Part IV

Telecommunication

Workshop 1: Communication Channel Characteristics & Effects of Noise

---

Objective: To Observe Communication Channel Characteristics & Effects of Noise

Outcome: After successful completion of this session, the student would be able to

1. Identify the cut-off frequencies and bandwidth associated with channel characteristics
2. Estimate the cut-off frequencies and bandwidth by measurement
3. Identify the effects of bandwidth limitation on signals carried by a channel
4. Examine the effect S/N ratio in analog signal transmission
5. Estimate the Bit Error Rate (BER) in digital signal transmission as a function of S/N

Equipment Required:

1. Oscilloscope
2. Signal Generator
3. Proto board
4. A Personal Computer
5. MATLAB software or MATLAB online

Components Required:

1. Resistors 270Ω (1), 330Ω, (1), 1 kΩ (1)
2. Capacitors 1µF (1), 0.33µF (1)

1.1 Communication Channel

Figure 1.1 shows the block diagram of a basic communication system (analog/digital).

The **source** is the device or the person which generates the data to be transmitted. The **transmitter** processes and transmits the data. **Channel** is the medium through which the data is transmitted. **Receiver** receives and processes the transmitted data. The **destination** is the final device or person to which the data is intended.

In this practical we will be focusing on the effects induced by the channel. The channels typically induce three major effects to the transmitted data.

- Attenuation
- Noise Addition
- Bandwidth limitation

Let the transmitted signal be $X(t)$. Attenuation is simply the reduction of the amplitude of the signal. The signal after attenuation is $AX(t)$, where $0 < A < 1$ is the attenuation factor.

Noise is the random fluctuation of the signal around it's actual value. The signal after attenuation and noise can be written as $AX(t) + N(t)$, where $N(t)$ represents a random signal.

The channels typically limit the bandwidth of the transmitted signal. In other words certain frequencies of the transmitted signal are filtered. Hence, the signal after all three effects can be written as bandlimit$(AX(t) + N(t))$ where, the bandlimit function represents the bandwidth limitation.

In this practical we will be focusing on understanding these effects in detail.

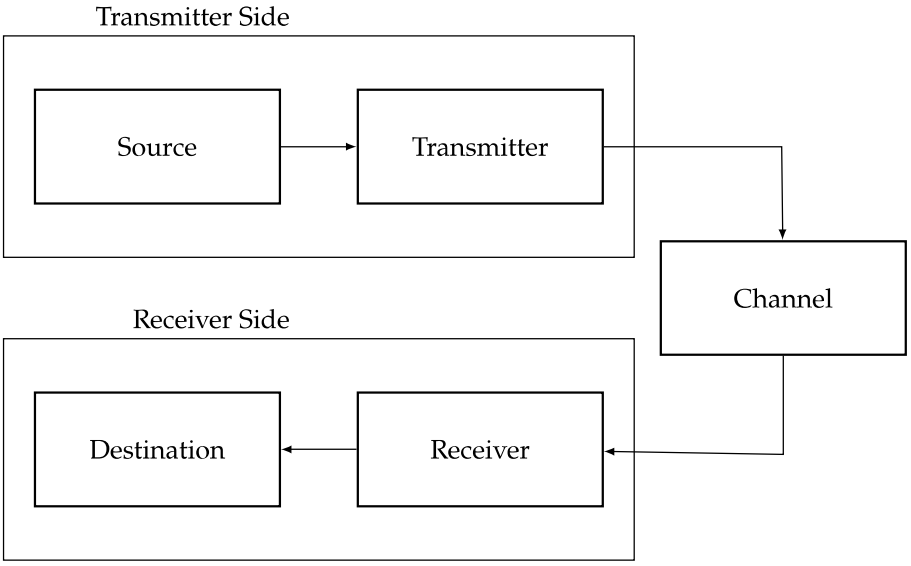


Figure 1.1: The block diagram of a communication system

1.2 Pre-Lab

Prior to the lab you will model a basic communication channel using Simulink. Channels for both analog and digital transmission can be modelled using the above effects.

1.2.1 Analog Channel

We will first focus on an analog channel. In an analog channel an analog signal is transmitted through the channel. In this case we model the noise $N(t)$ as a Gaussian noise. The bandlimiting filter is a low-pass filter. Build the following simulink model (Figure 1.2) in bring it to the lab.

Use the following parameters.

- Transmitted sinusoidal signal [**Sine Wave** block in **Simulink** > **Sources**]
  - Amplitude: 10
  - Frequency: 200 rad/s
  - Sample time: 0.001 s
- Channel attenuation [**Gain** block in **Simulink** > **Commonly used blocks**]
  - Gain: 0.8
- Attenuated signal, Signal corrupted by noise, Signal output from channel, Information signal, Noise, SNR grpah, SNR average [**Scope** block in **Simulink** > **Commonly used blocks**]
- Add [**Add** block in **Simulink** > **Math Operations**]
- Channel noise [**Random Source** block in **DSP System Toolbox** > **Sources**]
  - Source type: Gaussian
  - Mean value: 0
  - Sample mode: Discrete
  - Variance: 1
  - Sample time: 0.001 s

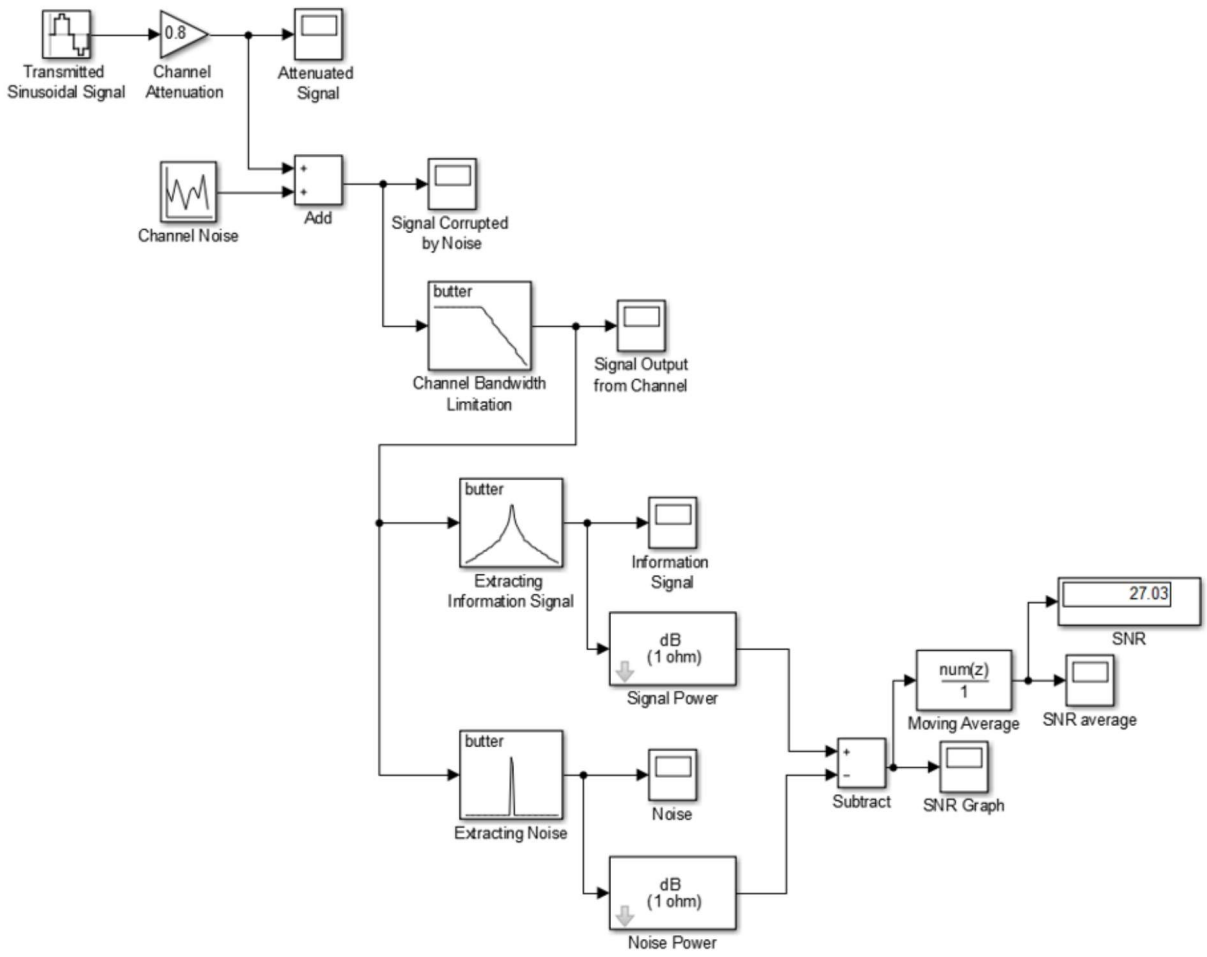


Figure 1.2: Simulink Model for Analog Channel.

- Channel bandwidth limitation [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]
  - Design method: Butterworth
  - Filter type: Lowpass
  - Filter order: 10
  - Passband edge frequency: 1000 rad/s
- Extracting information signal [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]
  - Design method: Butterworth
  - Filter type: Bandpass
  - Filter order: 10
  - Lower passband edge frequency: 195 rad/s
  - Upper passband edge frequency: 205 rad/s
- Extracting noise [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]

  - Design method: Butterworth
  - Filter type: Bandstop
  - Filter order: 10
  - Lower passband edge frequency: 195 rad/s
  - Upper passband edge frequency: 205 rad/s
- Signal power, Noise power [**dB Conversion** block in **Communications System Toolbox** > **Utility Blocks**]
  - Convert to: dB
  - Input signal: Amplitude
  - Load resistance: 1 ohm
- Subtract [**Subtract** block in **Simulink** > **Math Operations**]
- SNR [**Display** block in **Simulink** > **Sinks**]
- Moving average [**Discrete FIR Filter** block in **Simulink** > **Discrete**]
  - Filter structure: Direct form
  - Sample time: 0.001
  - Coefficients: ones(1,1000)/1000

Task 1. *Set the simulation time to 20s and run the simulation. Observe the effects of attenuation, channel noise and channel bandwidth limitation. Explain how the SNR is calculated.*

1.2.2 Digital Channel

In a digital channel a bit stream is transmitted. The bit 1 is transmitted with a pulse of amplitude 4, and the bit 0 is transmitted with a pulse of amplitude -4. We model the noise as Gaussian noise similar to the analog channel. Build the following simulink model (Figure 1.3) in bring it to the lab.

Use the following parameters.

