CS122A: Intermediate Embedded and Real Time Operating Systems

Jeffrey McDaniel

University of California, Riverside

Error Detection

- Errors can be caused by Electrostatic Discharge (ESD)

Error Detection

- Errors can be caused by Electrostatic Discharge (ESD)
- Preventative measures can be taken, but errors are inevitable

Error Detection

- Errors can be caused by Electrostatic Discharge (ESD)
- Preventative measures can be taken, but errors are inevitable
- Errors are generally detected using inserted redundancy

Error Detection

- Errors can be caused by Electrostatic Discharge (ESD)
- Preventative measures can be taken, but errors are inevitable
- Errors are generally detected using inserted redundancy
- Additional bits are added to the end of the data to detect when errors occur

Error Detection and Correction Schemes

- Redundancy checks

Error Detection and Correction Schemes

- Redundancy checks
  - Parity

Error Detection and Correction Schemes

- Redundancy checks
  - Parity
  - Checksum

Error Detection and Correction Schemes

- Redundancy checks
  - Parity
  - Checksum
- Gray code

Error Detection and Correction Schemes

- Redundancy checks
  - Parity
  - Checksum
- Hamming Distance
- Gray code

Error Detection and Correction Schemes

- Redundancy checks
  - Parity
  - Checksum
- Hamming Distance
- Gray code
- Forward Error Correction

Parity Bits

7 bits of data	(Count of 1 bits)	8 bits including parity	
		even	odd
000 0000	0	000 0000**0**	000 0000**1**
101 0001	3	101 0001**1**	101 0001**0**
110 1001	4	110 1001**0**	110 1001**1**
111 1111	7	111 1111**1**	111 1111**0**

- A bit added to the end of binary code

Parity Bits

7 bits of data	(Count of 1 bits)	8 bits including parity	
		even	odd
000 0000	0	000 0000**0**	000 0000**1**
101 0001	3	101 0001**1**	101 0001**0**
110 1001	4	110 1001**0**	110 1001**1**
111 1111	7	111 1111**1**	111 1111**0**

- A bit added to the end of binary code
- Indicates if the number of bits with a 1 is even or odd

Parity Bits

7 bits of data	(Count of 1 bits)	8 bits including parity	
		even	odd
000 0000	0	000 0000**0**	000 0000**1**
101 0001	3	101 0001**1**	101 0001**0**
110 1001	4	110 1001**0**	110 1001**1**
111 1111	7	111 1111**1**	111 1111**0**

- A bit added to the end of binary code
- Indicates if the number of bits with a 1 is even or odd
- Used as the simplest form of error detection

Parity Bits

7 bits of data	(Count of 1 bits)	8 bits including parity	
		even	odd
000 0000	0	000 0000**0**	000 0000**1**
101 0001	3	101 0001**1**	101 0001**0**
110 1001	4	110 1001**0**	110 1001**1**
111 1111	7	111 1111**1**	111 1111**0**

- A bit added to the end of binary code
- Indicates if the number of bits with a 1 is even or odd
- Used as the simplest form of error detection
- Single errors are identified, but the error cannot be recovered without re-sending data

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	
Odd Parity	

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	A wants to transmit: 1001
Odd Parity	A wants to transmit: 1001

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1)\%2=0$
Odd Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1+1)\%2=1$

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 + 0 + 0 + 1)\%2 = 0$ A adds parity bit and sends: 10010
Odd Parity	A wants to transmit: 1001 A computes parity bit: $(1 + 0 + 0 + 1 + 1)\%2 = 1$ A adds parity bit and sends: 10011

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1)\%2=0$ A adds parity bit and sends: 10010 B receives: 10010
Odd Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1+1)\%2=1$ A adds parity bit and sends: 10011 B receives: 10011

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1)\%2 = 0$ A adds parity bit and sends: 10010 B receives: 10010 B computes parity: $(1+0+0+1+0)\%2 = 0$
Odd Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1+1)\%2 = 1$ A adds parity bit and sends: 10011 B receives: 10011 B computes parity: $(1+0+0+1+1)\%2 = 1$

Parity Bits - Successful Transmission

Type of bit parity	Successful transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1)\%2 = 0$ A adds parity bit and sends: 10010 B receives: 10010 B computes parity: $(1+0+0+1+0)\%2 = 0$ B reports correct transmission.
Odd Parity	A wants to transmit: 1001 A computes parity bit: $(1+0+0+1+1)\%2 = 1$ A adds parity bit and sends: 10011 B receives: 10011 B computes parity: $(1+0+0+1+1)\%2 = 1$ B reports correct transmission.

