Digital Communication Using GNU Radio

Prof Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

Week-03

Lecture-14

In the previous lecture, we have gone through the raised cosine waveform and how using sinc and raised cosine waveform of course, can solve the problem of the rectangular pulse being band unlimited, but we also saw that using the sinc has disadvantages especially in terms of the rate of decay and it is not being robust in the case there is jitter. In this lecture, we are going to put together some simple simulations on GNU radio wherein we will explore the characteristics of the raised cosine and in particular using the root raised cosine waveform. We will then repeat our amplitude shift keying simulation that we did some time ago using root raised cosine. We will use both the root raised cosine at the transmitter as well as the receiver and show that this particular approach yields a way wherein you can recover the symbols that are sent at the transmitter at the receiver without any issues while generating a signal that is band limited. So that is the purpose of this particular lecture. I urge you to try it out on your own as well.

Before we begin our pursuit of pulse shaping for modulation, we will first inspect a couple of useful blocks that allow us to do interpolation and decimation easily. First let us introduce the interpolating FIR filter block that you have seen already. So, Ctrl F or Cmd F interpolating FIR filter and we will place it over here. We will begin with float to float real taps.

We will then also use a vector source that will provide initial data for this filter. So, Ctrl F, Cmd F type vector grab this vector source, double click the vector source. To keep it simple, we will set the vector source to float and we will give the inputs as some set of numbers let us say 1, -1, 3, 2. Let us say these are the sequence of 4 numbers 1, -1, 3, and 2. Now these numbers are going to be generated at around the sample rate and fed to the interpolating FIR filter.

Now we will also add a throttle Ctrl F or Cmd F type throttle, double click the throttle, we will set the throttle to float, connect the interpolating FIR filter to the throttle and we will then get a time sink. Ctrl F or Cmd F type time get this time sink, set this time sink

to float, we will add a grid and auto scale say ok and then we will connect this over here. There is only one thing missing which is setting the interpolation and the taps. Let us do this right now. As an example, let us set the interpolation to 2 which basically means that for every input sample, this filter is going to output 2 samples.

Then what are the taps? This is the filter that is going to be applied on the input samples before outputting them from the output of this interpolating FIR filter. As a simple example, let us set this to be [1,1]. What this will do is that it will repeat every sample twice. Let us actually have a look at whether that is the case. If we execute this flow graph, let us actually middle click and choose stem plot and let us zoom in to a particular region.

What you can see is that you have this 1 1, you have this -1 -1 and then you have this 3 3 and then this 2 2 which is exactly the same samples as the original vector source except that they are repeated 2 times to the same weightage. Let us actually change this and let us set this to let us say 0.5. So the second tap of this filter is 0.5. Now if I execute this flow graph, again middle click, set it to stem plot, zoom in. You can see that we start with, let us say we go to the absolute beginning so that we have no ambiguities. It is 1/2 because it is one followed by 1/2. So the impulse response of your filter which is 1, 1/2 is imposed here, -1 -1/2. Here you have 3 and 1 and a 1/2 followed by 2 and 1.

An important point is that this filter did not have the same length as the interpolation. As an example, let us actually set this to something like 1 1 1 1. In which case, you can imagine that this 1 is essentially replaced with 1 1 1 1 and then this -1 is going to become -1 -1 -1 shifted by 2 samples and added. Let us visualize this one. We will go to the stem mode.

We will go to the absolute beginning. So the original 1 became 1 1 1 1. Subsequently, the -1 became -1 -1 -1 -1 and is added after a shift of 2 because our interpolation is 2 and those 1s and -1s got cancelled. These subsequently, the -1 -1 -1 -1 got added with these 3s. So 3 3 3 3 on the second and third.

So you have a 2 2 over here and then the effect of this 3 and 2 come together and become 5. So an important point to remember is that the taps of the filter need not be the same length as the interpolation. In fact, if it is less, for example, let us say it is square bracket 1, this is going to do the traditional upsampling by insertion of zeros. In other words, we are going to get 1 0 -1 0 3 0 2 0. If you look over here, make it stem 1 0 -1 0 3 0 2 0 which is exactly what happens over here.

The corresponding undo operation is called the decimation. So in this particular case, let us actually say go back to our 1, 1 which is basically upsampling by repeating the

sample twice. So now your 1 -1 3 2 is going to become 1 1 -1 -1 3 3 2 2. Let us make this stem. As you can see, it is 1 1 -1 -1 3 3 2 2.