- Binary Generator [**Bernoulli Binary Generator** block in **Communication System Toolbox** > **Comm sources** > **Random Data Sources**]
  - Probability of a zero: 0.5
  - Source of initial seed: Parameter
  - Initial seed: 61
  - Sample time: 0.1 s
- Gain [**Gain** block in **Simulink** > **Math Operations**]
  - Gain: 8
- Constant [**Constant** block in **Simulink** > **Sources**]
  - Constant value: -4
- Add [**Add** block in **Simulink** > **Math Operations**]
- Digital information signal, Signal corrupted by noise, Channel output, Output from sampler, Output from thresholder, Output from detector [**Scope** block in **Simulink** > **Commonly used blocks**]
- Channel attenuation [**Gain** block in **Simulink** > **Math Operations**]
  - Gain: 0.5

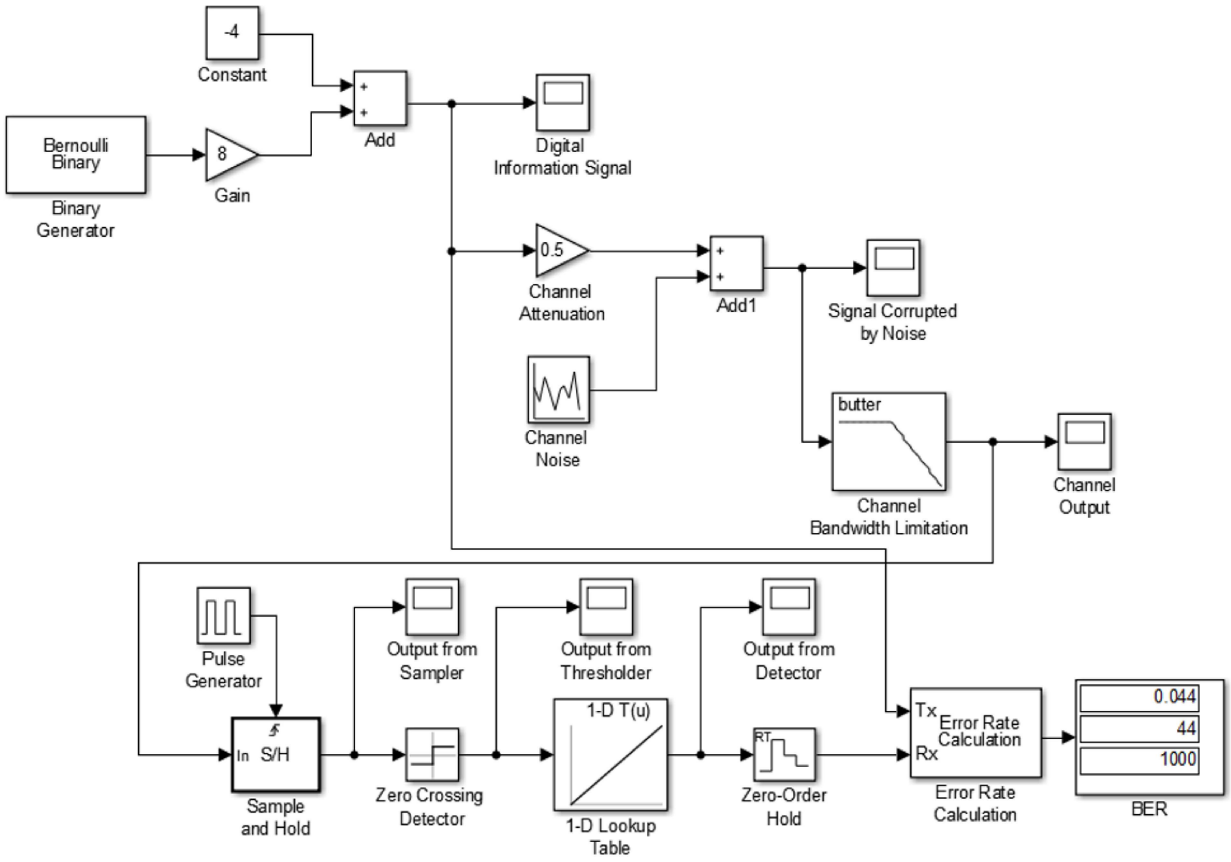


Figure 1.3: Simulink Model for Digital Channel.

- Channel noise [**Random Source** block in **DSP System Toolbox** > **Sources**]
  - Source type: Gaussian
  - Mean value: 0
  - Variance: 1
  - Sample time: 0.01 s
- Channel bandwidth limitation [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]
  - Design method: Butterworth
  - Filter type: Lowpass
  - Filter order: 10
  - Passband edge frequency: 1000 rad/s
- Pulse Generator [**Pulse Generator** block in **Simulink** > **Sources**]
  - Pulse type: Time based
  - Time: Use simulation time
  - Amplitude: 1
  - Period: 0.1 s
  - Pulse width: 50
  - Phase delay: 0 s

- Sample and Hold [**Sample and Hold** block in **DSP System Toolbox** > **Signal Operations**]
  - Trigger type: Rising edge
  - Initial condition: 0
- Zero crossing detector [**Sign** block in **Simulink** > **Math Operations**]
  - Enable zero crossing detection
  - Sampling time: -1
- Lookup Table [**1-D Lookup Table** block in **Simulink** -> **Lookup Tables**]
  - Table Data: [-4,4]
  - Break Points: [-1,1]
  - Sample time: -1
- Zero-Order Hold [**Zero-Order Hold** block in **Simulink** > **Discrete**]
  - Sample time: 0.1 s
- Bit Error Rate Calculator [**Error Rate Calculation** block in **Communication Systems Toolbox** > **Comm Sinks**]
  - Receive delay: 1 s
  - Computation delay: 0 s
  - Output Data: Port
- BER [**Display** block in **Simulink** > **Sinks**]

Task 2. *Set the simulation time to 100s and run the simulation. Observe the signal at each scope output and identify the operation at each stage. Explain how the BER is calculated.*

1.3 Analyzing the Analog Channel

Answer the following questions using the simulink model developed in section 1.2.1.

Task 3. *Complete the Table in the Task Sheet by changing the channel noise variance as given.*

Task 4. *Complete the Table in the Task Sheet by changing the channel bandwidth as given. Keep the noise variance as 1 for all cases.*

Task 5. *Discuss the reasons for the observed variation of SNR with the channel bandwidth.*

Task 6. *Discuss the observed variation of SNR with the information signal amplitude and its relevance in signal transmission over communication channels in practice.*

Task 7. *It is often required to transmit composite analog signals through transmission channels. To evaluate the impact of noise and other factors on such applications, replace the previously transmitted analog signal to the following signal by using suitable blocks available in Simulink. Show your work to an instructor.*

$$x(t) = \sum_{k=0}^{4} (10 - 2k)\sin\left((200 + 20k)t\right). \tag{1.1}$$

Task 8. *Include the time domain behavior of the new composite signal.*

Task 9. *Adjust the passband of the Butterworth bandpass filter for information extraction and stopband of the Butterworth bandstop filter for noise extraction. What are the new cutoff frequencies for passband and stop band of the two filters for information extraction and noise extraction?*

Task 10. *How do you adjust the cutoff frequencies for passband and stop band of the two filters for information extraction and noise extraction? Elaborate your answer from the results obtainable from the simulations.*

1.4 Analyzing the Digital Channel

Answer the following questions using the simulink model developed in section 1.2.2.

Task 11. *Complete the Table in the Task Sheet by changing the channel noise variance as given.*

Task 12. *Discuss the reasons for the observed variation of BER with noise variance.*

Task 13. *Plot the BER with respect to signal to noise ratio (Eb/No).*

1.5 Modelling the Bandlimiting Effect of a Communication Channel Using Hardware

In this section we will be modelling the bandlimiting effect of a communication channel using hardware.

- Low-pass channel - filters the frequencies above a particular cutoff
- High-pass channel - filters the frequencies below a particular cutoff
- Band-pass channel - allows frequencies in a particular range to pass through while filtering the other frequencies
- Band-stop channel - filters frequencies in a particular range

1.5.1 Signal Transmission Through a Low-Pass Channel

We will first implement a low pass channel using hardware. Implement the circuit shown in Figure 1.4.

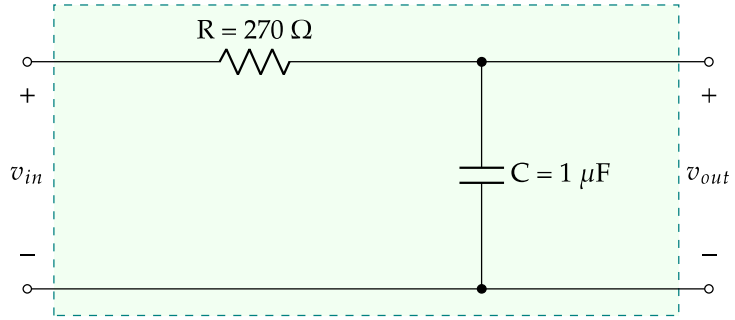


Figure 1.4: Low-Pass Channel

Connect the signal generator to v_{in} and the two channels of the oscilloscope to v_{in} and v_{out}.

Task 14. *Select the sinusoidal mode of the signal generator and apply 1V peak-to-peak signals having frequencies as shown in the Table in the Task Sheet. Complete the Table by taking measurements of v_{out}. Please note that you need to keep v_{in} at 1V peak-to-peak in each measurement.*

Task 15. *Plot the output voltage vs. Log (frequency) in the Task Sheet, complete the curve and write down the cut-off frequency obtained from graph.*

Task 16. *What is the obtained cut-off frequency?*

Task 17. *Find the theoretical cut-off frequency of this channel and compare with the value obtained by the above graph.*

1.5.2 Distortion in Signals When Transmitted Through Low-Pass Channels

The signals are subjected to distortion (change from the original signal) as a result of passing through channels. This is due to the partial or full loss of signal energy in certain frequencies due to bandwidth limitation. In this section we will try to observe the distortion.

Select the square wave mode of the signal generator. Set the peak-to-peak input voltage to 2V, and increase the frequency from 0 while looking at the output on the oscilloscope.

Task 18. *Complete the Table in the Task Sheet with sketches showing relative amplitudes correctly.*

Task 19. *What is the frequency at which you begin to see distortion in the shape of the output signal.*

1.5.3 Signal Transmission Through a Band-Pass Channel

In this section we will implement a band pass channel using hardware. Implement the circuit shown in Figure 1.5.

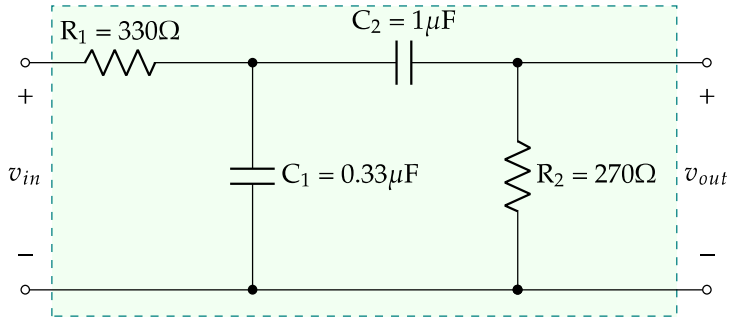


Figure 1.5: Band-Pass Channel

Connect the signal generator to the input and select the sinusoidal mode and apply 1V peak-to-peak signal.

Task 20. *Measure the output voltages v_{out} at the frequencies shown in the Table in the Task Sheet. Please note that you need to keep v_{in} at 1V peak-to-peak in each measurement.*

Task 21. *Plot the output voltage vs. Log (frequency) in the Task Sheet, complete the curve and write down the lower cut-off frequency and upper cut-off frequency obtained from the graph.*

Task 22. *What is the obtained lower cut-off frequency?*

Task 23. *What is the obtained upper cut-off frequency?*

Task 24. *Find the theoretical Lower cut-off and Upper cut-off frequencies and bandwidth of the channel.*

Task 25. *What is the frequency of resonance?*

1.5.4 Distortion in Signals When Transmitted Through Band-Pass Channels

Select the square wave mode of the signal generator, set the input signal amplitude to 1V peak-to-peak, and vary the frequency in both directions from the frequency at which you got the resonance and observe the output.

Task 26. *Complete the Table in the Task Sheet with sketches showing relative amplitudes correctly.*

♣ The End ♣

Workshop 2: Baseband Communication

Objective: To simulate and analyze baseband transmission.

Outcome: After successful completion of this session, the student would be able to

1. Identify the basic elements of a baseband communication system
2. Implement a simple baseband communication system using MATLAB
3. Simulate and analyze the bit-error rate of a baseband communication system under different settings
4. Implement a simple error-correction mechanism

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online
3. A headphone set (to be brought by the student)

Components Required: None.

2.1 Baseband Communication

In telecommunications the signals are usually transmitted after converting them to higher frequencies. This process is called modulation and the transmitted signals are known as broadband signals. But the properties of the transmission can be analyzed using a communication system without modulation. Such a system is known as a baseband communication system. Figure 2.1 shows a baseband and a broadband signal in frequency domain side-by-side.

