Unit 10: Convolutional Codes

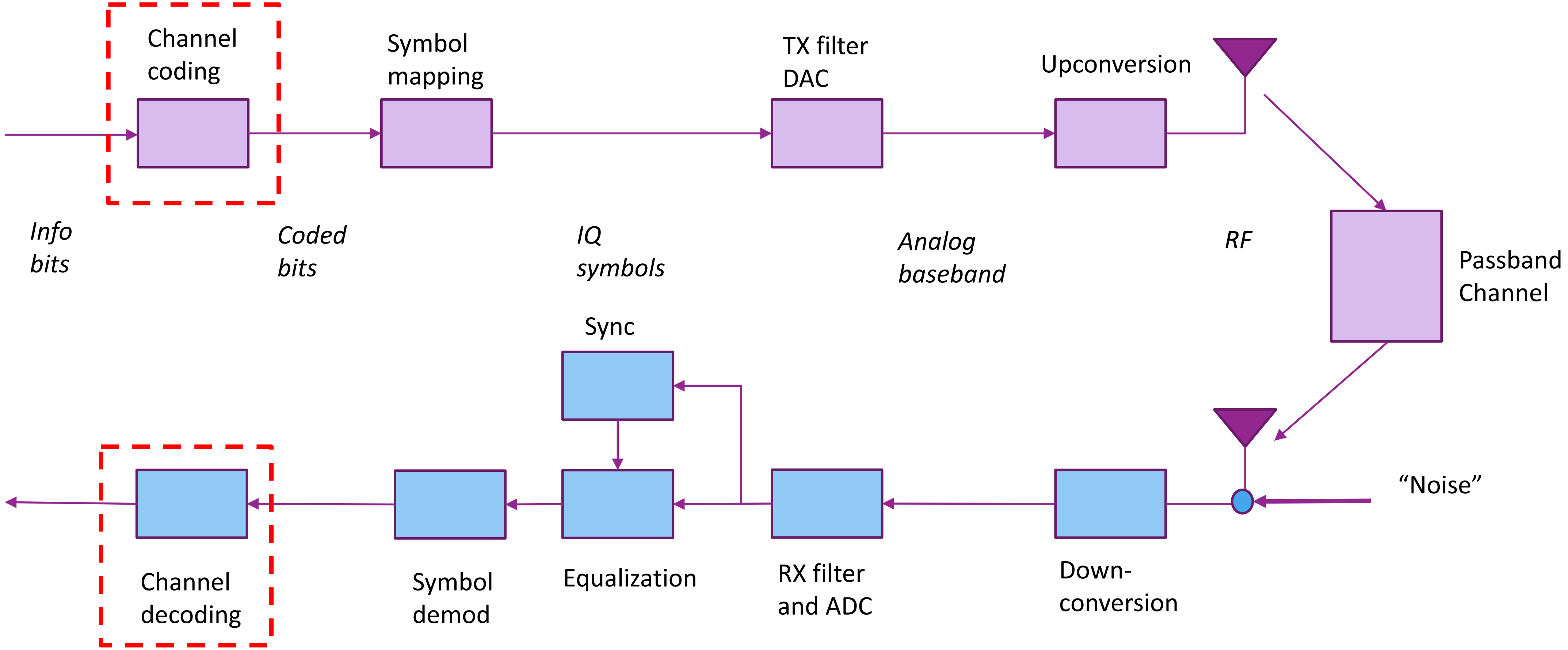
EL-GY 6013: DIGITAL COMMUNICATIONS

PROF. SUNDEEP RANGAN

Learning Objectives

- Encode data using a convolutional code for given a generator polynomial
- Compute the rate of the code including tail bits
- Represent the code via a trellis diagram and finite state machine
- Compute the branch metrics of a code for a memoryless channel
- Decode the code via the Viterbi algorithm

This Unit



Channel
coding
Symbol
mapping
TX filter
DAC
Upconversion
Info
bits
Coded
bits
IQ
symbols
Analog
baseband
RF
Passband
Channel
Sync
"Noise"
Channel
decoding
Symbol
demod
Equalization
RX filter
and ADC
Down-
conversion

Outline

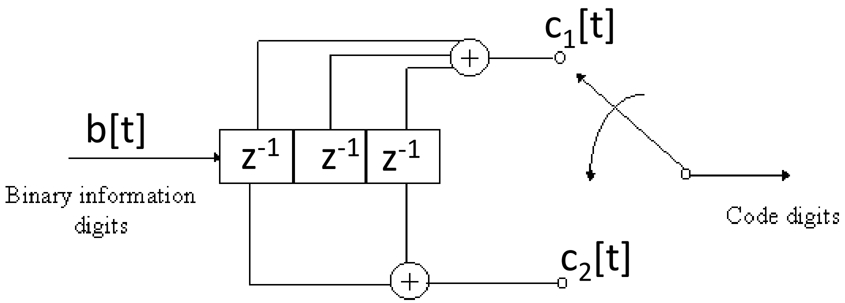
- Convolutional codes: encoding and representations
- Tree, trellis and state diagrams
- Decoding with branch metrics
- Viterbi decoding

Convolutional Codes History

- Generates a stream of coded bits from uncoded bits
  - Block codes form by terminating the stream
- Output stream created by binary FIR filters
- Developed originally by Elias (1955)
  - Key challenge was decoders. Much study in 1960s.
  - Practical, optimal decoders developed by Viterbi,1967.
- Can perform within 2-3 dB of Shannon limit.
- Instrumental in Pioneer Space program (along with RS codes)
- Most widely-used code in industry today: WiFi, cellular
  - In the mid 1990s, longer block length cellular codes were replaced with Turbo codes,
  - But, these use convolutional codes as basis

Convolutional Codes

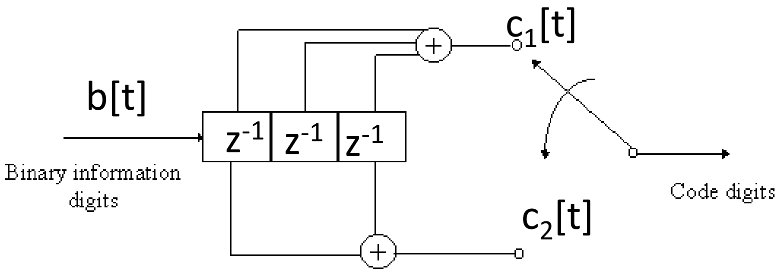
- Encode data through parallel binary (usu. FIR) filters
- Example convolutional code:
  - Rate = ½ (two output bits ($c_1[t], c_2[t]$) for each input bit b[t].
  - Constraint length K=3 (size of shift register)



$$c_1[t] = b[t] + b[t-1] + b[t-2]$$
$$c_2[t] = b[t] + b[t-2]$$

Convolutional Codes

Block Implementation



$$c_1[t] = b[t] + b[t-1] + b[t-2]$$
$$c_2[t] = b[t] + b[t-2]$$

- To implement as block code:
  - Start with L input bits b[0],b[1],…,b[L-1]
  - Append K-1 zero b[L]=b[L+1]=…=b[L+K-2]=0 (called tail bits)
  - Generate output bits from each branch
    - $c_j[0]$, $c_j[1]$, … , $c_j[L+K-2]$, j=1,…,N where N = number of branches
  - Final codeword is concatenation of branch output
    - **c**=($c_1[0]$, $c_1[1]$, … , $c_1[L+K-2]$,…, $c_N[0]$, $c_N[1]$, … , $c_N[L+K-2]$)
- Effective rate: $R = \frac{L}{N(L+K-1)} \approx \frac{1}{N}$ for large L

