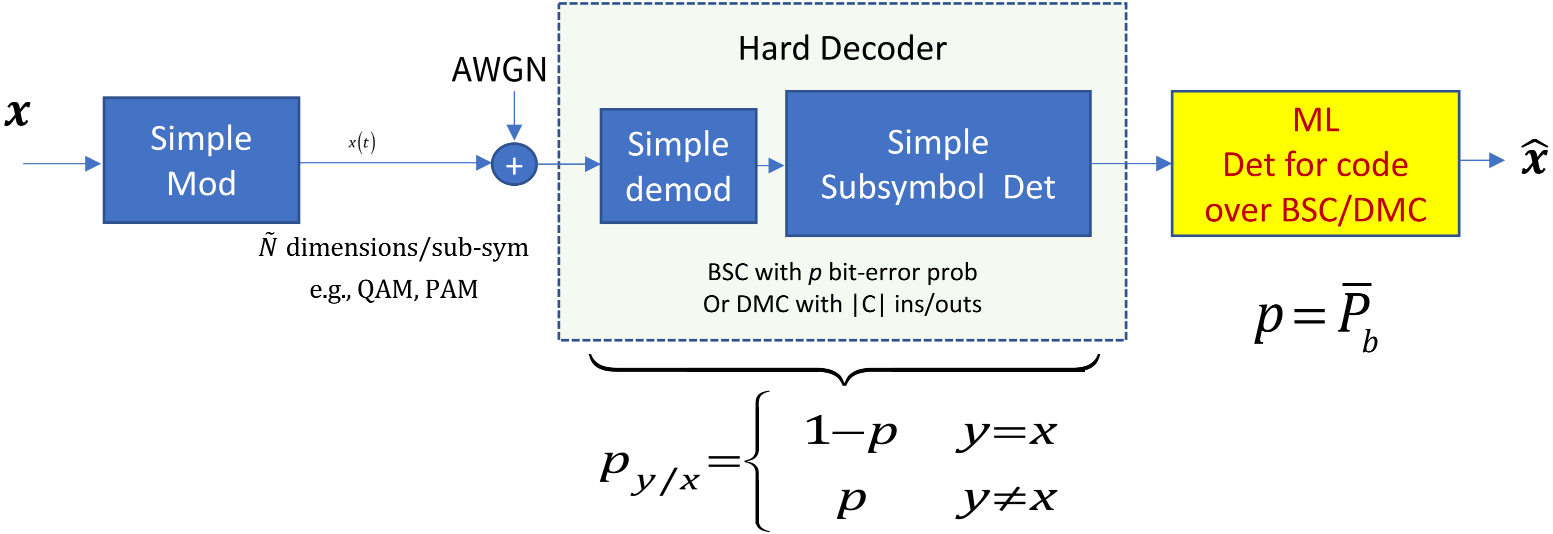
$\tilde{N}$ dimensions/sub-sym
e.g., QAM, PAM

$$\boxed{\hat{\boldsymbol{x}} = \arg\left\{\min_{\hat{\boldsymbol{x}}} \sum_{n=1}^{\bar{N}} \|\tilde{\boldsymbol{y}}_n - \hat{\tilde{\boldsymbol{x}}}_n\|^2\right\}}$$

Finite
Real-Time
Complexity

- The demodulator samples ($\in \mathbb{C}$) pass to the detector for comparison of codewords (subsymbol sequences).
- The $\boldsymbol{y}$ information is “soft” in that it is not pre-quantized into a decision (or at least not to $|C|$ **subsymbol values).**
- Deployed systems often have ADC on y_n ; quantize $\frac{d_{min}(|C|)}{\sigma_q}$ = 4^3 ; i.e., 3 bits cover intra-point distance.
  - This 3-bit quantization **of dmin** limits decoder loss (w.r.t. infinite precision) to .25 dB distortion (one more bit reduces to .06 dB distortion).
  - Same rule applies per dimension for both ADCs if receiver is in quadrature.
  - Total ADC bits will then be these 3, plus $\bar{b}$, plus 1-2 bits for peak-to-average (analog coverage), so $b_{ADC} = \bar{b}$ +4, or possibly $\bar{b}$ +5 .

