AMRITA VISHWA VIDYAPEETHAM

DEEMED TO BE UNIVERSITY

20CYS301

Digital Communications

L27-L30 Sep 30, 2024

Error Detection & Correction

Amit Agarwal, Professor

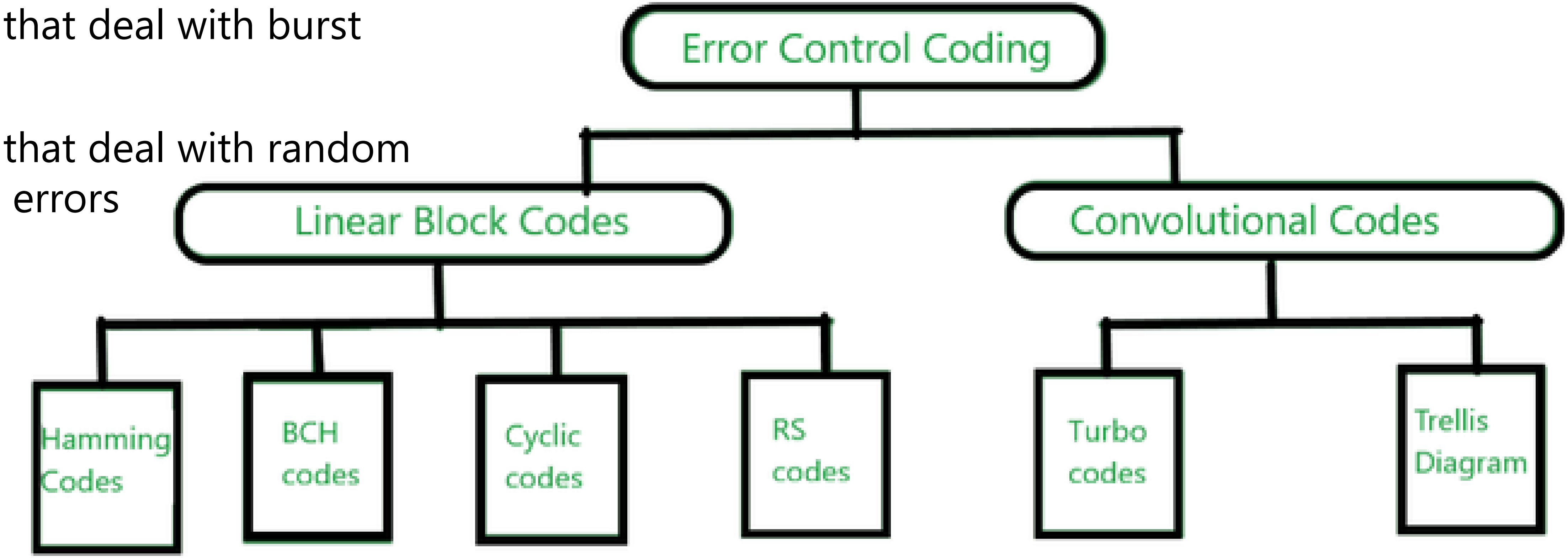
Department of Electrical & Electronics Engineering

TIFAC-CORE in Cyber Security

1. EDC Introduction [1/5]

3 types of EDCs

- The ones that deal with random errors
- The ones that deal with burst errors
- The ones that deal with random and burst errors



If channel characteristics can be determined and stable, use EDC.

1. EDC Introduction [2/5]

Error Correction Codes (ECC) are of two types:

- Convolutional codes are processed on a bit-by-bit basis. They are particularly suitable for implementation in hardware, and the Viterbi decoder allows optimal decoding.
- Block codes are processed on a block-by-block basis. Early examples of block codes are repetition codes, Hamming codes and multidimensional parity-check codes, Reed–Solomon codes, Turbo codes and low-density parity-check codes (LDPC).

Code Rate = k/n where k is message length and n is encoded message length (< 1.0)

If channel characteristics cannot be determined or is fast-changing then use a combination of EDC and automatic repeat request (ARC) – hybrid automatic repeat request (HARC). 3-types:

- Stop-and-wait ARQ: send a message, wait for ACK, once received, send the next packet
- Go-Back-N ARQ: same as above but wait for N window transmission before requiring an ACK

1. EDC Introduction [3/5]

- Selective Repeat ARQ: keep sending frames within a window size even if some frames are lost. These are buffered at the receiver. Once the window has been transferred, an ACK is sent, and frames contained in the ACK are re-sent.

ARQ requires availability of a back channel.

Hamming codes are a class of binary linear code.

Due to the limited redundancy that Hamming codes (HCs) add to the data, they can only detect and correct errors when the error rate is low. This is the case in RAM where bit errors are extremely rare and Hamming codes are widely used.

