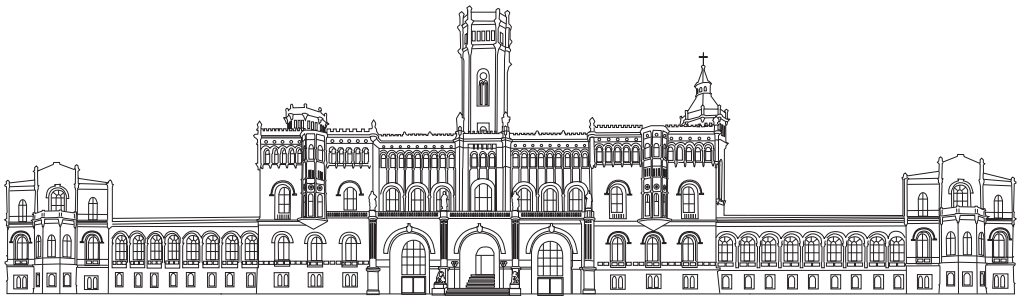
Rechnernetze - Computer Networks

Problem Set 2: Encoding & Framing

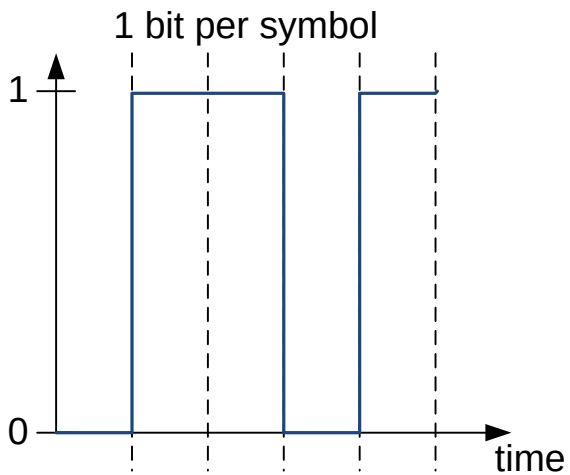
Markus Fidler, Mark Akselrod, Lukas Prause

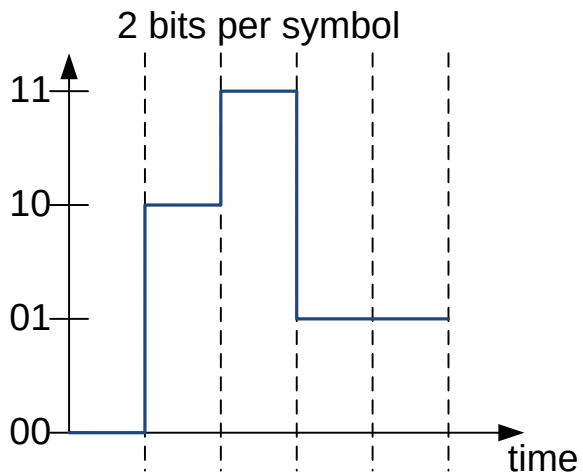


Institute of Communications Technology
Leibniz Universität Hannover

April 15, 2024

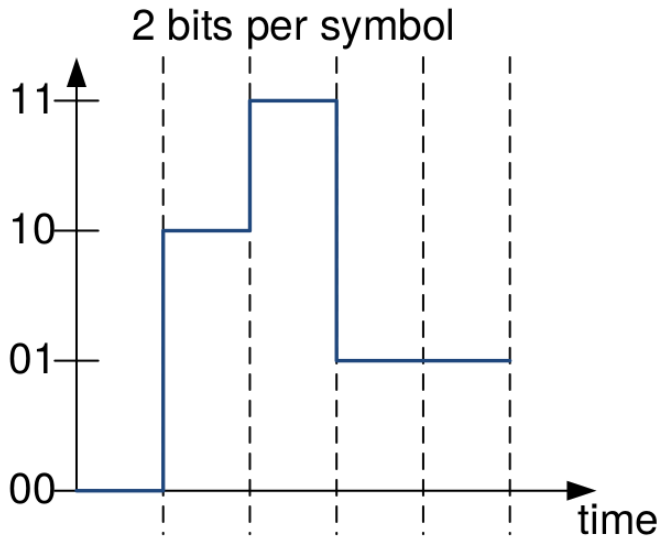
1 Bit Rate and Baud Rate





- baud rate: number of symbols transmitted per second
  - number of possible signal changes per second
  - changes in amplitude, frequency, phase
- bit rate: number of bits transmitted per second
  - bit rate = baud rate $\cdot \log_2$ (number of different symbols)

