**Digital Communication Using GNU Radio**

**Prof Kumar Appaiah**

**Department of Electrical Engineering**

**Indian Institute of Technology Bombay**

**Week-02**

**Lecture-10**

Hello, welcome to this lecture on Digital Communication Using GNU Radio. We have been looking at aspects connected to digital modulation, in particular how discrete symbols can be mapped to waveforms. In this lecture, we will be performing a simulation of digital communication or digital messages and mapping them to waveforms using GNU radio, wherein we will generate a random set of symbols that correspond to different amplitudes, convert it to waveforms, a baseband waveform, also take it to the carrier frequency and get a passband waveform and finally we will also get the passband waveform to, we will use the passband waveform to return to the baseband and recover our symbols. In this particular lecture, we are going to begin with amplitude shift keying or pulse amplitude modulation, wherein the amplitude of the signal is used to convey information. Modulation in digital communication involves mapping messages such as bits to waveforms. The key idea is for us to select a waveform based on the input message and transmit it and the receiver will match these waveforms across its available list and figure out the most likely waveform that was sent and map it back to the transmit message.

That is what we call demodulation. Right now, our focus will be on mapping these bits or symbols to waveforms. The constraints involved in the design of these modulation waveforms are bandwidth, power, hardware and various other aspects that the design may offer. The question is how can we most efficiently design reliable high-speed modulation schemes within the constraints that are given to us by the system.

In this lecture, we will be looking at some of the most common modulation schemes and discuss how we can put them together by looking at them in GNU radio as well. So some of the most common modulation waveforms are ASK which is amplitude shift keying, PSK which is phase shift keying and FSK which is frequency shift keying. In each of these, some aspects of the waveform is modified in order to convey some message. At the simplest level, if your aim is to transmit bits which are zeros or ones, one way to achieve this would be to send a signal which is zero for the duration of the symbol to

send the bit zero and raise it to a 1 volt level to send the one message. So in this case, if we assume that the duration of a bit is let us say one second, then this is held at zero for one second, one at one second, one at one second, again zero at one second and so on in order to send an amplitude shift keyed waveform.

The next waveform is a frequency shift keyed waveform wherein to send zero for the duration of the zero symbol, a partial sinusoid is sent. Subsequently to send one, a sinusoid of a different frequency is sent. Again the sinusoid of a different frequency is sent to indicate one and it falls back to the slower frequency sinusoid to send zero. In this way, by modifying the frequency of the sinusoid within the duration of a symbol, we are able to send bits by modifying the frequency. Finally, this is a phase shift keying waveform.

In this case, you can see that there is a sinusoid which has two cycles which is used to send zero. There is another sinusoid which has two cycles to send the one except that it starts with a $\pi$ phase shift. In other words, the first waveform can be considered as a sine, the second waveform looks like a minus sine. The minus sign continues to give you another one and then you again have to send a zero, so you send the positive sinusoid. In this way, by appropriately adjusting the phase of the sinusoid during the duration of a symbol, you can use phase shift keying to convey information.

One simplifying assumption that we will make is that we will look at linear modulation. If we consider messages as

$b[k]$

and our baseband pulse as

$g_{TX}(t)$

and the symbol rate as one symbol for every

$T$

seconds, we have the transmit waveform

$$s(t)=\sum_{k=-\infty}^{\infty} b[k]g_{TX}(t-kT) \quad .$$

Some things to note are that waveforms generally are translated to passband before transmission. This is something that we have seen in the context of the complex baseband equivalent. The challenge for us is to design

$g_{TX}(t)$

in order to honor the bandwidth and power constraint.

An important note is that in this simplifying assumption, we are fixing

$g_{TX}(t)$ ,

but a more general modulation scheme may involve

$g_{TX}(t)$

also depending on k. We should now keep in mind what are constellations and their connection to waveforms. It is convenient to map messages. It could be bits like 0 and 1 or it could be a set of 10 messages, but these have to be mapped to something more convenient wherein we choose complex symbols also known as constellation due to their appearance on a 2 dimensional space. These constellations consist of complex symbols that are sent and have to be detected at the receiver.

$b[k]$

is a value that is chosen appropriately from this set in order to obtain or design a transmit waveform. We will see some specific modulation formats and their waveform versions to get an idea of how these messages are effectively translated into waveforms. For all our simulations on GNU radio, we will be using 1 kilo symbol per second which means that we will be sending a 1000 symbols of the form

$b[k]$

per second and for the pass band we will be taking 8 kHz as our carrier frequency. The first modulation scheme that we will be looking at is amplitude shift keying. Here

$b[k]$

is a number that is among

$\{0,1,2,...M-1\}$

or it can be an offset variant like let us say it can be this can be -3, this can be -1, this can be 1, this can be 3.