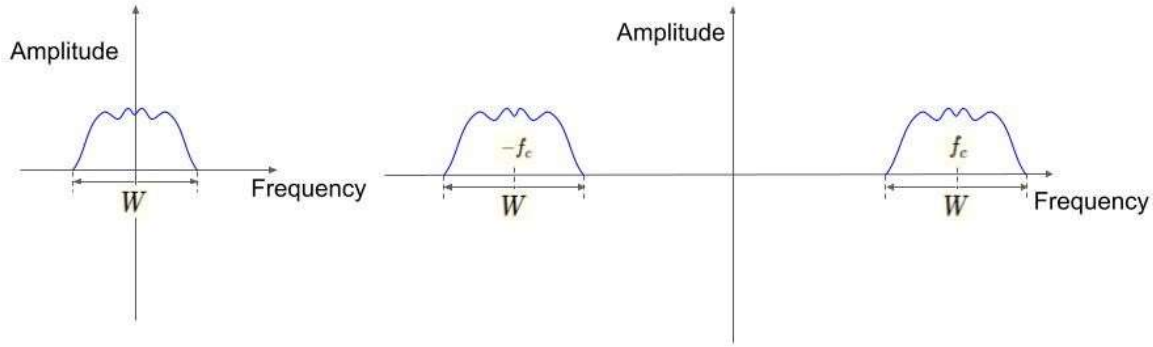


Figure 2.1: Frequency domain representation of a signal in baseband (left) vs broadband (right). Bandwidth of the signal is W and the center frequency of the broadband signal is f_c.

In this practical we will be simulating a baseband communication system using MATLAB and we will be analyzing the bit error rate of the system under different settings. Finally we will study a simple error-correction mechanism.

2.2 Pre-Lab

Prior to the lab we will implement the baseband communication system using MATLAB. You will have to record audio from your headphones. The recorded audio will be tramsitted through the communication system and will be reconstructed at the receiver.

2.2.1 Implementing a Baseband Communication System

Figure 2.2 illustrates the block diagram of a simple baseband communication system.

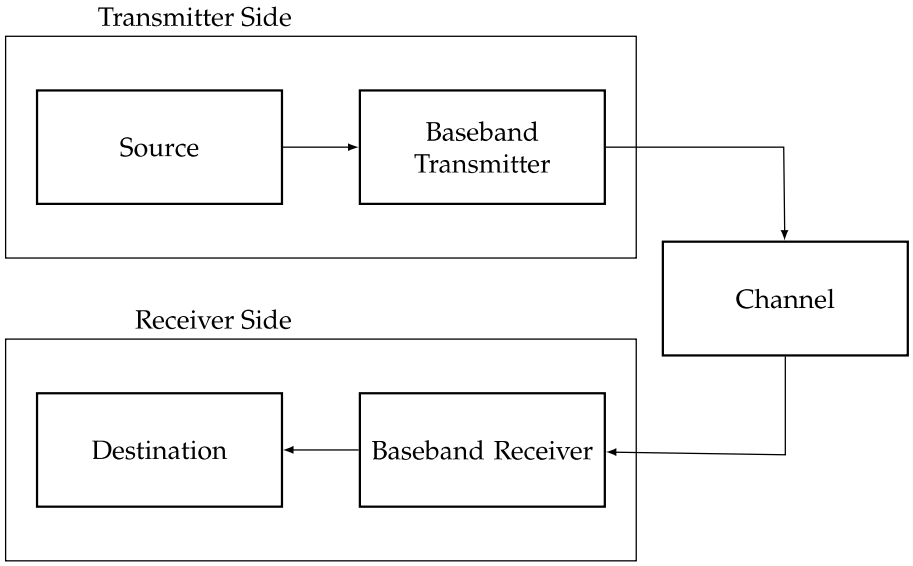


Figure 2.2: The block diagram of a baseband communication system

Next we will understand each block along with the implementation details.

Source

Source is the device/person which generates the signal to be transmitted. In this case you will be the source since you are generating the audio waveform. The analog signal from the source is sent to the transmitter.

Transmitter

Transmitter performs several modifications to the signal received from the source. First, the analog signal is sampled at discrete time intervals (The analog signal is represented as a set of samples). This process is known as **sampling**. Your headphone will perform the sampling.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
recObj = audiorecorder(Fs,qbits,1);  %Starting the audio recorder object
disp('Start speaking.')      %Recording audio fro 5 seconds
recordblocking(recObj, 5);
disp('End of Recording.');
audio_samples = getaudiodata(recObj);  %array containing samples of the recorded audio
```

If you have trouble accessing the audio recorders (this can happen in MATLAB online), you can use recorded audio. An audio clip named "Recording.wav" will be uploaded to the Moodle. You can directly

use this audio clip. In case of MATLAB online, upload the file to MATLAB online. Then use the following code instead of the above code.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
[audio_samples,Fs] = audioread('Recording.wav'); %Replace filename with the location of your file.
```

If you prefer to record your own audio, first you will have to record a 5 second audio clip from your PC. Then convert it to .wav format using an online converter. You can use https://www.aconvert.com/audio/. Make sure you set the sampling rate to 6000. Rename the file as "Recording.wav" and use it for the workshop.

Next, we have to represent the set of samples using a bit stream in order to transmit as a digital signal. For that, we **quantize** each sample, where we approximate each sample with a discrete level. In this case we represent each sample with one of 256 discrete levels.

```
siz = size(audio_samples);   %Quantizing the audio samples (Each sample is quantized with 8 bits)
size_audio_samples = siz(1);
number_of_bits = qbits*size_audio_samples
bit_stream = zeros([1,number_of_bits]); %array which is used to store the binary representation
                            %of the recorded audio

for i = 1:size_audio_samples
 num = audio_samples(i,1);
 if(num<0)
   bit_stream(1,(i-1)*qbits+1) = 1;
 else
   bit_stream(1,(i-1)*qbits+1) = 0;
 end
 num = abs(num);
 for j = 2:qbits
    num = 2*num;
    if(num>=1)
     bit_stream(1,(i-1)*qbits+j) =  1;
     num = num - 1;
    else
     bit_stream(1,(i-1)*qbits+j) =  0;
    end
 end
end
```

Now we have the binary stream of 1's and 0's which we aim to transmit. Although we transmit bits we still have to represent (**encode**) them using analog signals. We will have to represent bit zero and bit one using two distinct finite duration signals. For simplicity, we use two sinusoidal signals with different phases (phase_0 and phase_1). For this part we will use $\text{phase}_1 = 0$ and $\text{phase}_1 = \pi$ (This formation is called as antipodal signalling since a bit 0 is represented by the negative of the signal used to represent bit1).

$$\begin{aligned} \text{bit0} &\longrightarrow \text{Amplitude} \times \sin(W_c t + \text{phase}_0) \quad 0 \leq t \leq T, \\ \text{bit1} &\longrightarrow \text{Amplitude} \times \sin(W_c t + \text{phase}_1) \quad 0 \leq t \leq T, \end{aligned} \tag{2.1}$$

where T is the bit period.

Although in a practical implementation these signals are implemented in analog domain, in MATLAB we approximate the actual analog signal with a set of samples (We cannot create analog signals in MATLAB).

```
samples_per_bit = 100;  %Number of samples used to approximate the
                %sinosoid for a single bit (We approximate
                % sin(Wc*t+phase0) or sin(Wc*t+phase1) with 100
                % bits in this case
```

```
Amplitude = -5; %Amplitude in decibels
sampling_rate = Fs*qbits*samples_per_bit; %The sampling rate used to represent the
                                          %anlog singal in MATLAB
fc = sampling_rate/100; % frequency of the sinosoid
Wc = 2*pi*fc;
delt = 1/sampling_rate; %Sample time period
phase0 = pi;
phase1 = 0;


number_of_samples = number_of_bits*samples_per_bit; %Total number of samples of the transmitted signal
X = zeros([1,number_of_samples]); %X(t) - This array is used to store the transmitted analog signal


%This section does the encoding which is described previously

power = 10^(Amplitude/20); %Signal power in watts.
t_bit = delt:delt:delt*samples_per_bit;
bit1  = sqrt(power)*sin(Wc*t_bit+phase1); %Representation of bit 1
bit0  = sqrt(power)*sin(Wc*t_bit+phase0); %Representation of bit 0
for i = 1:number_of_bits
  if(bit_stream(i)==1)
    X(samples_per_bit*(i-1)+1:samples_per_bit*i) = bit1;
  else
    X(samples_per_bit*(i-1)+1:samples_per_bit*i) = bit0;
  end
end
```

At this point the transmitted signal is stored in the array X and is ready to be transmitted (X is a discrete time approximation of the actual transmitted signal $X(t)$).

Channel

The signals are transmitted through channels (cables for wired transmissions and free space for wireless transmissions). The channel induces several effects to the transmitted signal. Below we model the induced effects.

First, the power of the signal is reduced when it is propagating through the channel. This is known as **attenuation**. Next, channels add **noise** to the transmitted signal. In addition, channels also filter out certain frequencies of the signal (This effect is known as channel **bandwidth limitation**).

The signal after attenuation and noise is

$$\tilde{X}(t) = AX(t) + N(t). \tag{2.2}$$

Here A is the attenuation factor (The proportion of the power of the transmitted signal received at the receiver. $N(t)$ is the noise which is a random signal containing random values. The signal $\tilde{X}(t)$ is filtered (bandlimited) to produce the signal $Y(t)$. We use a low-pass butterworth filter of order 10 to model the filtering effect.

```
att = 0.8;%attenuation factor
mu = 0; %Paramenters of the noise signal (mu, and sigma). We
      % model noise as a Gaussian random variable
sigma = 1;
N = normrnd(mu,sigma,1,samples_per_bit*number_of_bits); %N(t) - noise signal
X_hat = att*X + N; %signal after noise

%Filtering Effect
fc_butter = fc*25;
```

```
[fil_b,fil_a] = butter(10,fc_butter/(sampling_rate/2));
Y = filter(fil_b,fil_a,X_hat);%recived signal after bandwidth limitation (filtering)
```

Receiver

The receiver receives $Y(t)$. Since $Y(t)$ and $X(t)$ are not the same, we have to figure out a method to obtain (**decode**) the transmitted bit stream.

In this case we use phase estimation to perform decoding. Observe that if the transmitted bit was a zero, the phase of the encoding sinosoid was phase_0, and if the transmitted bit was a one, the phase of the encoding sinosoid was phase_1. But due to noise addition and bandwidth limitation the phase may get changed. If the estimated phase is close to phase_0 than phase_1, we interpret the bit as a zero and vice versa. Observe that depending on the strength of the noise signal, bit errors may occur at the receiver.

```
decoded_bit_stream = zeros([1,number_of_bits]); %The array to store the decoded bit stream
H = [cos(Wc*delt*[1:samples_per_bit]'),sin(Wc*delt*[1:samples_per_bit]')]; %Phase estimation matrox

for i = 1:number_of_bits %This loop deals with the phase estimation
  f = inv(H'*H)*H'*Y((i-1)*samples_per_bit+1:i*samples_per_bit)';
  decoded_phase = 0;
  if(f(2) == 0)
   if(f(1) >= 0)
     decoded_phase = pi/2;
   else
     decoded_phase = -pi/2;
   end
  else
    decoded_phase = atan(f(1)/f(2));
    if(f(1)<=0 & f(2)<0)
      decoded_phase = decoded_phase-pi;
    elseif(f(1)>0 & f(2)<0)
      decoded_phase = decoded_phase+pi;
    end
  end
  p0 = min(abs(decoded_phase - phase0),2*pi-abs(decoded_phase - phase0));
  p1 = min(abs(decoded_phase - phase1),2*pi-abs(decoded_phase - phase1));

  if(p0<p1)
    decoded_bit_stream(i) = 0;
  else
    decoded_bit_stream(i) = 1;
  end
end
```

Next, we **reconstruct** the audio samples from the bit-stream.

```
decoded_audio_samples = zeros([size_audio_samples,1]);
for i=1:size_audio_samples
   st = 0.5;
   for j=2:qbits
    decoded_audio_samples(i) = decoded_audio_samples(i)+st*decoded_bit_stream((i-1)*qbits+j);
    st = st/2;
   end
   if(decoded_bit_stream((i-1)*qbits+1) == 1)
     decoded_audio_samples(i) = -decoded_audio_samples(i);
   end
```

```
end
```

Destination

Destination is the device/person to which the signal transmission is intended. In this case you are the destination. Now we will listen to the reconstructed signal.

```
pause(10);
sound(5*decoded_audio_samples, Fs);
```

Change the amplitude of the transmitted signal and observe the impact of amplitude on the quality of the reconstructed signal.