Consider a transmission over a channel with symbols transmitted every $T = 1ms$. The number of different symbols is 4, according to below illustration.



- 1.1 Calculate the baud rate and the bit rate of the example above.

$$\text{baud rate} = \frac{1}{T} = \frac{1}{1 \cdot 10^{-3} s} = 10^3 \, \frac{1}{s} = 1 \; kBd$$

symbols per second

$$\text{bit rate} = \text{baud rate} \cdot log_2(\text{number of different symbols})$$

$$\text{bit rate} = 10^3 \frac{1}{s} \cdot log_2(4) = 2 \cdot 10^3 \frac{1}{s} = 2 \; kbps$$

(kilo-)bits per second

- 1.2 Consider a different example with a bit rate of 128 *Mbps* and a baud rate of 16 *million symbols per second*. Determine the number of different symbols and the number of bits accordingly.

Hint: $y = z \cdot log_b(x) \Leftrightarrow x = b^{\frac{y}{z}}$

bit rate = baud rate $\cdot$ log_2(number of different symbols) $\Leftrightarrow$

number of different symbols = $b^{\frac{bitrate}{baudrate}}$

number of different symbols = $2^{\frac{128 \cdot 10^6 1/s}{16 \cdot 10^6 1/s}} = 2^8 = 256$

$2^{\text{number of bits}}$ = number of different symbols $\rightarrow$ 8 bits were used for the 256 different symbols.

Nyquist's theorem specifies a theoretical upper bound (Nyquist rate) for a bandwidth-limited but otherwise perfect channel with bandwidth B if a signal consists of V different discrete levels:

$$\text{maximum data rate} = 2\text{B} \cdot log_2 V \text{ bit/s}$$

Bits that are transmitted with a higher rate cannot be received correctly.

Consider a bandwidth-limited but otherwise perfect channel with bandwidth [0, B = 4 kHz]. Per baud one bit (0/1) is transfered to the receiver.

- 2.1 Determine the maximum data rate at which transmitted bits can be recovered correctly at the receiver.

  maximum data rate $= 2\ \mathrm{B}\ log_2(V)$

  number of different discrete levels: $2 \rightarrow log_2(2) = 1 \rightarrow$ maximum data rate $= 2\ \mathrm{B}$

  maximum data rate $= 2 \cdot 4\ \mathrm{kHz} = 8 \cdot 10^3\ \frac{1}{s} = 8\ \mathrm{kbps}$

- 2.2 Assume that the available bandwidth B drops to 1 kHz and to 250 Hz, (i.e., to $^1/_4 B$). Additional discrete levels V can be introduced with the goal to compensate the lower available bandwidth and to sustain the maximum data rate. In this case, which component in above equation, (i) the number of bits or (ii) the number of discrete levels, needs to be multiplied by 4 (according to the $^1/_4$ B) to achieve an equal data rate?

- ... which component in above equation, (i) the number of bits or (ii) the number of discrete levels, needs to be multiplied by 4 (according to the $^1/_4$ B) to achieve an equal data rate?