Hard decoder – decode the bit sequence



$$p = \bar{P}_b$$

$$p_{y/x} = \begin{cases} 1-p & y=x \\ p & y \neq x \end{cases}$$

- Subsymbols are decoded independently – e.g., a “hard” decision.
- The remaining channel is a DMC (most often a BSC) model, to which an outer binary code may also be applied.
- The BEC with the “erasure” output is a first step from hard to soft.

AWGN Error Probability for Conv Codes

- AWGN $\bar{P}_e = \bar{N}_e \cdot Q\left(\frac{d_{min}}{2\sigma}\right) = \bar{N}_e \cdot Q\left(\sqrt{d_{free}\cdot\frac{\mathcal{E}_x}{\sigma^2}}\right) = \bar{N}_e \cdot Q\left(\sqrt{d_{free}\cdot\frac{k}{n}\cdot SNR}\right)$
  - Because $d_{min}=\sqrt{d_{free}\cdot 4\cdot\mathcal{E}_x}$

energy-spread reduces energy/subsym
(assumes $\frac{1}{T'}$ can increase, so no filter on AWGN)

- AWGN $\bar{P}_b = \frac{N_b}{b}\cdot Q\left(\sqrt{d_{free}\cdot r\cdot SNR}\right)$
  - Where $N_b=\sum_{i=1}^{\infty} i\cdot N\left(i,d_{free}\right)$ and $N(i,d)$ for conv code is the number of i-input-bit error events with distance d .
  - Finding N_b can require exhaustive search in general, but Section 7.2 (Lecture 8) show how to compute $N(i,d)$ for CC.
  - Yes, it is equal to Chapter 1's $\sum_{i=1}^{\infty} p_x(i)\cdot n_b(i)$, which is actually harder to compute.
- BC **coding gain** $\gamma = 10\cdot\log_{10}\left(r\cdot d_{free}\right)$ (for AWGN with binary subsymbols ..) and **energy/bit** $\bar{\mathcal{E}}_b$.

HAZARD WARNING☠ – BINARY CODING THEORIST'S FALLACY – assumes "free bandwidth"

Binary-code fair comparison: hold 2 of 3 $\{\bar{b} \quad \bar{\mathcal{E}}_x \quad \bar{P}_e\}$ fixed and compare 3rd ;

But $N_{coded} = \frac{1}{r}\cdot N_{uncoded}$ so then BOTH $\bar{\mathcal{E}}_x$ & $\bar{b}$ decrease for coded w.r.t uncoded (~ holding power & rate constant), not fair.

$\bar{b}_{coded} = r\cdot\bar{b}_{uncoded} \quad \bar{\mathcal{E}}_{x,coded} = r\cdot\bar{\mathcal{E}}_{x,coded}$; So $\mathcal{E}_b = \frac{\bar{\mathcal{E}}_x}{\bar{b}}$ is the same, **BUT** $W\cdot T \rightarrow {}^{W\cdot T}/_{r}$

So, either the coded design increased bandwidth (may not be possible) or otherwise reduced rate;
adding a code to reduce rate is somewhat antithetical to Shannon if $R < C$. Increasing W is "cheating."

BSC Error Probability

- BSC $\bar{P}_e = \bar{N}_e \cdot [4p(1-p)]^{\left\lceil \frac{d_{free}}{2} \right\rceil}$
- BSC $\bar{P}_b = \frac{N_b}{b} \cdot [4p(1-p)]^{\left\lceil \frac{d_{free}}{2} \right\rceil}$

- Chapter 1’s B-Bound can be used to show that this is roughly 3dB inferior to soft decoding (AWGN).

- Fair-comparison discussion is for AWGN.
  - Strictly speaking with BSC, data rate must reduce to improve with codes.
  - From BSC capacity, $r \le \underbrace{1 + p \cdot \log_2 p + (1-p) \cdot \log_2(1-p)}_{\mathcal{C}} \le 1$ for reliable transmission with a code $0 < p < \frac{1}{2}$.

