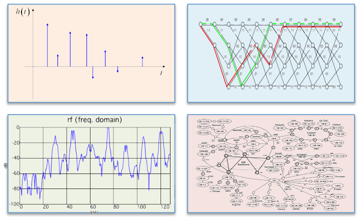


INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2011
Lecture #3

- Analog woes, and the motivation for *digital abstraction*
- Recipes for sending digital data mapped to analog signals
- Layered communication model
  - Messages → packets → bits → signals

Sources of Data (i.e., Messages)

- Computers – with input from people or from programs ("machine-generated" data)
  - Inherently digital (i.e., bit streams)
- Cameras – audio and video
  - Inherently analog (but made digital on purpose)
- Telephones, televisions, broadcast radio, walkie-talkies, … → Inherently digital? Analog?
- Sensors – GPS, accelerometers, MEMS devices, climate sensors, …
  - Inherently either digital or analog
- Regardless of the inherent nature of a source, converting to *digital* form is the modern way
- Why?

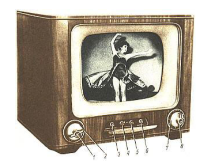
Why Digital?

- Enables composition of modules to build large systems
- Enables sophisticated processing of data
- But… physical communication links turn out to be analog at the lowest level, so we're going to have go between digital and analog and vice versa
- A story about picking the right abstractions…

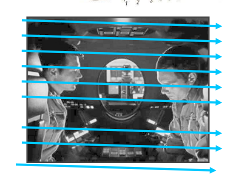
Example: Analog TV
Representing luminance with voltage

Representation of each point (x, y) on a B&W Picture:

0 volts: BLACK
1 volt: WHITE
0.367879 volts: 36.7879% white
(i.e., a shade of gray)



Representation of a picture:
Scan points in some prescribed raster order… generate voltage waveform



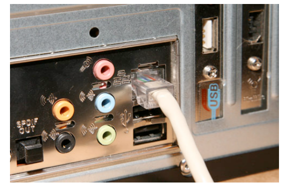
Example: Analog Telephone System

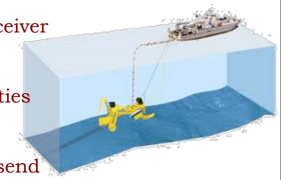
Pic from wikipedia

Sound waves → Electric signals → Sound waves

http://en.wikipedia.org/wiki/Telephone

Analog Representation Maps Well to Physical Link Capabilities





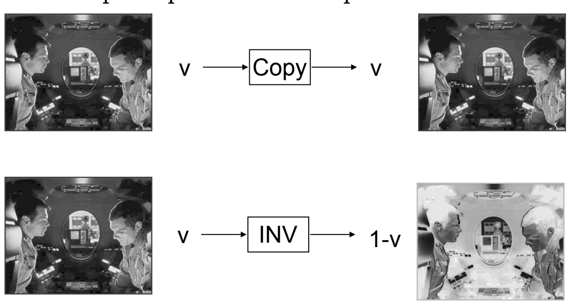
Wire: Send signals of different voltages; receiver measures voltage

Optical: send signals with different intensities (possibly at different wavelengths)

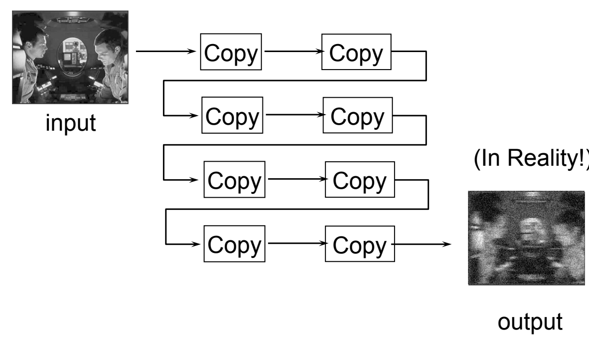
Radio/Acoustic: A bit trickier, but we can send at different amplitudes

