This file contains the solutions!
To remove, set "\def\sol{x}" to 0

3 Pulse Amplitude Modulation

In this lab, you will implement a simple communication system based on pulse amplitude modulation (PAM). First using uni-polar PAM and then, through simple additions to the flowgraph, switch to bi-polar PAM. Finally you will transmit an audio signal and observe the effect of quantization on the audio quality.
Here is a list of all the blocks that will be helpful throughout this lab. Based on the instructions, it will be up to you to use them accordingly:

- `Add`
- `Add Const`
- `Multiply Const`
- `Float to UChar`
- `UChar to Float`
- `Random Source`
- `Interpolating/Decimating FIR Filter`
- `Root Raised Cosine Filter`
- `Skip Head`
- `Noise Source`
- `Repack Bits`
- `Throttle`

In case of doubt, you can access the documentation of a block by double clicking on it and going under the `Documentation` tab of each block.

As in previous labs, you can find a template code on Moodle that you can use to start this lab. The template code contains a few variables and GUI element to simplify the visualization of some signals. All the block will be useful during the lab, but some only in later steps.

3.1 Transmitter

We will start by implementing the transmitter. In the first part of the lab, we will use a uni-polar PAM scheme with data being randomly generated bits. The transmitter will perform the following operations:

1. Generate the data to be transmitted
2. Map the data to symbols
3. Shape the symbols into pulses

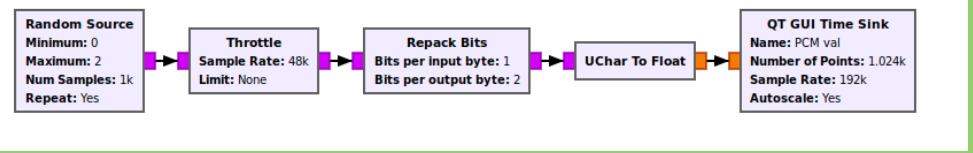
3.1.1 Data Generation

The first step is to generate the data that will be transmitted. We will use a binary sequence of 1's and 0's. A *Random Source* block can generate a sequence of bytes with values $\in [0, 2)$. When you have no block that have a fixed sampling rate, you have to use a *Throttle* block to limit the rate at which the data is generated. Without such a block in your datapath, your computer will generate data as fast as possible, using 100% of the CPU.

3.1.2 Symbol Mapping

We want our system to support any M-ary PAM schemes. To do so, we will use a *Repack Bits* block to gather multiple bits into a single symbol based on the variable *bit_per_symb* already present in the flowgraph. Finally, convert the byte value of the symbol to a float value to be able to make further signal processing operation in a "continuous" domain.
You should now be able to visualize the generated symbols in the time domain and vary the number of bits contain in each symbol using the corresponding slider already available in the GUI.

Answer:



3.1.3 Upsampling

The next step is to shape the symbols into pulses. Currently our discrete signal only contains the symbol values at specific time instances. To simulate a continuous time signal we should first create an upsample signal where each symbol is a Dirac pulse. As we remain in the digital domain, we will simply insert (*os_factor*-1) zeros between each symbol.
Conveniently, the block *Interpolating FIR Filter* can do this for us. This block does two consecutive operations: upsampling and filtering.
The upsampling is done by inserting zeros between each symbol based on the interpolation rate. The filtering is done by convolving the upsampled signal with the filter taps specified as a python list (e.g. [1,0.7,1]).

Configure the block such that you obtain a serie of dirac pulses, each one corresponding to one symbol.

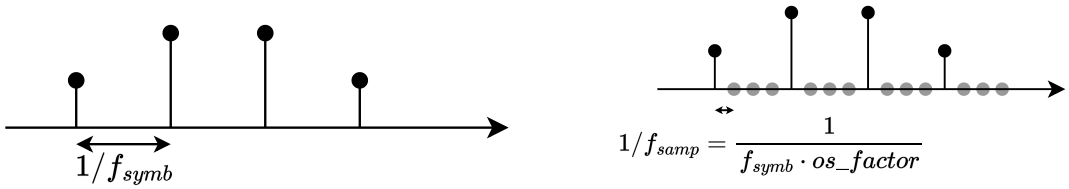


Figure 3.1: Signal before and after the upsampling

3.1.4 Pulse Shaping

By connecting the output of the *Interpolating FIR Filter* to a *QT GUI Frequency Sink* block, you should be able to visualize the signal in the frequency domain.

Question: How much of the bandwidth is used by the signal, is it expected?