2.2.2 Comparing the Original and the Reconstructed Signals

Now we will plot the original recorded analog audio signal and the signal reconstructed at the receiver, side by side.

```
figure(1)
subplot(1,2,1)
plot(audio_samples)
title('Original Audio Signal')
xlabel('t (s)')
ylabel('Amplitude')
subplot(1,2,2)
plot(decoded_audio_samples);
title('Decoded Audio Signal')
xlabel('t (s)')
ylabel('Amplitude')
```

Task 1. *Compare the two signals and comment on the observed differences.*

2.2.3 Comparing the Original and the Transmitted Signals in Frequency Domain

Use the following code to plot the frequency domain representations of the original recorded analog audio signal and the transmitted signal, side by side.

```
L_1 = size_audio_samples;
Audio_freq = fft(audio_samples);
Audio_freq_norm = abs(Audio_freq/L_1);
Audio_freq_norm_one = Audio_freq_norm(1:L_1/2+1);
Audio_freq_norm_one(2:end-1) = 2*Audio_freq_norm_one(2:end-1);

L_2 = samples_per_bit*number_of_bits;
X_freq = fft(X);
X_freq_norm = abs(X_freq/L_2);
X_freq_norm_one = X_freq_norm(1:L_2/2+1);
X_freq_norm_one(2:end-1) = 2*X_freq_norm_one(2:end-1);

f_1 = Fs*(0:(L_1/2))/L_1;
f_2 = sampling_rate*(0:(L_2/2))/L_2;

figure(2)
```

```
subplot(1,2,1)
plot(f_1,Audio_freq_norm_one);
title('Single-Sided Amplitude Spectrum of the Original Audio Stream');
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(1,2,2)
plot(f_2,X_freq_norm_one);
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

Task 2. *Compare the two frequency spectra and comment on the observed differences*

2.3 Analyzing the Bit-Error Rate (BER) Using a Baseband Communication System

Next we will calculate the bit-error rate of the system.

Task 3. *Write a code snippet to calculate the bit-error rate (BER) for the transmission discussed in section 2.2.1. Include your code at the bottom of the "Pre_Lab.m" file given in the Moodle, and submit the updated MATLAB file with the BER calculation to Moodle. In your code, assign the calculated BER to a variable named "BER". (Hint: The original bit stream is the array "bit_stream" and the decoded bit stream is the array decoded_bit_stream". Your code should calculate the error rate using the difference between these two arrays.)*

Task 4. *What is the calculated bit-error rate?*

Now we will analyze the bit-error rate of a baseband communication system. You will have to vary the signal amplitude and observe it's effect on the bit-error rate. Note that since we keep the noise variance constant, the Signal-to-Noise ratio is proportional to the signal amplitude.

Task 5. *Repeat section 2.2.1 with integer amplitudes ranging from -10 to 10 to 10 dB (Hint: You can use a for loop on the Amplitude variable to achieve this). Plot the Amplitude (dB) vs BER graph in the Task Sheet.*

Task 6. *Repeat the same procedure of task 5 with* $phase_1 = 0$ *and* $phase_1 = \pi/2$ *and plot the graph in the Task Sheet.*

Task 7. *Comment on the differences of the two graphs and the reasons for the difference.*

2.4 Implementing a Simple Error Correction Mechanism

We observed that bit-errors occur due to the channel noise and bandwidth limitation. We can perform error correction at the receiver in order to correct some of the errors occurred. In this section we will be studying a simple error-correction mechanism (Known as an error correcting code).

In this mechanism we will transmit a single bit three times. At the receiver we will observe the three bits and consider the most frequent bit out of the three to be the correct bit.

Replace the code we used for 4.2 with the following code.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;  %The number of bits used to encode a single sample of the audio
recObj = audiorecorder(Fs,qbits,1); %Starting the audio recorder object
disp('Start speaking.')      %Recording audio fro 5 seconds
recordblocking(recObj, 5);
disp('End of Recording.');
```

```
audio_samples = getaudiodata(recObj);  %array containing samples of the recorded audio
siz = size(audio_samples);   %Quantizing the audio samples (Each sample is quantized with 8 bits)
size_audio_samples = siz(1);
number_of_bits = qbits*size_audio_samples
bit_stream = zeros([1,number_of_bits]); %array which is used to store the binary representation
                                %of the recorded audio
for i = 1:size_audio_samples
 num = audio_samples(i,1);
 if(num<0)
   bit_stream(1,(i-1)*qbits+1) = 1;
 else
   bit_stream(1,(i-1)*qbits+1) = 0;
 end
 num = abs(num);
 for j = 2:qbits
    num = 2*num;
    if(num>=1)
     bit_stream(1,(i-1)*qbits+j) =  1;
     num = num - 1;
    else
     bit_stream(1,(i-1)*qbits+j) =  0;
    end
 end
end
samples_per_bit = 100;
Amplitude = -5; %Amplitude in decibels
sampling_rate = Fs*qbits*samples_per_bit; %The sampling rate used to represent the
                                    %anlog singal in MATLAB
fc = sampling_rate/100; % frequency of the sinosoid
Wc = 2*pi*fc;
delt = 1/sampling_rate; %Sample time period
phase0 = pi;
phase1 = 0;


number_of_samples = number_of_bits*samples_per_bit; %Total number of samples of the transmitted signal

X = zeros([1,3*number_of_samples]); %X(t) - This array is used to store the transmitted analog signal
power = 10^(Amplitude/20); %Signal power in watts.
t_bit = delt:delt:delt*samples_per_bit;
bit1  = sqrt(power)*sin(Wc*t_bit+phase1); %Representation of bit 1
bit0  = sqrt(power)*sin(Wc*t_bit+phase0); %Representation of bit 0
for i = 1:number_of_bits
  if(bit_stream(i)==1)
    X(3*samples_per_bit*(i-1)+1:3*samples_per_bit*(i-1)+samples_per_bit) = bit1;
    X(3*samples_per_bit*(i-1)+samples_per_bit+1:3*samples_per_bit*(i-1)+2*samples_per_bit) = bit1;
    X(3*samples_per_bit*(i-1)+2*samples_per_bit+1:3*samples_per_bit*i) = bit1;
  else
    X(3*samples_per_bit*(i-1)+1:3*samples_per_bit*(i-1)+samples_per_bit) = bit0;
    X(3*samples_per_bit*(i-1)+samples_per_bit+1:3*samples_per_bit*(i-1)+2*samples_per_bit) = bit0;
    X(3*samples_per_bit*(i-1)+2*samples_per_bit+1:3*samples_per_bit*i) = bit0;
  end
end

att = 0.8;%attenuation factor
mu = 0; %Paramenters of the noise signal (mu, and sigma). We
      % model noise as a Gaussian random variable
sigma = 1;
N = normrnd(mu,sigma,1,3*number_of_samples); %N(t) - noise signal
```

```
X_hat = att*X + N; %signal after noise
fc_butter = fc*25;
[fil_b,fil_a] = butter(10,fc_butter/(sampling_rate/2));
Y = filter(fil_b,fil_a,X_hat);%recived signal after bandwidth limitation (filtering)
decoded_bit_stream = zeros([1,3*number_of_bits]); %The array to store the decoded bit stream
H = [cos(Wc*delt*[1:samples_per_bit]'),sin(Wc*delt*[1:samples_per_bit]')]; %Phase estimation matrox

for i = 1:3*number_of_bits %This loop deals with the phase estimation
  f = inv(H'*H)*H'*Y((i-1)*samples_per_bit+1:i*samples_per_bit)';
  decoded_phase = 0;
  if(f(2) == 0)
   if(f(1) >= 0)
     decoded_phase = pi/2;
   else
     decoded_phase = -pi/2;
   end
  else
    decoded_phase = atan(f(1)/f(2));
    if(f(1)<=0 & f(2)<0)
      decoded_phase = decoded_phase-pi;
    elseif(f(1)>0 & f(2)<0)
      decoded_phase = decoded_phase+pi;
    end
  end
  p0 = min(abs(decoded_phase - phase0),2*pi-abs(decoded_phase - phase0));
  p1 = min(abs(decoded_phase - phase1),2*pi-abs(decoded_phase - phase1));

  if(p0<p1)
    decoded_bit_stream(i) = 0;
  else
    decoded_bit_stream(i) = 1;
  end
end
decoded_bit_stream_error_corrected = zeros([1,number_of_bits]); %The array to store the decoded bit
                                     %stream after error-correction
for i = 1:number_of_bits %This loop does the task of correcting erros by looking at the most frequent
                %bit from the three bits
  for j = 1:3
    decoded_bit_stream_error_corrected(i) = decoded_bit_stream_error_corrected(i)+decoded_bit_stream(3*(i-1)+j);
  end
  decoded_bit_stream_error_corrected(i)= round(decoded_bit_stream_error_corrected(i)/3);
end
decoded_audio_samples = zeros([size_audio_samples,1]);
for i=1:size_audio_samples
   st = 0.5;
   for j=2:qbits
    decoded_audio_samples(i) = decoded_audio_samples(i)+st*decoded_bit_stream_error_corrected((i-1)*qbits+j);
    st = st/2;
   end
   if(decoded_bit_stream_error_corrected((i-1)*qbits+1) == 1)
     decoded_audio_samples(i) = -decoded_audio_samples(i);
   end
end
```

Go through the above code and understand how the code for section 2.2.1 is changed to perform error-correction.

Task 8. *Similar to task 3 implement the code to calculate the bit-error rate (Hint: the arrays "bit_stream" and*

"decoded_bit_stream_error_corrected" contain the transmit and received bit streams after error correction). What is the calculated BER?

Task 9. *Repeat task 5 and task 6 with error correction and include the graphs in the Task Sheet.*

Task 10. *Comment on the observations and the effect of using the error-correction mechanism on the bit-error rate.*

Task 11. *What are the disadvantages of using this error-correction mechanism*

♣ The End ♣

Workshop 3: Digital Modulation Schemes

Objective: To identify digital modulation schemes using Matlab

Outcome: After successful completion of this session, the student would be able to

1. Demonstrate an understanding of amplitude shift keying (ASK)
2. Demonstrate an understanding of frequency shift keying (FSK)
3. Demonstrate an understanding of phase shift keying (PSK)
4. Develop skills in using Matlab as a tool for communication systems study

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online

Components Required:

1. None

3.1 Digital Modulation

At this point we understand the basics of digital data transmission. In the previous practical we analyzed a baseband communication system, where one's and zero's are simply represented using two finite duration signals.

In this practical we will be analyzing digital modulation schemes. Digital modulation is the process of encoding a digital information signal into the amplitude, phase, or frequency of the transmitted signal. Hence, there are three major digital modulation techniques.