Convolutional Codes

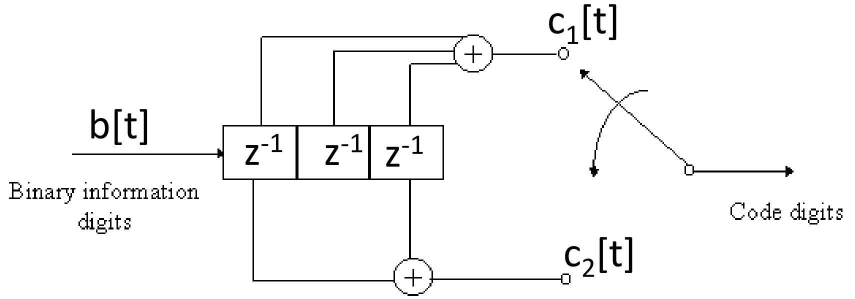
Encoding Example

- Encode message **b** = [1 0 1]
- Branch outputs **c**1=[11011] c2=[11001]
- Final output **c**=[11,10,00,10,11] (interleaved)
- Rate = 3/10

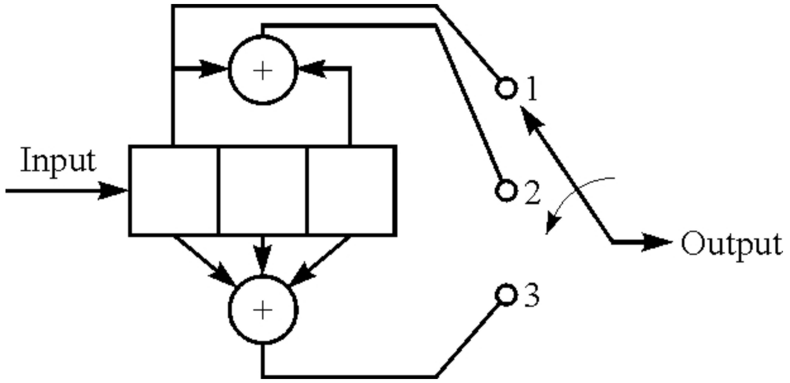
t	b[t]	c1[t]	c2[t]
0	1	1	1
1	0	1	0
2	1	0	0
3	0	1	0
4	0	1	1

Tail bits

$$c_1[t] = b[t] + b[t-1] + b[t-2]$$
$$c_2[t] = b[t] + b[t-2]$$



Generator Polynomials: Binary Form



Rate 1/3, K=3 example:

$$c_1[t] = b[t]$$
$$c_2[t] = b[t] + b[t-1]$$
$$c_2[t] = b[t] + b[t-1] + b[t-2]$$

- Code polynomials: Binary vector of filter coefficients

$g_1 = [100]$

$g_2 = [101]$

$g_3 = [111]$

Generator Polynomials: Octal Form

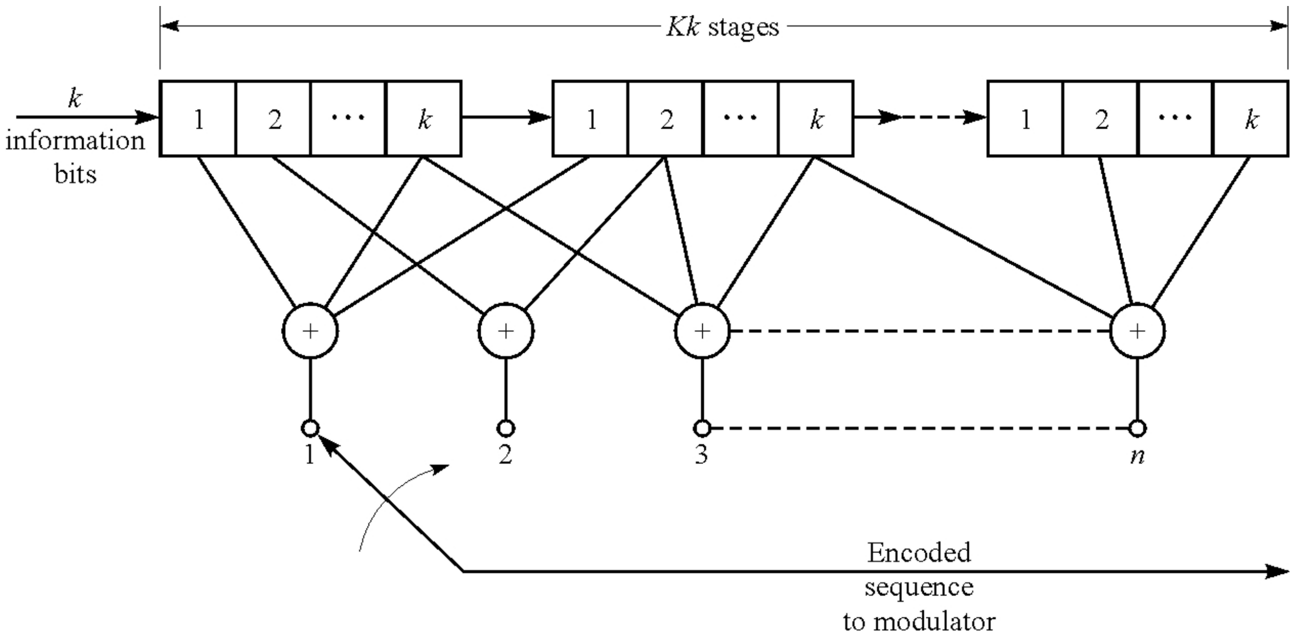
- For large constraint lengths (large K), binary form is inefficient
  - Engineers often use octal form
  - Base 8, each digit 0…7
  - Each digit represents three bits
- Example:

$$\begin{aligned} g_1 &= [1\ 011] \\ g_2 &= [1\ 101] \\ g_3 &= [1\ 010] \end{aligned} \quad \Longrightarrow \quad \begin{aligned} g_1 &= [13] \\ g_2 &= [15] \\ g_3 &= [12] \end{aligned}$$

Binary Octal

Multiple Inputs

- Examples up to now are R=1/n
- Can extend to rate R=k/n
  - Add k bits at a time



Outline

- ❑ Convolutional codes: encoding and representations
- ❑ Tree, trellis and state diagrams
- ❑ Decoding with branch metrics
- ❑ Viterbi decoding