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	
Even Parity	

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001
Even Parity	A wants to transmit: 1001

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...**
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...**

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1**1**010
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1001**1**

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1**1**010 B computes parity: $(1 \wedge 1 \wedge 0 \wedge 1 \wedge 0) = 1$
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1001**1** B computes parity: $(1 \wedge 0 \wedge 0 \wedge 1 \wedge 1) = 1$

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1**1**010 B computes parity: $(1 \wedge 1 \wedge 0 \wedge 1 \wedge 0) = 1$ B reports incorrect transmission.
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1001**1** B computes parity: $(1 \wedge 0 \wedge 0 \wedge 1 \wedge 1) = 1$ B reports incorrect transmission.

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...**

Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1**1**01**1**

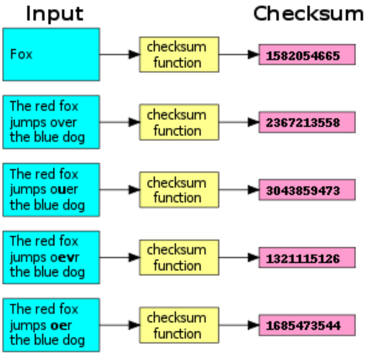
Parity Bits - Failed Transmission

Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1**1**01**1** B computes parity: $(1 \wedge 1 \wedge 0 \wedge 1 \wedge 1) = 0$

Parity Bits - Failed Transmission

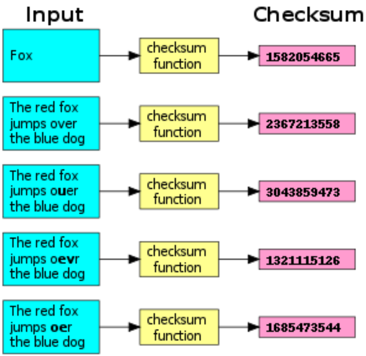
Type of bit parity	Failed transmission scenario
Even Parity	A wants to transmit: 1001 A computes parity bit: $(1 \wedge 0 \wedge 0 \wedge 1) = 0$ A adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** B receives: 1**1**01**1** B computes parity: $(1 \wedge 1 \wedge 0 \wedge 1 \wedge 1) = 0$ B reports **correct** transmission.

Checksum



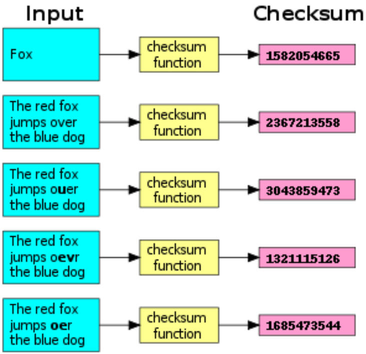
- **Checksum function** gives checksum from data input

Checksum



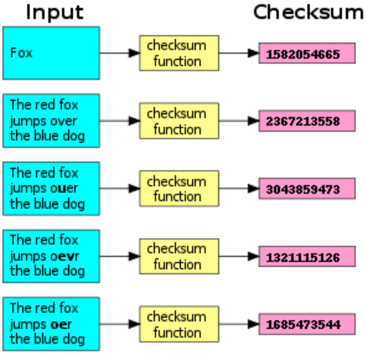
- **Checksum function** gives checksum from data input
- Produces significantly different data even for small changes

Checksum



- **Checksum function** gives checksum from data input
- Produces significantly different data even for small changes
- Parity bits are simple checksum functions

Checksum



- **Checksum function** gives checksum from data input
- Produces significantly different data even for small changes
- Parity bits are simple checksum functions
- Some error-correcting codes are based on checksums

Checksum Algorithms

- Parity byte or word

Checksum Algorithms

- Parity byte or word
  - Break data into words

Checksum Algorithms

- Parity byte or word
  - Break data into words
  - Compute **XOR** of words

Checksum Algorithms

- Parity byte or word
  - Break data into words
  - Compute **XOR** of words
  - Check if resulting word has *n* zeros

