LABORATORY 4
PSK DEMODULATION

OBJECTIVES:

- Understand issues of signal distortion and channel effects.
- Recognize the timing recovery stage required to recover a signal.

In this laboratory, we will be focused on simulation rather than over-the-air transmission. It will discuss the timing issues with signal transmitting and receiving and channel effects. We will go through setting up our simulation and then step-by-step how to recover the signal.

This is just one way of handling digital signal reception. There are various algorithms and methods that have been designed for these steps, and different types of digital signals will behave differently. Here, we go through a set of stages and use of algorithms readily available in GNU Radio for PSK signal reception and demodulation.

TRANSMITTING A SIGNAL

Modulating a Signal

The first stage is transmitting the QPSK signal. We generate a stream of bits and modulate it onto a complex constellation.

The constellation object allows us to determine how the symbols are coded. The modulator block can then use this modulation scheme with or without differential encoding [1]. The constellation modulator expects packed bytes, so we have a random source generator providing bytes with values 0 - 255.

When dealing with the number of samples per symbol, we want to keep this value as small as possible (minimum value of 2). Generally, we can use this value to help us match the desired bit rate with the sample rate of the hardware device we'll be using. Since we're using simulation, the samples per symbol is only important in making sure we match this rate throughout the flowgraph. We'll use 4 here, which is greater than what we need, but useful to visualize the signal in the different domains.

Run the **mpsk_stage1.grc** flowgraph and see the effects of changing the number of samples per symbol.

Effect of the Excess Bandwidth

Finally, we set the excess bandwidth value. The constellation modulator uses a root raised cosine (RRC) pulse shaping filter, which gives us a single parameter to adjust the roll-off factor of the filter.

Run the **mpsk_rrc_rolloff.grc** flowgraph to see the effect of different values of the excess bandwidth. Typical values are between 0.2 (red trace) and 0.35 (green trace).

Matched Filter

In the constellation plot, we see the effects of the up-sampling (generating 4 samples per symbol) and filtering process. In this case, the RRC filter adds intentional self-interference, known as inter-symbol interference (ISI). ISI is bad for a received signal because it blurs the symbols together. If we didn't put a shaping filter on the signal, we would be transmitting square waves that produce a lot of energy in the adjacent channels. By reducing the out-of-band emissions, our signal now stays nicely within our channel's bandwidth.

On the receiver side, we get rid of ISI by using another filter. Basically, what we've done is purposefully used a filter on the transmitter, the RRC filter, that creates the ISI. But when we convolve two RRC filters together, we get a raised cosine filter, which is a form of a Nyquist filter. So, knowing this property of the transmit RRC filter, we can use another RRC filter at the receiver. Filtering is just a convolution here, so the output of the receive-side RRC filter is a raised cosine pulse shaped signal with minimized ISI. The other benefit is that, absent effects of the channel, what we are doing is using a matched filter at the receiver.

Run the **mpsk_stage1.grc** flowgraph and add a RRC at the receiving side to produce a RC filter.

Adding Channel Impairments

Now we will look into the effects of the channel and how the signal is distorted between when it was transmitted and when we see the signal in the receiver. The first step is to add a channel model, which is done using the example **mpsk_stage2.grc** below. To start with, we'll use the most basic Channel Model block of GNU Radio.

This block allows us to simulate a few main issues that we have to deal:

1. Additive White Gaussian Noise (AWGN)
2. Frequency offset
3. Timing offset

Run the **mpsk_stage2.grc** flowgraph and see the effects of modifying each parameter.

RECOVERY TIMING

There are many algorithms we could use for recovery of each stage. Some can do joint recovery of multiple stages at the same time. We will use a polyphase clock recovery algorithm here.

Timing Recovery Problem
We're trying to find the best time to sample the incoming signals, which will maximize the Signal to Noise Ratio (SNR) of each sample as well as reduce the effects of Inter Symbol Interference (ISI).

Run the **symbol_sampling.grc** flowgraph and compare both results. As you can see, if we don't use a second RRC filter, we are not operating a matched filter at the Nyquist frequency.

Run the **symbol_sampling_diff.grc** flowgraph and see the effect of the different clocks in the transmitter and receiver. We simulate this by adding a resampler that adjusts the symbol sampling time slightly between the transmitted signal and the receiver. Notice that with the samples being collected at different points in time, the ideal sampling period is not known and any sampling done will also include ISI.

Details of the Polyphase Clock Sync Block

There are various algorithms that we can use to recover the clock at the receiver, and almost all of them involve some kind of feedback control loop. Those that don't are generally data aided using a known word like a preamble. We'll use a **polyphase filterbank clock recovery technique**. This block does three things for us:

1. Performs the clock recovery.
2. Does the receiver matched filter to remove the ISI problem.
3. Down-samples the signal and produces samples at 1 sps.

The block works by calculating the first differential of the incoming signal, which will be related to its clock offset.

Run the flowgraph **symbol_differential_filter.grc**, we can see how everything looks perfect when our rate parameter is 1 (i.e., there is no clock offset). The sample we want is obviously at 0.22 ms. The difference filter ([-1, 0, 1]) generates the differential of the symbol, and as the following figure shows, the output of this filter at the correct sampling point is 0. We can then invert that statement and instead say when the output of the differential filter is 0 we have found the optimal sampling point. What happens when we have a timing offset?

Instead of using a single filter, what we can do is build up a series of filters, each with a different phase. If we have enough filters at different phases, one of them is the correct filter phase that will give us the timing value we desire. Let's look at a simulation that builds 5 filters, which means 5 different phases. Think of each filter as segmenting the unit circle (0 to 2pi) into 5 equal slices.

Run the flowgraph **symbol_differential_filter_phases.grc**, and see how each filter with different phase produces the difference calculation. Here we are using the fractional resampler because it makes it easy to do the phase shift (between 0 and 1), but it also changes the filter delays of the signals, so we correct for that using the follow-on delay blocks. Which filter finds the correct sampling time?

We have a large bank of filters where one of them is at (or very close to) the ideal sampling phase offset. How do we automatically find that? Well, we use a 2nd order control loop, like we almost always do in these recovery situations. The error signal for the recovery is the output of the differential filter. The control loop starts at one of the filters and calculates the output as the error signal. It then moves its way up or down the bank of filters proportionally to the error signal, and so we're trying to find where that error signal is closest to 0. This is our optimal filter for the sampling point. And because we expect the transmit and receive clocks to drift relative to each other, we use a second order control loop to acquire both the correct filter phase as well as the rate difference between the two clocks.

Using the Polyphase Clock Sync Block in Our Receiver

Now we will use the Polyphase Clock Sync Block in our simulation. We use a 32 filter bank because this gives us a maximum ISI noise factor that is less than the quantization noise of a 16 bit value. If we want more than 16 bits of precision, we can use more filters. We also use a loop bandwidth of 2pi/100. The block also takes in a value for the expected samples per symbol, but this is just our guess at what we think this value should be. Internally, the block will adapt around this value based on the rates of the incoming signal.

Run the flowgraph **mpsk_stage3.grc** and observe the effects of changing each parameter.

This laboratory is based on the GNU Radio Tutorial called: Guided Tutorial PSK Demodulation [2] with some minor modifications. All credit to them.

LINKS

[1]
https://www.allaboutcircuits.com/technical-articles/differential-quadrature-phase-shift-keying-dqpsk-modulation/

[2]
https://wiki.gnuradio.org/index.php/Guided_Tutorial_PSK_Demodulation#Adding_Channel_Impairments