Coding Tables –best known rate ½ conv codes

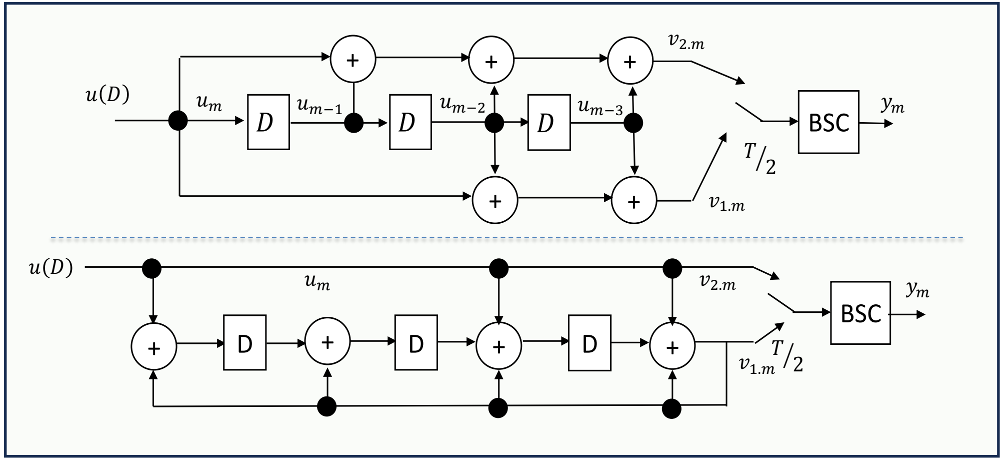
- Section 8.2 – Conv Code Tables see the octal entries, chap 8 [6])

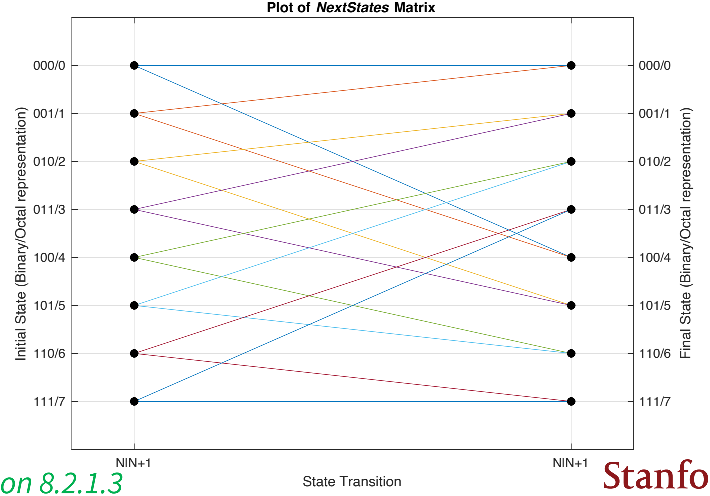
2^{ν}	$g_{11}(D)$	$g_{12}(D)$	d_{free}	γ	(dB)	N_e	N_1	N_2	N_b	L_D
4	7	5	5	2.5	3.98	1	2	4	1	3
8	17	13	6	3	4.77	1	3	5	2	5
16	23	35	7	3.5	5.44	2	3	4	4	8
(2G) 16	31	33	7	3.5	5.44	2	4	6	4	7
32	77	51	8	4	6.02	2	3	8	4	8
64	163	135	10	5	6.99	12	0	53	46	16
(802.11a) 64	155	117	10	5	6.99	11	0	38	36	16
(802.11b) 64	133	175	9	4.5	6.53	1	6	11	3	9
128	323	275	10	5	6.99	1	6	13	6	14
256	457	755	12	6	7.78	10	9	30	40	18
(3G) 256	657	435	12	6	7.78	11	0	50	33	16
512	1337	1475	12	6	7.78	1	8	8	2	11
1024	2457	2355	14	7	8.45	19	0	80	82	22
2048	6133	5745	14	7	8.45	1	10	25	4	19
4096	17663	11271	15	7.5	8.75	2	10	29	6	18
8192	26651	36477	16	8	9.0	5	15	21	26	28
16384	46253	77361	17	8.5	9.29	3	16	44	17	27
32768	114727	176121	18	9	9.54	5	15	45	26	37
65536	330747	207225	19	9.5	9.78	9	16	48	55	33
131072	507517	654315	20	10	10	6	31	58	30	27

Table 8.1: Rate 1/2 Maximum Free Distance Codes

L_D = length of Min-dist event

```
>> t8=poly2trellis(4,[17  13]) =
    numInputSymbols: 2
   numOutputSymbols: 4
          numStates: 8
         nextStates: [8 × 2 double]
            outputs: [8 × 2 double]
>> plotnextstates(t8.nextStates)
```