Now let us get back these original samples by decimating this interpolated value. To decimate this interpolated value, we are going to fetch our decimation filter. So control F for command F, type decimate and we will get the decimating FIR filter. We will double click it, set it to float to float real taps, set the decimation to 2 and now the question is what should the taps be. Let us start with the most basic one which is square bracket 1 which is the same as square bracket 1, 0.

And let us repeat a QT-GUI time sink. I will do control C, control V or command C command V. Connect this and connect the interpolating FIR filter to the decimating filter and see what we get. Let us label our sinks so that we do not have any confusion. We will double click this and call it interpolated.

We will double click this and call it decimated. Let us now execute this. So this is the decimated and I am just going to middle click and say stem plot. This is the interpolated. So now let us zoom into the beginning of both of these.

So here we have 1 1, -1 -1, 3 3, 2 2. Over here we have 1 -1, 3 2, 1 -1, 3 2 which is the exact same input sequence at the exact same rate which makes complete sense. Now let us change this decimating filter to make it 1 1 and see what we get. If you make it 1 1 and let us look at the stem plot, we end up getting something like 3 0, 2 5, 3 0, 2 5. Why do we get this? This is because this filter is applied on the interpolated output and the result is essentially presented after decimation.

So in this particular case, let us make both stems. Over here what we have is, let me go to the beginning of this as well. We have a 3 here. The reason is because if you essentially take this 1 1 filter and flip it, what is to the left of -1? To the left of -1 over here there is nothing but if you go to the right, let us look at this particular 1. To the left of -1 is 3.

Over here you can see that 2 rather. The left of -1 is 2. So the 1 multiplies this 2 and the 1 multiplies this 3 and you end up getting 3, then the one multiplies this one and the one multiplies this -1 and you end up getting 0, then the one multiplies this particular 3 and this particular -1 and you end up getting 2. Finally, the 1 ends up multiplying this particular 2 and 3 and you end up getting 5. Therefore, in this case, the filter essentially applies on the interpolated input and the result is presented to you and because of downsampling you have to go one off.

So, every other output is what you get. In other words, you apply the filter 1 1 onto this

input and take every other sample to get this decimated output. You can see that this particular pattern repeats and you end up getting what you expect if the input is 1 1 -1 -1 3 3 2 2, you filter take every other sample. Pulsing and decimating are very common operations and this will come in handy when we do our pulse shaping. Now, to get an idea of how we can do the pulse shaping, let us actually inspect how our root raised cosine pulse looks like.

Let us first see how we can create the RRC filter using GNU radio. Now, to this end we are going to use an inbuilt GNU radio functionality which is in python to create the filter taps. So, let us do that by just putting it in a variable. So, I am going to say control f for command f and variable and grab the variable over here, double click it and call it rrc_taps. Now, this rrc_taps is going to contain our RRC filter samples.

To design this RRC filter, we are going to use the filter design tools inbuilt in GNU radio, but write the code for it or rather the one line code for it. You double click this and you say FIRDES short for FIR filter design dot root_raised_cosine. We will first need to specify the gain which is 1 followed by the symbol rate which in our case is a 1000 symbols per second. Then we need to give the excess bandwidth which we can choose as let us say 0.35 to begin with and finally, the number of filter samples.

Let us say 8192 seems to be a good number although this is only an indicative number your actual filter may be a slightly different length. Let us say ok. Now, if you put your mouse over this, you will get an 8193 length filter which looks like some you know it looks like there are 0 values and it looks to be symmetric also because it starts and ends with similar values fine. Let us now inspect how this filter looks like. We will get a vector source control f command f vector source.

Double click the vector source. The output type we will set to float and we will just make the vector rrc_taps. Next we will add a throttle. The throttle we will double click make it to float. I notice that our sampling rate should be 64 k. So, let us change by double clicking it to 64000.

We will then connect our vector source to the throttle and just add a time sink. So, control f or command f and get a time sink. Double click the time sink and make it float. Add a grid, add auto scale and for viewing it conveniently let us just set the number of points to the length of the rrc_taps. We can do that by just writing a python expression len and in brackets rrc_taps.

This will make it convenient to visualize. We will say ok, connect this, run the flow graph and you will essentially see what looks like a sink or what looks like a raised cosine, but it is not really a sink it is a root raised cosine because if you inspect it very

carefully we want the zero crossings to be perfect. So, let us say that if you look over here this is going to be at 64 milliseconds there is a peak and at 65 milliseconds you expect a zero crossing which is not there. This is not a cause for concern because the raised cosine pulse is what is guaranteed to have the appropriate zero crossing not the root raised cosine. Let us see what happens if we get a raised cosine.