If you want a bit picture, this can correspond to 00, this can correspond to 01, this can correspond to 11, this can correspond to 10 meaning that you are able to send 2 bits per symbol. In this case, your

$s_k(t)$

corresponding to a single symbol is just

$b[k]\psi_1(t)$

which corresponds to one dimensional signaling. Let us now look at how this appears on GNU radio. We will begin by first introducing a couple of new blocks that will come in handy for signal generation, especially random signal generation. The first block that we will consider is the random source.

We will use our control f or command f and type random to obtain a random source and put it on our flow graph. Since we want to generate an amplitude shift keyed signal consisting of 4 possible values, we will modify the random source by double clicking it and choosing the maximum to be 4. And let us also make it 1024 symbols and set repeat to yes. Next, we will introduce the next block called chunks to symbols. So, control f or command f chunks to symbols.

Chunks to symbols can be thought of like a lookup table if this random source generates messages, then this chunks to symbols generates symbols which also correspond to constellation points. Let us double click the chunks to symbols, set the dimension to 1 and define our 4 symbols. In this case, I am going to use 0 1 2 3 which I have written as a python list by putting a square bracket and putting the number 0 to 3 separated by commas. The next thing I am going to do is to introduce a throttle control f or command f throttle get our throttle here and finally, I am going to introduce a time sink. The time sink will allow us to visualize what this random source gives us.

Control f or command f press time QT-GUI time sink. Let us now connect our blocks and execute this flow graph. Upon executing this flow graph, you will see that you get these numbers or values. Let us middle click and choose the control panel and say auto scale. Let us also middle click and choose stem plot and then zoom into a particular region.

You can see that the values are either 0 1 2 or 3. We get two signals. The reason for this is our choice of complex symbols. Let us address that first. We will set the chunks to symbols to have type float, our throttle to have type float, our time sink to have type float.

Now we middle click control panel auto scale or we can even set the auto scale here. If we say stem plot and zoom into a particular region, you will see that the values are 0 1 2 or 3 and they appear in somewhat of a random fashion. This is expected because we are

connecting the output of a random source to the time sink. The next task for us is to ensure that we generate 1 kilo symbol per second or in other words the symbols must be separated by a 1000 they must be separated by 1 millisecond each which is how you can get a 1000 symbols per second. To do that we will use the simplest pulse which is the rectangular pulse.

To approximate a rectangular pulse, we will just repeat each of these symbols for 1 millisecond duration time. Let us first increase the sampling rate for convenience. We are going to make the sampling rate 64000. So, I double click the sampling rate change it to 64000. Then before I go to the time sink, I will click this arrow, hit the delete button to remove this connection.

I am now going to introduce an interpolating filter. So, control f for command f we will type interpolate and we will get the interpolating FIR filter which I drag and place here. We double click this filter. We want float to float real taps. Now we want this to be interpolated so that we get exactly a 1000 symbols per second.

So, before we fill this in let us do some computations. When we have 64 k as the sampling rate to have this symbol repeated for 1 millisecond implies that we need this 64 k to be divided by multiplied by 1 millisecond rather that corresponds to exactly 64 samples. In other words, let us create a variable. So, control f for command f and type variable, drag this variable. We will double click this variable and we will call it interpolation factor as the id and we will set this to sample rate upon 1000.

Note that there are 2 slashes. This will ensure that we have our symbols last for exactly 1 millisecond. Why? Because if 64000 samples correspond to 1 second, 64000 divided by 1000 will correspond to 1 millisecond. By doing it this way, our flow chart will still continue working correctly even if we change the sample rate later. Therefore, it is always a good idea to keep variables wherever needed.

We will say ok. We will double click this and say interpolation factor as our interpolation and taps. Currently, we need to provide a list of filter coefficients that are used to fill in the gap that is introduced when you interpolate. I am going to simply say square bracket 1 indicating that I want only a 1 and then nothing else during the interpolation. We will say ok.

Connect these inputs and run. Now, you can see that you have a picture like this. Let us first stop and let us make it stem plot. As you can see, you get a few symbols. This value is 2, this value is 3, this value is 1. Let us see by how much they are separated.