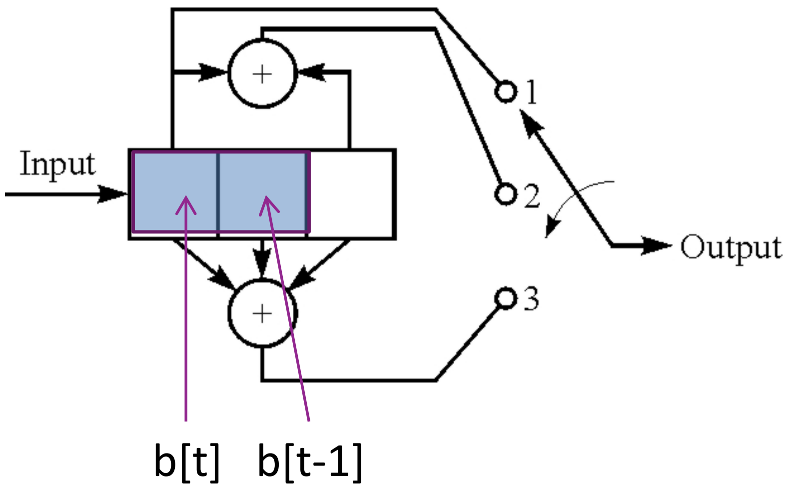
Convolutional Codes as State Machines

- Convolutional codes have memory
  - Stored in contents of shift registers
  - Each input bit changes memory contents
  - Output bits are a function of memory and input

- Three common ways to represent evolution of the memory:
  - Tree diagram
  - Trellis diagram
  - State diagram

Encoder States

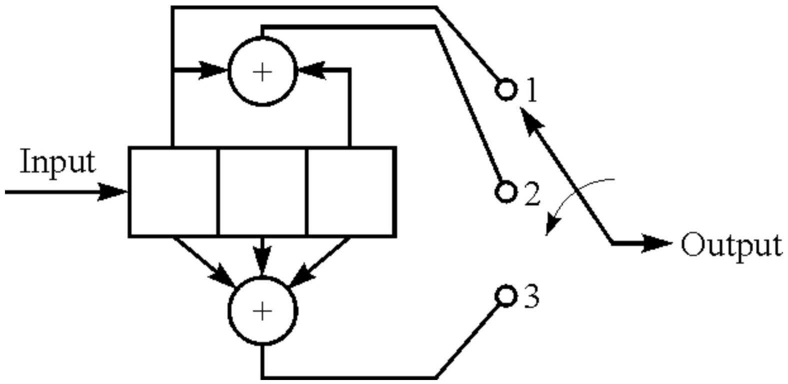
- State of the encoder determined by contents of shift register
- Only need to look at most recent K-1 bits
  - Last bit will be shifted out
  - There are 2^{K-1} states

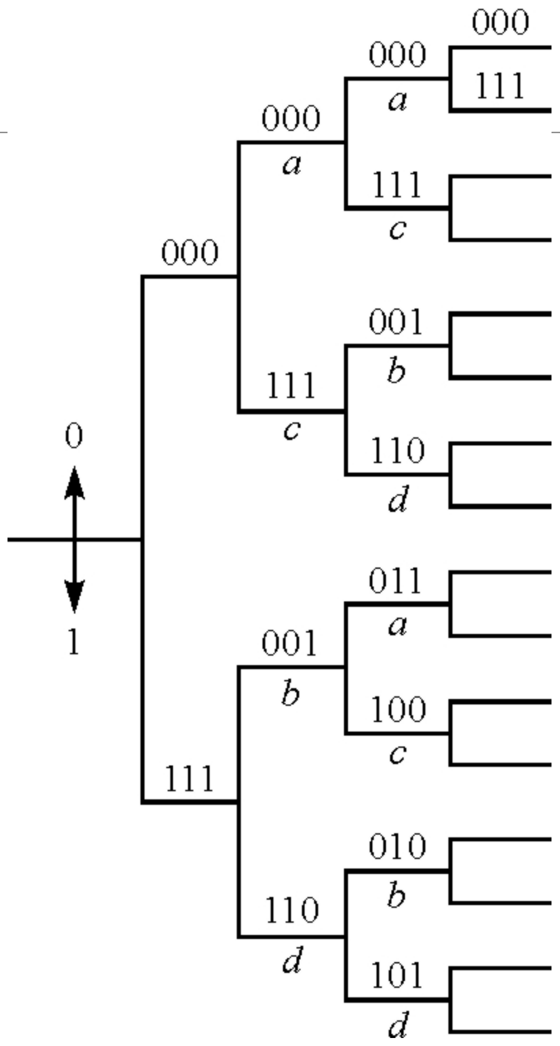


State label	$b[t]$	$b[t-1]$
a	0	0
b	0	1
c	1	0
d	1	1

Tree Diagram

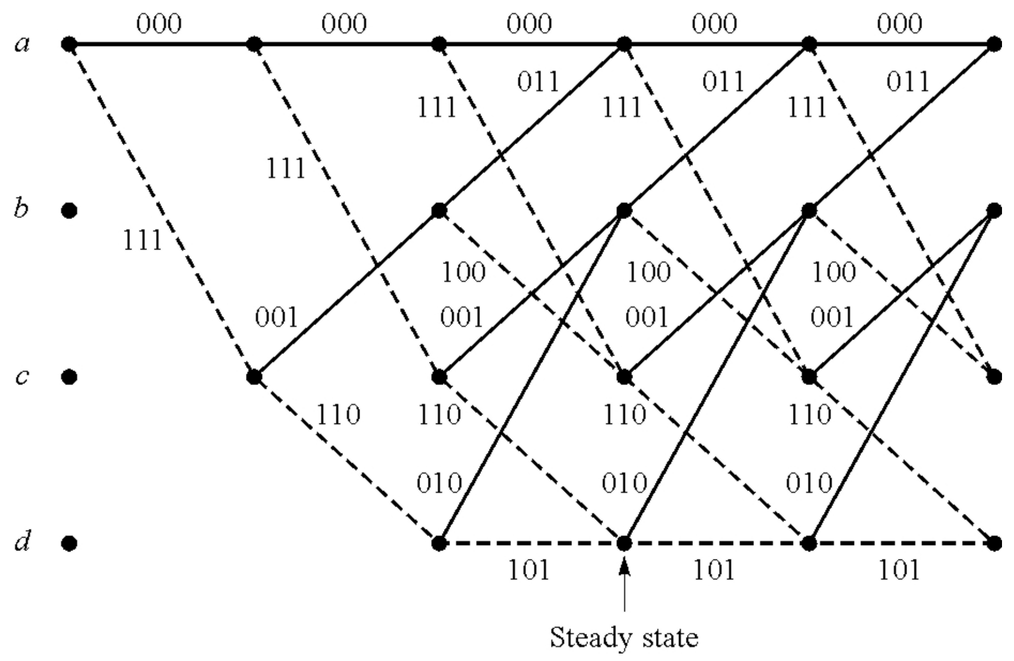
- Two branches for each input bit 0 or 1.
- Branches labeled by output bits (n bits)
- Difficult to use since branches infinitely grow