Example Building Blocks: Two Simple Components

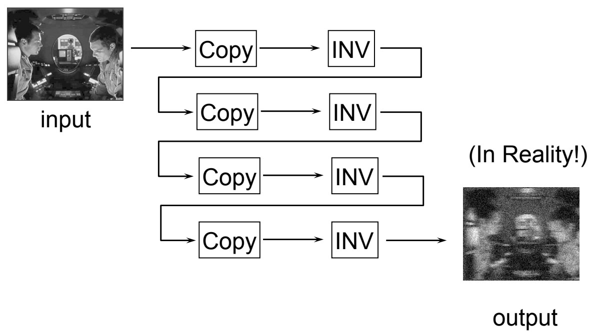
- Copy is the simplest imaginable processing element
- INVert is perhaps the next simplest



Let's build a system!



Let's build a system!



Analog Woes

.12345678 → INV $V_{OUT} = 1 - V_{IN}$ → Expected: .87654322 Actual: .87???????

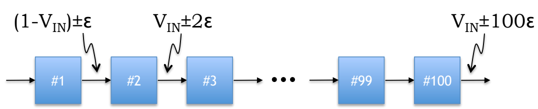
The actual value of V_{OUT} depends on many factors:

- Manufacturing tolerance of internal components
- Environmental factors (temp, power supply voltage)
- External influences (EM effects that affect voltages)
- How long we're willing to wait
- How much we're willing to spend
- Many distortions, which we can collectively think of (for now) as "*noise*"

Truth in advertising: $V_{OUT} = (1 - V_{IN}) \pm \varepsilon$

If we call it ε maybe it'll seem small ☺

Analog Errors Accumulate (or Cascade)

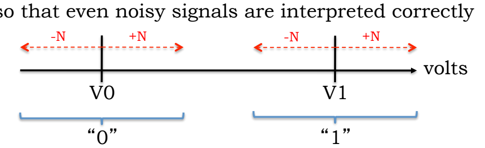


- If, say, ε = 1%, then result might be 100% off (urk!)
- Accumulation is good for money, bad for errors
- As system builders we want to guarantee output without having to worry about exact internal details
  - Bound number of processing stages in series (doesn't work because noise can be unbounded!)
  - Figure out a way to eliminate (or reduce) errors at each processing stage. So how do we know *which part of the signal is correct and which is in error?*

Digital Signaling: Map Bits to Signals

To ensure we can distinguish signal from noise, we'll *map bits to signals* using a fixed set of discrete values. For example, in a binary mapping (or signaling) scheme we would use two voltages (V0 and V1) to represent the two binary values "0" and "1".

- Voltages near V0 would be interpreted as representing "0"
- Voltages near V1 would be interpreted as representing "1"
- If we would like our encoding to work reliably with up to ±N volts of noise, then we can space V0 and V1 far enough apart so that even noisy signals are interpreted correctly

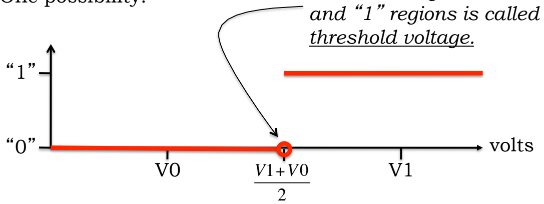


Digital Signaling: Receiving

We can specify the behavior of the receiver with a graph that shows how incoming voltages are mapped to "0" and "1".

One possibility:

The boundary between "0" and "1" regions is called the *__threshold voltage.__*

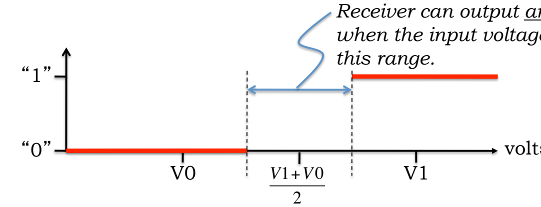


It would be hard to actually build a receiver that *precisely* met this specification since it's very expensive and time consuming to *accurately* measure voltages (e.g., those near the threshold voltage).

We Need a "Forbidden Zone"

We need to change our specification to include a "forbidden zone" where there is *no mapping* between the continuous input voltage and the discrete output:

Receiver can output __any value__ when the input voltage is in this range.