- Amplitude-Shift Keying (ASK)
- Frequency-Shift Keying (FSK)
- Phase-Shift Keying (PSK)

In simple terms, we will use sinusoids to encode bits or a group of bits. Consider the sinusoid $x(t) = A\sin(2\pi ft + \phi)$, where A is the amplitude, f is the frequency and ϕ is the phase. We can change one of these three properties to obtain distinct finite duration signals, which we use to encode bits or groups of bits.

For example we will consider ASK. Here, we first divide the bit stream into blocks of length n, where n is referred to as the order of the modulation scheme. Observe that each block may have 2^n possible strings. Hence, we will create 2^n sinusoids with distinct amplitudes each of which is mapped to a single string of length n. Then we encode each block with the respective sinusoid. This is known as 2^n-level ASK modulation. In case of FSK and PSK, we change the frequency and the phase of the sinusoids instead of the amplitude to generate the 2^n sinusoids.

3.2 Pre-Lab

We use constellation diagrams to represent digital modulation schemes. Read about constellation diagrams and understand how they are drawn (https://en.wikipedia.org/wiki/Constellation_diagram).

Task 1. *Draw the constellation diagrams of a 2-level ASK schemes where the amplitudes are (1) bit 0 → -1 and bit 1 → 1 (2) bit 0 → 0 and bit 1 → 1 side by side.*

Task 2. *Draw the constellation diagrams of a 2-level PSK schemes where the phases are (1) bit 0 → 0 and bit 1 → π (2) bit 0 → 0 and bit 1 → $\pi/2$ side by side.*

3.3 Amplitude-Shift Keying

We will first generate a 2-level ASK. Save the following function as ask.m and change the current directory path in Matlab to the location where you saved this M-File.

```
function []=ask(bit_pattern,n)
Cf = 1.2E6; % Carrier frequency 1.2 MHz;
% We will represent the signal using samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
% and a bit rate of 400 kbps i.e. 250 samples per bit
delt = 1E−8;
fs = 1/delt;
samples_per_bit=250;
tmax = (samples_per_bit∗length(bit_pattern)−1)∗delt;
t= 0:delt:tmax; % Time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)∗samples_per_bit+sample)=bit_pattern(bit_no);
 end
end

% See what it looks like
figure;
subplot(2,1,1);plot(t,bits);
ylabel('Amplitude');
title('Info signal');
axis([0 tmax −2 2]);
% ASK modulation
ASK=[];
if n==2
 for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (1)∗sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (0)∗sin(Wc);
 end
 ASK=[ASK mod];
 end
 subplot(2,1,2); plot(t,ASK);
 ylabel('Amplitude');
 title('ASK Modulated Signal');
 axis([0 tmax −2 2]);
end
end
```

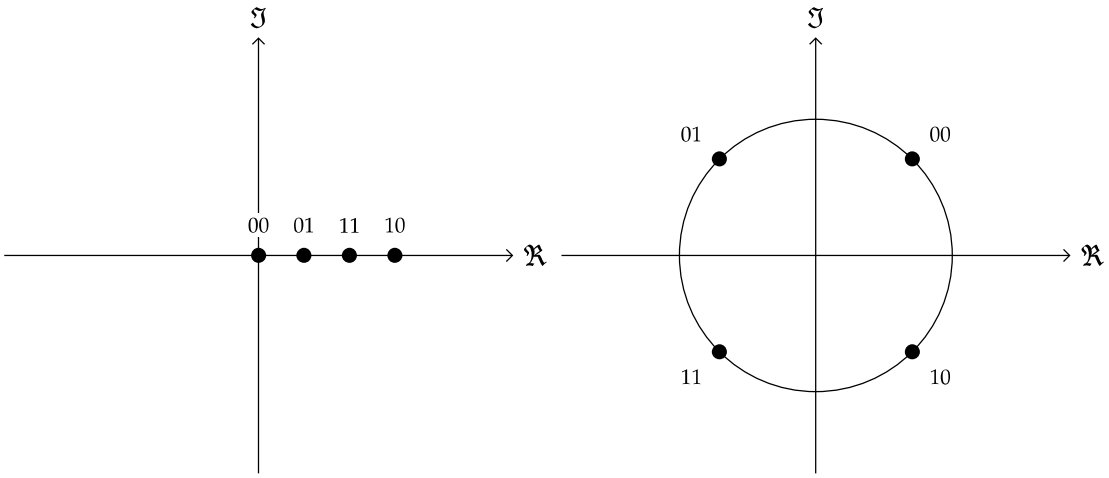
Generate the binary ASK modulation using the following.

```
close all;
clear all;
bit_pattern= [ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ] ;
ask(bit_pattern,2);
```

Observe the binary ASK modulation and show the output to a Laboratory Instructor.

Task 3. *Fill the Table in the Task Sheet based on you observations.*

Task 4. *Extend the function to generate 4-level ASK modulation, as specified by the signal constellation in Figure 3.1a, when the second parameter of the function is set as 4. Show the code and the output to a Laboratory Instructor. Fill the Table in the Task Sheet.*



(a) 4-level ASK (b) QPSK

Figure 3.1: Constellation diagrams

3.4 Frequency-Shift Keying

We will now move onto FSK. Generate the FSK modulation using the following script.

```
%fsk.m
close all;
clear all;
bit_pattern=[ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ];
Cf0 = 0.8E6; % Carrier frequency to encode binary 0, 0.8 MHz;
Cf1 = 2.4E6; % Carrier frequency to encode binary 1, 2.4 MHz;
% We will represent the signal using samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
% and a bit rate of 400 kbps i.e. 250 samples per bit
delt=1E−8;
fs=1/delt;
samples_per_bit=250;
tmax = (samples_per_bit*length(bit_pattern)−1)*delt;
t = 0:delt:tmax; %time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)*samples_per_bit+sample)=bit_pattern(bit_no);
 end
end
% See what it looks like
figure;
subplot(2,1,1); plot(t,bits);
```

```
ylabel ('Amplitude');
title ('Info signal');
axis([0 tmax -2 2]);
% FSK Modulation
FSK=[];
for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit =
(bit_no-1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit-1)*delt;
 Wc = Cf1*2*pi*t_bit;
 mod = (1)*sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit =
(bit_no-1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit-1)*delt;
 Wc = Cf0*2*pi*t_bit;
 mod = (1)*sin(Wc);
 end
 FSK=[FSK mod];
end
subplot(2,1,2); plot(t,FSK);
ylabel ('Amplitude');
title ('FSK Modulated Signal');
axis([0 tmax -2 2]);
```

Observe FSK modulation scheme and show the output to a Laboratory Instructor.

Task 5. *Fill the Table in the Task Sheet.*

3.5 Phase-Shift Keying

Finally, we will analyze phase-shift keying. Save the Matlab function given below as npsk.m and change the current directory path in Matlab to the location where you saved this M-File.

```
function []=npsk(bit_pattern,n)
Cf = 4E5; %Carrier frequency 0.4 MHz;
% We will represent the signal as samples taken at intervals of le-8 S
% i.e., a sampling frequency of 100 MHz
delt=1E-8;
fs=1/delt;
samples_per_bit=250;
tmax = (samples_per_bit*length(bit_pattern)-1)*delt;
t = 0:delt:tmax; %time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no-1)*samples_per_bit+sample)=bit_pattern(bit_no);
 end
end
% See what it looks like
figure;
subplot(2,1,1); plot(t,bits);
ylabel ('Amplitude');
title ('Info signal');
axis([0 tmax -2 2]);
if n==2
 % BPSK Modulation
```

```
BPSK=[];
for bit_no=1:1:length(bit_pattern)
if bit_pattern(bit_no)==1
t_bit = (bit_no1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit-1)*delt;
Wc = Cf*2*pi*t_bit;
mod = (1)*sin(Wc);
elseif bit_pattern(bit_no)==0
t_bit = (bit_no1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit-1)*delt;
Wc = Cf*2*pi*t_bit;
mod = (1)*sin(Wc+pi);
end
BPSK=[BPSK mod];
end
subplot(2,1,2); plot(t,BPSK);
ylabel ('Amplitude');
title ('BPSK Modulated Signal');
axis([0 tmax -2 2]);
end
end
```

Generate the binary PSK modulation (BPSK) using the following code.

```
close all;
clear all;
bit_pattern= [ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ] ;
npsk(bit_pattern,2);
```

Observe BPSK modulation and show the output to a Laboratory Instructor.

Task 6. *Fill the Table in the Task Sheet.*

Task 7. *Extend the function to generate the Quaternary Phase Shift Keying (QPSK) modulation as specified by the signal constellation in Figure 3.1b, when the second parameter of the function is set as 4. Show the code and the output to a Laboratory Instructor. Fill the Table in the Task Sheet.*

Task 8. *Sketch constellation diagrams for 8-level and 16-level PSK, side by side.*

Task 9. *Comment on the advantages and disadvantages of higher order modulation schemes.*

♣ The End ♣

Workshop 4: Communication Networks and Protocols

Objective: To observe and analyze the basic operation of a communications network using packet sniffing.

Outcome: After successfully completion of this session, the student would be able to

1. Become familiar with the Wireshark packet sniffing tool
2. Be able to appreciate how the layered protocol stack facilitates the functioning of a communications network.

Equipment Required:

1. A Personal Computer with a network interface
2. Internet connectivity
3. Wireshark (the most recent version)
4.

Components Required:

None

This lab is adapted from a series of `Wireshark` Labs provided as a supplement to *Computer Networking: A Top-Down Approach,* 7th ed., J.F. Kurose and K.W. Ross.

4.1 Introduction

Our understanding of communication networks and protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols". In this lab, we will be working with a "real" network environment, and use `Wireshark` to capture data packets that are passing through. You will run various network applications in different scenarios using your own computer and observe the network protocols "in action," interacting and exchanging messages with entities elsewhere in the Internet. Thus, you and your computer will be an integral part of these "live" labs. You'll observe, and you'll learn, by doing.

In this lab, you'll get acquainted with `Wireshark`, make some simple packet captures with it, and gain some insight into how communications networks operate.

4.2 Pre-Lab

Step 1: Please read Annex 4.5: "An Introduction to `Wireshark`" and install the software on your computer using the given instructions.
Step 2: Take `Wireshark` for a Test Run following the instructions given in Annex 4.5: "Wireshark Test Run".
Step 3:

Task 1. *Copy the protocol listing obtained after the last step in the Test Run in the space given in the Task Sheet.*

4.3 Observing the TCP/IP protocol stack

In this lab, you will observe the TCP/IP protocol in action using the common data communication activity of browsing the web. You will also learn some more features of `Wireshark` during this activity. Follow the steps given below.
Step 1: Erase your recent browsing history, execute the Test Run again and answer the following questions. Explain how you obtained your answer in each case.

Task 2. *How long did it take from when each HTTP GET message was sent until the corresponding HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since* `Wireshark` *tracing began. To display the Time field in time-of-day format, select the* `Wireshark` *View pull down menu, then select Time Display Format, then select Time-of-day.).*

Task 3. *What is the IP address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the IP of your computer?*

Step 2: Select the first HTTP GET message and from the top menu select

Analyze -> Conversation Filter -> TCP.

Task 4. *Sketch the sequence of activities that you see between your computer and the web server at gaia.cs.umass.edu. The Conversation Filter extracts the entire sequence of messages (involving TCP in this case) between the endpoints in the selected message.*

Step 3: Remove the packet filtering.

Task 5. *List 3 different protocols that appear in the protocol column in the packet-listing window.*

Step 4: Using the filter, select only the messages involving your computer (either as source or destination).