  maximum data rate $= 2\ \mathrm{B}\ log_2(V)$

  $$8\ \mathrm{kbps} = 2 \cdot 4\ \mathrm{kHz} \cdot \underbrace{log_2(2)}_{1}$$

  $$8\ \mathrm{kbps} = 2 \cdot 1\ \mathrm{kHz} \cdot \underbrace{log_2(16)}_{4}$$

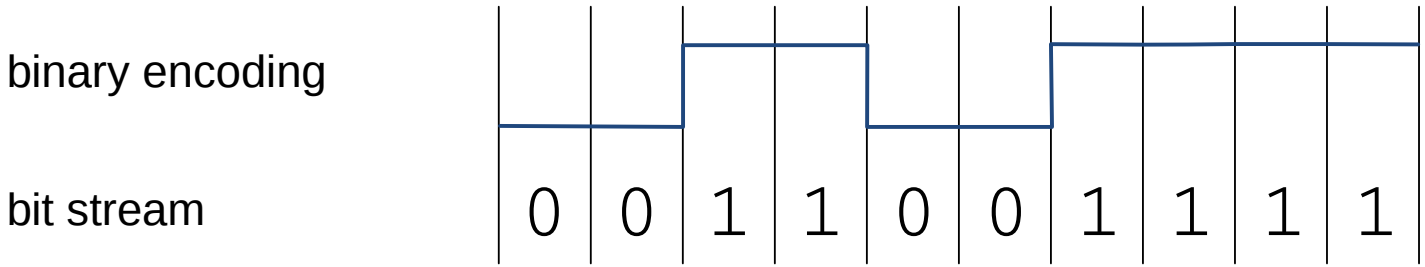
  $$8\ \mathrm{kbps} = 2 \cdot 250\ \mathrm{Hz} \cdot \underbrace{log_2(\underbrace{65536}_{\text{number of discrete levels}})}_{16\ =\ \text{number of bits}}$$

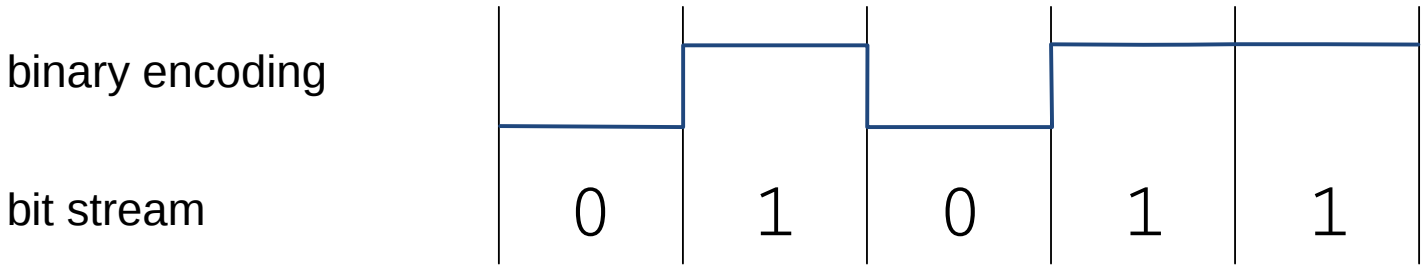
- Binary encoding: 0: low voltage, 1: high voltage
- Manchester encoding: first 1/2 of the interval T: equal to binary encoding, then transition
  - 0: low to high transition
  - 1: high to low transition

  $\rightarrow$ inherent self-clocking
- Differential Manchester encoding: only transitions matter
  - 0: transition at the beginning of the interval T
  - 1: no transition at the beginning of the interval T

