

Lecture 23: Modulation

CSE 123: Computer Networks
Alex C. Snoeren



HW 4 due Mon 12/4



Overview

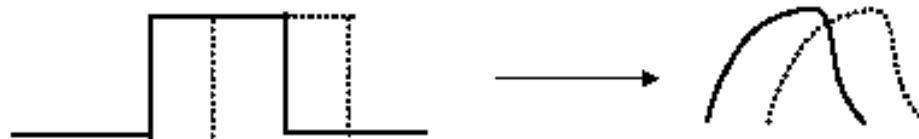


- Signaling constraints
 - ◆ Shannon's Law
 - ◆ Nyquist Limit
- Encoding schemes
 - ◆ Clock recovery
 - ◆ Manchester, NRZ, NRZI, etc.



Intersymbol Interference

- Bandlimited channels cannot respond faster than some maximum frequency f
 - ◆ Channel takes some time to settle
- Attempting to signal too fast will mix symbols
 - ◆ Previous symbol still “settling in”
 - ◆ Mix (add/subtract) adjacent symbols
 - ◆ Leads to **intersymbol interference** (ISI)

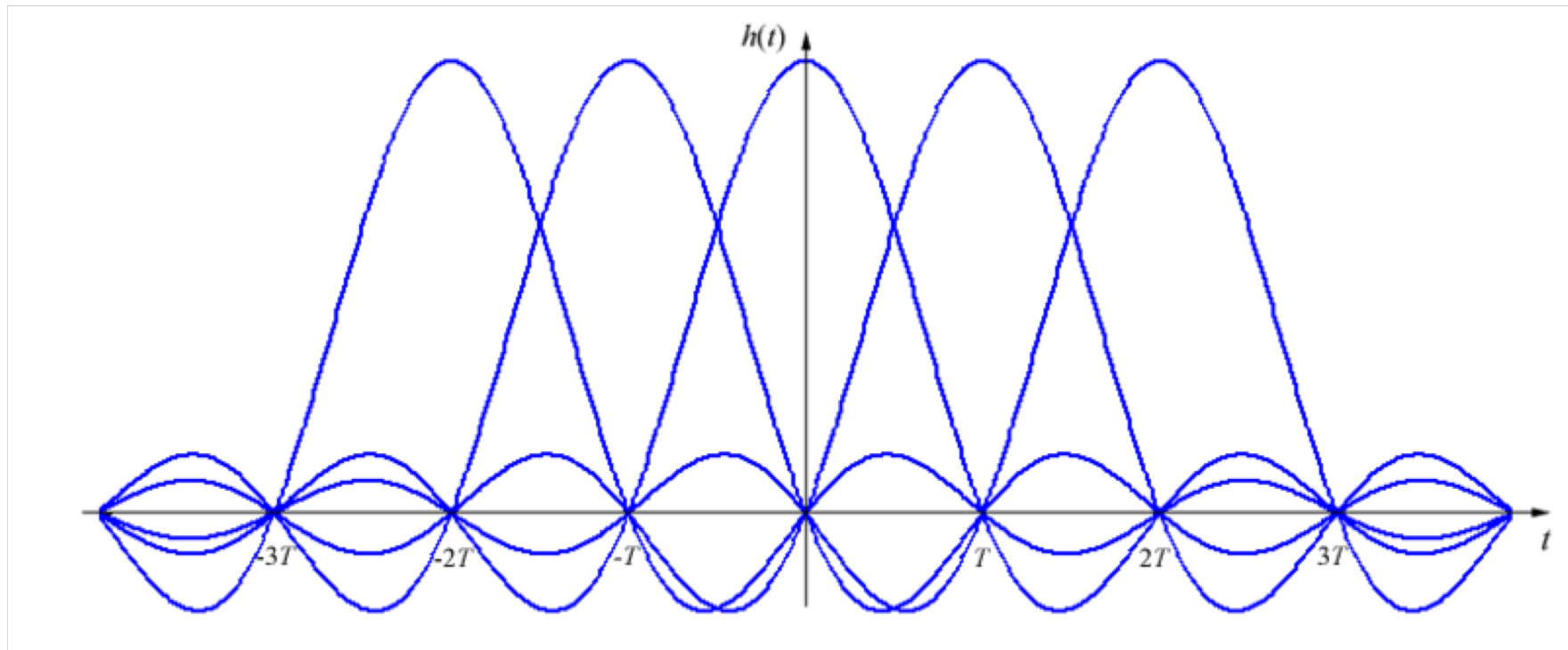


- OK, so just how fast can we send symbols?

Speed Limit: Nyquist



- In a channel bandlimited to f , we can send at maximum symbol (**baud**) rate of $2f$ without ISI





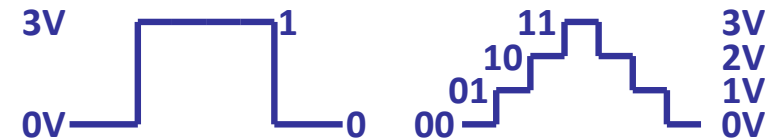
Multiple Bits per Symbol

- Nyquist limits the number of symbols per second we can send, but doesn't talk about the information content in each symbol
- Couldn't we send *multiple* bits per symbol
 - ◆ E.g., multiple voltage levels instead of just high/low
 - ◆ Four levels gets you two bits, $\log_2 M$ in general (M levels)
- Can combine this observation with Nyquist
 - ◆ *Channel capacity: $C < 2 B \log_2(M)$*
- Why not infinite levels? Infinite bandwidth no?

Noise matters



- Real channels are **noisy**... noise creates measurement challenges
- Example:
 - ◆ Encode 4 values using voltage
 - » 2 bits per symbol
 - » Symbols at 3V, 2V, 1V and 0V
 - ◆ What if noise is 0.5V?
 - » If you get line level of 2.5V then what symbol is it? 11 or 10?
- Limited to $\sim \log_2 (S/2N)$ bits per symbol (S = signal power, N = Noise)
 - ◆ Previous example: $S = 3V - 0V = 3V$, $N = 0.5V$, so we can have $\log_2(3/1) = 1.58$ bits per symbol



Shannon's Law



- Shannon considered noisy channels and derived

$$C = B \log (1 + S/N)$$

- Gives us an upper bound on any channel's performance regardless of signaling scheme
- Old school modems approached this limit
 - ◆ $B = 3000\text{Hz}$, $S/N = 30\text{dB} = 1000$
 - ◆ $C = 3000 \times \log(1001) \approx 30\text{kbps}$
 - ◆ 28.8Kbps – anyone remember dialup?

