UCSD CSE CS 118 Networks

Homework 1, Fall 2022

Due: in a week

Directions: No late homeworks. You can talk it over but you must write up and do all the computation by yourself. This one is a lot of work but it will really teach you the main ideas of the Physical layer. Don't worry if you can't solve them all, but put down your ideas. We will grade **one** of Problems 1, 2, and 3 but you must hand in all of them for completion credit for the non-graded HWs. Try them to prepare for the midterm!

1. **Fourier Analysis**, 25 points: In this problem, we will learn (not for the exam, do not fear, I know that many of you are not EEs) how to calculate Fourier coefficients and how much bandwidth you need for a good approximation.

   -***A***. *The square wave:* Consider the square wave that we claimed in class was approximated by the Fourier series 4/ π (sin(2 π *t)* + sin(6 π *t)/3* + sin(10 π *t)/ 10* + …) Why do the scaling factors go down linearly for increasing frequencies? To see this lookup how to find Fourier coefficients (scaling factors) by integration. To find the Fourier analysis you want the coefficient of the n-th harmonic, in other words the scaling factor for the sine wave sin(2 π *n x).* To find the coefficient for an arbitrary function f(x) simply integrate f(x) sin(2 π *n x)* from negative infinity to positive infinity. All you have to remember (or lookup!) is how to integrate a sine function. You could look up the proof on the net but in 1 sentence in your own words, why does the integral result in a scaling factor is inversely proportional to n?

   -***B***. *The approximation:* Go to https://www.desmos.com/calculator (you may have to login using Google) and visualize the series of approximations of the series sin(2 π *t)* + sin(6 π *t)/3* + sin(10 π *t)/ 10* + . . . First, go to the spanner icon (graph settings) on the upper right corner. Set the x-limits from 0 to 1 and the y-limits from -1 to +1. Also, choose radians. Now do the following. Start with sin(2 π *t),* then sin(2 π *t)* + sin(6 π *t)/3,* then do sin(2 π *t)* + sin(6 π *t)/3* + sin(10 π *t)/ 10* Do two more terms that you should be able to guess. For each of these 5 approximations to the square wave, answer the following questions:

   - **B.1** Turn in the graph of all 5 approximations (you should be able to save, worst case do a screenshot) in your report.
   - **B.2** How much bandwidth in Hertz do you need for the approximation? Assume 0 Hz waves can go through the system.
   - **B.3** If you sample the signal anywhere in the middle 2/3rd (in other words anywhere from 0.1 to 0.4 seconds for the first cycle as clock recovery is not perfect), what is the largest percentage error in the amplitude you find (it

should reduce with better approximations (click points to see values) for each approximation.

- B.4 How long does it take to ramp up? That is for each approximation, how long does it take to reach 0.9 (roughly 90% of its peak). For the first basic sine approximation, it should be around t=0.18. Click over points to see the x-coordinate. Even though the real peak is $4/\pi$, assume the peak is 1.