An HC is also called single error correction/double error detection code (SECDED).

1. EDC Introduction [4/5]

Hamming Code: Hamming(7,4) code adds 3 parity bits to 4 bits of data. It can detect and correct single-bit errors but not double-bit errors.

Hamming code efficiency increases as block size increases.

Problem: Find the the effective data rate of Hamming(7, 4) and Hamming(255, 247)

Answer: 0.571 and 0.969, respectively (show the calculation)

Problem: For HCs which meet the following definition, find min # of parity bits needed for 7 bits of data to be sent: $2^p \geq d + p + 1$

Answer: If data bits to be sent are 7 then RHS is 8 + p.

$\therefore\ p \geq \log_2(8+p)$

If p=1 then FALSE If p=2 then FALSE

If p=3 then LHS is 3, RHS is $\log_2 11 = 3.46$ i.e., FALSE

If **p = 4** then LHS is 4 & RHS is $\log_2 12$ which is 3.58 and thus, TRUE

1. EDC Introduction [5/5]

ECC RAM, imagine a bank database server was recording a deposit of ₹100.

As an 8-bit integer, that amount would be stored as the binary number 01100100.

If a cosmic ray changed the first bit, it would change the deposit amount to ₹228, or 11100100. ECC memory would catch and correct this error automatically.

Hamming code is not used in modern data transmission.

Wired data connections are not noisy enough to warrant the overhead of added parity data. If an error is encountered, it may be faster to ask for a retransmission of the faulty packet.

Also, low-density parity-check (LDPC) codes are more efficient to transmit but require more computation than Hamming code. LDPC is used for Wi-Fi 6 and 10GBASE-T Ethernet.

Hamming code is used for data transmission is in satellite and space communication.

Because of the great distances involved, long transmit times and the requirements for accurate data, it is preferred to use the slower but more precise.

1. RAID System [1/2]

RAID (Redundant Array of Inexpensive Disks) is used to achieve data redundancy Hamming code is not used in modern data transmission.

If a drive in an independent, 3-drive system fails then, it can be recovered from the third drive that is redundant. RAID works when drive failures are rare and independent.

Problem: Assume Drive 1 data is 11010101 and Drive 2 data is 01111111 then show the process of recovery via a back-up Drive 3.

Answer: Drive A: 1 1 0 1 0 1 0 1

Drive B: 0 1 1 1 1 1 1 1 … bitwise XOR the 2 drives and store in Drive 3.

→ Drive C: 1 0 1 0 1 0 1 0

If Drive A fails then it can be recovered by XOR-ing Drives B & C!

Drive B: 0 1 1 1 1 1 1 1

Drive C: 1 0 1 0 1 0 1 0 … bitwise XOR gives

1 1 0 1 0 1 0 1 which is the same as Drive A.