Next problem: Clock recovery



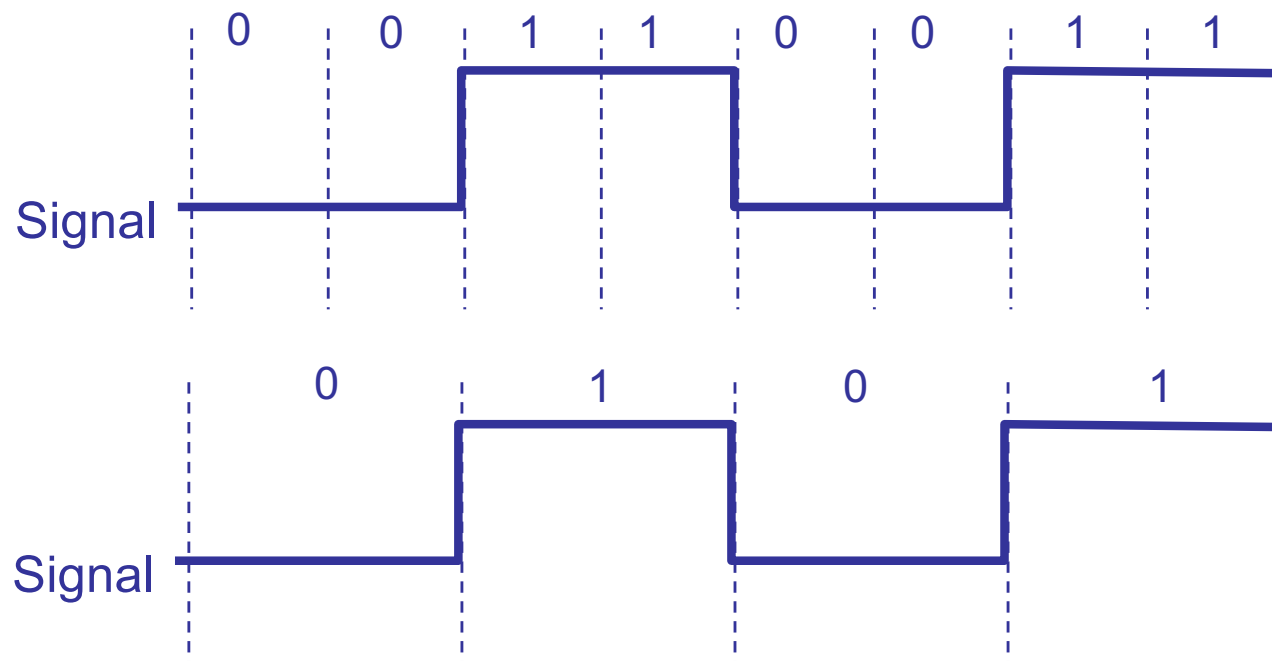
- How does the receiver know when to sample the signal?
 - ◆ Sampling rate: How often to sample?
 - ◆ Sampling phase:
 - » When to start sampling? (getting in phase)
 - » How to adjust sampling times (staying in phase)

Why sampling rate matters...



- Signal could have multiple interpretations

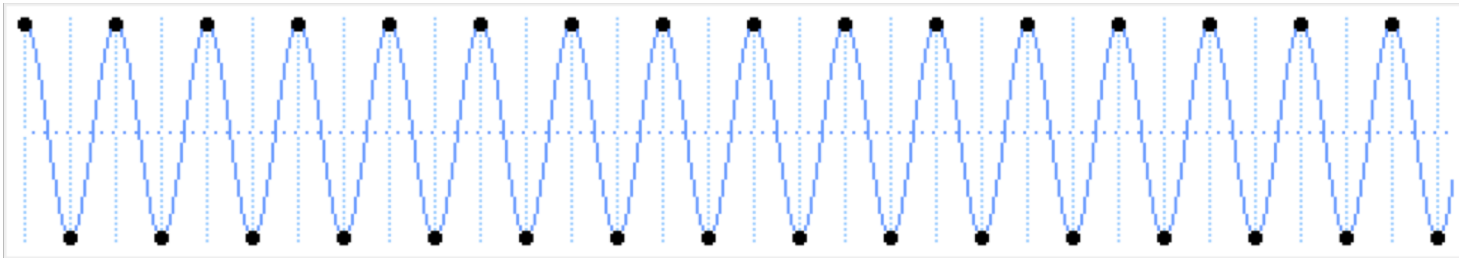
Which of these is correct?