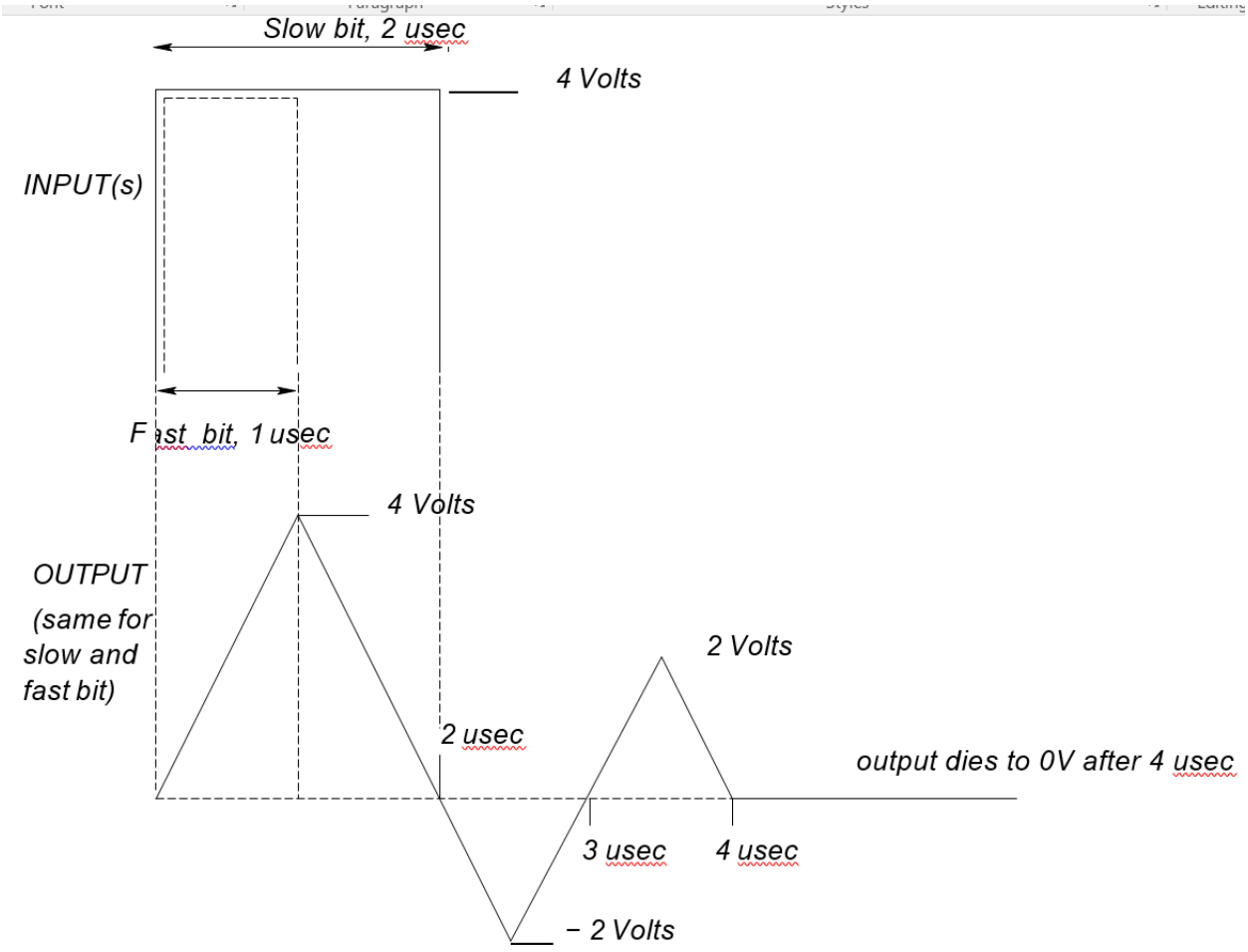
2.Nyquist Limit: This will take some work but at the end you should have a good idea about how bits are sampled, and what inter-symbol interference means. Figure 1 shows the response of a wire to two input bits, a slow bit of width 2 usec and a fast bit of width 1 usec. The response to both slow and fast bits (this shows that the wire cannot respond faster than once every 2 usec, its Nyquist limit) is shown at the bottom of Figure 1. Note that I have chosen to make the response a triangular approximation to the sinc function we studied in class. It rises to 4 Volts in 1 usec, falls to 0 after 2 usec. It then falls to -2 Volts at 2.5 usec, climbs to 0V at 3 usec, rises to 2 V again at 3 usec, and finally falls to zero at 4 usec. Unlike the real sinc function, this idealized output goes to 0 after 4 usec.

Assume that a 1 is encoded as a 4 Volt signal, and a 0 is encoded as 0 Volts. Assume that the output to a 0 Volt input of any bit width is also 0 Volts for all time. If at any sampling instant, the receiver measures a voltage of 2 Volts or more, it assumes it has received a 1; else it assumes it has received a 0. The sender is going to send just 3 bits 101

2.1 (7 points) First assume the sender sends bits at the slow rate of once every 2 usec. Use graph paper and color pens (or a program) to draw the 3 bits: the first bit as red, the second in blue, and the third in black (using these colors will help the grader). Assume the sender sends its 3 bits at times 0, 2 usec, and 4 usec. The sampling instants the receiver uses are 1 usec, 3 usec and 5 usec. At any sampling instant, the receiver measures the output voltage as the sum of the voltage values of the red, blue, and black waves. Write down the measured outputs. What bits does the receiver output?

2.2 (7 points) Repeat the same process you did in Part B but this time using the "fast bit" of width 1 usec (you are now signalling at the fabled Nyquist limit). The sender now sends its bits at times 0, 1 usec, and 2 usec and the receiver samples at 1 usec, 2, and 3 usec. Write down the measured outputs. What bits does the receiver output?