This is at around 2 milliseconds, this is at around 3 milliseconds, this is at around 4

milliseconds. In other words, because the interpolating filter essentially says give me a 1 and then nothing else, you have all these symbols separated exactly by 1 millisecond but after a single impulse, there is no other signal present. To address this, we will ask our interpolating filter to fill in a rectangle for the full duration of 1 millisecond. To do this easily, we can either sit and write that many 1s or use some python. In this case, I am going to use a numpy function to do this.

We need to first import numpy. So, I am going to hit ctrl f or command f and type import, and then grab the import module, double click it and type numpy, rather import numpy. Now over here, we are going to just use the function numpy.ones interpolation factor. This will essentially populate 64 1s that will essentially fill the duration for 1 millisecond. Let us now execute this flow graph. As you can see, we now have these rectangles.

Let me first enable the auto scale by double clicking this and enable the grid, enable auto scale. Now by executing this flow graph, you are able to see the values being 0, 1, 2 or 3. Let us stop this and let us look at the duration. It is very evident that this starts at 6 milliseconds, this starts at 7 milliseconds, this starts at 8 milliseconds and thus what we have now is our baseband waveform for amplitude shift keying with 4 messages 0, 1, 2 and 3 corresponding to those symbols and this is the baseband waveform that we wanted. As an exercise, you can try to change the interpolating filter taps and see what happens.

For example, rather than use numpy.ones, you can use a different set of taps to see that those taps are reproduced in this particular time sink. Our next task is to see how this waveform would appear once we go to passband. But before we do that, we will now introduce another pair of blocks which make it convenient to visualize our flow graphs when we have multiple elements. When we have to take multiple signals across the flow chart, sometimes it becomes messy if you start connecting arrows across the flow chart. Therefore, we will introduce a couple of convenient blocks which are called virtual sources and virtual sinks.

Let me just press ctrl f or command f and type virtual and we will first choose a virtual sink. A virtual sink's input appears white. It has no output. This is because the virtual sink takes in a particular signal and consumes it and it can take in a signal of any type which is what the white color indicates. Let us double click the virtual sink and give it a stream name.

I am going to call it passband1 and I am going to connect the output of our interpolating filter to this virtual sink. Now, if I want to get this signal somewhere else in the flow chart, I need a virtual source. So, let us grab our virtual source, put it over here as you can see the virtual source is the counterpart to the sink. We just need to double click it

and give it the same name which is passband1 and we now have the float detected because it is the same name as the virtual sink. By connecting this further, we have essentially avoided having to pull this arrow across the flow chart and thus making the layout of our flow chart much more convenient.

Our next task will be to modulate this baseband signal to carrier frequency. Before that, let us first introduce a variable called fc. So, control f or command f, variable, drag the variable block, double click it, we will call it fc and we will choose it as 8000 hertz. Again, the advantage of having these is because if you want to change one of these parameters across the flow graph, it is much easier for you to change it in one place rather than having to change every block which is very cumbersome and error prone. Now, if you recall from our complex baseband to passband conversion, we have to multiply this signal by a cos and the other baseband signal by a sine.

But in this case, we have just a single signal only the real or the I part and therefore, we are just going to multiply this with a cos and that will give us our passband signal. So, I am going to press control f or command f, type signal, grab a signal source, I am going to double click the signal source, make it float, the frequency will be fc and now we are just going to multiply these. So, control f or command f, multiply, grab the multiply block, we double click this multiply, change it to float and we connect these signals and thus this will be our passband signal and let us visualize how the passband signal looks like. Let us actually create a separate time sink just to visualize this. So, control f or command f, we can grab a time sink or let us just double click this time sink, hit copy control c or command c and control v or command v and connect this time sink.

If you now execute the flow graph, you now see that the signal above looks very much like the signal below except that it has been modulated. In fact, let us actually look at these in the same time sink as well. So, if you double click this time sink and set the number of inputs to 2 and drag it across. If you now see, the modulation essentially just multiplies the signal by a cos, the peak value of the cosine has the same value as the rectangular portions of your amplitude shift keyed waveform. Therefore, it is very easy to understand how the passband waveform looks like for amplitude shift keying.

It is just a cosine or sine which has peak amplitudes resembling that of the baseband waveform. Our next challenge will be to recover the baseband waveform from this passband signal. Temporarily, let me remove this and let me now work on how we can get back the baseband signal. So, if you remember the way we got back our baseband signal from the passband signal, we essentially have to multiply it by a cos and a sine and get it back.

