Supplementary Lecture 3

Line Codes with Memory

January 13, 2026

JOHN M. CIOFFI

Hitachi Professor Emeritus (recalled) of Engineering

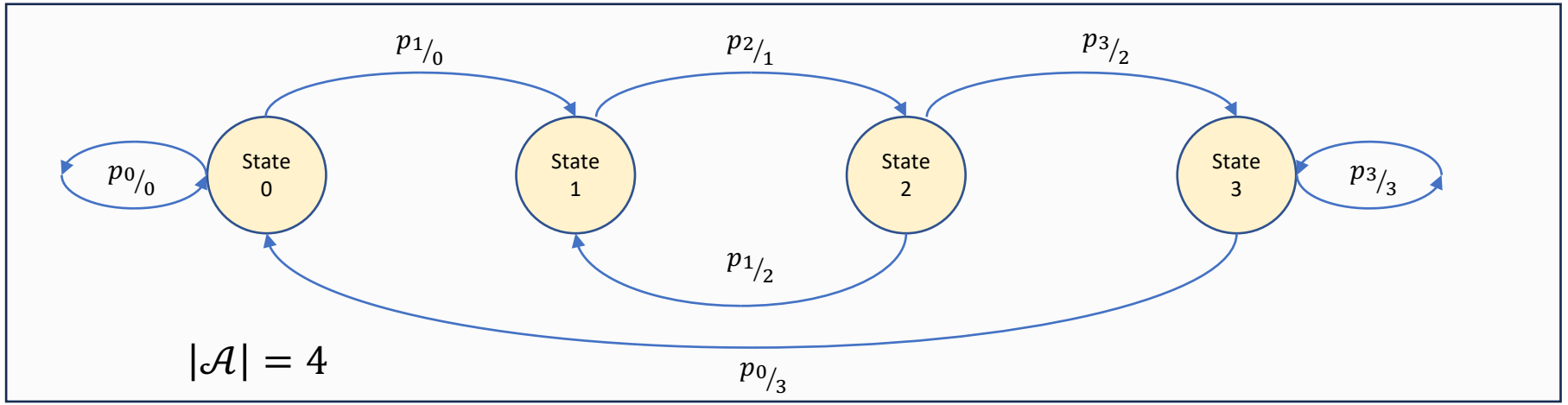
Instructor EE379A – Winter 2026

Announcements & Agenda

- See Appendix A on Markov Processes as well as Section 1.3.3.5
- Line Code Examples
  - Miller
  - Modified Miller
  - Run-Length-Limited Codes
- Near-Field Communication

Markov Process ~ State Machine or Graph

- Probability depends only on last step $p_{k/k-1,k-2,\ldots,-\infty} = p_{k/k-1}$.
- **Transition probability matrix P** and **state-machine** descriptions, with $|\mathcal{A}|$ = # of states = matrix size.



$$P = \begin{bmatrix} p_{3/3} & p_{3/2} & 0 & 0 \\ 0 & 0 & p_{2/1} & 0 \\ 0 & p_{1/2} & 0 & p_{1/0} \\ p_{0/3} & 0 & 0 & p_{0/0} \end{bmatrix}$$

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

Adjaceny matrix A

- **Irreducible,** any state can eventually get to any other: $P \cdot \boldsymbol{\pi} = \boldsymbol{\pi}$ (eigen value 1, stationary distribution)

Some Markov basics

- When transitions/edges are symbol values, then maximum rate is $\log_M\left(\max_{\lambda}(eig(A))\right)$, usually $M = 2$.
  - See Appendix A
- So, there is an encoder that maps input symbols to output symbols with some loss of forbidden string.
  - This leads to variable rate (but there is usually an average data rate with some ”jitter” about it.
  - There is also correlation between successive symbols.
- Matrix X contains the symbol transmitted for each $\left({}^{j}/_{i}\right)$ in P (or A).

$$X = \begin{bmatrix} +1 & +1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 \\ +1 & 0 & 0 & +1 \end{bmatrix}$$

- Matrices P^k and A^k correspond to k successive transitions.
- Matrix $X^{(k)}$ corresponds to a string of symbols on the corresponding transitions of P^k and A^k.