Task 6. *What new destinations do you observe? What new protocols do you see in action? What are their purposes?*

4.4 Discussion

Task 7. *Describe how this lab has helped sharpen your interest in communication networks and protocols. Also state how this lab can be improved in future. Use 150 – 300 words.*

4.5 Annex: An introduction to Wireshark

4.5.1 What is a Packet Sniffer ?

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Figure 4.1 shows the structure of a packet sniffer. At the right of Figure 4.1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 4.1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP (that we use to browse the web) all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable or a wireless medium. In Figure 4.1, the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

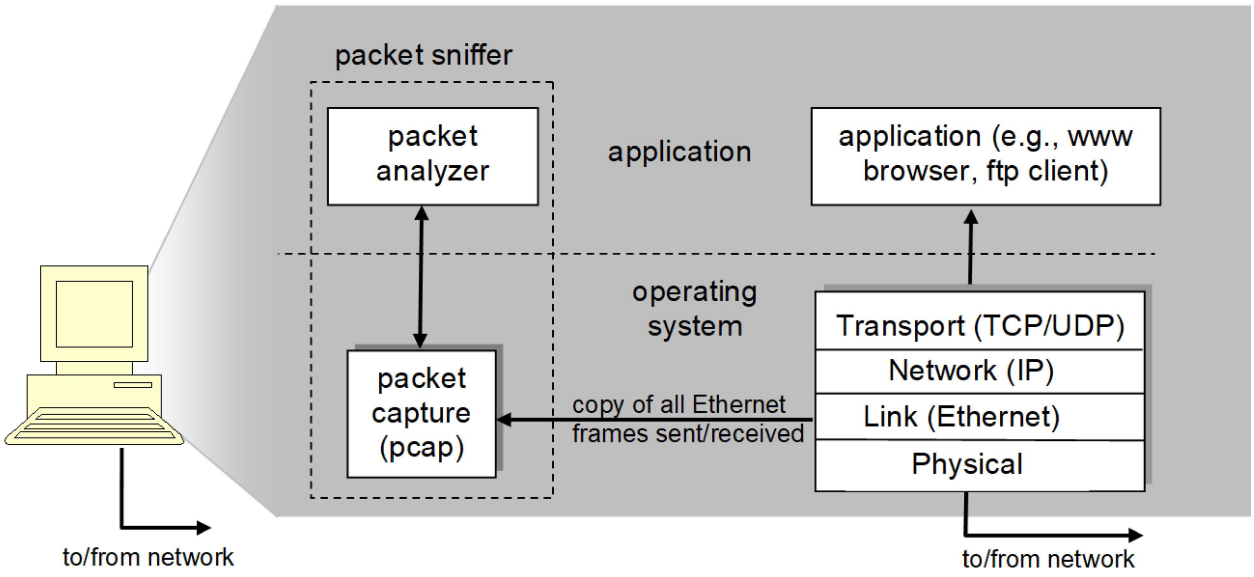


Figure 4.1: Packet Sniffer Structure

The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 4.1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, is able to identify the contents of HTTP message.

4.5.2 Wireshark

`Wireshark` is a packet sniffer http://www.wireshark.org/ which allows us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, `Wireshark` is a packet analyzer that uses a packet capture library in your computer). `Wireshark` is a free network protocol analyzer that runs on Windows, Mac, and Linux/Unix computer. It's an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user-guide (http://www.wireshark.org/docs/wsug_html_chunked/), man pages (http://www.wireshark.org/docs/man-pages/), and a detailed FAQ (http://www.wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface.

4.5.3 Installing Wireshark

In order to run `Wireshark`, you will need to have access to a computer that supports both `Wireshark` and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed for you, if it is not installed within your operating system, when you install `Wireshark`. See http://www.wireshark.org/download.html for a list of supported operating systems and download sites

Download and install the `Wireshark` software:

- Go to http://www.wireshark.org/download.html and download and install the `Wireshark` binary for your computer.

4.5.4 Running Wireshark

When you run the `Wireshark` program, you'll get a startup screen that looks similar to Figure 4.2. (Note: Different versions of `Wireshark` will have different startup screens.) In the Capture section of the screen, there is a list of communication interfaces available in the computer. To capture packets, you need to first select an interface from this list.

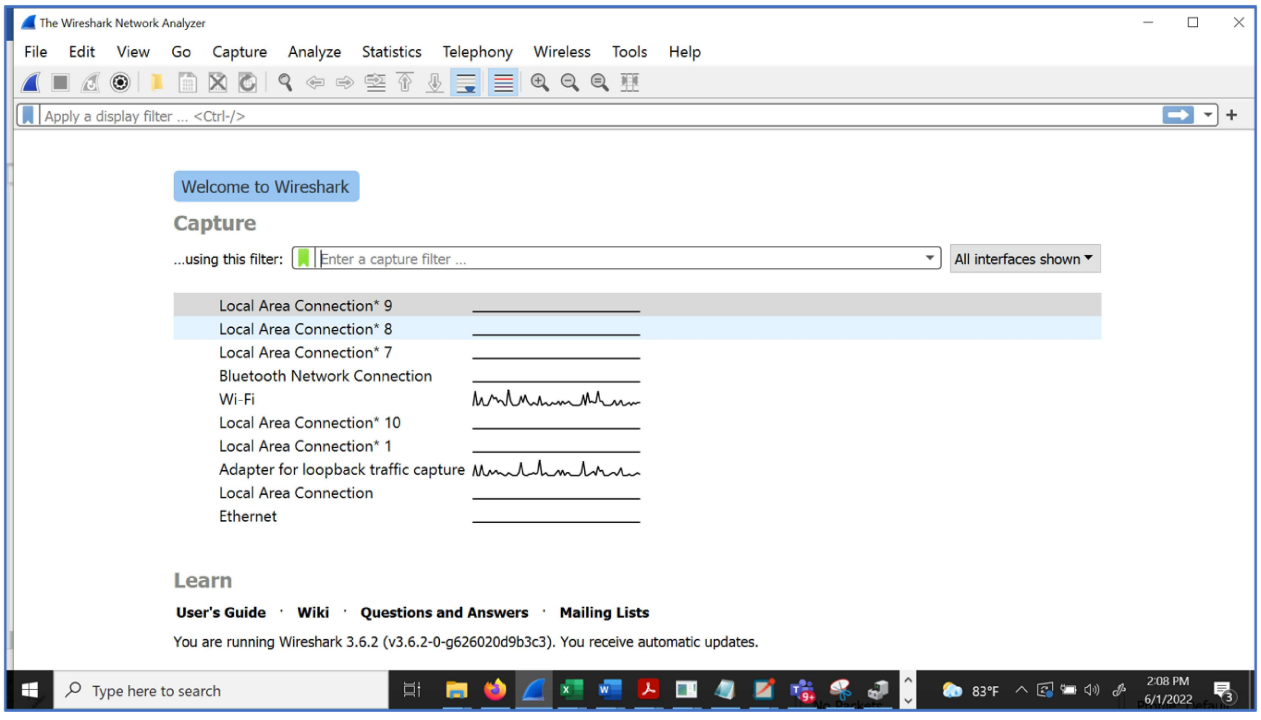


Figure 4.2: The Wireshark start-up screen

When you click on one of these interfaces to start packet capture (i.e., for `Wireshark` to begin capturing all packets being sent to/from that interface), a screen like the one in Fig. A16.1.3 will be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the *Capture* pull down menu and selecting *Stop*.

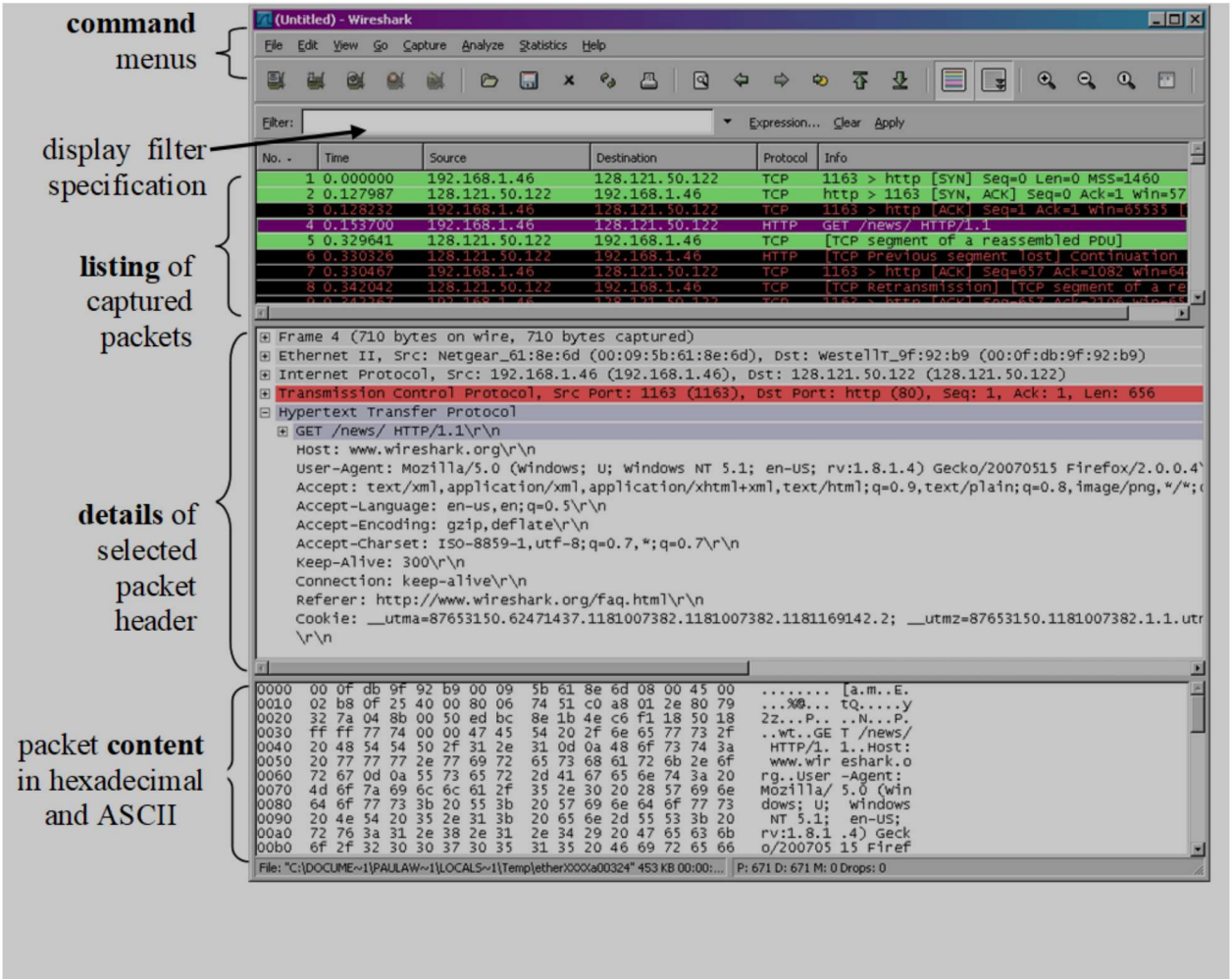


Figure 4.3: Wireshark Graphical User Interface during packet capture and analysis

The `Wireshark` interface has five major components as shown in Figure 4.3. These are described below:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the `Wireshark` application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by `Wireshark`; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the

packet's one-line summary in the packet-listing window and click with the left mouse button.) These details include information about all the protocols over which the packet has been carried, up to the highest level.

- The **packet-contents** window displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the `Wireshark` graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (e.g. to select and display only packets using a particular protocol).

4.6 Annex: A Wireshark Test Run

Follow the steps given below to complete your first run with `Wireshark`. You will access a web page, and examine the protocols that enable you to do so.