Section 8.2.1.3

Best rate-1/3 convolutional codes

- Codes listed for other rates, example 1/3 here, see Sec 8.2 for ¼, 2/3, ¾,

2^ν	$g_{11}(D)$	$g_{12}(D)$	$g_{13}(D)$	$g_{14}(D)$	d_{free}	γ	(dB)	N_e	N_1	N_2	N_b	L_D
4	7	7	7	5	10	2.5	3.98	1	1	1	2	4
8	17	15	13	13	13	3.25	5.12	2	1	0	4	6
16	37	35	33	25	16	4	6.02	4	0	2	8	7
32	73	65	57	47	18	4.5	6.53	3	0	5	6	8
64	163	147	135	135	20	5	6.99	10	0	0	37	16
128	367	323	275	271	22	5.5	7.40	1	4	3	2	9
256	751	575	633	627	24	6.0	7.78	1	3	4	2	10
512	0671	1755	1353	1047	26	6.5	8.13	3	0	4	6	12
1024	3321	2365	3643	2277	28	7.0	8.45	4	0	5	9	16
2048	7221	7745	5223	6277	30	7.5	8.75	4	0	4	9	15
4096	15531	17435	05133	17627	32	8	9.03	4	3	6	13	17
8192	23551	25075	26713	37467	34	8.5	9.29	1	0	11	3	18
16384	66371	50575	56533	51447	37	9.25	9.66	3	5	6	7	19
32768	176151	123175	135233	156627	39	9.75	9.89	5	7	10	17	21
65536	247631	264335	235433	311727	41	10.25	10.1	3	7	7	7	20

- Code complexity measure $N_D = \underbrace{2^\nu}_{states} \cdot \left(\underbrace{2^k}_{adds} + \underbrace{2^k - 1}_{compares} \right)$

Design Example

- An AWGN has SNR = 5 dB.

- The uncoded ($M = 2$) error rate is $P_e = Q\left(10^{5/20}\right) = .0377\ \ (not\ very\ good)$.

- A better design uses best 64-state rate $r =$ ½ code, so bandwidth expands by 2x.
  - The gain is 7 dB.
  - New $P_e = Q\left(10^{(5+7)/20}\right)$= 3.4303e-05 (better, see Slide L7:33's table for this code).

- To get $P_e \approx 10^{-6}$?
  - Need 8.5 dB of coding gain with rate ½ , so use this table's 1024-state code
  - $P_e = Q\left(10^{(5+8.5)/20}\right) \approx 10^{-6}$

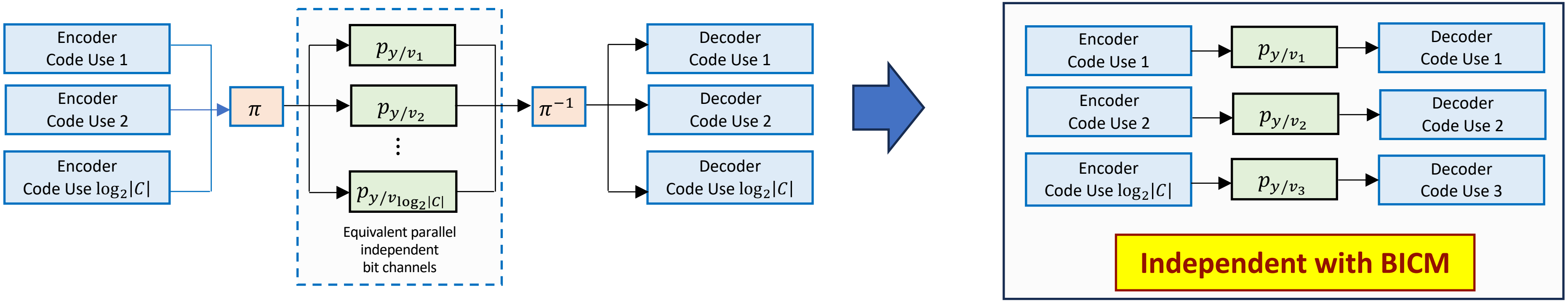
- Encoder is $G(D)$ = $\left[\underbrace{1 + D + D^2 + D^3 + D^5 + D^8 + D^{10}}_{2457} \quad \underbrace{1 + D^2 + D^3 + D^5 + D^6 + D^7 + D^{10}}_{2355}\right]$

1024 is a lot of states: larger distances may have large N_i that increase P_e.
Design instead should use better (not CC) code (see Lectures 9-10).
The 7 dB and 8.5 dB here often reduce in practice to about 5.5-6.0 dB, because of large N_i.