Checksum Algorithms

- Parity byte or word
- Modular sum

Checksum Algorithms

- Parity byte or word
- Modular sum
  - Variant to above

Checksum Algorithms

- Parity byte or word
- Modular sum
  - Variant to above
  - Add all "words" as unsigned binary numbers

Checksum Algorithms

- Parity byte or word
- Modular sum
- Position-dependent

Checksum Algorithms

- Parity byte or word
- Modular sum
- Position-dependent
  - Above algorithms fail to detect common errors which affect many bits

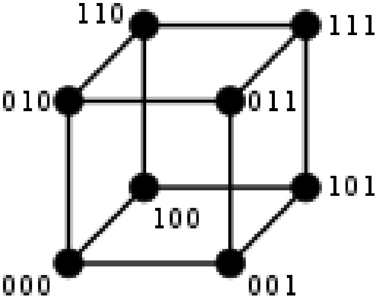
Checksum Algorithms

- Parity byte or word
- Modular sum
- Position-dependent
  - Above algorithms fail to detect common errors which affect many bits
  - Position dependent algorithms look not just at values, but position as well

Checksum Algorithms

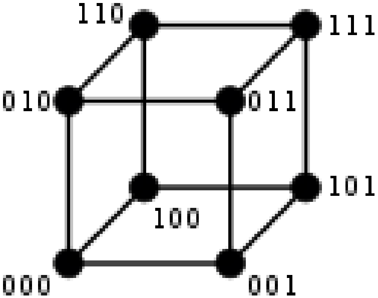
- Parity byte or word
- Modular sum
- Position-dependent
  - Above algorithms fail to detect common errors which affect many bits
  - Position dependent algorithms look not just at values, but position as well
  - Generally increases the cost of computing the checksum

Hamming Distance



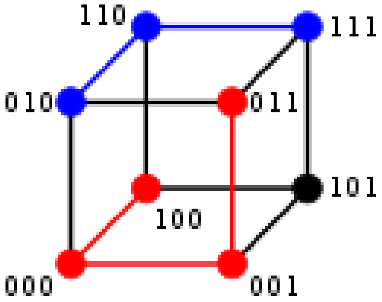
- **Hamming distance**
  between two strings of equal length is the number of positions at which they differ

Hamming Distance



- **Hamming distance** between two strings of equal length is the number of positions at which they differ
- Minimum number of *substitutions* required to change on string into the other

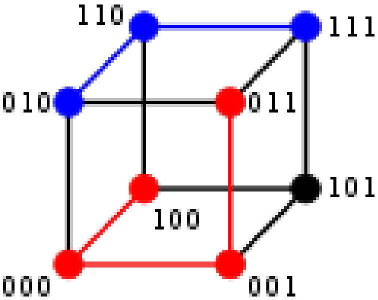
Hamming Distance



100→011 has distance 3 (red)

- **Hamming distance** between two strings of equal length is the number of positions at which they differ
- Minimum number of *substitutions* required to change on string into the other
- Or the minimum number of *errors* that could have occurred

Hamming Distance



100→011 has distance 3 (red)
010→111 has distance 2 (blue)

- **Hamming distance** between two strings of equal length is the number of positions at which they differ
- Minimum number of *substitutions* required to change on string into the other
- Or the minimum number of *errors* that could have occurred

Gray Code

2-bit	**4-bit**
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

- Also known as **reflected binary code**

Gray Code

2-bit	**4-bit**
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

- Also known as **reflected binary code**
- Named after Frank Gray of Bell Labs (1887 - 1969)

Gray Code

2-bit	4-bit
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

- Also known as **reflected binary code**
- Named after Frank Gray of Bell Labs (1887 - 1969)
- Changes only one bit at a time

Gray Code

2-bit	**4-bit**
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

- Also known as **reflected binary code**
- Named after Frank Gray of Bell Labs (1887 - 1969)
- Changes only one bit at a time
- Hamming Distance of one for adjacent values

Gray Code

2-bit	**4-bit**
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

- Also known as **reflected binary code**
- Named after Frank Gray of Bell Labs (1887 - 1969)
- Changes only one bit at a time
- Hamming Distance of one for adjacent values
- Gray codes are “cyclic”