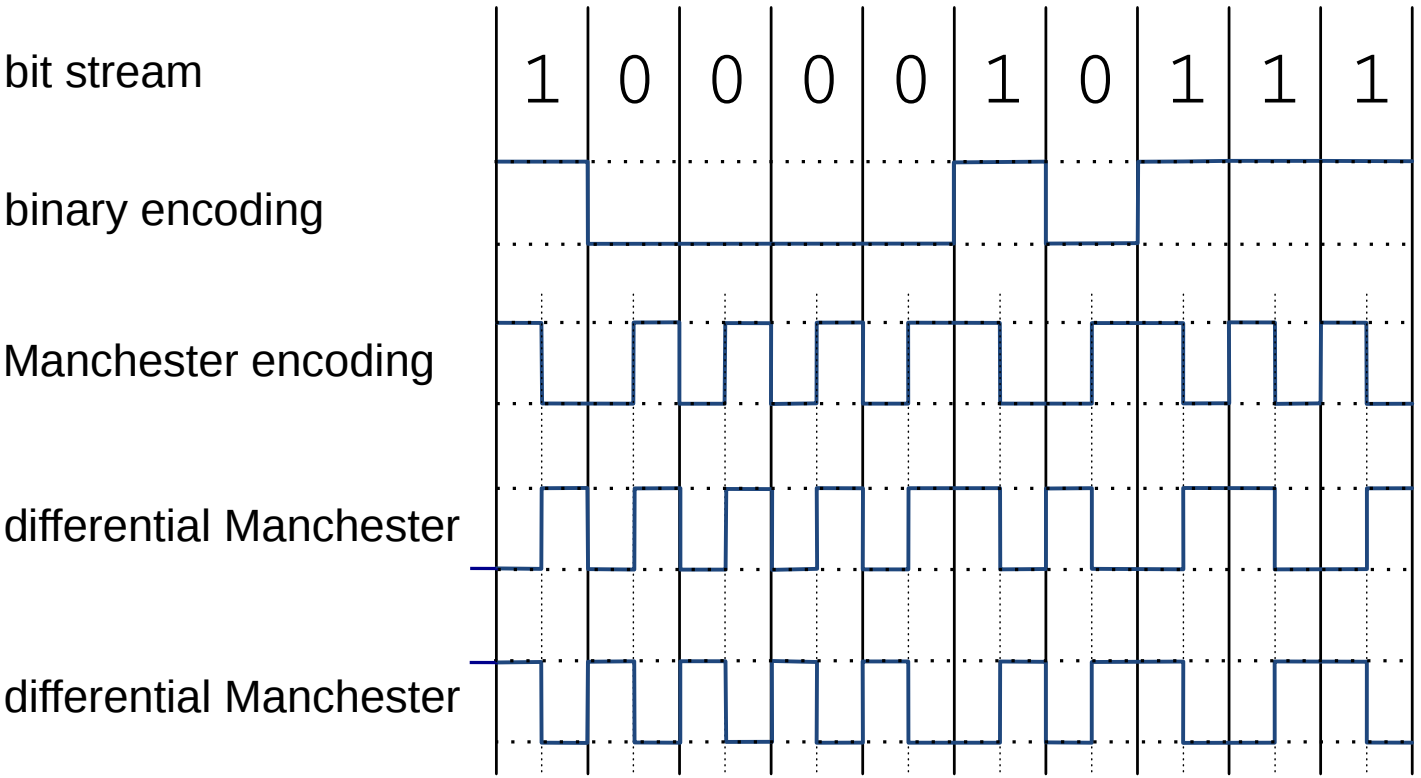
  $\rightarrow$ works with swapped wires