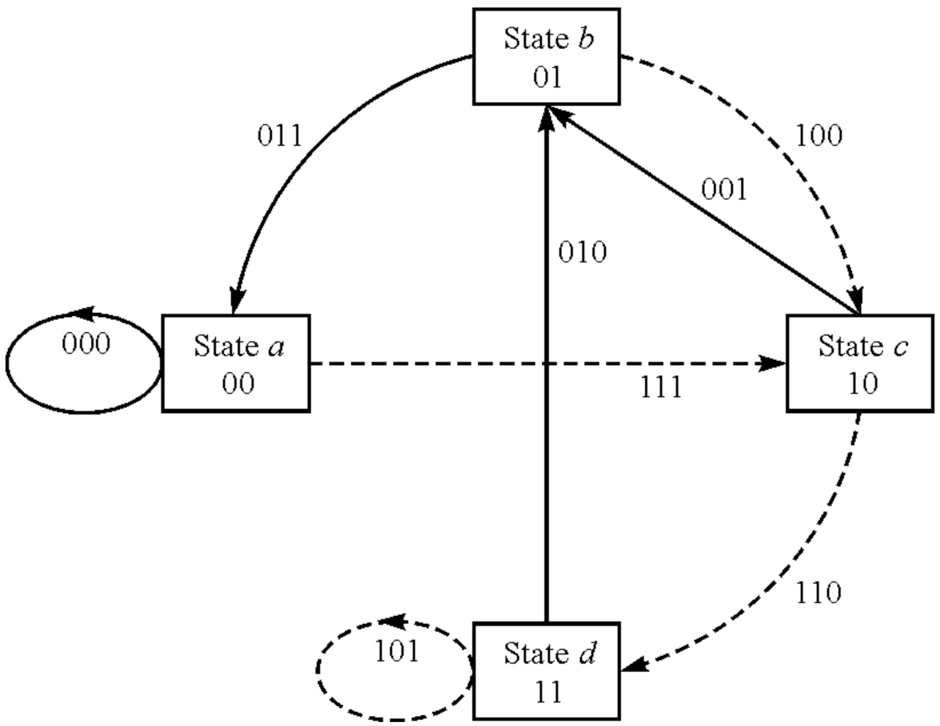
Trellis Diagram

- Show trajectory through the states
- Solid lines: paths with b[t]=0, Dash lines: b[t]=1



Labels are the outputs along each path

State Diagram



- One node per state
- Solid lines:
  - Transitions with b[t]=0
- Dashed lines:
  - Transitions with b[t]=1
- Labels indicate outputs on each state

State Machine Functional Description

- Finite state machine:
  - $x[t]$: Sequence of states, $x[t] \in \{0, \ldots, 2^{K-1} - 1\}$
  - $b[t]$: Input sequence
  - $c[t]$: Output sequence
- Iterative generating sequence:

$$x[t+1] = f(x[t], b[t])$$
$$c[t] = h(x[t], b[t])$$

State function

Output function

- Initial condition: $x[0]$

Outline

- Convolutional codes: encoding and representations
- Tree, trellis and state diagrams
- Decoding with branch metrics
- Viterbi decoding

ML Estimation

- Assume the following dimensions:
  - N outputs per time steps: $c[t] = (c_1[t], \ldots, c_N[t])$
  - T time steps (including tail bits!)
- Channel model:
  - For each bit $c_i[t]$, we make some observation $r_i[t]$
  - Output is probabilistic $P(r_i[t]|c_i[t])$
  - Assume all outputs are independent:

$$P(\boldsymbol{r}|\boldsymbol{c}) = \prod_{t=0}^{T-1} \prod_{i=1}^{N} P(r_i[t]|c_i[t])$$

- Find ML estimate:

$$\hat{c} = \arg\max_{c} P(r|c)$$

  - Maximum over all codewords

Branch Metrics

- Perform ML estimation in log domain

$$\hat{c} = \arg\max_{c} \sum_{t=0}^{T-1} \sum_{i=1}^{N} \log P(r_i[t] | c_i[t])$$

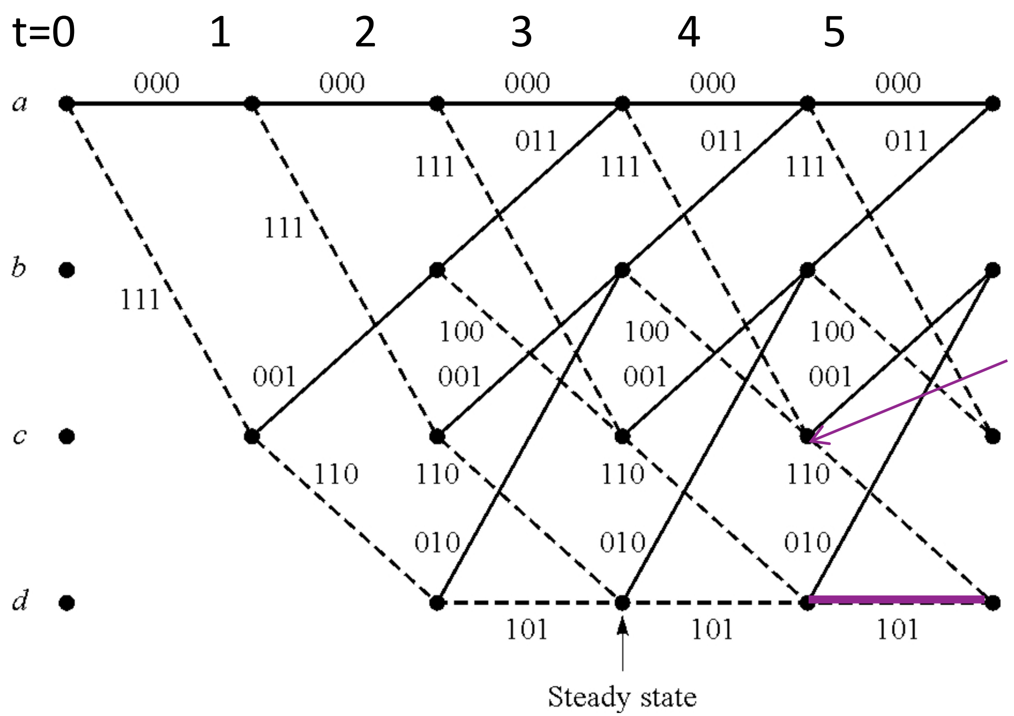
- Define the branch metric:

$$\mu_t(x_{t+1}, x_t) = \sum_{i=1}^{N} \mu_{t,i}(x_{t+1}, x_t),$$
$$\mu_{t,i}(x_{t+1}, x_t) = \log P(r_i[t] | c_i[t])$$

  - $c[t]$ is the output whenever there is a transition from x_t to x_{t+1}

- Sometimes use a negative log likelihood.