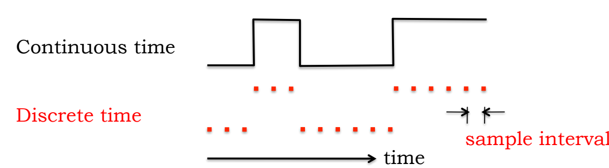
Now the specification has some "elbow room" which allows for manufacturing and environmental differences from receiver to receiver.

Signals in 6.02

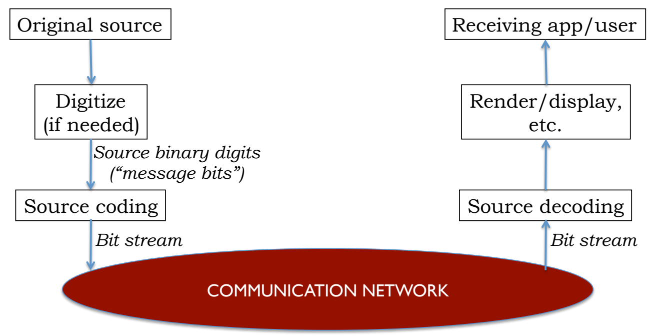
- Each individual transmission signal conceptually a *fixed-voltage waveform* held for some period of time
  - 0 → V0 volts, 1 → V1 volts, held for some time duration
- In 6.02, we'll represent signals, i.e., voltage waveforms, using sequences of *samples*
- *Sample rate* = number of samples/second
- Reciprocal is the *sample interval* (time between samples)
- 4 million samples/second means the sample interval is 0.25 * 10^{-6} seconds = 0.25 microseconds = 250 nanoseconds.

Signals Sent and Received as Samples

Each transmission of a single bit ("0" or "1") will entail sending some number of consecutive voltage samples (V0 or V1 volts); we'll choose an appropriate number of *samples_per_bit* depending on various factors. Goal: smaller is better!



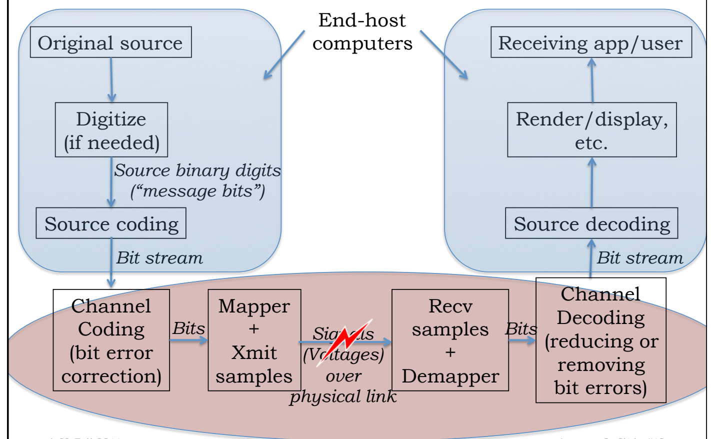
Popping Up: Packaging Messages for Transmission and Reception



The rest of 6.02 is about the colored oval
Simplest network is a single physical comm link
We'll start with that, then get to networks with many links

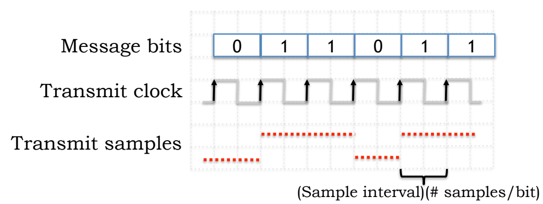
6.02 Fall 2011 Lecture 3, Slide #17

Single Link Communication Model



6.02 Fall 2011 Lecture 3, Slide #18

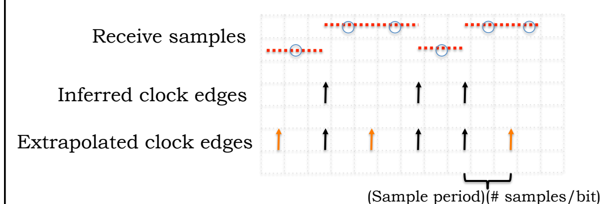
Mapper + Transmitting Signal as Samples



- Periodic events are timed by a clock signal
  - Sample period is controlled by the sample clock
  - Transmit clock is a submultiple of the sample clock
- Can receiver do its job if we only send samples and not the transmit clock?
  - Save a wire and the power needed to drive clock signal
  - Over wireless, may not even be able to send separate clock