Answer:
The upsampled signal uses the entire bandwidth. The fourier transform of a dirac is a constant value.

Ideally, we would like to transmit signals with a bandlimited spectrum. Limiting the frequency domain of a signal can be interpreted as multiplying the frequency domain of the signal with some type of window function. One of the most common windows respecting the Nyquist ISI criterion and offering a good tradeoff between duration in time domain and bandwidth in frequency domain is the raised cosine.

The *Root Raised Cosine Filter* block can be used to apply a raised cosine filter to the upsampled signal. One should note that this is a **root** raised cosine filter, meaning that the filter should be applied twice successively to the signal.

Filter the pulses with a raised cosine. The sample and symbol rate parameters are already set in the flowgraph and the parameter alpha corresponds to the rolloff factor. Setting the number of taps to 10 times the *os_factor* should be enough for this lab.

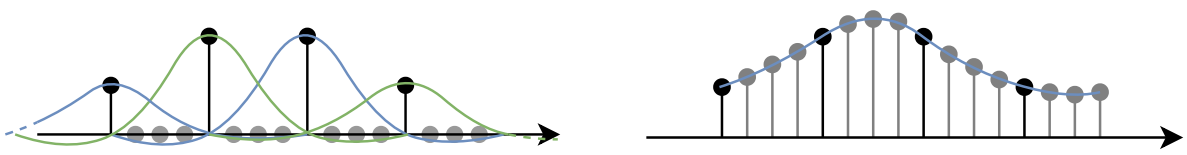


Figure 3.2: Signal before pulse shaping with symbol pulses illustration and signal after pulse shaping

Connect to the Time and Frequency sinks to visualize the signal before and after the pulse shaping on the same plot.

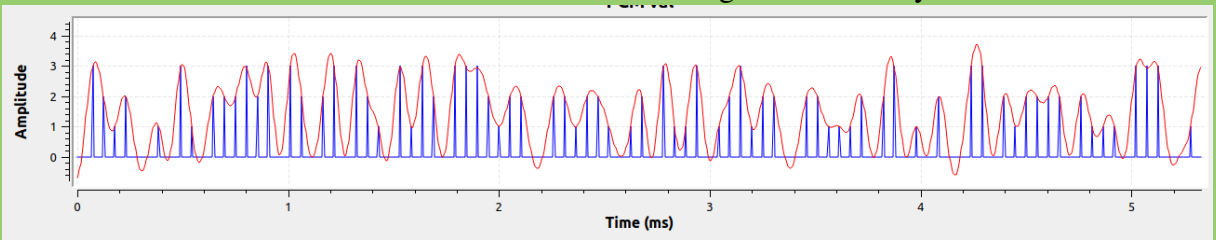
The signal after the pulse shaping is not aligned in the time domain with the upsampled signal.

Question: Can you compensate the delay and the scaling introduced by the FIR filters?

Note on GNU Radio Root Raised Cosine FIR filters:

- The delay introduced by the filter is half the number of taps.
- The energy of the signal is the same before and after the filter for a gain of 1.

Answer:
The gain of the RRC can be set to os_factor**0.5 as the filter would keep the power constant if the gain is set to 1, causing the energy of the dirac pulse to be spread across more samples.
For the time delay introduced by the FIR filters, you can add a *Skip Head* block with the number of taps in the RRC filter. This number comes from twice half of the filter length added by each RRC filter.



Bandwidth Usage

In this lab we assume that we want to keep the bandwidth of the transmitted signal as low as possible while still being able to send the data in real time. It means that we have an original source with a fixed rate (e.g. an audio file with a sampling rate of $48\,\mathrm{kHz}$).
The main contributors to the bandwidth of the transmitted signal are: the number of bits per audio sample and the number of bit per symbol.

Question: How does the bandwidth of the transmitted signal changes with respect to the number of bit sent every symbol?

Question: How does the bandwidth of the transmitted signal changes with respect to the number of bit per audio samples?

Question: For which combinations of bit per symbol and bit per audio sample is the bandwidth the same as the original source (i.e. the bandwidth of the analog audio signal)?

Answer:

1. The required bandwidth is inversely proportional to the bit per symbols.
2. It is proportional.
3. When the number of bits per audio sample is equal to the number of bits per symbol,i.e. the symbol rate is the same as the audio sampling rate, the bandwidth is the same as the original source.