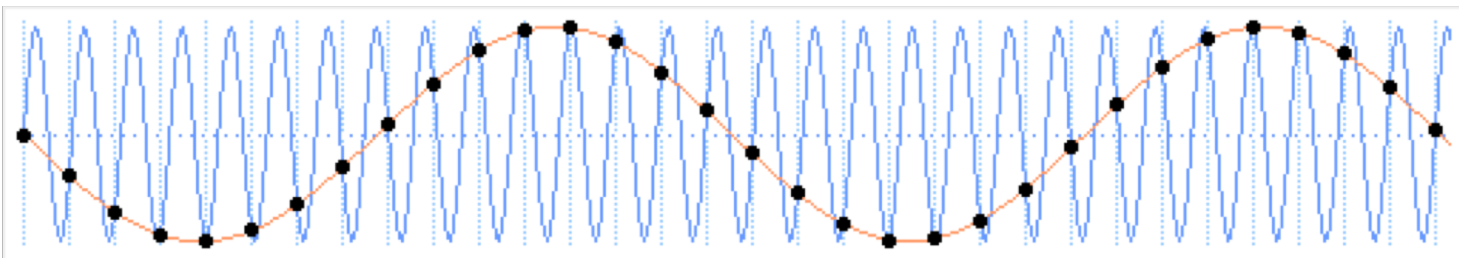
Nyquist Revisited



- Sampling at the correct rate ($2f$) yields actual signal
 - ◆ Always assume lowest-frequency wave that fits samples