6.02 Fall 2011 Lecture 3, Slide #19

Receiving Samples: Data-Clock Recovery



- Receiver can infer presence of clock edge every time there's a transition in the received samples.
- Using sample period, extrapolate remaining edges
  - Now know first and last sample for each bit
- Choose "middle" sample to determine message bit

6.02 Fall 2011 Lecture 3, Slide #20

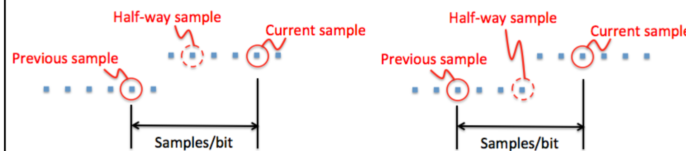
Data-Clock Recovery Challenge: Clock Drift Between Sender and Receiver

- No two crystals have identical frequencies

Pics from wikipedia

- Oscillation frequency depends on temperature, hysteresis, mechanical stresses, radiation, supply voltage, EM fields, age of crystal, …
- Sender's and receiver's clocks therefore do not "tick" at the same rate; one is faster than the other

Data-Clock Recovery with Clock Drift



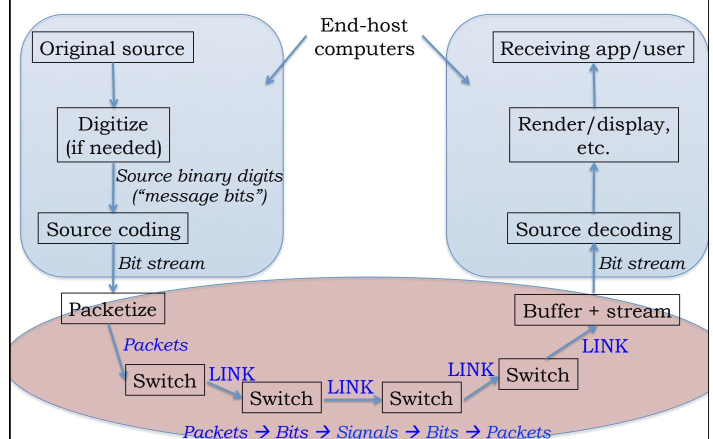
- Don't want receiver to extrapolate over too long
  - Differences in xmit & rcv clock speeds will eventually cause receiver to mis-sample the incoming waveform
- Recode data stream to ensure frequent transitions
- Then, data-clock recovery using a simple "control loop": if halfway (samples_per_bit/2) is same as current, *reduce sample index by 1*, else *increment*
- Formal name for this controller: *bang-bang* because it switches rapidly between two states
- More details in tomorrow's recitation & PSet 2

Packets

- Bit streams could be long, and many different conversations could be sharing links
- Packets help share links between different apps; they also act as good units *of loss recovery* (so we don't have to re-send the entire stream)
- Bits in a packet are sent *synchronously* according to the clock, but packets themselves are *asynchronous*
- So how does receiver at end of a link know when a packet starts (and ends)?
- Solution: use special SYNC bit sequences to periodically synchronize packet start. These SYNC sequences must be *distinguishable* from bits in the packet body.

Network Communication Model

Three Abstraction Layers: Packets, Bits, Signals



Summary

- Analog signaling has issues
  - Real-world channels introduce errors
  - Errors accumulate at each processing step
- Digital Abstraction
  - Mapping bits to discrete signals allows us to tolerate noise better
  - Recover digital data by comparing against threshold
  - And later in 6.02: error correcting codes
- Physical links: mapping and digital signaling
  - We don't send xmit clock, receiver does clock recovery
  - Determine bit from samples in "middle" of bit cell + encoding to ensure frequent transitions
  - Tune in to recitation tomorrow – useful for PSet 2!
- The big picture: three layers – packets, bits, and signals