General spectrum calculation

- Encoder produces output bits at higher clock 1/T'
- The autocorrelation for lag k that corresponds to P^k is

$$x(t) = A_0 \cdot \sum_{n=1}^{N} x'_n \cdot \phi\left(\frac{t}{T'};\right)$$

$$r_{xx,k-1} = \sum_i \sum_j \pi_j \cdot P_{ji}^k \cdot \left(X_{ji}^{(k)} \cdot X_{ji}^{(1)*}\right) \quad \forall\, k = 1,,,,,\infty$$

- The prob-transition matrix has singular value decomposition
  - Appendices C and D.

$$P = F \cdot \Lambda \cdot M^*, \text{ then } P^k = F \cdot \Lambda^n \cdot M^*$$
$$P^k = \sum_n \lambda_n^k \cdot \boldsymbol{f}_n \cdot \boldsymbol{m}_n^* \ ,$$

- So then (algebra)

$$r_{xx,k-1} = \sum_n c_{\lambda_n} \cdot \lambda_n^k$$

$$c_{\lambda_n} \triangleq \sum_{i,j} \pi_j \cdot \boldsymbol{f}_{i,n} \cdot \boldsymbol{m}_{n,j}^* \cdot X_{ji}^{(k)} \cdot X_{ji}^{(1)*}$$

- The power spectrum is

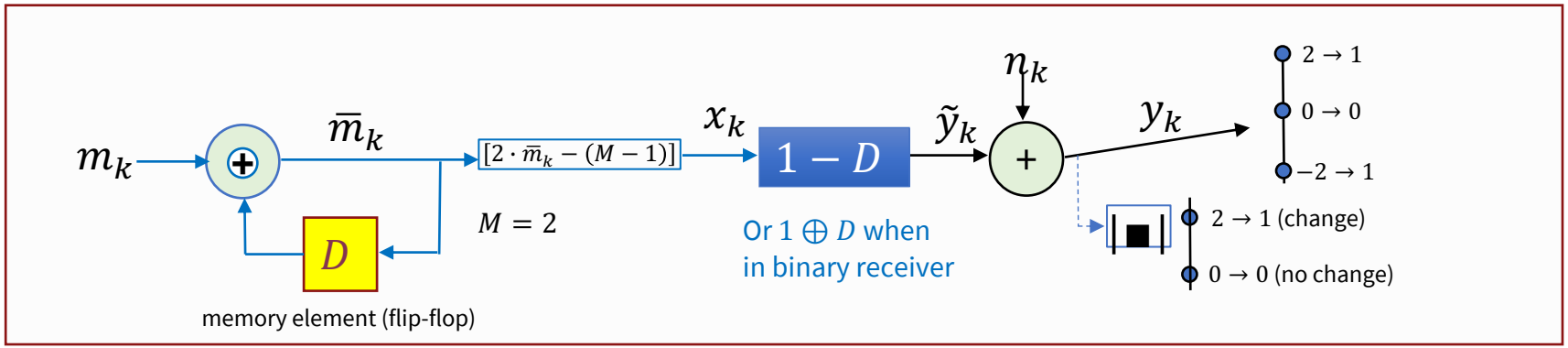
$$S_x(e^{-\jmath 2\pi \cdot f \cdot kT}) = \sum_{k=-\infty}^{\infty} r_{xx,k} \cdot e^{-\jmath 2\pi \cdot f \cdot kT} = -\sum_{n=1}^{N} c_{\lambda_n} + 2 \cdot \sum_{n=1}^{N} \frac{c_{\lambda_n}}{1 - \lambda_n \cdot e^{-\jmath 2\pi \cdot f \cdot T}}$$

Line Code Examples

Section 1.3.3

Line Codes with Memory

- Simplest is the **differential encoder**.