3.2 Channel

We will now simulate the transmission of the signal over an additive white gaussian noise (AWGN) channel. This noise model corresponds to the thermal noise added by electronic components (mainly the amplifiers) in the receiver chain.
Use an *Noise Source* to generate the noise and control its amplitude using the variable *noise_pow*. Note that we want to give the value of *noise_pow* in dB, however, the noise source takes the power in linear scale, therefore you should convert the value.
Hint: You can use inline python code in the variable field of the blocks in gnuradio companion (e.g, Amplitude 10**noise_pow*+5).
Add the noise to the signal at the input of your receiver.

Answer:
Since we have a real noise, the amplitude is given by $10^{\text{power\_dB}/10}$

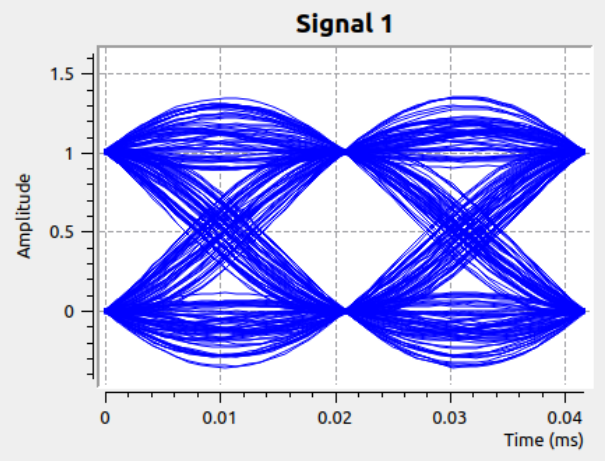
3.3 Receiver

The receiver will mostly perform the reverse operations of the transmitter.
By connecting the output of the channel to a *QT GUI Eye Sink*, you can observe the eye diagram of the received signal. The eye diagram is a useful tool to visualize the effect of the channel on the signal.

Question: Looking at an eye diagram, how would you recognize if the pulse used respect the Nyquist ISI criterion?

Answer:

The values at the time instant 0 should all be exactly the values of the symbols.



When the noise power is low, you should be able to easily identify the different symbol values (e.g. for a noise power of -20 dB). The objective of the receiver is to:

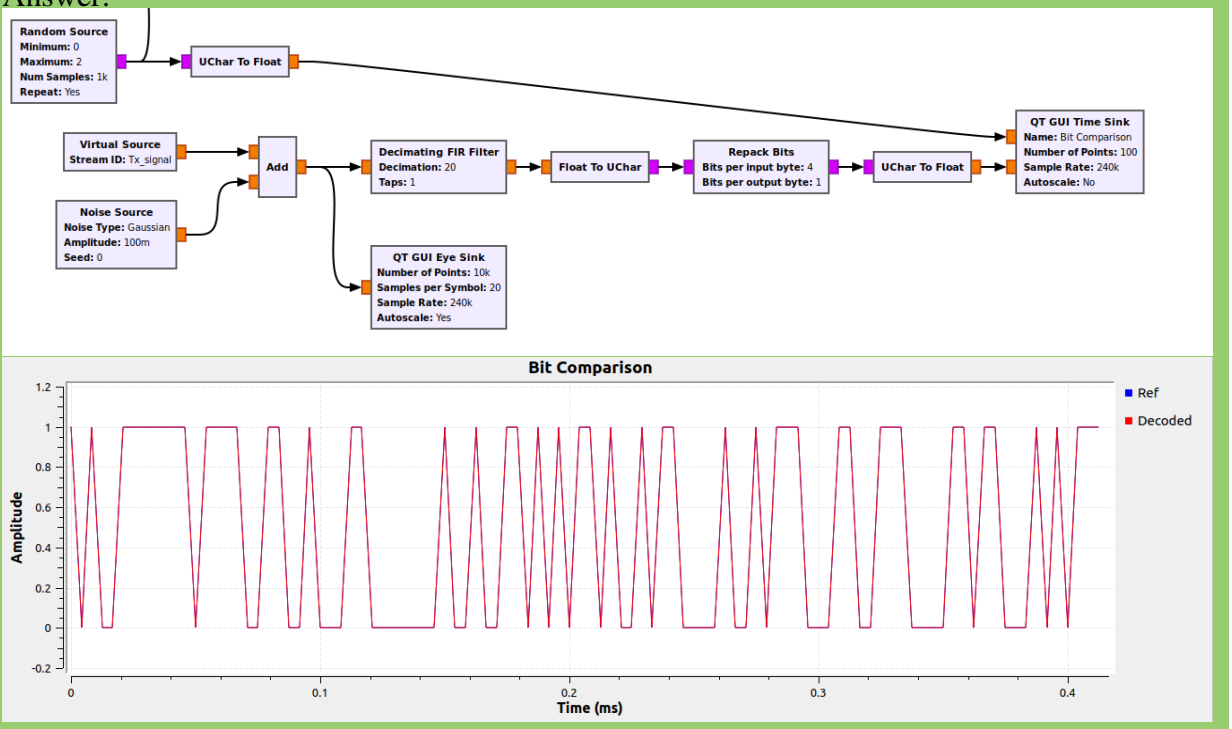
- Extract only one value per symbol by decimating the signal at the correct time.
- Evaluate which symbol was transmitted based on the extracted value.
- Repack the symbols into bits.