1. RAID System [2/2]

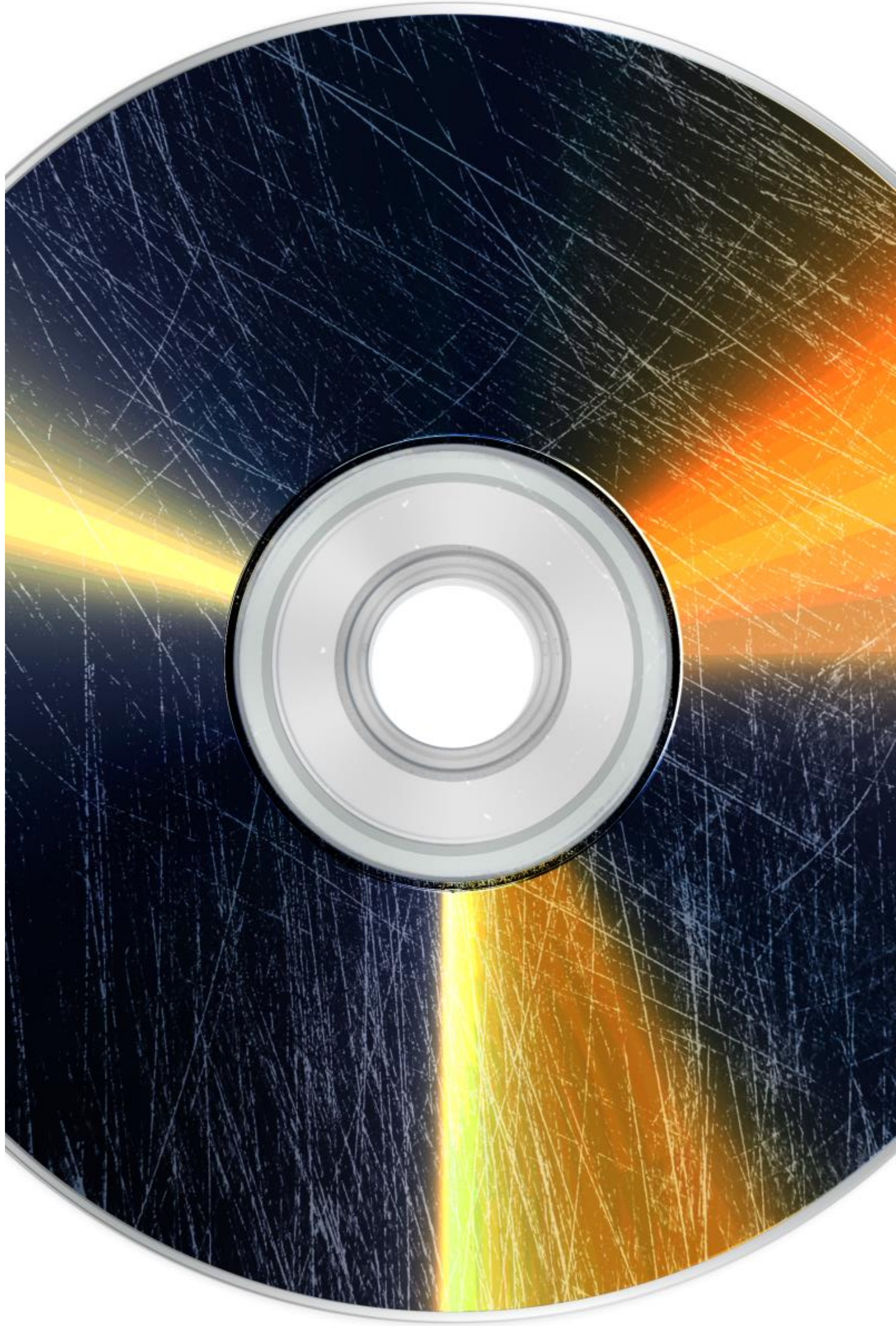
Problem: What is the code rate for storage in a RAID system?

Solution: Assuming that the back-up drive is needed for recovery, data is stored in 2 drives and backed-up using the back-up drive. Where M is the memory of each drive in the RAID system, code rate = (memory for useful storage)/(memory for useful storage + memory for recovery) = (2M)/(2M+M) =2M/3M = 66.67%.

However, in the event of a failure of any of the 3 storage media, recovery will take a long time as storages as large. Therefore, in practise, one storage for recovery is used for only equal sized data instead of two.

Thus, both A & B will store the same data and C stores the XOR of A & B. Thus, efficiency is (M)/(M+2M) =M/3M = 33.33%. That is quite inefficient!

Thus, use Hamming Codes …



1. Hamming Code [1/2]

Hamming Code Parity Bit Algorithm

STEP 1: Bit positions are counted L→R e.g. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, …

1 2 3 4 5 6 7 8 9 10

STEP 2: Note bit positions in binary format e.g. 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, …

STEP 3: Parity bits are placed in only bit positions where the position's binary representation is a whole number power of 2 – marked in saffron 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, …

STEP 4: Rest are data bit positions.

STEP 5: Parity bit coverage: Parity bit 1: all bits where LSB is 1. e.g. 1, 3, 5, 7, 9, …