1. Start up your web browser, which will display your selected homepage.
2. Start `Wireshark`.
3. To begin packet capture, select the Capture pull down menu and select *Options*. This will cause the "Wireshark: Capture Options" window to be displayed with the available interfaces as shown in Figure 4.1 earlier.
4. Double-click on the interface that you wish to capture packets on (i.e., the network interface you use for Internet access). Packet capture will now begin - `Wireshark` is now capturing all packets being sent/received from/by your computer! Once you begin packet capture, a window similar to that shown in Figure 4.3 will appear. This window shows the packets being captured. By selecting `Capture` pulldown menu and selecting *Stop*, you can stop packet capture. But don't stop packet capture yet.
5. Let's capture some interesting packets now. To do so, we'll need to generate some network traffic. We will look at the HTTP protocol that is used to download content from a website. While `Wireshark` is running, enter the URL: `http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html` and have that page displayed in your browser.

   In order to display this page, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by `Wireshark`.
6. After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop `Wireshark` packet capture. The main window should now look similar to Figure 4.3.
7. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the *Protocol* column in Figure 4.3). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. We'll learn much more about these protocols as we progress through the text! For now, you should just be aware that there is often much more going on than "meet's the eye"!
8. Type in "http" (without the quotes, and in lower case – all protocol names are in lower case in `Wireshark`) into the display filter specification window at the top of the main `Wireshark` window. Then select *Apply* (to the right of where you entered "http"). This will cause only HTTP message to be displayed in the packet-listing window.

♣ The End ♣

Workshop 5: Point-to-Point Communication

Objective: To build and test a point-to-point communication system.

Outcome: After successfully completion of this session, the student would be able to

1. Identify the basic elements of a point-to-point communication system
2. Implementing a point-to-point communication system using 315/433 transmitter-receiver modules
3. Analyzing the packet-error rate of a point-to-point communication link

Equipment Required:

1. A Personal Computer Installed with Arduino software and RadioHead library
2. Two Arduino UNO Boards

Components Required:

315/433MHz transmitter-receiver modules

Two copper cables of length 17cm

Jumper wires

5.1 Point-to-Point Communication System

A point-to-point communication link connects a transmitter to a single receiver. The communication between an aircraft and a control tower is an example of a point-to-point communication link. In contrast, in point-to-multi-point (or a broadcast) communication, the transmitter can be heard by multiple receivers. A radio or TV broadcast system is an example. In this lab, we will be implementing a point-to-point communication link using 315/430 MHz transmitter-receiver modules. The link will transmit data in the form of packets.

5.2 Pre-Lab

In the pre-lab you will understand the setup for the implementation. You will also install and configure the necessary software.

5.2.1 Hardware Setup

We will be using 315/433MHz transmitter-receiver modules for communications. The module has a transmitter and a receiver. Both the modules are controlled by two separate Arduino UNO boards. We use the RadioHead Library to handle the communication. Figure 5.1 illustrates a block diagram of the communication system.

315/433 MHz transmitter module

The transmitter module implements OOK (On-Off Keying). It mainly consists of three sections.

- SAW resonator generating a 433MHz
- Switching transistor
- Antenna

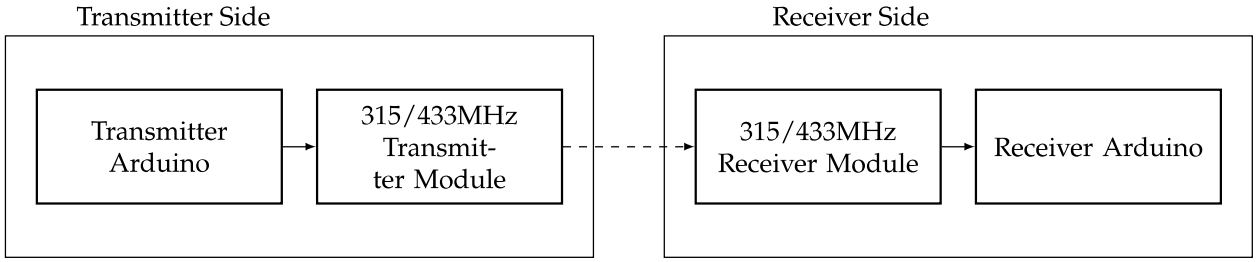


Figure 5.1: The block diagram of the point to point communication system we will be implementing

The working principle is simple. The resonator generates a continuous sinusoidal waveform. The switching transistor governs the connection between the antenna and the resonator. The binary data stream is connected to the the switching transistor. When binary 0 is given the switch is in off state and when binary 1 is given switch is in on state thereby providing a sinusoidal waveform to the antenna which is subsequently transmitted. The frequency of the waveform can be either 315 MHz or 433MHz depending on how the device is tuned. This technique is known as On-Off Keying (OOK), a basic digital modulation technique. The sinusoidal waveform is the carrier.

Figure 5.2 shows the transmitted waveform for the bit stream "111010011" using this technique. The Figure 5.3 shows the connections and the components of the transmitter module.

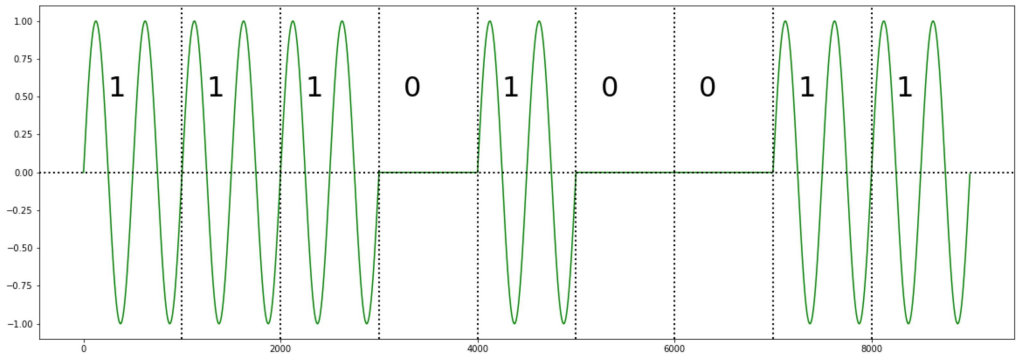


Figure 5.2: OOK waveform for the bit stream "111010011

5.2.2 315/433 MHz receiver module

The receiver module is a bit more complex than the transmitter but still has a simple implementation. The receiver contains four major parts

- Antenna
- RF Tuner Circuit
- Amplifier
- Phase-Locked Loop

The Antenna receives the RF signal transmitted by the transmitter. The RF Tuner Circuit is responsible for tuning the receiver to the frequency of the transmitted signal. The amplifier amplifies the transmitted signal. The Phase-Locked Loop is responsible for decoding the OOK signal and to generate the bit stream. Figure 5.4 shows the connections and the components of the receiver module.

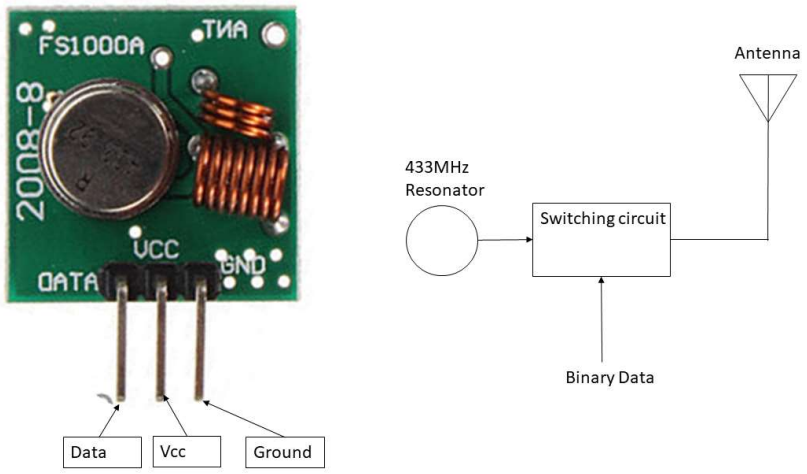


Figure 5.3: The transmitter module

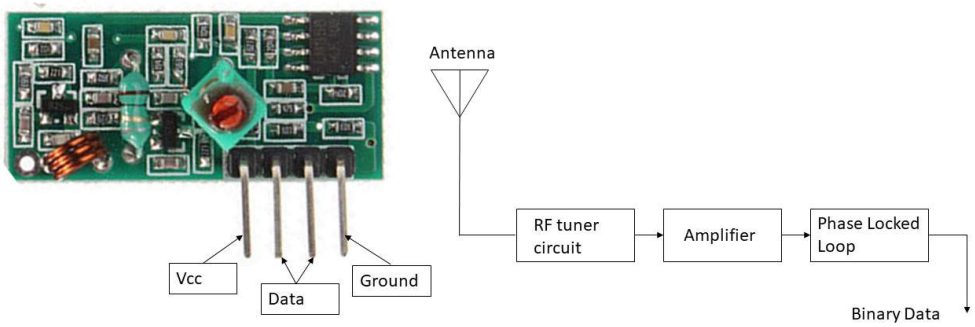


Figure 5.4: The receiver module

5.2.3 Setting Up the Arduino Software

Before moving onto the practical you will have to download the Arduino software. You can download the version compatible with your PC from https://www.arduino.cc/en/software. After downloading, install the software.

5.2.4 Setting Up the RadioHead Library

After downloading the Arduino IDE (Integrated Development Environment), you will have to add the RadioHead software. RadioHead provides many libraries which can be configured easily for different applications in wireless communications. If RadioHead is not already added to Arduino you can download the zip folder from http://www.airspayce.com/mikem/arduino/RadioHead/RadioHead-1.121.zip. After that you can add the zip library by **Sketch → Include library → Add .ZIP Library...**

In order to ensure reliable transmission of data, we transmit data in the form of packets. The RadioHead library is responsible for encapsulating data into packets. In simple terms it appends a training preamble, start symbol and a frame check sequence to the data. The composition of a RadioHead packet is depicted in Figure 5.5.

Training preamble: consists of 36 alternating 1's and 0's. Used by the receiver to adjust it's gain.

Start symbol: consists of 12 bits. These 12 bits indicate the receiver that a new data packet is arriving and it indicates when the actual data block will start.

Frame check sequence (FCS): These 16 bits are used by the receiver to check whether bit-errors have occurred.

Payload: Payload is the part which contains the actual data. In addition to the data it may contain the receiver address and the packet identification number. Receivers address is important when there are several transmitter-receiver pairs. In that case the receiver needs to know that the message is intended to it. Packet identification number is unique for each packet intended to a particular receiver. Depending on the size of the actual data block, number of bits in the payload may vary.

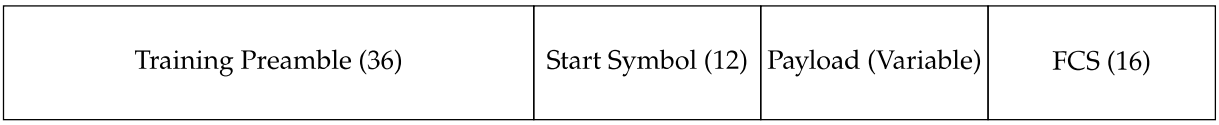


Figure 5.5: The RadioHead Packet

‘

5.3 Implementation of The Point to Point Communication System

Since the transmitter module implements OOK, we utilize the ASK (Amplitude Shift Keying) library of RadioHead. OOK is a special case of the more general digital modulation scheme ASK. Now we are ready to build the point to point communication link. First we will program the Arduino boards to send a message from the transmitter to the receiver.

5.3.1 The Transmitter Side

In our implementation we let the length of the payload section to be 6 bytes (48 bits). Since we are building a point-to-point communication link, the transmitted packets are intended only to a single receiver. First two bytes of the payload represents the receiver address. The instructor will provide you with the address for your group. The address is an integer between 0 and 99. For example if the address is 73, the first byte will represent the ASCII code for 7 and the second byte will represent the ASCII code for 3. The next four bytes will represent the packet ID which is a number from 0 to 1023 (Note that we are not transmitting actual data using these packets).