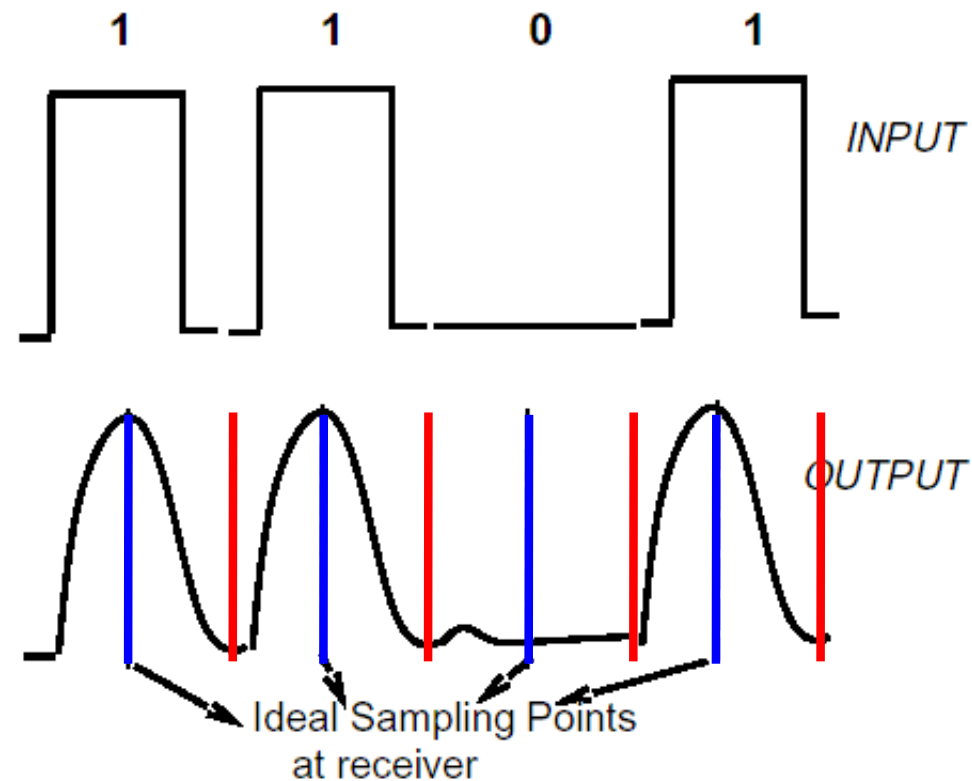
- Sampling too slowly yields aliases





The Importance of Phase

- Need to determine when to START sampling, too



Clock Recovery

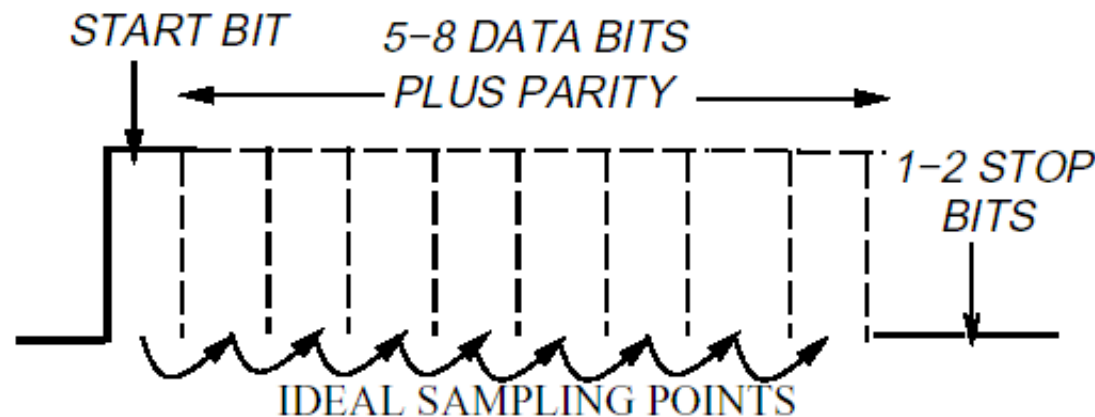


- Using a training sequence to get receiver lined up
 - ◆ Send a few, known **initial training bits**
 - ◆ Adds inefficiency: only m data bits out of n transmitted
- Need to combat clock drift as signal proceeds
 - ◆ Use **transitions** to keep clocks synched up