Trellis Diagram Interpretation



- The ML estimate is the shortest path on the trellis diagram.
- Each branch labeled with the metric

Scaling and Shifting Branch Metrics

- Up to now we have used branch metric

$$\mu_{t,i}(x_{t+1}, x_t) = \log P(r_i[t] | c_i[t])$$

- Can scale and shift branch metrics

$$\mu_{t,i}(x_{t+1}, x_t) = A \log P(r_i[t] | c_i[t]) + B_i[t]$$

  - Does not affect the arg max
  - Constant A must be positive
  - Constant $B_i[t]$ cannot depend on $c_i[t]$
    (Although it can depend on $r_i[t]$)

Example 1: Binary Symmetric Channel

- Binary symmetric channel: Output $r_i = 0$ or 1
- Error probability $p < 1/2$

$$P(r_i|c_i) = \begin{cases} 1-p & \text{if } c_i = r_i \text{ (no error)} \\ p & \text{if } c_i \neq r_i \text{ (error)} \end{cases}$$

- Branch metric: After appropriate shifting:

$$\mu_{ti} = \begin{cases} L & \text{if } c_i = r_i \text{ (no error)} \\ 0 & \text{if } c_i \neq r_i \text{ (error)} \end{cases}, \qquad L = \log \frac{1-p}{p}$$

- Scaling: Can take $L = 1$
  - Computes path with minimum Hamming distance

Example 2: AWGN Channel

- Binary modulation

$$r_i = s_i + w_i, \qquad w_i \sim N(0, N_0/2), \qquad s_i = \pm 1$$

- Log likelihood ratio

$$LLR_i = -\frac{(r_i - s_i)^2}{N_0}$$

- Branch metric: After appropriate shifting:

$$\mu_i = \frac{2}{N_0} r_i s_i$$

- Scaling: Can take $N_0 = 1$
  - Computes path with maximum correlation

Example 3: General LLRs

- Consider general memoryless channel $P(r_i[t]|c_i[t])$
- Take constant $B_i[t] = -\log P(r_i[t]|c_i[t] = 0)$
- Then, branch metric given by LLR:

$$\mu_{t,i}(x_{t+1}, x_t) = \begin{cases} LLR_i[t] & c_i[t] = 1 \\ 0 & c_i[t] = 0 \end{cases}$$

- Log likelihood ratio

$$LLR_i[t] = \log \frac{P(r_i[t]|c_i[t] = 1)}{P(r_i[t]|c_i[t] = 0)}$$

Summary

Channel type	Unscaled branch metric (decoding maximizes metric)	Scaled branch metric
BSC error probability p	$L = \log\left(\frac{1-p}{p}\right)$ when $c_i = r_i$ 0 when $c_i \neq r_i$	-# bit errors
AWGN with binary modulation and noise N_0	$\frac{r_i c_i}{N_0}$	$r_i c_i$
General binary channel	$L = \log\frac{P(r_i \mid c_i=1)}{P(r_i \mid c_i=0)}$ when $c_i = 1$ 0 when $c_i = 0$	

Examples

- Simple two state problem on board.

Outline

- Convolutional codes: encoding and representations
- Tree, trellis and state diagrams
- Decoding with branch metrics
- Viterbi decoding
- Practical considerations

Two Variable Optimization (1)

- Consider two variable optimization problem:

$$(\hat{x}_0, \hat{x}_1) = \arg\min_{x_0, x_1} f(x_1, x_0)$$

- This is equivalent to a nested minimization:

$$\hat{x}_1 = \arg\min_{x_1} \left[\min_{x_0} f(x_0, x_1) \right]$$

  - Minimize over x_0
  - Then minimize over x_1
- Hence, we can write the minimum over x_1 as:

$$\hat{x}_1 := \arg\min_{x_1} V(x_1), \qquad V(x_1) = \min_{x_0} f(x_0, x_1)$$

Two Variable Optimization (2)

- To obtain $\hat{x}_0$ define:

$$P(x_1) := \arg\min_{x_0} f(x_1, x_0)$$

- Then, we obtain three step procedure:

- Step 1: Minimization over x_0:
  - Compute $V(x_1) = \min_{x_0} f(x_0, x_1)$ and $P(x_1) = \arg\min_{x_0} f(x_1, x_0)$

- Step 2: Minimization over x_1: Compute $\hat{x}_1 = \arg\min_{x_1} V(x_1)$

- Step 3: Go back and find x_0: Select $\hat{x}_0$=P($\hat{x}_1$).

Example (Calculations on Board)

- Binary example: $x_0, x_1 \in \{0,1\}$:

	x_1 **= 0**	x_1 **= 1**
x_0 = 0	4	6
x_0 = 1	3	2

Values for $f(x_0, x_1)$

- Quadratic example
  - $f(x_0, x_1) = x_0^2 + 2x_1^2 + x_0 x_1 + x_1$

Convolutional Decoding Problem

- Recall convolutional code is a finite state machine:

$$x[t+1] = f(x[t], \boldsymbol{b}[t])$$
$$\boldsymbol{c}[t] = g(x[t], \boldsymbol{b}[t])$$

State update

Output

- Decoding problem is to find path with minimum path metric:

$$\boldsymbol{b} = \min_{b[0:T-1]} \sum_{t=0}^{T-1} \mu_t(c[t])$$

  - $\mu_t(c(t))$ is the branch metric
  - Write the branch metric as a function of the output instead of the state transition $(x[t], x[t+1])$

Iterative Solution

- Define partial solutions up to time $t \le T$:
- Minimum partial value ending at state x_t

$$V_t(x_t) = \min_{b[0:t-1]} \sum_{s=0}^{t-1} \mu_s(c[s]) \;\; s.t.\; x[t] = x_t$$

- Minimum path

$$P_t(x_t) = \arg\min_{b[0:t-1]} \sum_{s=0}^{t-1} \mu_s(c[s]) \;\; s.t..\; x[t] = x_t$$

- Initial conditions: $V_0(x_0) = 0, P_0(x_0) =$ empty set.

Value Function Recursion

- Value function obeys simple recursive rule:

$$V_{t+1}(x_{t+1}) = \min_{\boldsymbol{b}[0],\ldots,\boldsymbol{b}[t]} \sum_{s=0}^{t} \mu_s(c[s]) \quad s.t.\ x[t+1] = x_{t+1}$$

$$= \min_{\boldsymbol{b}(t)} \left[\mu_t(c[t]) + \min_{\boldsymbol{b}(0),\ldots,\boldsymbol{b}(t-1)} \sum_{s=0}^{t-1} \mu_s(c[s]) \right]$$

$$= \min_{\boldsymbol{b}(t),x_t} \left[\mu_t(c[t]) + V_t(x_t) \right]$$

  - Last minimization is subject to: $x_{t+1} = f(x_t, \boldsymbol{b}[t]), \boldsymbol{c}[t] = g(x_t, \boldsymbol{b}[t])$

- Minimization over previous state and current input

Path Recursion

- To update path, first solve minimization:

$$\hat{\boldsymbol{b}}(t), \hat{x}_t = \underset{\boldsymbol{b}(t), x_t}{\arg\min}\ [\mu_t(c[t]) + V_t(x_t)]$$

  - Subject to: $x_{t+1} = f(x_t, \boldsymbol{b}[t]), \boldsymbol{c}[t] = g(x_t, \boldsymbol{b}[t])$