So, let us actually make the amplitudes correct first. So, remember that if you now want

to have the correct values, you must choose $\sqrt{2}$ over here. So, we will make this 1.414 and we will copy this cosine source, paste it. We will copy this cosine source, paste it again.

Now, the second one we will make a sine source. Now, if you remember how we can recover our baseband signal, we will multiply the modulated waveform with each of these signal sources and then place a low pass filter that cuts off frequencies around 2fc. So, let us first get multipliers. I will just click this multiplier and hit control c or command c and command v, control v command v and again repeat command v or control v. Take this signal, connect it over here, take this source, take the signal, connect it over here, take the sine source.

I now need a pair of low pass filters. So, I am going to hit control f for command f and say low pass filter, drag drop the low pass filter, double click it, change it to float to float decimating and we will set the cutoff frequency to be it should cut off frequencies above to or near to fc, but we will just make the cutoff frequency fc and we will make the transition with about a 1000 hertz. We will copy this filter, control c, command c, control v or command v, paste it and send these signals through the respective filters and this will be our baseband signal if you now convert this to a complex waveform. What we will do is, we will actually directly just view this on our original time sink. Let me double click this time sink, make it have three inputs, connect this over here, connect this over here and visualize. Now, you can see that there is some resemblance over here to the original signal.

Let me first stop this. You can see that there is a blue, oops, let me first enable this auto scale again. You can see that there is a blue followed by a red, a blue followed by a red and this red looks like the blue, but there are a couple of things that you must observe. The first is that the red has this ringing. The reason for this is that we chose a rectangular waveform as our gtx and as you know rectangular waveform is not band limited and therefore by passing it through the low pass filter you end up getting these ringing effects. This will not happen if you choose a more band limited pulse such as the raised cosine or sink or any other waveform that we will see in subsequent lectures.

Next, there is a green signal. That green signal corresponds to the Q part and as you recall we were modulating only a single signal in the baseband as opposed to a single, rather a single real signal in the baseband as opposed to two real signals or a complex signal. Therefore, we can safely ignore the other signal. Now, to verify whether these two are indeed the same we will now try to match these. The cause of this delay is because we are using the filters over here. These filters because they are practical filters will inevitably possess some delay.

Let us try to match the waveforms by adding a delay block to the original signal and then seeing how we can match the delay to see whether the signals are indeed the same. So, I am going to do do ctrl f or command f and get a range. We will get a QT-GUI range widget.

We will double click it. Set the type to integer. Set the id to delay. Set the default value to 0. We will start at 0 stop at let's say 1000.

We will next grab a delay block. So, ctrl f or command f. The delay block. We will double click this delay block. Make it float because we are going to deal with float signals and the delay will be the variable delay. It is from the QT-GUI range. Now, rather than connect this output directly, we will connect it through this delay.

And now let us run our example. As you can see we have a range now. Let's actually set it to something over here. As we move it, you can see that the blue one gets closer. So, if we move it over here roughly at around 75 samples, we see that we are almost matching the output after converting the passband signal down. And it is a reasonable match and obviously this ringing effect is the Gibbs phenomenon because we have essentially filtered what is fundamentally a band unlimited signal in the form of the rect. Therefore, as you can see, we have converted an amplitude shift keyed waveform to passband and recovered it at the baseband as well.

As a final step, in order to view more of the waveform, let us just adjust our GUI time sinks to show more points. Let us say 16384.

Double click this one over here. Say 16384. And now let us visualize. And let us again set this delay to 75. And you can clearly see a very nice match. And over here, you can clearly see that the waveforms look in this way. If you really want, you can also overlay this waveform in the same.

You will actually get the modulated signal as well. I leave that as an exercise. But with this, we have essentially built a simple approach to making a flow graph wherein an amplitude shift keyed waveform can be obtained from random source, converted to passband, then obtained back in the baseband. In this particular case, remember that we did not have an imaginary signal in the passband.

Therefore, this will yield zero. The other part yields the original signal. And because you chose

$g_{TX}(t)$

as a rectangular signal, you got these rigging effects. In this lecture, we have carefully designed a digital communication system wherein discrete set of values are mapped to waveforms. We have seen how the waveforms look in the baseband when using a rectangular pulse and how they go to passband as well. And from the passband when we recover, we see that using the rectangular waveform allows us to recover the data. But there are some minor issues like those rigging effects thanks to the band unlimited nature of the rectangular waveform.

In the next few lectures, we will be continuing this exploration. And we will be seeing how not only amplitude but the phase variations in the baseband signal can be used to convey information. Thank you.