- Question is, how often do we do this?
 - ◆ Quick and dirty every time: **asynchronous** coding
 - ◆ Spend a lot of effort to get it right, but amortize over lots of data: **synchronous** coding

Asynchronous Coding



- Encode several bits (e.g. 7) together with a leading “start bit” and trailing “stop bit”
- Data can be sent at any time

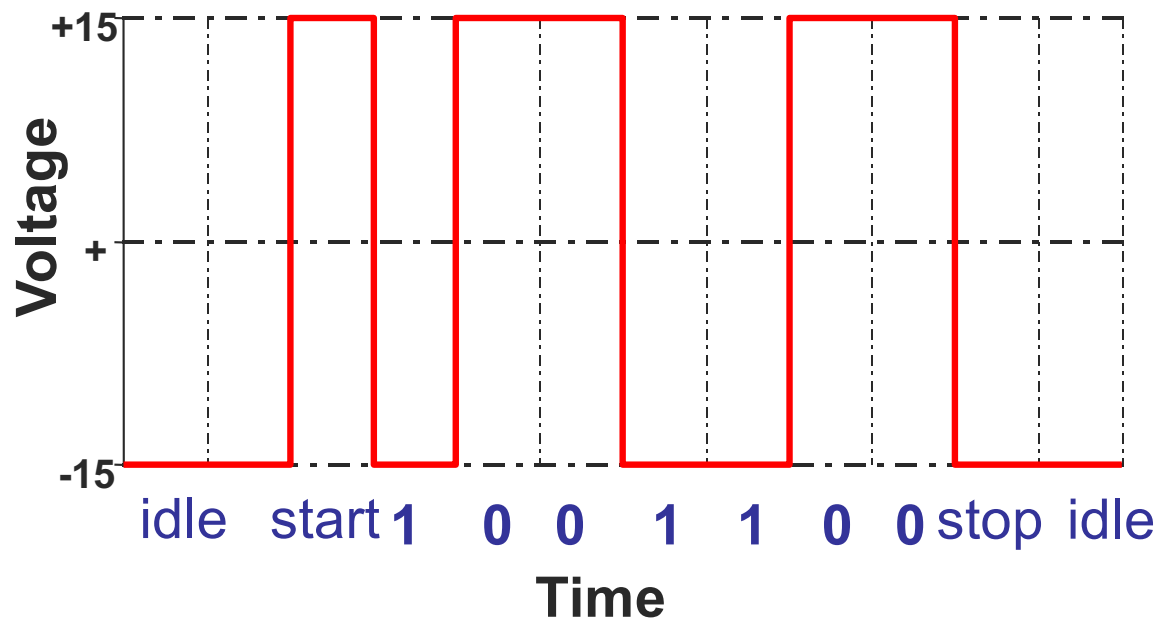


- Start bit transition kicks off sampling intervals
 - Can only run for a short while before drifting