The following code will iteratively transmit packets with ID's ranging from 0 to 1023. Once it transmits a packet with ID 1023 it will start again from a packet with ID 0.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>

// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

// This array will store the six bytes representing the payload
char payload[6];
```

```
// Ask the instructor for the number of the receiver you
// are communicating with
int rec_num = 0;

// Stores the current id of the current transmitted packet

int current_packet = 0;

void calcRecNum(int receiver_number){
 // Calculates the two bytes representing the receiver_number
 int  p = receiver_number;
 int  i;
 for(i=0;i<2;i++){
   payload[1-i] = char((p%10)+int('0'));
   p = p/10;
 }
}

void calcId(int packet_ID){
 // Calculates the four bytes representing the packet ID
 int  p = packet_ID;
 int i;
 for(i=0;i<4;i++){
   payload[5-i] = char((p%10)+int('0'));
   p = p/10;
 }
}

void setup()
{
  // Initialize ASK Object
  rf_driver.init();
  // Set the receiver number
  calcRecNum(rec_num);
  Serial.begin(9600);
}



void loop()
{
  calcId(current_packet);
  rf_driver.send((uint8_t *)payload, strlen(payload));
  rf_driver.waitPacketSent();
  delay(100);

  // Incrementing the packet ID
  current_packet = current_packet + 1;
  if(current_packet == 1024){
    current_packet = 0;
  }

}
```

Connect the Arduino and the 315/433MHz transmitter module as shown in Figure 5.6. Enter the above code to a new Arduino file. Connect the Arduino to the PC. Using **Tools** → **Port** select the correct COM port to which the Arduino is connected. Click on the Upload button and wait until the uploading finishes. Now

you can unplug the Arduino from the PC.

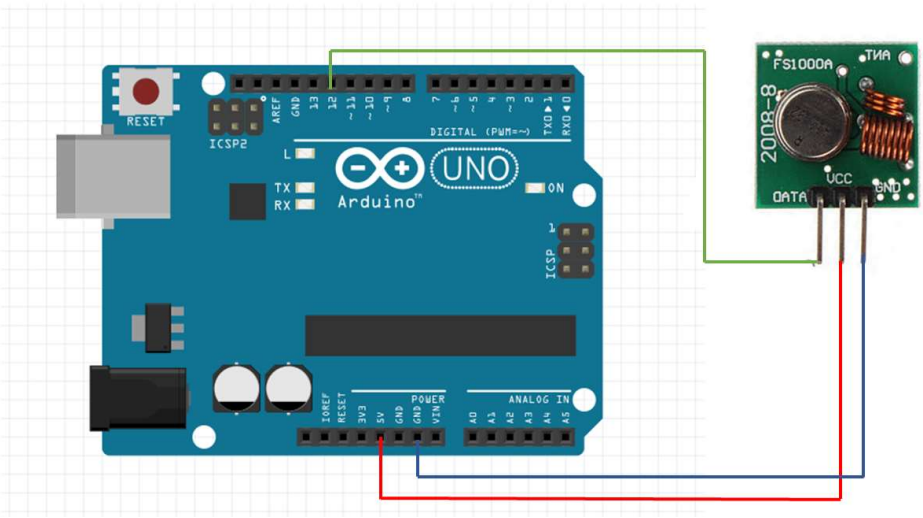


Figure 5.6: Configuration of the transmitter side

5.3.2 The Receiver Side

Next you will program the Arduino board for the receiver. When a packet arrives at the receiver, it checks whether the packet is valid by comparing the frame check sequence with the rest of the packet. If the packet in invalid (if bit errors have occurred during transmission) the packet is discarded. Then the receiver checks whether the packet is intended for it. If so the receiver calculates the ID of the packet which is subsequently printed to the serial monitor.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>
// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

const int sent_size = 1024;

//This array will store whether a packet with a particular ID is being received
bool rec[sent_size];

// Number of this receiver
int rec_num = 0;


// This array stores the received message
uint8_t buf[6];
uint8_t buflen;


bool checkRecNum(int receiver_number){
  //Checks whether the message is intended to the receiver
  int c = 0;
```

```
  int i;
  int pow10 = 1;
  for(i=0;i<2;i++){
     c = c + (pow10*(buf[1-i]-'0'));
     pow10 = pow10*10;
  }
  if(c == receiver_number){
    return true;
  }
  return false;

}



int calcPacID(){
//Calculates the packet ID
   int i;
   int pac_ID = 0;
   int pow10 = 1;
   for(i = 0;i<4;i++){
      pac_ID = pac_ID + (pow10*(buf[5-i]-'0'));
      pow10 = pow10*10;
   }
   return pac_ID;
}

void setup()
{
   // Initialize ASK Object
   rf_driver.init();
   // Setup Serial Monitor
   Serial.begin(9600);
}

void loop()
{

   buflen = sizeof(buf);
   // Check if received packet is correct size
   if (rf_driver.recv(buf, &buflen))
   { // Message received with valid checksum
     if(checkRecNum(rec_num)){
       // Message is intended to the receiver
       //Calculating the received packet ID
       int pac_ID = calcPacID();
       rec[pac_ID]=1;

       Serial.print("Packet Received: ");
       Serial.println(pac_ID);
     }
   }

}
```

Connect the Arduino and the 315/433MHz receiver module as shown in Figure 5.7. Similar to the transmitter side Arduino, upload the code to the Arduino board for the receiver. Do not disconnect the Arduino from the PC. Now connect the transmitter side Arduino to a power source. Navigate to **Tools** → **Serial Monitor**.

The outputs should appear on the serial monitor (Make sure the correct COM port is selected).

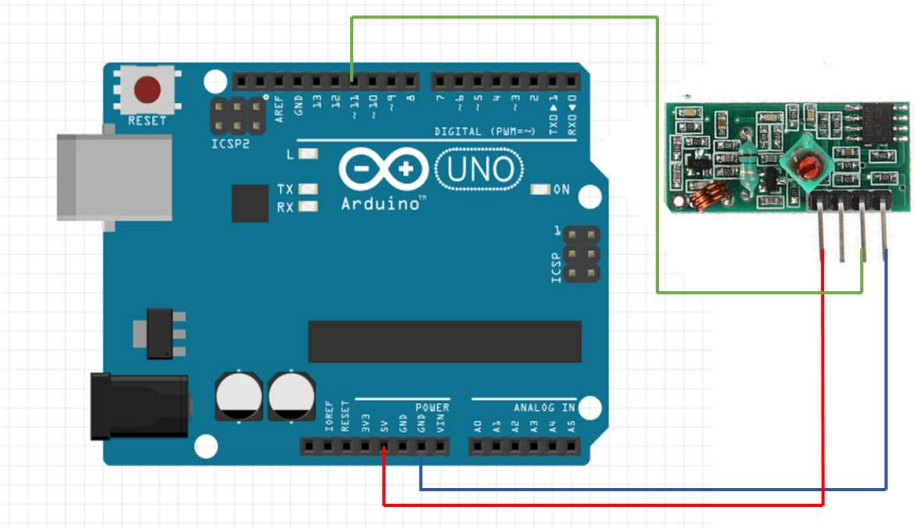


Figure 5.7: Configuration of the receiver side

Task 1. *Increase the distance between the transmitter and the receiver and identify the minimum distance beyond which packets are no longer received.*

5.4 Analyzing the Packet Error Rate of a Point to Point Communication System

Now we modify our code to send each packet only once (The packets with ID's from 0 to 1023 are sent only once). At the receiver we check how many of the transmitted packets are correctly received. Then the packet error rate is calculated as,

$$\text{Packet Error Rate} = 1 - \frac{\text{Number of Packets Correctly Received}}{1024} \tag{5.1}$$

Use the following code for the transmitter.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>

// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

// This array will store the six bytes representing the payload
char payload[6];

// Ask the instructor for the number of the receiver you
// are communicating with
int rec_num = 0;

// Stores the current id of the current transmitted packet
```

```
int current_packet = 0;

void calcRecNum(int receiver_number){
 // Calculates the two bytes representing the receiver_number
 int  p = receiver_number;
 int  i;
 for(i=0;i<2;i++){
   payload[1−i] = char((p%10)+int('0'));
   p = p/10;
 }
}

void calcId(int packet_ID){
 // Calculates the four bytes representing the packet ID
 int  p = packet_ID;
 int i;
 for(i=0;i<4;i++){
   payload[5−i] = char((p%10)+int('0'));
   p = p/10;
 }
}

void setup()
{
  // Initialize ASK Object
  rf_driver.init();
  // Set the receiver number
  calcRecNum(rec_num);
  Serial.begin(9600);
}



void loop()
{

  if(current_packet<1024){
    calcId(current_packet);
    rf_driver.send((uint8_t *)payload, strlen(payload));
    rf_driver.waitPacketSent();
    delay(100);
    // Incrementing the packet ID
    current_packet = current_packet + 1;
  }
}
```

Upload the following code to the receiver side Arduino.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>
// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

const int sent_size = 1024;
```

```
//This array will store whether a packet with a particular ID is being received
bool rec[sent_size];

// Number of this receiver
int rec_num = 0;


// This array stores the received message
uint8_t buf[6];


//Control paremeters to check whether the packet receiving has finished
long int last_time = 0;
long int cur_time = 0;
int printed = 0;
uint8_t buflen;
int cur_pac = 0;
long int time_per_pac = 225;

bool checkRecNum(int receiver_number){
  //Checks whether the message is intended to the receiver
  int c = 0;
  int i;
  int pow10 = 1;
  for(i=0;i<2;i++){
     c = c + (pow10*(buf[1-i]-'0'));
     pow10 = pow10*10;
  }
  if(c == receiver_number){
    return true;
  }
  return false;

}



int calcPacID(){
//Calculates the packet ID
   int i;
   int pac_ID = 0;
   int pow10 = 1;
   for(i = 0;i<4;i++){
      pac_ID = pac_ID + (pow10*(buf[5-i]-'0'));
      pow10 = pow10*10;
   }
   return pac_ID;
}

void setup()
{
   // Initialize ASK Object
   rf_driver.init();
   // Setup Serial Monitor
   Serial.begin(9600);
}

void loop()
{
```

```
  buflen = sizeof(buf);
  // Check if received packet is correct size
  if (rf_driver.recv(buf, &buflen))
  { // Message received with valid checksum
    if(checkRecNum(rec_num) && printed == 0){
      // Message is intended to the receiver

      //Calculating the received packet ID
      int pac_ID = calcPacID();
      rec[pac_ID]=1;
      cur_pac = pac_ID;
      Serial.print("Packet Received: ");
      Serial.println(pac_ID);
      last_time = millis();
    }
  }
  else{
    // If no valid packet is received for within the remaining
    // packet receiving has finished
    cur_time = millis();
    if(cur_time - last_time > time_per_pac*(sent_size-cur_pac)){
      if(printed ==0){
        int sum = 0;
        int i;
        for(i = 0;i<1024 ;i++){
         sum=sum+rec[i];
        }
        Serial.print("Packet Receiving Finished. Packet Error Rate: ");
        Serial.println(((float) (sent_size-sum)/(float) 1024));
        printed = 1;
      }
    }
  }
}
```

To ensure proper operation, make sure you power the transmitter side Arduino after powering the receiver side Arduino. Monitor the communication using the serial monitor as mentioned in section 5.3.2. Wait for the communication to complete. The packet error rate will appear on the serial monitor afterwards.

Task 2. *Calculate the PER for 5 different distances. Take 5 distances to cover the range between 0 and the distance obtained in task 1. For each distance repeat do the experiment for 5 iterations and obtain the average PER. Record the results in the table in the Task Sheet.*

Task 3. *Discuss the impact of the distance between the transmitter and the receiver antennas on reliable communication.*

Task 4. *Repeat task 2 with 17cm antennas fixed to the transmitter and the receiver and record the results in the table in the Task Sheet.*

Task 5. *Discuss the impact of adding an antenna.*

♣ The End ♣