Parity bit 2: all bits where the 2nd LSB is 1. e.g. 2-3, 6-7, 10-11, …

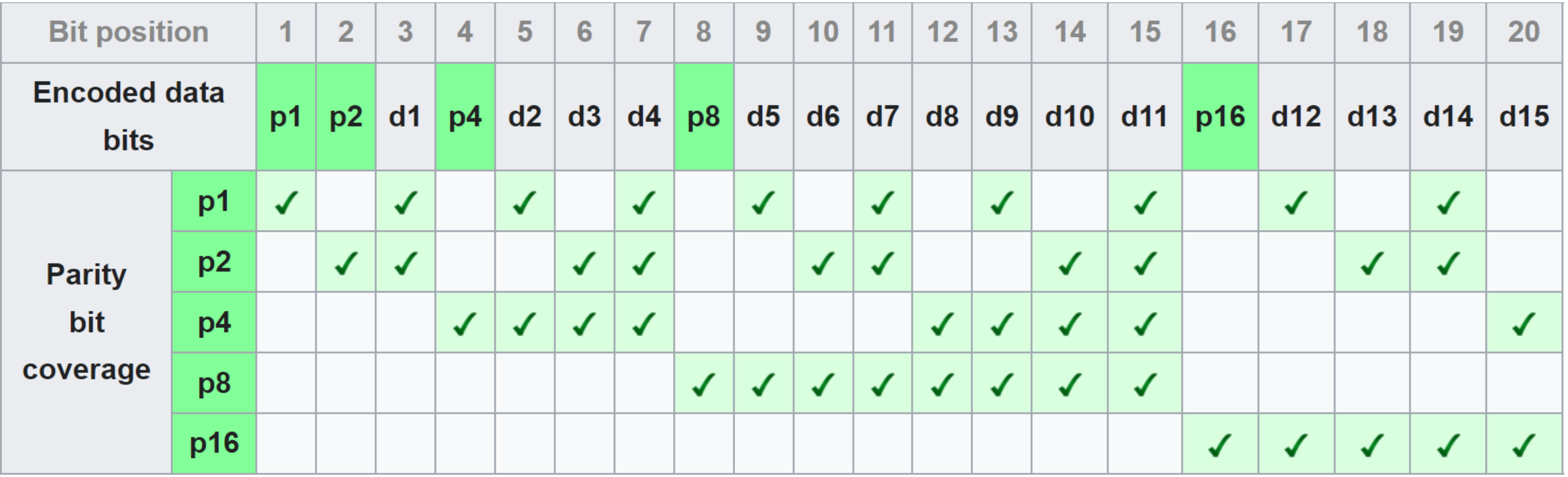
Parity bit 3: all bits where the 3rd LSB is 1. e.g. 4-7, 12-15, 20-23, …

Parity bit 4: all bits where the 4th LSB is 1. e.g. 8-15, 24-31, 40-47, …

This is illustrated on the next slide.

1. Hamming Code [2/2]

Parity Coverage



Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15
Parity bit coverage	p1	✓		✓		✓		✓		✓		✓		✓		✓		✓		✓	
	p2		✓	✓			✓	✓			✓	✓			✓	✓			✓	✓	
	p4				✓	✓	✓	✓					✓	✓	✓	✓					✓
	p8								✓	✓	✓	✓	✓	✓	✓	✓					
	p16																✓	✓	✓	✓	✓

If data to be encoded is 10011010, then the data word is p_1-p_2-D_1-p_4-D_2-D_3-D_4-p_8-D_5-D_6-D_7-D_8 would be p_1-p_2-1-p_4-0-0-1-p_8-1-0-1-0, which is 0-1-1-1-0-0-1-0-1-0-1-0.

Problem: Write the Hamming Code Algorithm and compute parity for 11010110. Do it for Hamming(7,4) for a message of your choice – 4-bit message with 3 parity bits. **Solution:** DIY.

Problem: How would you write a Hamming(8,4) for the message you chose?

1. Cyclic Redundancy Code [1/5]

Comparison with HC

CRC	HC
Block Code[1]	
Systematic Code[2]	
Not useful for protecting against intentional errors ∵ no authentication and easy reversibility	
Useful against burst errors	Not useful against burst errors
Code length is chosen a priori	Code length increases with data length
Code arrived after polynomial division	Code arrived after positional parity checks
Design challenge: finding the right polynomial	Design challenge: none
More efficient than HC	-

[1] *Block code operates on data blocks as opposed to individual bits*

[2] *Systematic Clock code is a code where the code and data are embedded in the encoded output*

1. Cyclic Redundancy Code [2/5]

Some Polynomials

Name	Use	Polynomial
CRC-1 [aka parity bit]	hardware	$x + 1$
CRC-5-USB	USB token	$x^5 + x^2 + 1$
CRC-8-Bluetooth	Wireless connectivity	$x^8 + x^7 + x^5 + x^2 + x + 1$
CRC-16-CCIT	Bluetooth, PACTOR HF Radio	$x^{16} + x^{12} + x^5 + 1$
CRC-30	CDMA	$x^{30} + x^{29} + x^{21} + x^{20} + x^{15} + x^{13} + x^{12} + x^{11} + x^8 + x^7 + x^6 + x^2 + x + 1$
CRC-32Q	Aviation AIXM	$x^{32} + x^{31} + x^{24} + x^{22} + x^{16} + x^{14} + x^8 + x^7 + x^5 + x^3 + x + 1$
CRC-40-GSM	GSM Channel Control	$(x^{23} + 1)(x^{17} + x^3 + 1)$

1. Cyclic Redundancy Code [3/5]

CRC Approach

I

STEP 1: Write the data in bit-form. E.g., If message is “I AM” then 01001001 00100000 01100001 01001101

A M

STEP 2: Choose a polynomial. E.g. $x^5 + x^2 + 1$

STEP 3: Write the polynomial in binary form with appropriate padding. E.g. 100101

STEP 4: Left align the data and the polynomial below it ∍ leftmost 1 of the polynomial aligns with the leftmost ‘1’ in the data. E.g. 01001001 00100000 01100001 01001101

100101

STEP 5: Do bit-wise division, i.e., XOR. Thus 01001001 00100000 01100001 01001101

100101 *& copy the rest of the code below*

00000011 00100000 01100001 01001101

STEP 6: Shift the divisor and left-align like in Step 4.