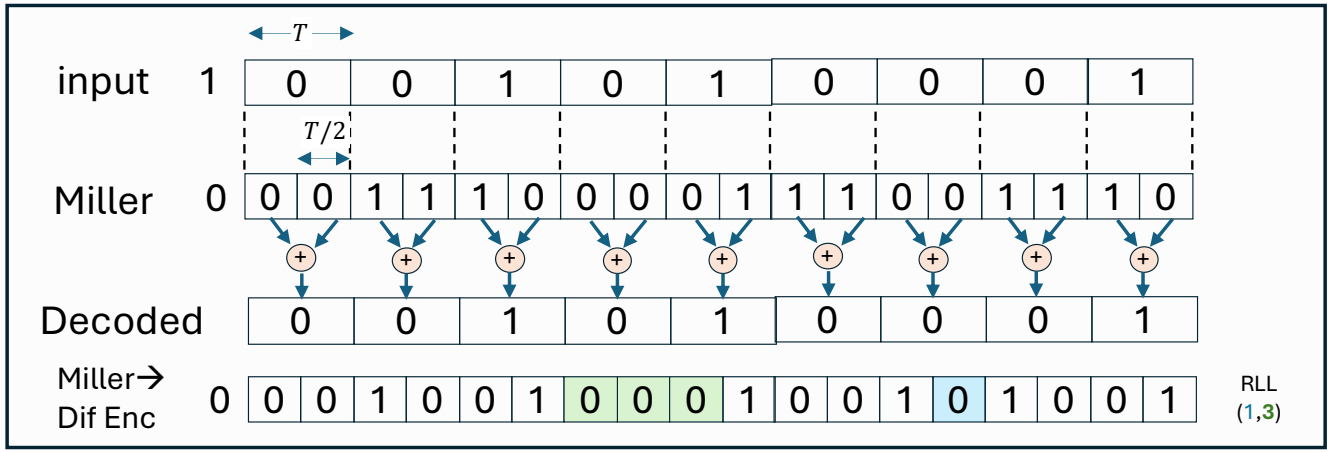
- The rule: $m_k = 1$, change output ; $m_k = 0$, no change.

- $1 - D$ in channel, then it is an **AC-Coupled Channel.**
  - Simple detectors look for a change (this not ML – ML comes later in Chapter 7 on Viterbi Detection).
  - Energy out → 1 ; no energy → 0 ; detector just samples the output at symbol/bit rate

- $1 - D$ in receiver after ML for binary, then the system is ML and differentially encoded
  - This measures change in successive outputs after ML ("hard decoding") to 1 or 0 stream.
  - Can be generalized to $M > 2$, see Section 3.8.4 that generalizes this precoder (L17).

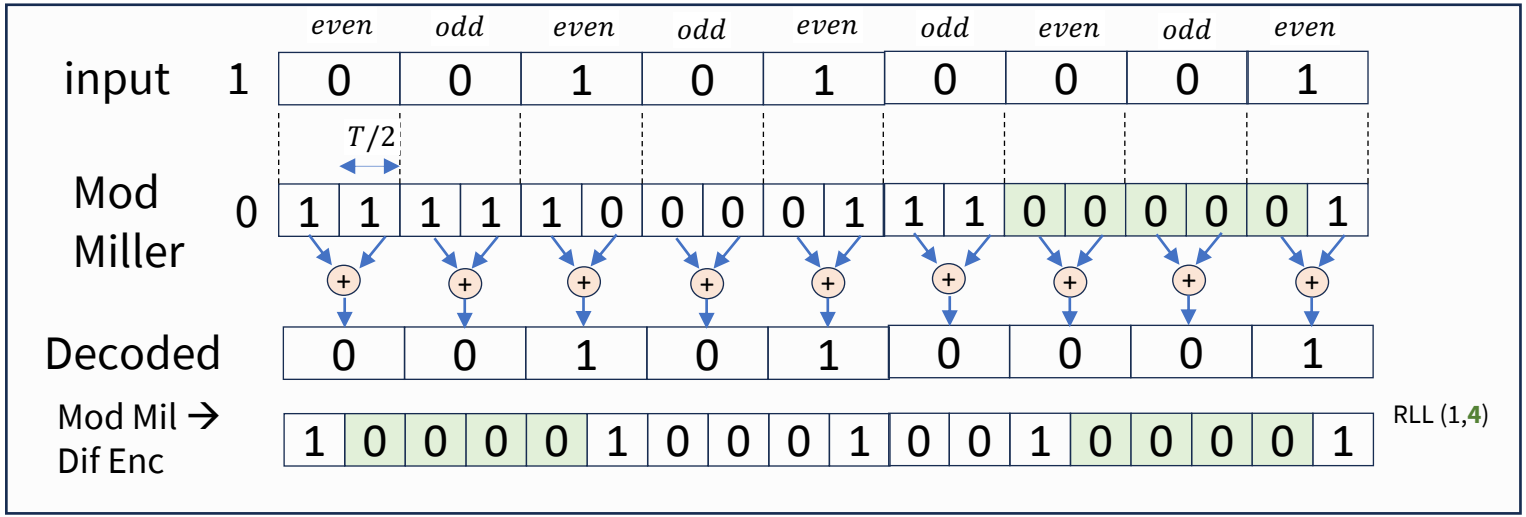
Miller Code

- Adds memory to RZ ($\bar{b} = \frac{1}{2}$).
- 2 output bits (symbol) per input bit
- 0 keeps same level
- 1 changes in middle