Mappings to M'ary Constellations: BICM

Sections 2.2, 8.1.7

BICM Basic concept



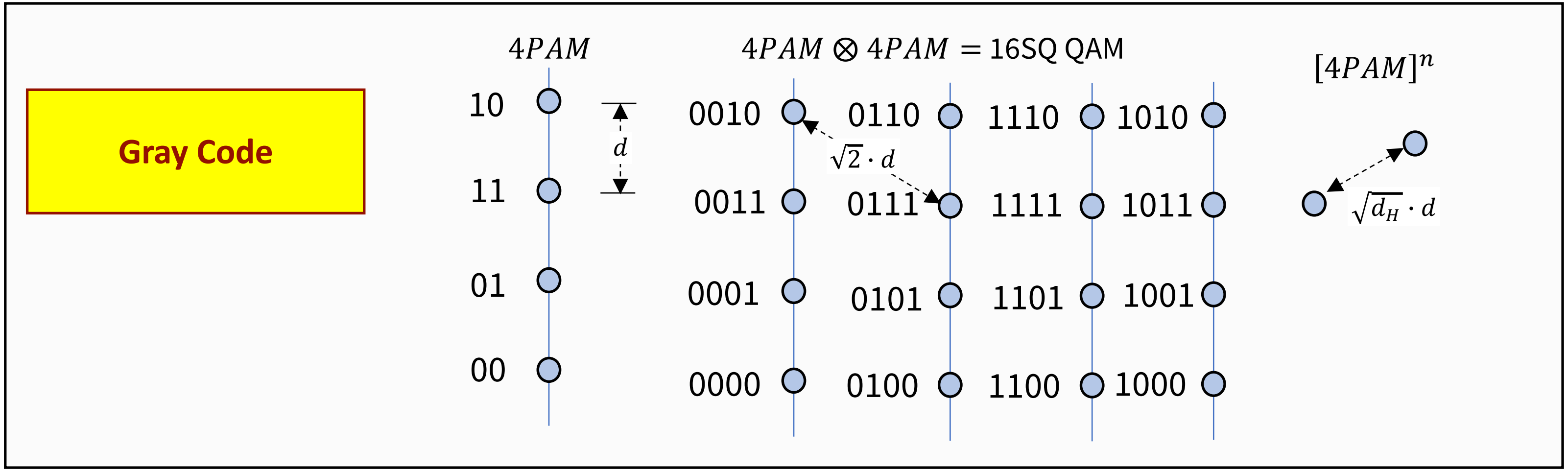
- The **interleaver** π reorders adjacent bits, and the **deinterleaver** π^{-1}causes $p_{y/[v_i,v_{i+L-1}]} = p_{y/v_i} \ldots p_{y/v_{i+L-1}}$.
  - Deinterleaving restores the original order but spreads a large channel-error/noise event over several codes.
  - L is the interleaver's "**depth**"– L9 has more on depth (Section 8.3).

- Each code sees an independent channel – so each is like a BSC or AWGN.

- EVEN WHEN AWGN and the SNR supports M-ary PAM (or SQ QAM) with $M > 2$ (4).
  - Without interleaving, a single large noise could cause multiple bit errors in presumably a single applied code.