STEP 7: Keep repeating till the division is completed …

1. Cyclic Redundancy Code [4/5]

CRC Approach

```
01001001 00100000 01100001 01001101
 100101
00000011 00100000 01100001 01001101
      10 0101
00000001 01110000 01100001 01001101
       1 00101
00000000 01011000 01100001 01001101
          100101
00000000 00010010 1100001 01001101
            10010 1
00000000 00000000 01100001 01001101
                  100101
00000000 00000000 00101011 01001101
                   100101
0000000000 000000 00001110 01001101
```

```
0000000000 000000 00001110 01001101
                      1001 01
0000000000 000000 00000111 00001101
                       100 101
0000000000 000000 00000011 00001101
                        10 0101
0000000000 000000 00000001 01011101
                         1 00101
0000000000 000000 00000000 01110101
                            100101
0000000000 000000 00000000 00111111
                             100101
0000000000 000000 00000000 00011010
```

However as 100101 > ~~0000000000 000000 00000000 000~~11010, no more integer polynomial division is possible.

1. Cyclic Redundancy Code [5/5]

CRC Approach

STEP 9: Append to the data to the left of the remainder of the polynomial division, i.e.,

- for the data I AM
- whose binary representation is 01001001 00100000 01100001 01001101
- where the CRC polynomial is 100101
- the CRC-encoded stream is 01001001 00100000 01100001 01001101 11010

Problem: What is the CRC-encoded stream for a data stream 11100010 01010100 when the polynomial is $x^4 + x^2 + x + 1$?

Solution: DIY

1. Convolution Code [1/6]

Convolution Code: a block of ‘n’ code digits generated by the encoder over a certain time depends upon a block of ‘k’ message digits within that time frame and ‘m-1’ message digits from the preceding time.

Input weights are stored in fixed-length shift registers where the shifting happens in time.

M_t	M_{t+1}	M_{t+2}

M_t is the current time point message bit.

M_{t+1} was the message bit that was current 1 unit of time back, etc.

Let $x_1 = [M_t \oplus M_{t+1} \oplus M_{t+2}]$ where $\oplus$ is the ‘modulo 2’ addition and

Let $x_2 = [M_t \oplus M_{t+2}]$ Here

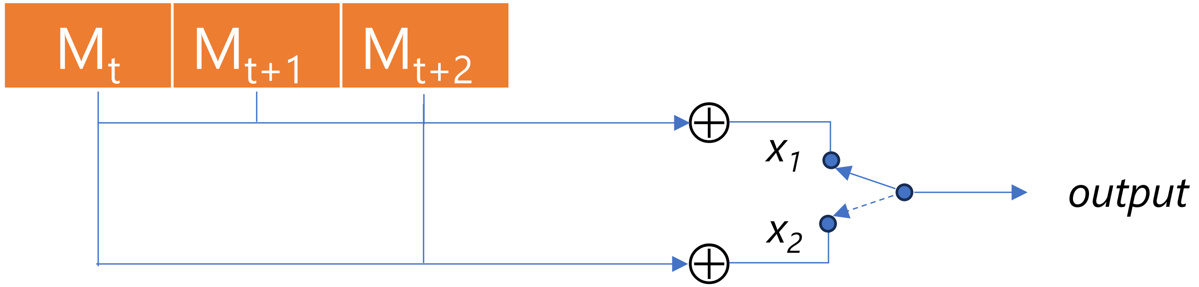
- K = 3 (*constraint length which is # of message bits that can influence encoded output*),
- k = 1 (*# of message bits*) &
- n = 2 (*# of encoded output bits for one message bits*)

Output is first x_1 and then x_2, i.e., $X = [x_1x_2]_t[x_1x_2]_{t+1}[x_1x_2]_{t+2}$ …

Code Rate: k/n = ½ in the example above.

1. Convolution Code [2/6]

Convolution Code: Encoder State, Code Tree



m_2	m_1	Encoder State
0	0	a
0	1	b
1	0	c
1	1	d

same as shift register state

The encoder state is used to generate the code tree. In a code tree, each branch of the tree represents an input symbol with the corresponding pair of output symbol on the branch.

E.g., for message M_t = 110. Initialize $\{M_{t+1}, M_{t+2}\} = \{0, 0\}$. Thus, $\boxed{M_t}\boxed{M_{t+1}}\boxed{M_{t+2}}$ is $\boxed{1}\boxed{0}\boxed{0}$.