Example: RS232 serial lines



- Uses two voltage levels (+15V, -15V), to encode single bit binary symbols
- Needs long idle time – limited transmit rate



Synchronous Coding

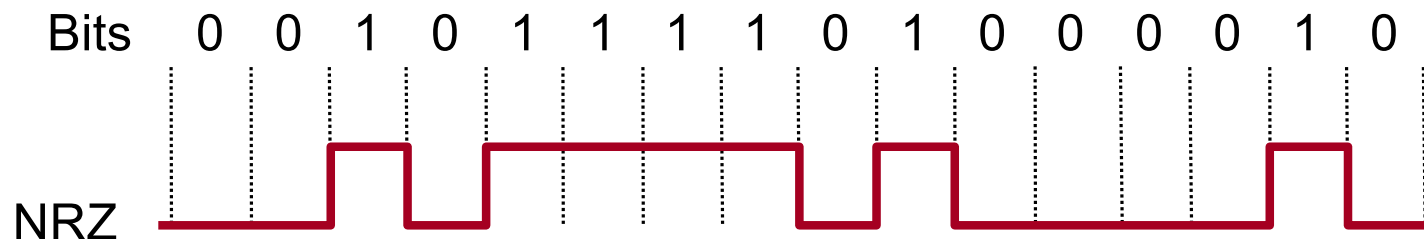


- Asynchronous receiver phase locks each symbol
 - ◆ Takes time, limiting transmission rates
- So, start symbols need to be extra slow
 - ◆ Need to fire up the clock, which takes time
- Instead, let's do this training once, then just keep sync
 - ◆ Need to continually adjust clock as signal arrives
 - ◆ Ever hear of Phase Lock Loops (PLLs) ?
- Basic idea is to use **transitions** to lock in

Non-Return to Zero (NRZ)



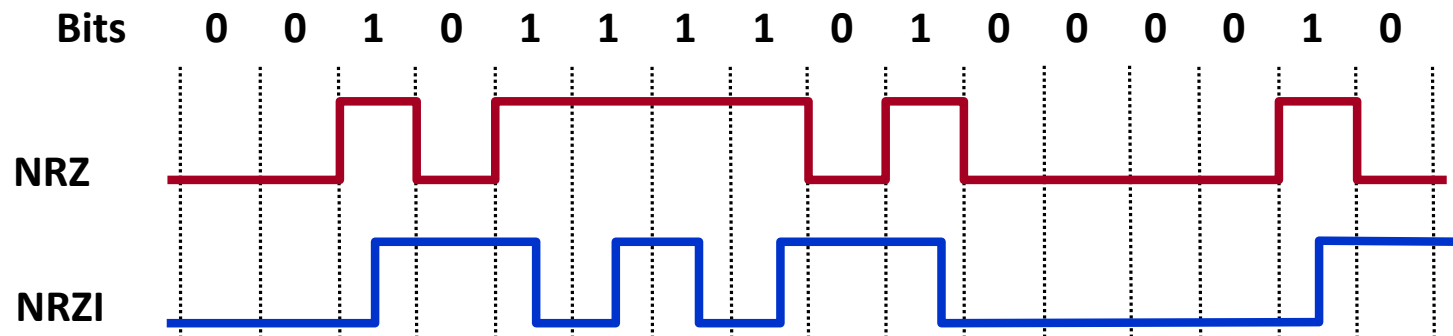
- Signal to Data
 - ◆ High $\Rightarrow$ 1
 - ◆ Low $\Rightarrow$ 0
- Comments
 - ◆ Transitions maintain clock synchronization
 - ◆ Long strings of 0s confused with no signal
 - ◆ Long strings of 1s causes *baseline wander*
 - » We use average signal level to infer high vs low
 - ◆ Both inhibit clock recovery