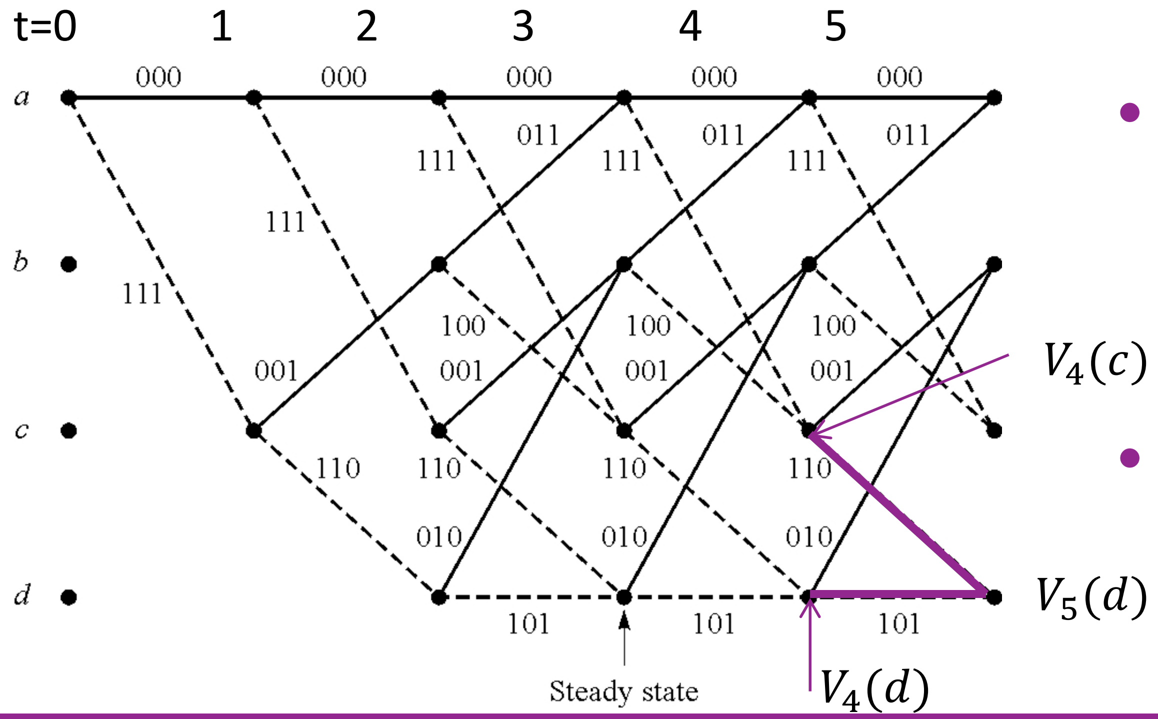
- Then, append bits to path:

$$P_{t+1}(x_{t+1}) = \{\hat{\boldsymbol{b}}(t), P_t(\hat{x}_t)\}$$

  - New path has t bits.

Trellis Diagram Interpretation

- Value function and path associated with each node.
  - Minimum value and path to get to that node.
- Updated left to right from inputs



- Consider example node

$$
\begin{aligned}
V_5(d) = \min\{ \\
& V_4(c) + \mu_4(110), \\
& V_4(d) + \mu_4(101) \}
\end{aligned}
$$

- Take min path from two possible incoming nodes

Example 1: HD Decoding

- A very simple rate ½ , K=2 convolutional code (too simple to be useful):

$$c_1(t) = b(t), \qquad c_{2(t)} = b(t) + b(t-1)$$

- Suppose received hard-decision decoded bits are:

$$r = \{01,10,11,10, \dots\}$$

- Draw state diagram and complete table (on board)

	t=0	1	2	3
$V_t(x=0)$				
$P_t(x=0)$				
$V_t(x=1)$				
$P_t(x=1)$				

Example 2: SD Decoding

- Consider a recursive code

$$c_1(t) = b(t), \qquad c_2(t) = b(t) + c_2(t-1)$$

- Suppose received soft-decision decoded bits are:

$$r = \{(0.1, 0.8), (-0.3, 1.5), (-1, -2), \ldots\}$$

- Draw state diagram and complete table (on board)

	t=0	1	2	3
$V_t(x=0)$				
$P_t(x=0)$				
$V_t(x=1)$				
$P_t(x=1)$				

Outline

- ❑ Convolutional codes: encoding and representations
- ❑ Tree, trellis and state diagrams
- ❑ Decoding with branch metrics
- ❑ Viterbi decoding
- ❑ Practical considerations

Complexity

- Update of each node requires maxima over 2^k branches
- There are $2^{k(K-1)}$ states, so complexity / time is $O(2^{kK})$
- Total complexity is $O(T2^{kK})$
- Storage is also $O(T2^{kK})$. (From the paths)
- Summary:
  - Viterbi algorithm is linear in block length.
  - Can have very long block lengths (often T in the 1000s)
  - But, complexity is exponential in constraint length
  - Practical decoders limited to $K = 7$ or $K = 9$.

Terminating the Trellis

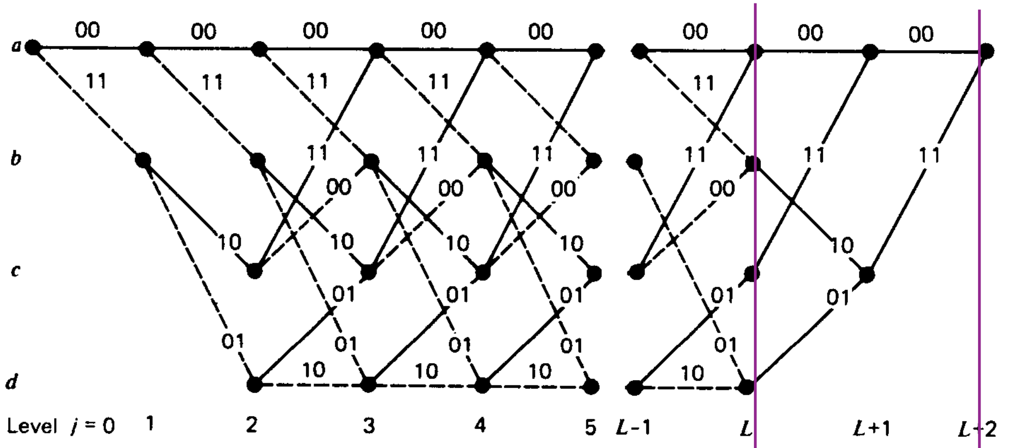


Figure 11.15 Trellis for the convolutional encoder of Fig. 11.13*a*.

During tail bits only paths with zero bit inputs.

- Recall tail bits are zero: $\boldsymbol{b}(L) = \cdots = \boldsymbol{b}(L+K-2) = 0.$
- Limits the paths at the end of the trellis
- Viterbi algorithm should only be done on the zero path.
- Very important to not forget the bits at the end.
- Otherwise, final bits are not protected.

Pruning the Path Memory

- In current algorithm, path $P_t(x_t)$ grows to full block length.
- Adds storage: Storage is O$\left(T2^{kK}\right)$. Linear in T
- Adds delay. No bits can be determined until code is fully decoded.
- Many practical implementations:
  - Store some finite length δ of each surviving path.
  - Can make decision on bit input $\boldsymbol{b}(t)$ after $\mathbf{r}(t+\delta)$.
  - Rule of thumb: Good performance if $\delta \geq 5K$.