Then, $x_1 = [M_t \oplus M_{t+1} \oplus M_{t+2}] = [1 \oplus 0 \oplus 0] = 1$ &

$x_2 = [M_t \oplus M_{t+2}] = [1 \oplus 0] = 1$. Thus, $[x_1x_2]_t = [1\ 1]$

After shifting, M_t is again 1 & right shifting $\boxed{1}\boxed{0}\boxed{0}$ gives $\boxed{}\boxed{0}\boxed{0}$. As $M_t = 1$, $\boxed{}\boxed{0}\boxed{0}$

becomes $\boxed{1}\boxed{1}\boxed{0}$.

1. Convolution Code [3/6]

To draw code tree, if input = 0, go to upper branch, else, go to lower branch.

Problem: Write code tree for the message 110

Solution:

1	0	0
M_t	M_{t+1}	M_{t+2}

$X_1 = 1$
$X_2 = 1$

1	1	0
M_t	M_{t+1}	M_{t+2}

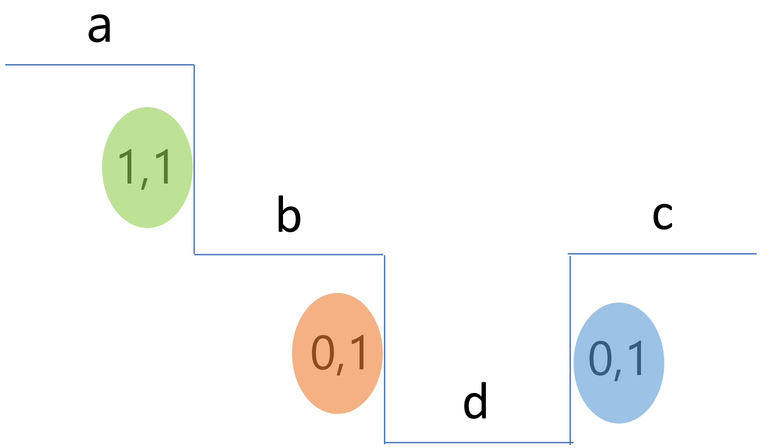
$X_1 = 0$
$X_2 = 1$

0	1	1
M_t	M_{t+1}	M_{t+2}

$X_1 = 0$
$X_2 = 1$

	0	1
	M_{t+1}	M_{t+2}

M_2M_1 denotes the state, e.g., a



m_2	m_1	Encoder State
0	0	a
0	1	b
1	0	c
1	1	d

1. Convolution Code [4/6]

	initial state		final state	
	of shift registers			
	m_2	m_1	m_2	m_1
a	0	0	0	0
b	0	1	0	1
c	1	0	1	0
d	1	1	1	1

Code Tree & Trellis Diagrams: The state of the shift register & output, $\{x_1, x_2\}$, changes every time a new message bit arrives. The goal of code tree or Trellis diagrams is to serve as a conceptual aid for understanding

- *changes to the state of the shift registers and,*
- *outputs* x_1 & x_2

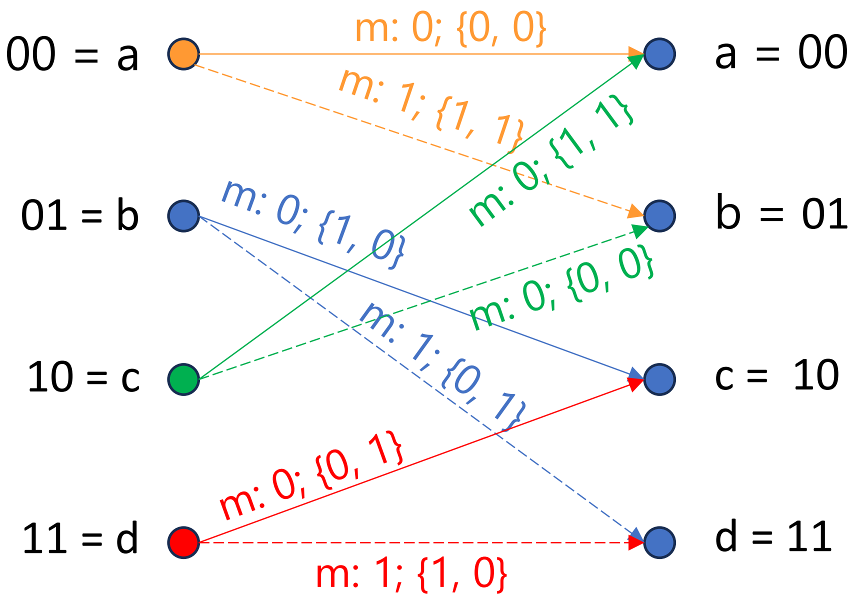
Represent '0', in a new incoming message bit with a solid arrow and, '1' with a dashed arrow.