- Encoder retains previous last output-bit's polarity, unless both it & current input bit are 0.
  - In this latter case, the two current output bits have same value, but opposite polarity from last output bit.
- Miller is used on AC-coupled channels so Miller outputs are differentially encoded.
  - Receiver decodes on output energy or not, i.e. a "transition."
- Miller code (with D.E.) spaces such transitions – RLL(1,3) – ***run-length-limited (RLL)*.**

Modified Miller Code

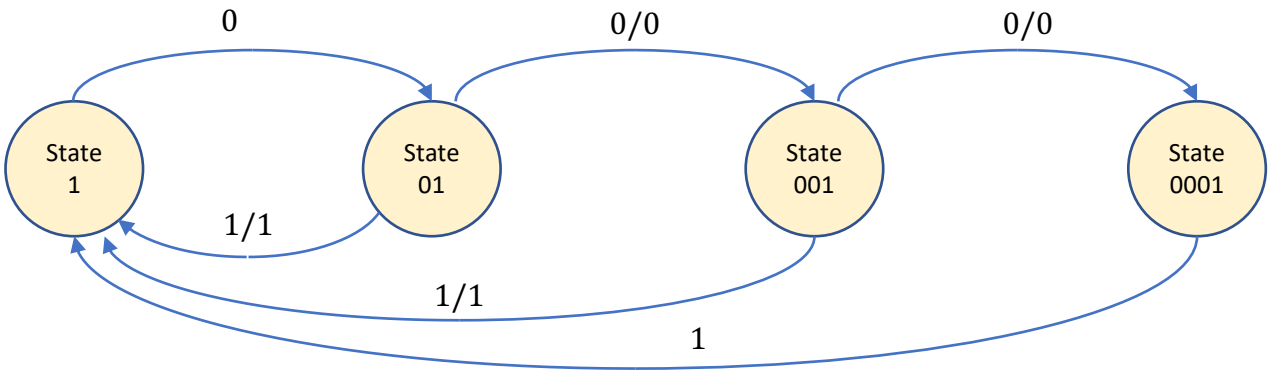


- Modified Miller allows slightly longer strings of zeros (no transition), up to 4 and
  - is slightly more low pass.
  - Two successive input 0's → one follows the normal Miller polarity inversion, while the other does not.

- Raises question of matching "modulation" (~basis) functions to channel.

RLL – Miller example

- Miller Encoder outputs → differential encoder
- (1 is change, 0 is stay same)→ ACC channel



- State Transition
  - Adjacency matrix A
  - Conditional probability matrix P

```
>> A =
   0   1   0   0
   0   0   1   0
   0   0   0   1
   1   1   1   0
>> transpose(eig(A)) % =
  1.4656  -1.0000  cplx)
log2(ans(1)) %=   0.5515
```

Allows $\bar{b} \leq .5515$ (so ½)

RLL (1,3) Code "= Miller," but it really need a mapping to actual input bits so that $\bar{b} = 1/2$ is maintained.

```
>> P % =
     0    0.50       0       0
     0       0    0.50       0
     0       0       0    1.00
  1.00    0.50    0.50       0
```

```
[V,D]=eig(P);
>> transpose(diag(D)) % =
  1.0000  -0.7718  -0.1141  cplx
pstat=V(:,1);
pstat=pstat/sum(pstat);
```

- transpose(pstat) %= 0.0909 0.1818 0.3636 0.3636
- (1/11)*[1 2 4 4] % = 0.0909 0.1818 0.3636 0.3636
- This is stationary distribution

You could do this for any RLL: st diagram, A, pstat

Iterating the state machine

- The new adjacency matrix is easy A^k

>> A^2 % =

0	0	1	0
0	0	0	1
1	1	1	0
0	1	1	1

>> A^3 % =

0	0	0	1
1	1	1	0
0	1	1	1
1	1	2	1

>> A^4 % =

1	1	1	0
0	1	1	1
1	1	2	1
1	2	2	2

>> A^5 % =

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

- So is new prob transition matrix P^k

>>P^2 % =

0	0	0.25	0
0	0	0	0.5000
1.00	0.5000	0.5000	0
0	0.5000	0.2500	0.5000

>> P^3 % =

0	0	0.25	
0.50	0.25	0.25	0
0	0.50	0.25	0.50
0.50	0.25	0.50	0.25

>> P^4 % =

0.25	0.1250	0.1250	0
0	0.2500	0.1250	0.2500
0.50	0.2500	0.5000	0.2500
0.25	0.3750	0.2500	0.5000