Non-Return to Zero Inverted (NRZI)



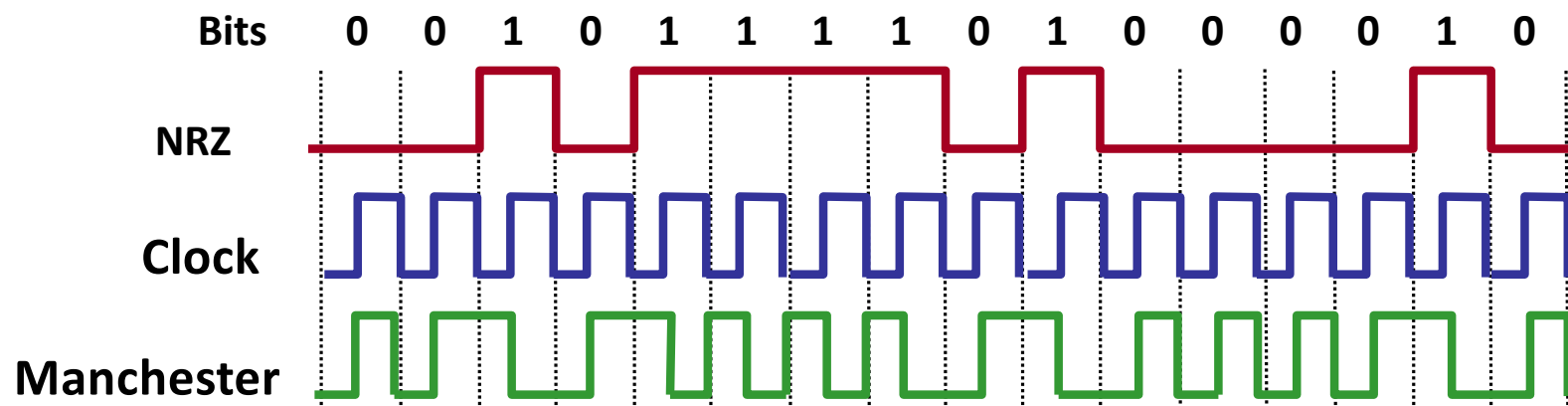
- Signal to Data
 - ◆ Transition $\Rightarrow$ 1
 - ◆ Maintain $\Rightarrow$ 0
- Comments
 - ◆ Solves series of 1s, but not 0s



Manchester Encoding (10Mbps Ethernet)



- Signal to Data
 - ◆ XOR NRZ data with senders clock signal
 - ◆ High to low transition $\Rightarrow$ 1
 - ◆ Low to high transition $\Rightarrow$ 0
- Comments
 - ◆ Solves clock recovery problem
 - ◆ Only 50% efficient ($\frac{1}{2}$ bit per transition)
 - ◆ Still need preamble (typically 0101010101... trailing 11 in Ethernet)



4B/5B (100Mbps Ethernet)



- Goal: address inefficiency of Manchester encoding, while avoiding long periods of low signals
- Solution:
 - ◆ Use five bits to encode every sequence of four bits
 - ◆ No 5 bit code has more than one leading 0 and two trailing 0's
 - ◆ Use NRZI to encode the 5 bit codes
 - ◆ Efficiency is 80%

4-bit	5-bit
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111

4-bit	5-bit
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

Encoding Summary



- Signaling & Modulation
 - ◆ Transforming digital signal to and from analog representation
 - ◆ Fundamental limits (Shannon)
 - ◆ Lots of ways to encode signal (modulation) onto a given medium
- Clock recovery
 - ◆ Receiver needs to adjust its sampling times to best extract signal from channel
 - ◆ Sender can code signal to make it far easier to do this

For Next Time



- HW 4 due next Monday
- Keep going on the project...
- No class on Friday