At state **0**0, if incoming message is '0' then 00**0**
Then, 0 ⊕ 0 ⊕ **0** is 0 and 0 ⊕ **0** is 0. Thus, output is {0,0} & the new state, obtained by right shifting by 1 bit is 00.

If incoming message is '1' then 10**0**
Then, 1 ⊕ 0 ⊕ **0** is 1 and 1 ⊕ **0** is 1. Thus, output is {1,1} & the new state, obtained by right shifting by 1 bit is 10, i.e., $\{m_1,m_2\}$ thus, $\{m_2,m_1\}$ is 01.

At state **1**0, if incoming message is '0' then 00**1**
Then, 0 ⊕ 0 ⊕ **1** is 1 and 0 ⊕ **0** is 1. Thus, output is {1,1} & the new state, obtained by right shifting by 1 bit is 00.

If incoming message is '1' then 10**1**
Then, 1 ⊕ 0 ⊕ **1** is 0 and 1 ⊕ **1** is 0. Thus, output is {0,0} & the new state, obtained by right shifting by 1 bit is 10, i.e., $\{m_1,m_2\}$ thus, $\{m_2,m_1\}$ is 01.



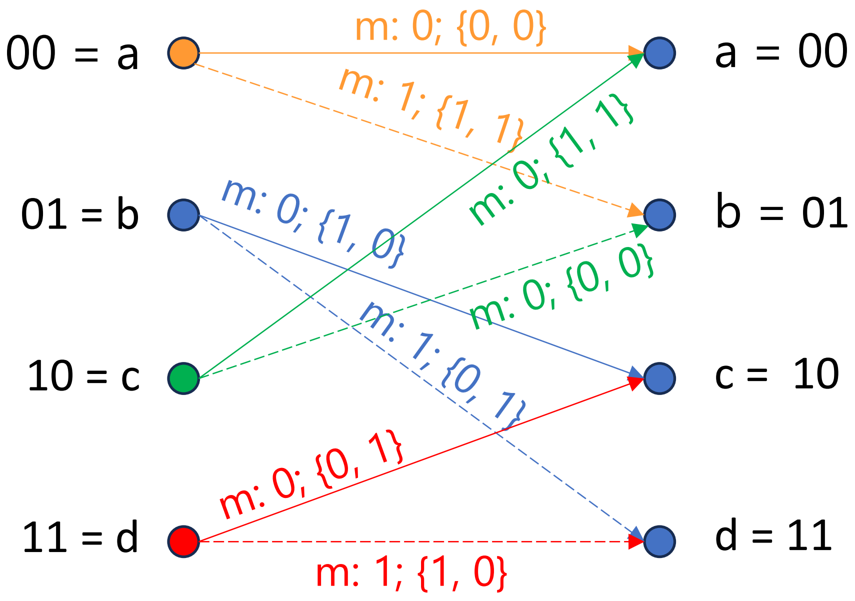
1. Convolution Code [5/6]

Viterbi Algorithm for Decoding Convolution Codes: Decode the convolution code sequence S = 11 01 10.