You can verify the well functionning of the receiver by comparing the transmitted and received bits in the *Time Sink* already in the flowgraph.

You can now see one of the main advantages of digital communication systems: the ability to transmit information over a noisy channel and still recover the original data perfectly.

Changing the number of bits per symbol while the flowgraph is running will fail as the bit unpacking may become unaligned. To change the bits per symbol (or bit per sample later) you need to change the default value in the *GUI Range* and restart the flowgraph.

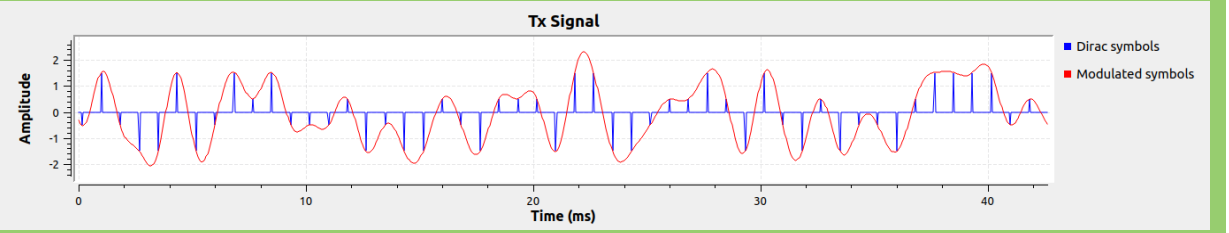
Answer:



3.4 Bi-polar PAM

As seen in the lecture, the PAM using a bi-polar scheme performs better than the uni-polar version.
Add two simple blocks to your flowgraph, change the PAM scheme from uni-polar to bi-polar.

Answer:
An *Add Const* block after the conversion from byte to float in the transmitter and before the float to byte in the receiver to shift the dirac values.
The value to subtract (and add) is $((2^{bit\_per\_symb}) - 1)/2$.



3.5 Audio Transmission

Instead of transmitting just random bits, you will now transmit an audio file and observe the effect of quantization.

3.5.1 Quantizing the Audio Signal

The audio file is read by the *Wav File Source* block. A possible way to quantize the signal is to use this method:

- Shift the signal to be included between 0 and 1. (the audio source is $[-0.5, 0.5]$).
- Multiply the signal by the number of quantization steps, based on the *bit_per_sample*.
- Round the signal to the nearest integer. (using a *Float to UChar* block)

Finally, repack the bits from the quantized value into single bits. You can replace the *Random source* by your new sequence of blocks.

3.5.2 Audio Playback

For the receiver, be careful to repack the received symbols into the correct number of bits per samples and connect the resulting float to the *to_audio virtual sink*.
You can disable the bit comparison Time sink as we should be able to hear if the audio signal makes sense.
Get a loudspeaker from the TA and you should now be able to hear the received audio signal!
Set the default bit per symbol to 1 and the bit per sample to 8.
Remember that you need to apply those changes in the default value of the *QT GUI Range* and restart the flowgraph to apply the changes without having bit misalignments.

Questions:

1. How does the digital audio quality evolves with respect to the noise power.
2. How does it differs from the analog audio quality?
3. With the current parameters, the bandwidth of the digital signal is much larger than the analog signal.
   (a) Can you reduce the quantization to use the same bandwidth as the analog signal?
   (b) Which of the analog or digital signal would you prefer to use with -20 dB of noise.
   (c) What about -7 dB?

Answer:

1. There is no degradation in the digital audio quality until the noise power is high enough to cause bit errors. Then the signal degrades quickly. This is very visible on the eye diagram plot.
2. The analog audio quality degrades continuously with the noise power.
3. When quantizing the audio signal with 1 bit, the quantization noise is very high. At some point, the noise power can become comparable to the quantization noise and it's not obvious which method is better.