Gray Code

2-bit	**4-bit**
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

- Also known as **reflected binary code**
- Named after Frank Gray of Bell Labs (1887 - 1969)
- Changes only one bit at a time
- Hamming Distance of one for adjacent values
- Gray codes are "cyclic"
- The value 7 can be changed to 0 with only one bit change

Forward Error Correction

- Data is encoded in a redundant way using an error-correcting code (ECC)

Forward Error Correction

- Data is encoded in a redundant way using an error-correcting code (ECC)
- Code allows limited number of errors to be detected

Forward Error Correction

- Data is encoded in a redundant way using an error-correcting code (ECC)
- Code allows limited number of errors to be detected
- Often those errors can be corrected without needing the data to be resent or recalculated

Forward Error Correction

- Data is encoded in a redundant way using an error-correcting code (ECC)
- Code allows limited number of errors to be detected
- Often those errors can be corrected without needing the data to be resent or recalculated
- **Systematic** codes include the data in the output without modification

Forward Error Correction

- Data is encoded in a redundant way using an error-correcting code (ECC)
- Code allows limited number of errors to be detected
- Often those errors can be corrected without needing the data to be resent or recalculated
- **Systematic** codes include the data in the output without modification
- **Non-systematic** codes modify the data before output

Forward Error Correction

Triplet received	Interpreted as
000	0 (error free)
001	0
010	0
100	0
111	1 (error free)
110	1
101	1
011	1

- Simple FEC (3,1) repetition code

Forward Error Correction

Triplet received	Interpreted as
000	0 (error free)
001	0
010	0
100	0
111	1 (error free)
110	1
101	1
011	1

- Simple FEC (3,1) repetition code
- Transmits each data bit 3 times

Forward Error Correction

Triplet received	Interpreted as
000	0 (error free)
001	0
010	0
100	0
111	1 (error free)
110	1
101	1
011	1

- Simple FEC (3,1) repetition code
- Transmits each data bit 3 times
- Errors corrected by "majority vote"

Forward Error Correction

- FEC works by “averaging” noise

Forward Error Correction

- FEC works by “averaging” noise
- Due to “risk-pooling”, FEC works well above a minimal signal-to-noise ratio

Forward Error Correction

- FEC works by "averaging" noise
- Due to "risk-pooling", FEC works well above a minimal signal-to-noise ratio
- *All-or-nothing tendency* becomes more pronounced closer to the **Shannon limit**

Forward Error Correction

- FEC works by "averaging" noise
- Due to "risk-pooling", FEC works well above a minimal signal-to-noise ratio
- *All-or-nothing tendency* becomes more pronounced closer to the **Shannon limit**

Shannon Limit

For any given degree of noise contamination
Data can be communicated nearly error-free up to a computable maximum rate

Forward Error Correction

- FEC works by "averaging" noise
- Due to "risk-pooling", FEC works well above a minimal signal-to-noise ratio
- *All-or-nothing tendency* becomes more pronounced closer to the **Shannon limit**
- Below the signal-to-noise ratio FEC systems don't work at all

Forward Error Correction

- FEC works by "averaging" noise
- Due to "risk-pooling", FEC works well above a minimal signal-to-noise ratio
- *All-or-nothing tendency* becomes more pronounced closer to the **Shannon limit**
- Below the signal-to-noise ratio FEC systems don't work at all
- More advanced techniques can be used here, or hybrid approaches

Forward Error Correction

- Two main categories of FEC codes

Forward Error Correction

- Two main categories of FEC codes
- Block codes
- Convolutional codes

Forward Error Correction

- Two main categories of FEC codes
- Block codes
  - Work on fixed-size blocks
- Convolutional codes

Forward Error Correction

- Two main categories of FEC codes
- Block codes
  - Work on fixed-size blocks
- Convolutional codes

Forward Error Correction

- Two main categories of FEC codes
- Block codes
  - Work on fixed-size blocks
  - Can be hard-decoded in polynomial time to block length
- Convolutional codes

Forward Error Correction

- Two main categories of FEC codes
- Block codes
  - Work on fixed-size blocks
  - Can be hard-decoded in polynomial time to block length
  - Hard-decision algorithm, makes a decision for each bit
- Convolutional codes