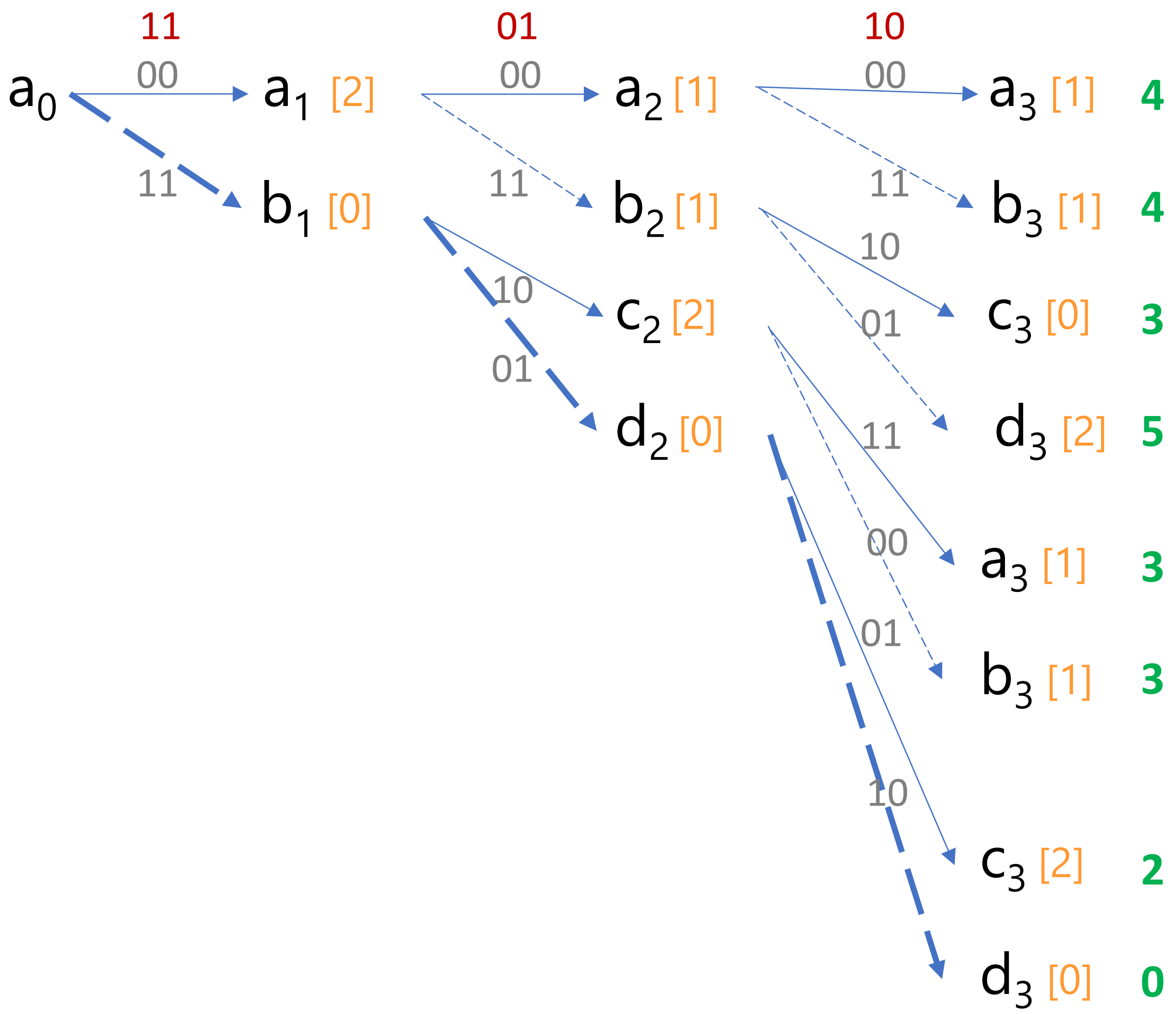
Step 1: Draw the Trellis diagram – it is not necessary but useful as a reference.

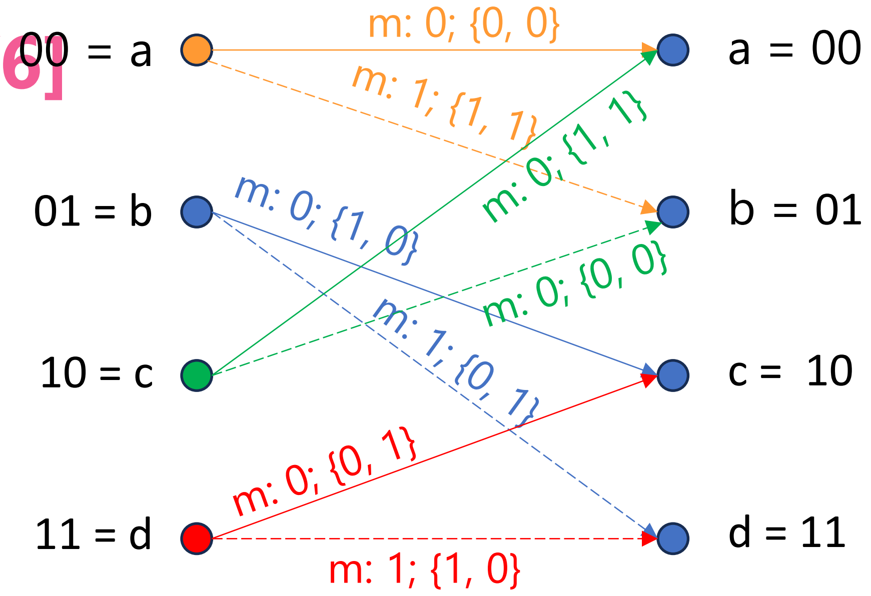
Step 2: Find the number of message bits – it is [# of message bits in S] ÷ 2 ∵ for each message bit we have 2 outputs
∴ 6/2 = 3.

Step 3: Rewrite the message tree as shown in the next slide



1. Convolution Code [6/6]





Step 4: Message is 111 as that is the minimum Hamming distance path to the bit tree leaves.