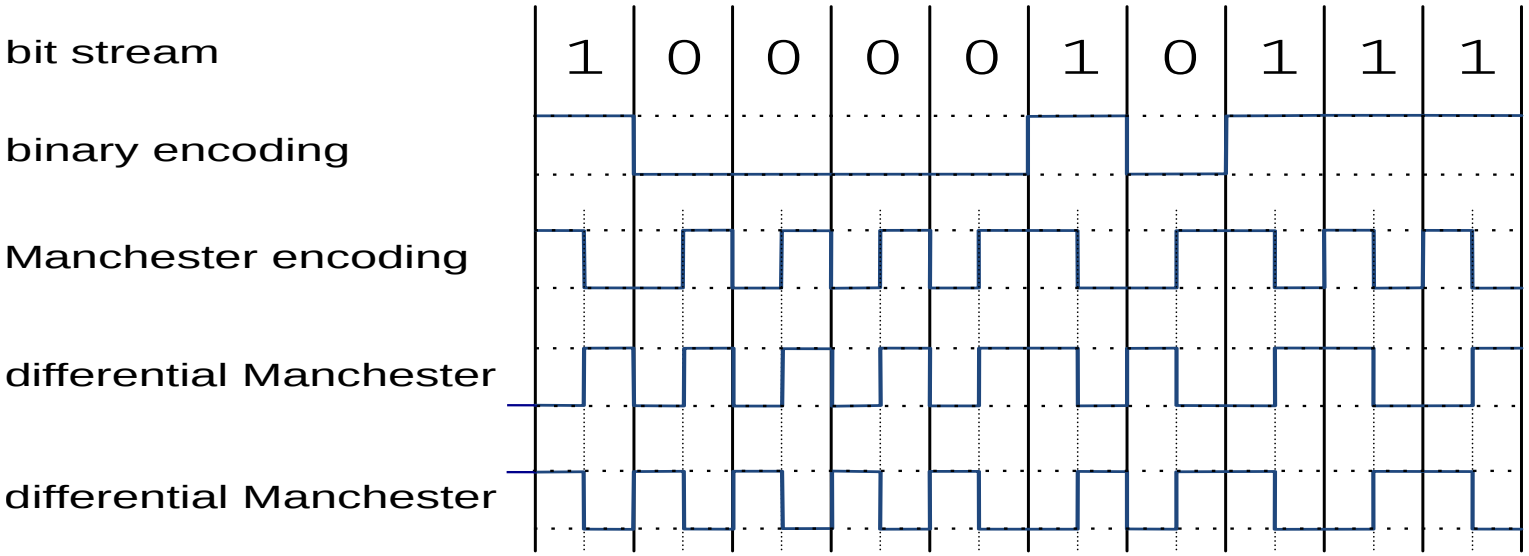
- 3.1 A receiver needs to decode the below signal and to retrieve the original bit stream as transmitted by the sender. The illustration shows two traces with a duration of 1 ms. The clock signal was (i) f = 10 kHz in the first case and (ii) f = 5 kHz in the second case. Reconstruct the received bit stream(s).





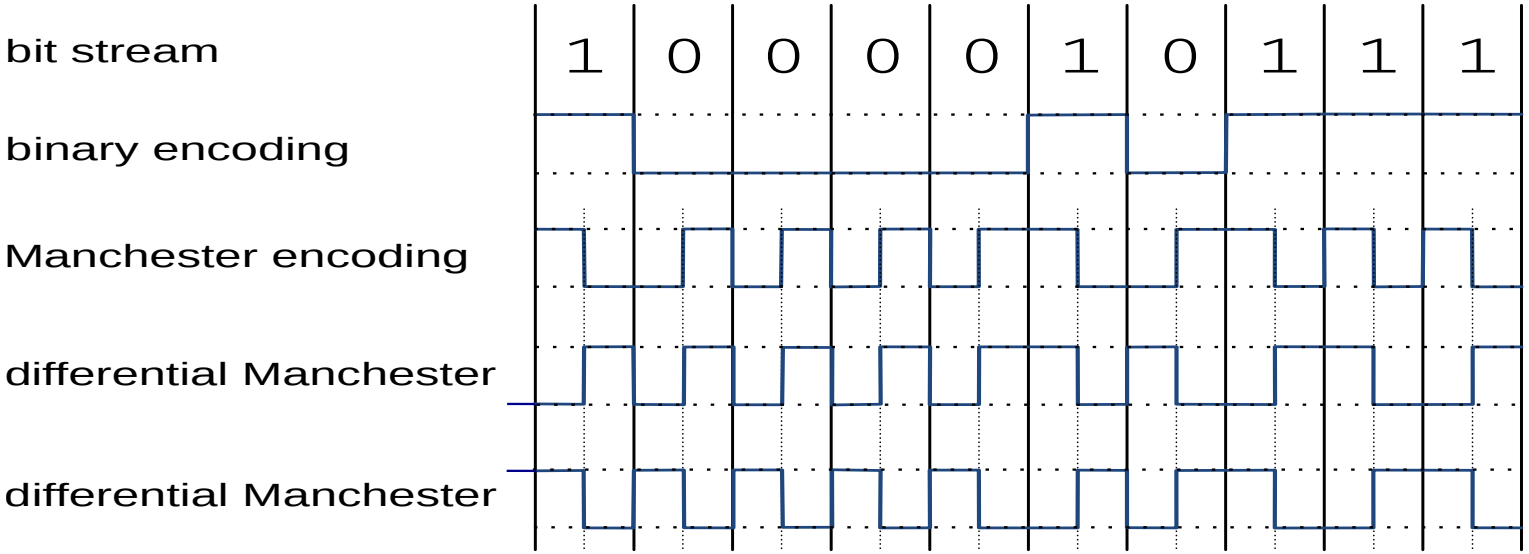
- 3.2 Consider below bit stream and a sender that uses different encoding methods to send out this data. Sketch the transmitted signal and use (i) binary encoding, (ii) Manchester encoding, and (iii), (iv) differential Manchester encoding. Take into account the different initial signal levels for (iii) and (iv).