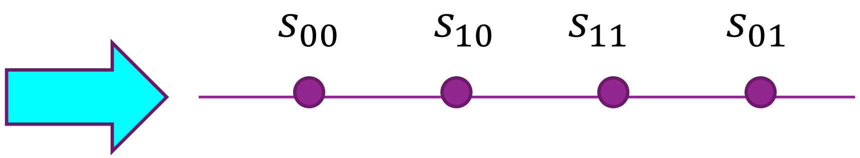
Rate Matching Convolutional Codes

- Convolutional codes have limited rates, usu. ½ or 1/3.
- Obtain other rates through
  - Puncturing: Remove coded bits to increase rate
  - Repetition: Repeat coded bits to decrease rate
- Puncture / repeat pattern is important (see Proakis)
  - Try to spread out modified bits
- For punctured bits, set corresponding LLRs to zero
  - Viterbi decoder just ignores those bits
- For repeated bits, add the corresponding LLRs
  - Viterbi decoder will increase confidence on that branch

High Order Constellations

- Higher order constellations (eg. 16- or 64-QAM)
- Each constellation is a point is a function of multiple bits.
- Likelihood does not factorize
  - Each symbol r(t) depends on multiple bits
- Example 4-PAM (or one dimension of 16-QAM):
  - Each symbol likelihood depends on two bits: $p(r|c_1, c_2)$

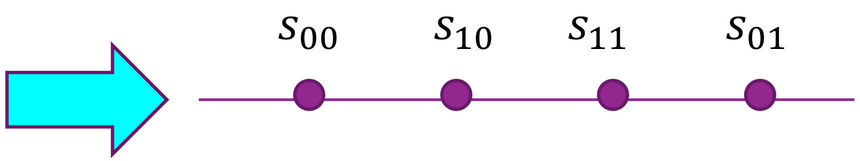
Two bits:
(c_1, c_2)



$r = s + n$

High Order Constellations

Two bits:
(c_1, c_2)



$r = s + n$

- To create LLRs for individual bits use total probability rule:

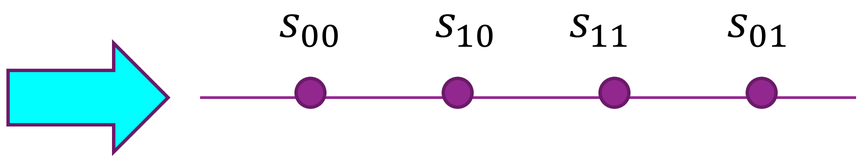
$$p(r|c_1) = \frac{1}{2}\left(p(r|c_1, c_2 = 0) + p(r|c_1, c_2 = 1)\right)$$

- Resulting bitwise LLR:

$$LLR \text{ for } c_1 = \log \frac{p(r|c_1, c_2 = 1,0) + p(r|c_1, c_2 = 1,1)}{p(r|c_1, c_2 = 0,0) + p(r|c_1, c_2 = 0,1)}$$

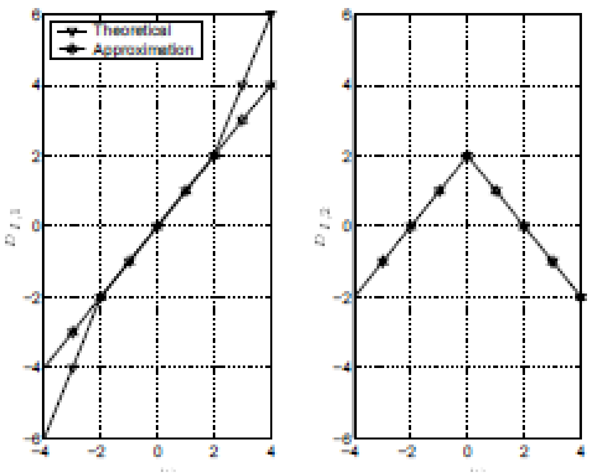
High Order Constellations

Two bits:
(c_1, c_2)



$r = s + n$

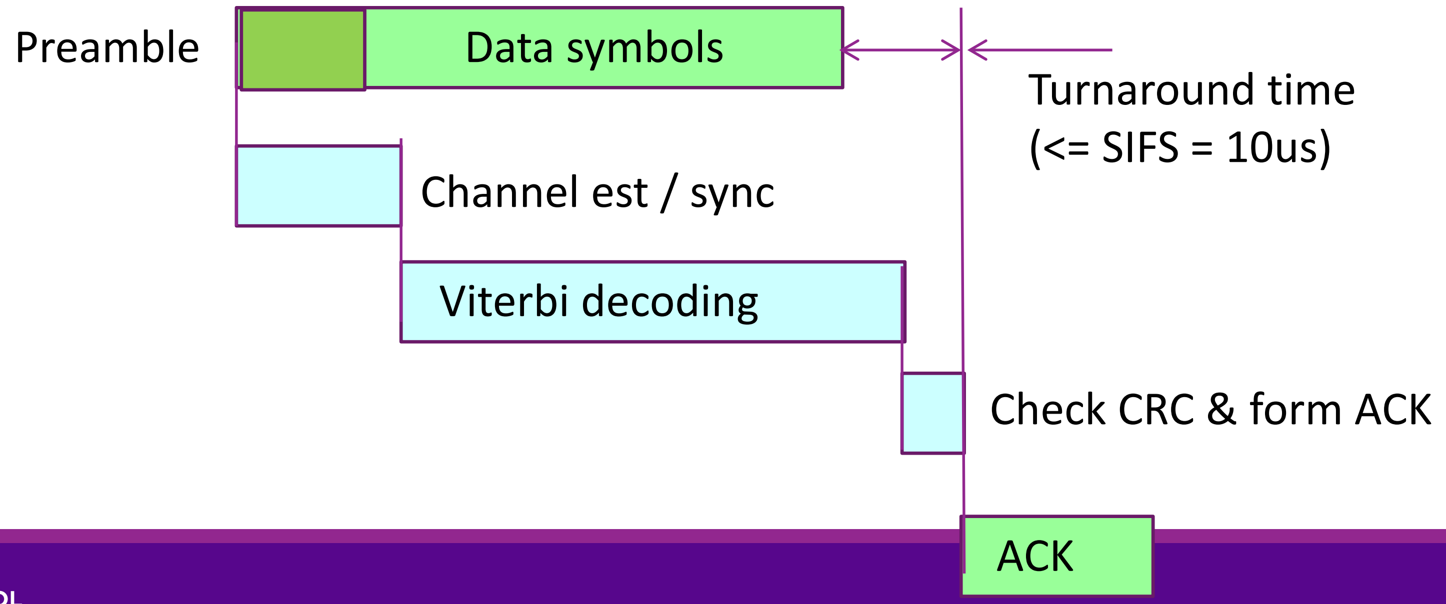
LLR for c2 LLR for c1



- LLRs can have irregular shapes
- Not simple linear function as in BPSK / QPSK case
- Often use approximations
- More info: Caire, Taricco and Biglieri, “Bit-Interleaved Coded Modulation," 1998.