Forward Error Correction

- Two main categories of FEC codes
- Block codes
- Convolutional codes
  - Work on arbitrary length streams

Forward Error Correction

- Two main categories of FEC codes
- Block codes
- Convolutional codes
  - Work on arbitrary length streams
  - Soft-decoded, processes discretized analog signals

Forward Error Correction

- Two main categories of FEC codes
- Block codes
- Convolutional codes
  - Work on arbitrary length streams
  - Soft-decoded, processes discretized analog signals
  - Higher error-correction performance

Forward Error Correction

- Two main categories of FEC codes
- Block codes
- Convolutional codes
- Classical block codes include Reed-Solomon coding (MLC NAND) and Hamming codes (SLC NAND)

Forward Error Correction

- Two main categories of FEC codes
- Block codes
- Convolutional codes
- Classical block codes include Reed-Solomon coding (MLC NAND) and Hamming codes (SLC NAND)
- Viterbi algorithm commonly used for Convolutional codes

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits
5. Each data bit is included in a unique set of 2 or more parity bits

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits
5. Each data bit is included in a unique set of 2 or more parity bits
   - Parity bit 1 covers positions with the least significant bit set: 1, 3, 5, 7, etc.

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits
5. Each data bit is included in a unique set of 2 or more parity bits
   - Parity bit 1 covers positions with the least significant bit set: 1, 3, 5, 7, etc.
   - Parity bit 2 covers positions with the second least significant bit set: 2, 3, 6, 7, 10, 11,etc.

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits
5. Each data bit is included in a unique set of 2 or more parity bits
    - Parity bit 1 covers positions with the least significant bit set: 1, 3, 5, 7, etc.
    - Parity bit 2 covers positions with the second least significant bit set: 2, 3, 6, 7, 10, 11,etc.
    - Parity bit 4 covers all bit positions which have the third least significant bit set: 4–7, 12–15

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits
5. Each data bit is included in a unique set of 2 or more parity bits
   - Parity bit 1 covers positions with the least significant bit set: 1, 3, 5, 7, etc.
   - Parity bit 2 covers positions with the second least significant bit set: 2, 3, 6, 7, 10, 11,etc.
   - Parity bit 4 covers all bit positions which have the third least significant bit set: 4–7, 12–15
   - Parity bit 8 covers all bit positions which have the fourth least significant bit set: 8–15, 24–31, etc.

Hamming Code

1. Number the bits starting from 1: bit 1,2,3,4, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions which are powers of two are parity bits: 1, 2, 4, 8,etc.
4. All other bit positions are data bits
5. Each data bit is included in a unique set of 2 or more parity bits
    - Parity bit 1 covers positions with the least significant bit set: 1, 3, 5, 7, etc.
    - Parity bit 2 covers positions with the second least significant bit set: 2, 3, 6, 7, 10, 11,etc.
    - Parity bit 4 covers all bit positions which have the third least significant bit set: 4–7, 12–15
    - Parity bit 8 covers all bit positions which have the fourth least significant bit set: 8–15, 24–31, etc.
    - In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero

Hamming Code

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**	X		X		X		X		X		X		X		X		X		X		
	p2		X	X			X	X			X	X			X	X			X	X		...
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

- Parity bit 1 covers positions with the least significant bit set: 1, 3, 5, 7, etc.
- Parity bit 2 covers positions with the second least significant bit set: 2, 3, 6, 7, 10, 11,etc.
- Parity bit 4 covers all bit positions which have the third least significant bit set: 4–7, 12–15
- Parity bit 8 covers all bit positions which have the fourth least significant bit set: 8–15, 24–31, etc.
- In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero

Burst Error

- So far the codes have been designed to detect and correct random errors

Burst Error

- So far the codes have been designed to detect and correct random errors
- Sometimes channels introduce errors localized in short intervals

Burst Error

- So far the codes have been designed to detect and correct random errors
- Sometimes channels introduce errors localized in short intervals
- **Burst errors** can be found in storage mediums

Burst Error

- So far the codes have been designed to detect and correct random errors
- Sometimes channels introduce errors localized in short intervals
- **Burst errors** can be found in storage mediums
- Random error correction codes tend to be inefficent here