- 3.3 What bit stream will be received if wires are swapped. Complete the following table.

Bit stream at sender	1 0 0 0 0 1 0 1 1 1
...received - binary encoding	0 1 1 1 1 0 1 0 0 0
- Manchester encoding	0 1 1 1 1 0 1 0 0 0
- differential Manchester encoding	1 0 0 0 0 1 0 1 1 1



- 3.4 Determine the baud rate and the bit rate for (ii) Manchester encoding and assume that the duration of the trace is 80 ms.

$$\text{baud rate} = \frac{1}{4ms} = 0.25 \text{ kBd}$$

$$\text{bit rate} = \frac{1}{2} \text{ baud rate} = \frac{1}{8ms} = 0.125 \text{ kbps}$$

- 3.5 Can the clock signal reliably be recovered from the received signal? Are there sequences that could yield an incorrect clock signal for (ii) Manchester encoding and (iii/iv) differential Manchester encoding?

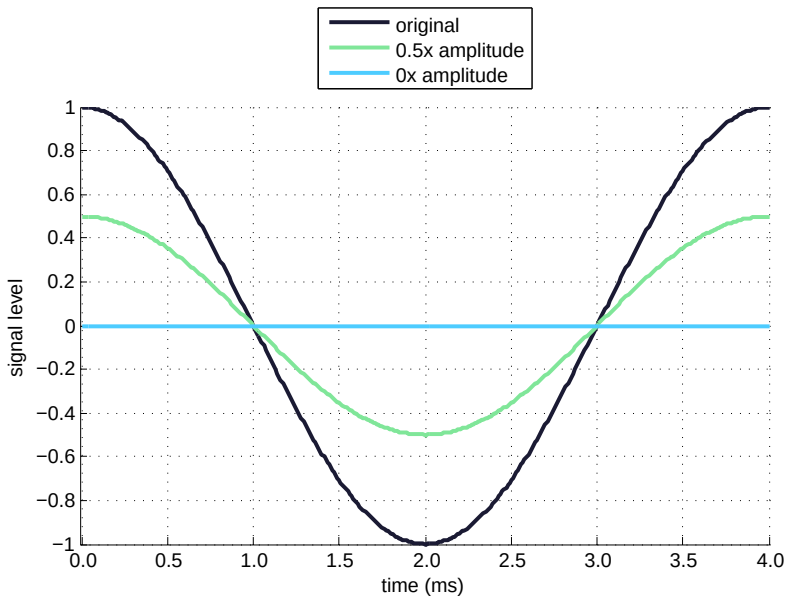
  Similarity of signals can delay clock synchronization. In the following cases, at least partial equality of signals can be observed for certain sequences if the clock frequency is halfed (resp. doubled):

  (ii) Manchester: ...01010101... $\leftrightarrow$ ...0000... or ...1111...

  (ii) differential Manchester: ...00000000... $\leftrightarrow$ ...1111...