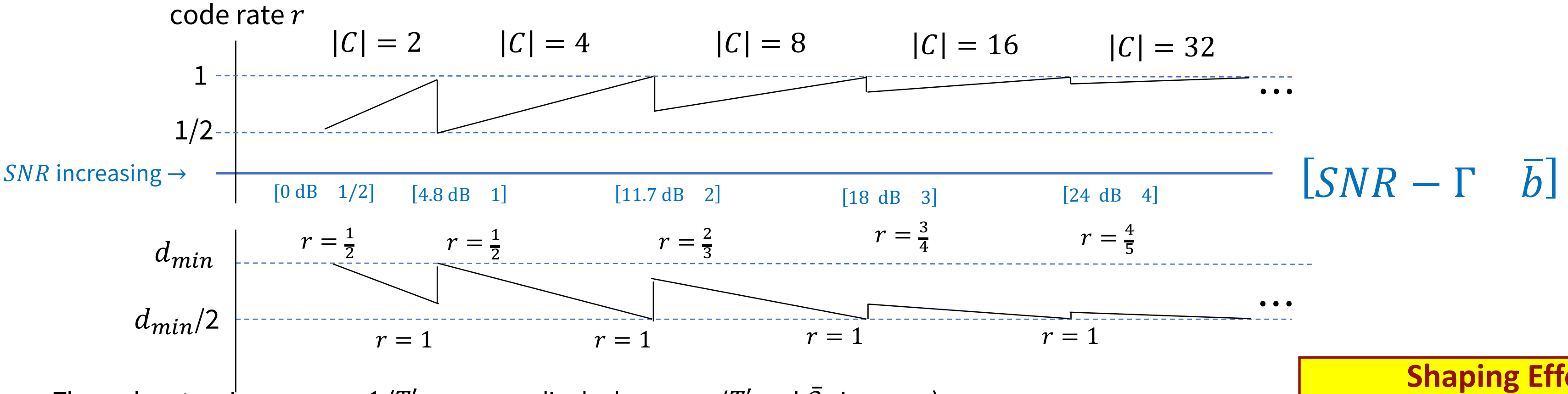
Gray Mapping and distance preservation

- Gray Coding (almost) **maintains coding gain** γ with (one-dimensional) $|C| = 2^{\bar{b}+\bar{\rho}} > 2$.
- Coded M-ary retains $d_{min} \geq 4 \cdot d_{free} \cdot \bar{\mathcal{E}}_x$ for M-ary SQ QAM.
- This applies well to PAM, or SQ-QAM, (in effect Cartesian product of 2 PAMs) and Gray Code.



- So, with Gray Code, $\bar{P}_b = \frac{N_b}{b} \cdot Q\left(\sqrt{\frac{3}{|C|^2-1} \cdot \gamma \cdot SNR}\right)$ where again $\gamma = r \cdot d_{free}$.
  - Exact ONLY IF 1-dimensional $|C|$ remains the same for coded and uncoded, but what about $\frac{1}{T\prime} \rightarrow \frac{1}{r \cdot T\prime}$??

M’ary PAM: approx constant-Γ puncturing with binary code



Shaping Effects
Reduce binary-code map’s
gain for larger
constellations

- The code rate r increases as $1/T'$ correspondingly decreases (T' and $\bar{\mathcal{E}}_x$ increase).
  - Puncturing carefully increases r with SNR until PAM constellation-size can double, which allows 1/T’ to reduce by $\frac{b}{b+1}$.
  - E.g., smooth transition @ 8-16 PAM has rate 3/4, then increasing from ¾ to 1 with puncturing until 32 PAM.
- Careful puncturing attempts to hold constant code gap.
  - d_{free}=∞ for Γ =0 dB gap, but also $|C| = \infty$, so theoretically must work with some good codes that look similar.
  - For the 64-state Wi-Fi code, γ=7 dB, but $\gamma_s \to 1.53$ dB for large $|C|$, and this reduces the 7 dB gradually. For this code the gap would be, at $P_e = 10^{-6}$, 8.8-7+1.5 or 3.3 dB, leaving γ_c=5.5 dB for the larger constellations.
    - Even with reasonable puncturing, this code eventually looses gain with large $|C|$, so has increasing gap (and thus needs more than 6 dB/bit-dimension to increase $|C|$, but they use it anyway).
    - There are larger-N binary block codes (LDPC, product) that offer more continuous puncturing options so the d_{min}choices (w.r.t r) help offset the constellation-increase.
  - In reality, with many nearest neighbors with BICM, puncturing is “about as good as it gets” with binary codes that ignore the constellation.
  - Iterative decoding (see L9) between constellation and binary-code can restore the constant gap at its best value (so account for the constellation).
  - 64-state $r = 1/2$ ′s 7 dB is really for $b < 1$ where shaping improvement is negligible. It can be restored with shaping codes (see Section 8.5, not taught).
  - There are “trellis codes” that well-hold constant gap, but their best gaps are below those of the BICM with convolutional codes.
- If there was one giant ML decoder for the aggregate of $\log_2 |C|$ codes (large N), the interleaving is unnecessary.
  - This aggregate code is NOT just the single binary code.

Mapping by set-partitioning (Trellis Codes)

- Trellis Codes (Ungerboeck, IBM) were popular, and a major intermediate step for M'ary in 1990's.
- They also have simpler ML decoders.
- TCs have coding gain limits below best codes (roughly 2 dB less than Gray codes with good binary codes).
- See Appendix B.

STANFORD

End Lecture 7