- But the symbol matix X expands to strings $X^{(k)}$

>> X % =

Ø	-1	Ø	Ø
Ø	Ø	-1	Ø
Ø	Ø	Ø	1
1	1	1	Ø

>> $X^{(2)}$ % =

Ø	Ø	[-1 -1]	Ø
Ø	Ø	Ø	[-1 -1]
[1 -1]	[1 -1]	[1 -1]	Ø
Ø	[-1 1]	[-1 1]	[-1 1]

Non-zero prob transitions can have up to 2 symbols (k in general)

Matlab programs to compute

- Inputs are P and X

```
markov_psd_transition_closedform
 PSD of a stationary Markov transition-output process:
  y_t = X(s_t, s_{t-1})
 with column-stochastic transition matrix:
  P(i,j)=Pr{s_t=i | s_{t-1}=j}

 Inputs
  P        NxN, column-stochastic (columns sum to 1)
  X        NxN, emission on transition j->i is X(i,j)
  f        frequency grid
           - if Fs empty: normalized cycles/sample (typically [0,0.5])
           - if Fs given: Hz (typically [0,Fs/2])
           If omitted/empty: generated as linspace(0,0.5,nfft) or linspace(0,Fs/2,nfft)
  Fs       sampling rate in Hz (optional; default [])
  remove_mean true removes DC impulse (subtract |mu|^2) (default true)
  nfft     number of frequency samples if f omitted (default 4096)

 Output struct out:
  out.pistat stationary distribution (P*pistat = pistat)
  out.mu     mean E[y_t]
  out.R0     R[0] = E[|y_t|^2]
  out.f      frequency grid (same as input or generated)
  out.S      PSD on that grid (nonnegative clipped)
```

- Because the output 1's correspond to transitions in writing,
  - which means a differential encoder follows the state machine
- Double number of states (so to 8 states from 4)

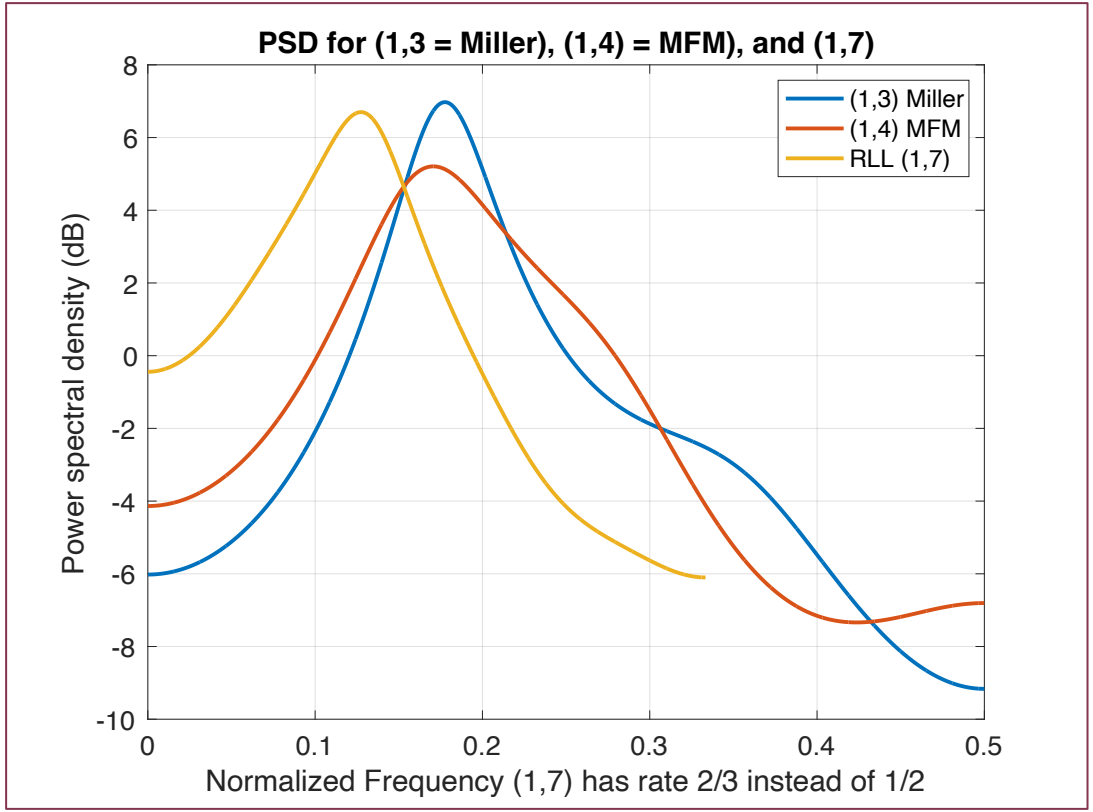
```
> function P = rll_coil_P(lmin, lmax, a)
 rll_coil_P Markov chain for RLL(lmin,lmax) with NRZI-like polarity flips on '1'.
 States: (r,+) and (r,-), r = 0..lmax.
 Column-stochastic: P(i,j)=Pr{next=i | prev=j}.

 Inputs
  lmin, lmax : integers with 0 <= lmin <= lmax
  a       : probabilities of writing '1' in the "free" region r=lmin..lmax-1.
          - If scalar: same probability used for all eligible r.
          - If vector: length must be (lmax - lmin), where
            a(1) corresponds to r=lmin,
            a(2) corresponds to r=lmin+1, ..., up to r=lmax-1.

 Example:
  P = rll_coil_P(1,3,[a1 a2]);  % produces (1,3) code
```