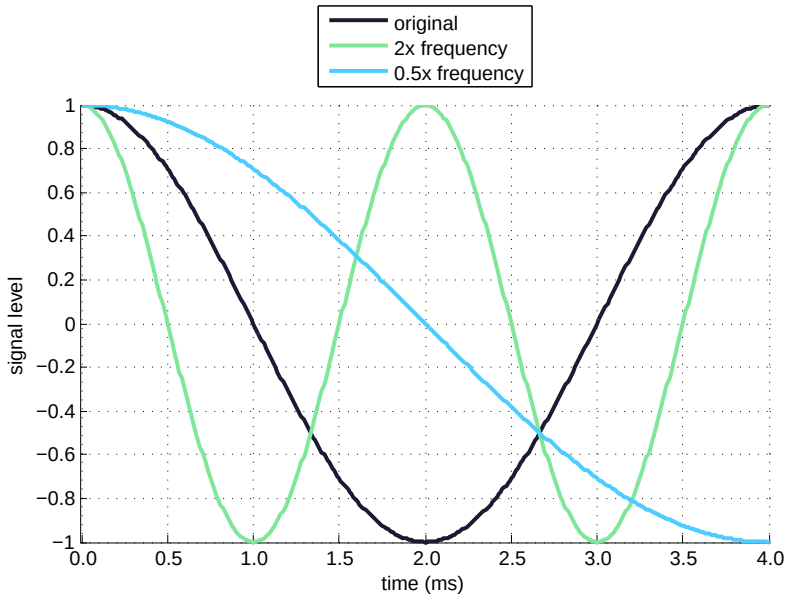
- 4.1 Below graphs show three times the following cosine function: s(t) = a · $cos(2\pi ft + \phi)$ with amplitude a = 1, frequency f = 250 Hz, and phase $\phi = 0$. Add to these graphs the same function with changed parameters: amplitude, frequency, phase - each two additional graphs.

  (i) 0.5x amplitude and 0x amplitude



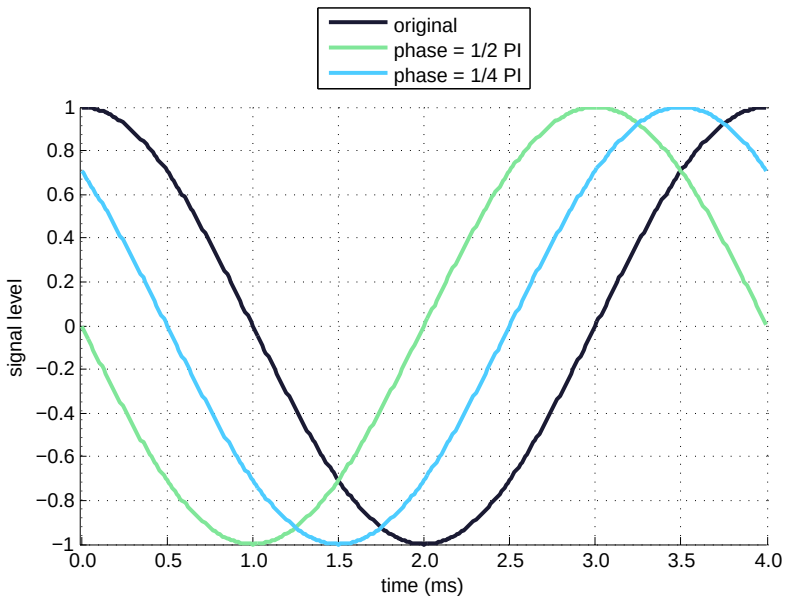
- 4.1 Below graphs show three times the following cosine function: s(t) = a · $cos(2\pi ft + \phi)$ with amplitude a = 1, frequency f = 250 Hz, and phase $\phi = 0$. Add to these graphs the same function with changed parameters: amplitude, frequency, phase - each two additional graphs.

(ii) 0.5x frequency and 2x frequency



- 4.1 Below graphs show three times the following cosine function: s(t) = a · $cos(2\pi ft + \phi)$ with amplitude a = 1, frequency f = 250 Hz, and phase $\phi = 0$. Add to these graphs the same function with changed parameters: amplitude, frequency, phase - each two additional graphs.