2.3 (11 points) Repeat the same process you did in Parts A and B except with a "super-sonic" bit whose width is 0.5 usec. The output response to the supersonic input bit is exactly the same as for the fast and slow bits. The sender now sends its bits at times 0, 0.5 usec, and 1 usec and the receiver samples at 1 usec, 1.5 usec, and 2 usec. Clearly, the sender is being cheeky and you should see intersymbol interference (at which sampling instants?). What bits does the receiver output?



3. **Clock Recovery,** 25 points: The first problem taught you about how outputs can be computed using Fourier series, the second problem taught you output distortions can cause inter-symbol interference if you send too fast. However, we assumed in Problem 2 that the sender and receiver clocks were perfectly synchronized. In reality, they are not and we need clock synchronization. In Problem 3 you will simulate the

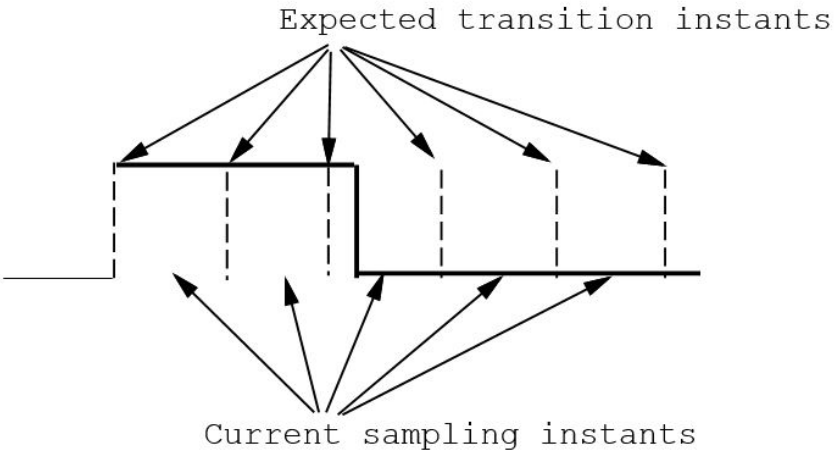
effect of clock recovery on some bits sent using 4-5 encoding using a clock synchronization algorithm I give you below. Assume a perfect channel, no distortion.

Assume the preamble has been received and the receiver is basically in sync except for possible clock drift. Thus the receiver is sampling according to its current clock (see figure below) and should be expecting transitions only at what it thinks are bit boundaries (see dotted lines in figure). However, because of clock drift the actual transitions may be a little off (see the solid line in the figure below).

Remember that in 4-5 coding you are guaranteed to get ***at least*** one transition in every 5 consecutive bits; however, you may get ***up to*** 5 transitions. Pseudocode for the receiver clock recovery algorithm is as shown on the next page.

Please run the code on the next page assuming a nominal bit time of 1 usec (e.g., T = 1 usec) and a sender who is sending 8% slower than the receiver. Thus the sender sends his first bit from 0 to 1.08, the second bit from 1.08 to 2.16, the third from 2.16 to 3.24. Without doing any clock recovery or lag adjustment the receiver would sample at what it thinks is the middle of a bit and so at 0.5, 1.5 etc. We would like to see what happens on the ten-bit sequence 1111101010 with and without clock recovery. Assume a 0 is encoded as 0 volts and a 1 as 1 volt.

- 3.1 Use graph paper to draw a waveform of the 10 bits sent by the sender. (2 points)
- 3.2 If the receiver does not run the clock recovery code, how far off is the sampling by the 10-th bit? (2 points)
- 3.3 On the picture of the waveform you drew, draw the sampling instants and values of lag assuming the receiver uses the pseudocode above for clock recovery and there is no noise. (17 points)
- 3.4 Now suppose there is a sharp noise spike of 1V at time 0.3 usec. How would it affect your sampling times? (2 points)
- 3.5 Finally, suppose there is a sharp noise spike of 1V at time 2.7 usec. How would it affect your sampling times? (2 points)



Variables

```
T: real constant; // nominal time to send a bit, input to program
P: real; // predicted next time at which a transition may occur
A: real; // actual real time at which a transition occurs
lag: real; // difference between predicted and expected
```

Code after preamble is detected

```
Initialize
Clock = 0
lag = 0
P = 0
StartTimer(T/2)
Wait(TimerExpiry)
Do until end of frame
   Output (Sample Signal)  // Output Sampled value when timer expires)
   P = P + T + lag
   StartTimer (T + lag)
   Wait(TimerExpiry)
   In parallel with Wait look for Transition if any
   If Transition is detected at actual time T
      lag = A - P // difference between real and predicted
end
```