Power Spectra of (1,3),(1,4), (1,7)

```
P8 = rll_coil_P(1, 3)
I = 1;
xstate = I * [ +1 +1 +1 +1  -1 -1 -1 -1 ].';
X8 = repmat(xstate, 1, 8);
out8 = markov_psd_transition_closedform(P8, X8)
P10 = rll_coil_P(1, 4)
xstate = I * [ +1 +1 +1 +1 +1 -1 -1 -1 -1 -1 ].';
>> X10 = repmat(xstate, 1, 10);
out10 = markov_psd_transition_closedform(P10, X10)
P16 = rll_coil_P(1, 7)
xstate = I * [ +1 +1 +1 +1 +1 +1 +1 +1 -1 -1 -1 -1 -1 -1 -1 -1 ].';
>> X16 = repmat(xstate, 1, 16);
>> out16 = markov_psd_transition_closedform(P16, X16)
plot(out8.f,10*log10(out8.S),out8.f,10*log10(out10.S),(2/3)*ou
t8.f, 10*log10(out16.S))
```



- The (1,7) is relaxed and can be implemented with 3 bits out for every 2 input bits ($\bar{b} = \frac{2}{3}$)

The (1,7) is better match to an AC-coupled channel with lowpass.

Near-Field Communication Line Codes

Section 1.3.3

ASK

- **Amplitude-Shift Keying** – BPSK with nonzero mean, but retains carrier. Values are 0 and $\sqrt{2 \cdot \mathcal{E}_x}$.
- ASK loses 3 dB (like binary orthogonal, and is essentially equivalent to that)
  - ML detector is bandpass filter with sampler at symbol rate (carrier frequency need not be known)
- ASK = 100% ASK
- 10% ASK - the two levels are $\sqrt{2 \cdot \mathcal{E}_x}$ and $.9 \cdot \sqrt{2 \cdot \mathcal{E}_x}$. Loses 20 dB w.r.t. 100% and 23 dB w.r.t. BPSK.
  - Carrier is always on so it can "power" a passive receiver's detector (and be used to power local xmit also)
- **Near-Field Communication**
  - The distance is roughly less than the wavelength (Fresnel distance, this is physics) OR
  - The receiver is close enough that it receives enough carrier power to operate (this is commercial NFC today)

Near-Field Comms (ISO Standards)

Standard	data rate (kbps)	downlink mod	uplink mod
14443 Type A	106	100% ASK mod Miller	load modulation
14443 Type B	106	10% ASK (NRZ)	BPSK 847.5 kHz subcar
18092 Passive	106	100% ASK	BPSK load mod 847.5 kHz subcar
18092 Passive	212 424	10% ASK	ASK load mod
18092 Active	106	100% ASK	100% ASK
18092 Active	212 424	10% ASK	10% ASK

- **13.56 MHz NFC carrier** - divided by powers of 2 (simple flip-flop-based counters) by up to 128
  - Divide by 16 (847.5 kHz) – sub-carrier offset to 13.56 MHz
  - Divide by 128 (~106 kHz) – symbol rate

End Lecture S3