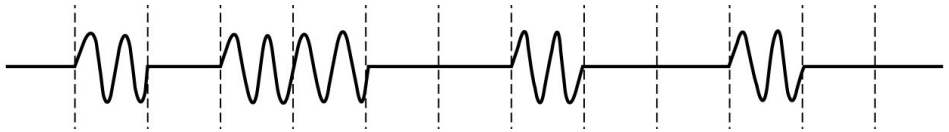
(iii) phase = 0.25π and 0.5π



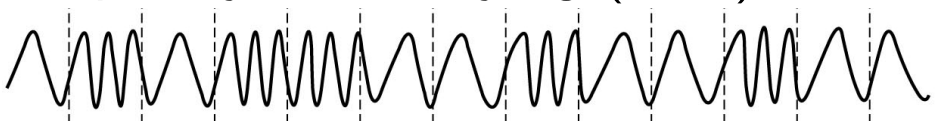
- 4.2 Name the basic modulation schemes used in the following three examples:

Phase Shift Keying (PSK)

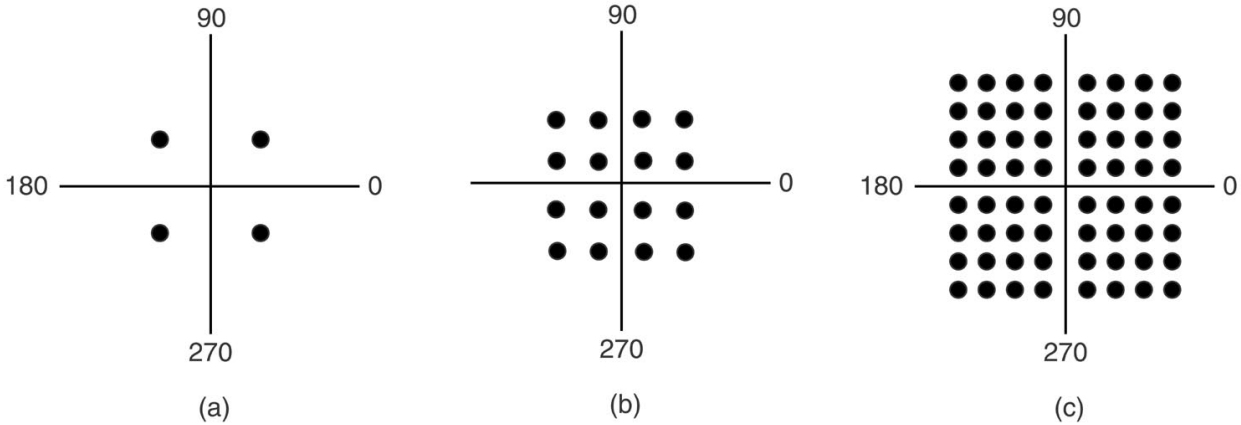
Amplitude Shift Keying (ASK)



Frequency Shift Keying (FSK)



- 4.3 Which modulation schemes are shown here and how many bits can be transmitted per symbol with these schemes?



(a): Quadrature phase shift keying (QPSK): 4 different phases $\rightarrow$ 2 bits per symbol

(b): Quadrature Amplitude Modulation (QAM): 16-QAM: 4 bits per symbol

(c): Quadrature Amplitude Modulation (QAM): 64-QAM: 6 bits per symbol

5 Flag Bytes with Bit Stuffing

- 5.1 Given is the flag 01111110 and the below payload bit pattern.
  0111011001111111
  How does the transmitted bit stream look like if the given pattern is divided into two frames of equal size (before stuffing)?

  - Divide into two frames of equal length:
  01110110 01111111

  - Do bit stuffing:
  01110110 011111011

  - Prepend/append flags:
  01111110011101100111111 01111110011111011011111110

5 Flag Bytes with Bit Stuffing

- 5.2 The following bit stream has been received.
  01111110111101111100011111100111111101111101001101000111111 0
  Given the flag 01111110, what does the payload bit pattern look like?

  - Identify and remove flags:
  01111110111101111100011111100111111101111101001101000111111 0
  →
  111101111100 111110100110100

  - Identify and remove stuffed zeros:
  111101111100 111110100110100 →
  111101111100 111111100110100