Convolutional Codes in WiFi

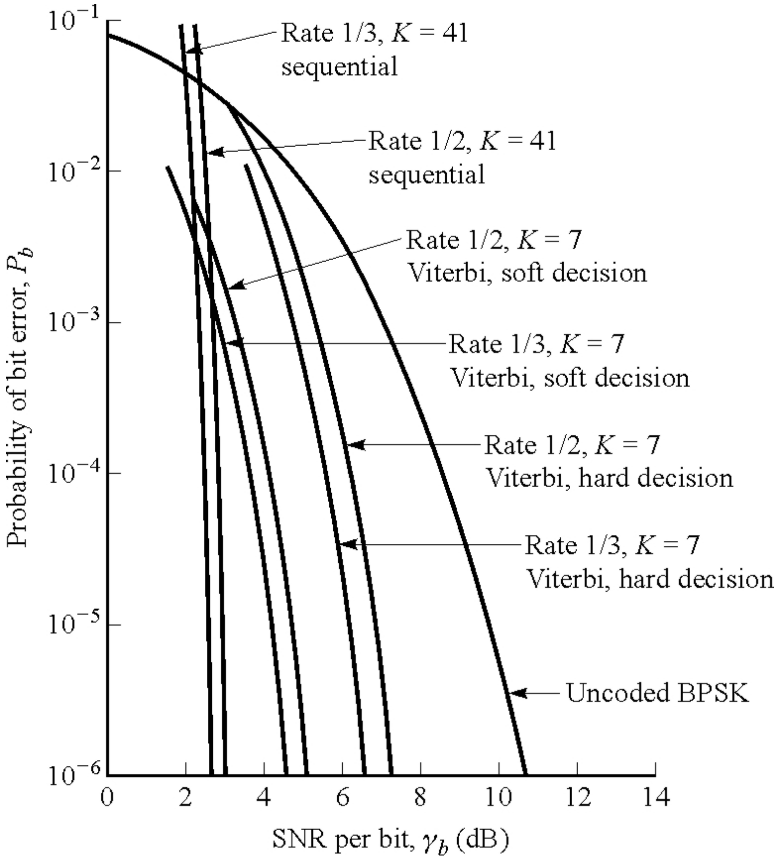
- 802.11 uses R=1/2 K=7 code.
- Length adjusted to packet size
- Higher rates (R=2/3 and ¾) achieved through puncturing
- Enables decoding as data arrives for ACK fast turnaround



Convolutional Codes in LTE

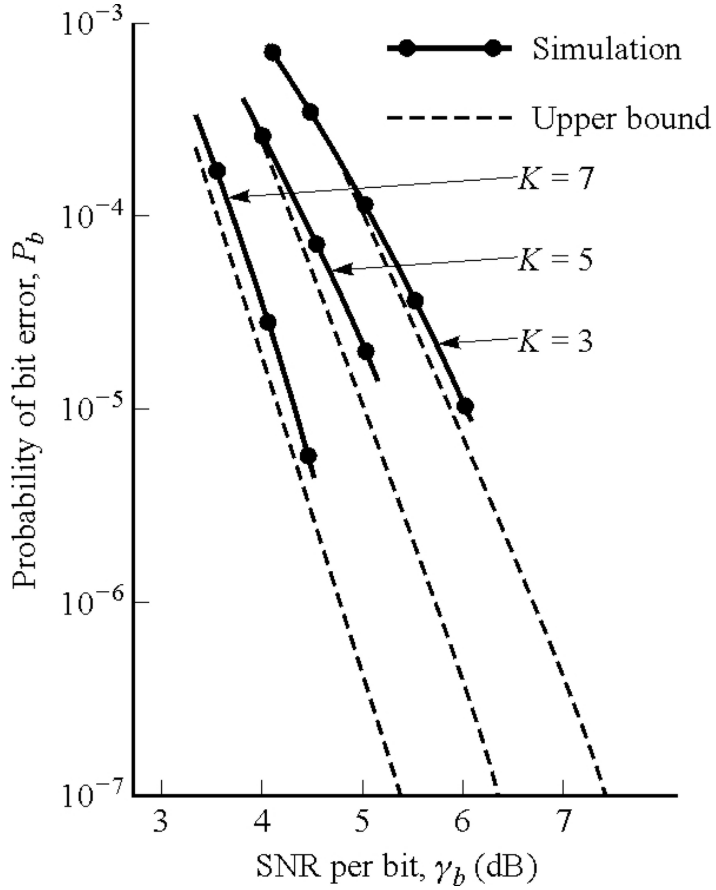
- Convolutional codes in LTE used for:
  - Control channels (payload typ 20-40 bits +CRC), and
  - Short (< 128 bit) data frames
- Larger payloads encoded with turbo codes (discussed later)
- Uses rate=1/3 base convolutional code with K=7.
- Higher rates achieved via puncturing
- Control channels use a sophisticated technique called tail biting to reduce loss on the tail bits

Decoding Performance



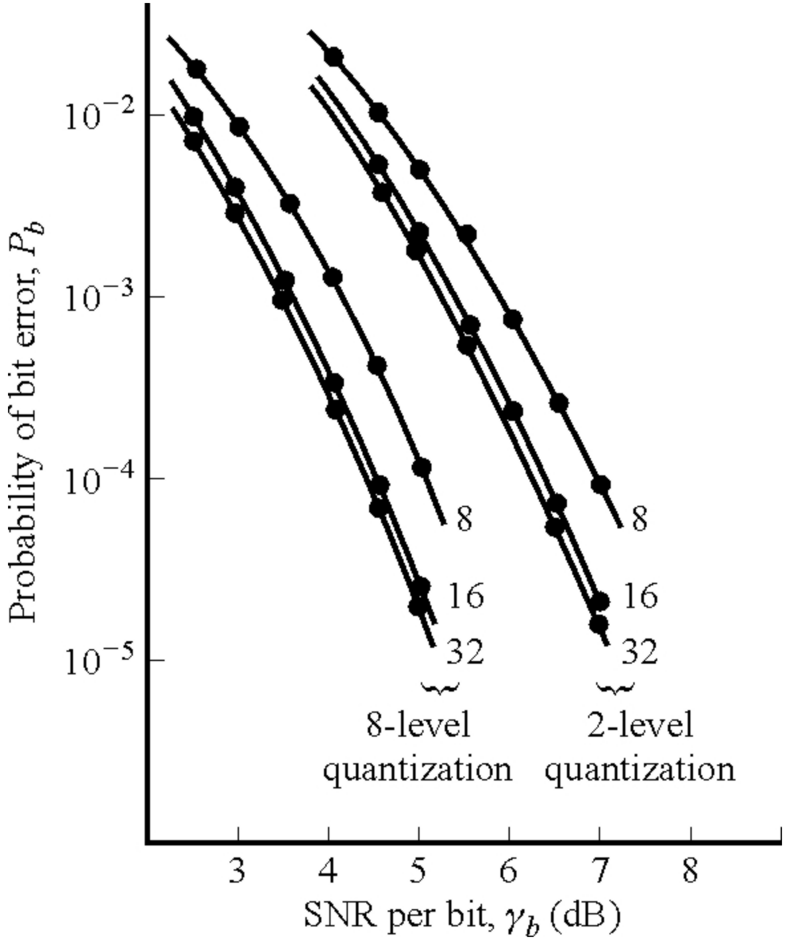
- Hard decision loses approximately 1.5 to 2 dB
- Constraint length K=7 is sufficient for very sharp error performance

Different Constraint Lengths



- ❑Approximate 1 dB improvement between K=3, 5 and 7
- ❑Higher constraint lengths become computationally intractable
- ❑Recall decoding complexity is exponential in K

LLR Quantization



- ❑Minimal gains after 16 bit quantization of branch metrics
- ❑Recall HD is equivalent to 1 bit quantization
- ❑Most commercial implementations use 6-bit LLRs