To get a raised cosine we just need to convolve the root raised cosine with itself. Alternately in the Fourier domain you have to multiply the Fourier transform of the root raised cosine filter with itself. So, let us do the convolution we will just say control f for command f and get an interpolating FIR filter take that and place it here we will double click it interpolation we will keep as 1 we will float to float real taps we will set the taps to rrc_taps and say ok and we will view it on the same time sink. So, we will double click it change the number of inputs to 2 then connect the rrc_taps over here connect this over here. Now if you visualize you will see that this pulse is essentially shifted out the reason is because there is a delay which this interpolating FIR filter encounters.

So, add another delay element. So, we will say control f for command f we will grab the delay block place it over here we will double click this delay block set the delay to int sorry float and the delay to len(rrc_taps) double divided by 2 and connect the original root raised cosine pulse through this delay and now we can visualize you can see that the both the peaks are close, but it is still very inconvenient. So, let us make let us remove this delay from here and let us instead connect it to the second one. So, connect the output directly connect this delay over here and let us execute this and now we can see that the red one is what we should expect should have the zero crossings. So, let us zoom in a little the red one seems to have a peak over here of 0.0157 why that is we will see momentarily it has a peak at 64 milliseconds and it goes to 0 at 65 milliseconds which satisfies our Nyquist ISI free criterion much like the sinc.

Another thing is that you can see these little oscillations. Let us actually handle those things one by one. The first thing is that we can change the gain over here for example, we can set the gain to be let us say 64. We set the gain to be 64, if you then run you can see that there is a high peak over here of 64. The reason this happens is because if you convolve this filter with itself there is a significant amplification.

So, what we can do is we can just take away that gain over here by dividing all the values by 64, but to do that we must first convert this to a NumPy array. So, let us do control f or command f and say import we will double click this and say import NumPy and then we will change this fir variable fir dash variable to NumPy dot array parenthesis and close the parenthesis over here and now we can just divide this by samp_rate upon 1000 which is 64 and let us inspect it over here. Now you can see the amplitudes match. This is an issue with the scaling which we have addressed by just adding a gain of 64

and taking it away when we do the raised cosine. So, now when you have the raised cosine at 64 milliseconds it is a peak and at 65 milliseconds there is a 0, at 66 milliseconds there is again a 0, 67 milliseconds again there is a 0 which is exactly what we need.

Next let us have a look at these oscillations. Let us say around 66, 67, around 68 milliseconds this goes away. Let us now change the rrc_taps to have this 0.

35 become something very small let us say 0.0001. 0.0001 is very close to a sink. So, let us execute this. You can clearly see that this is a very sink like behavior and even around 81, 83, 90, 100 milliseconds it does not really die down because this is very close to a sink. There is almost no excess bandwidth. Let us now double click our variable and let us actually make this we check 0.

35 let us make it 0.9 which is 90 percent excess bandwidth. Remember that at 0.35 around 69 milliseconds we had a very small amount of amplitude. Now if you zoom in you can see that even around 66 milliseconds the amplitude essentially dies down. So, here you can clearly see that the excess bandwidth which is being used makes the pulse thinner in the time domain at the cost of making it fatter or increasing the bandwidth usage in the frequency domain.

So, now this particular picture allows us to understand how the root raised cosine and raised cosine work. So, if you inspect this particular command that we have used to generate the root raised cosine this will come in handy when we generate root raised cosines for up sampling and down sampling with our interpolating and decimating FIR filters and that is something we will see subsequently. We are now going to repeat our or PAM-4 experiment, but this time we will use the root raised cosine pulse and use the root raised cosine to recover our symbols as well. Since we are going to use our same modus operandi let us double click this sample rate set it to 64000. First control F for command F we will grab a random source, the random source we will place here and let us say it goes from 0 to 4 which will give us numbers 0, 1, 2, 3 and we will make it 1024 samples which keep repeating.

We will then grab the chunks to symbols and the chunks to symbols will connect the random source symbol table let us make it 0, 1, 2, 3 output type let us keep it as float and one dimension. Next we are going to up sample this and using an interpolating FIR filter, but before that let us design our root raised cosine filter. So, I am just going to say control F or command F and say variable and grab the variable block place it here change the id to rrc_taps and the value will be FIRDIS dot root_raised_cosine and let us we will set the gain to exactly the amount of up sampling that we are going to do. So, sample rate upon 1000 then we are going to specify the sampling rate then we will

specify the symbol rate which is 1000 then we will set an excess bandwidth of about 0.4 and finally, we will set the length to be 8192 any reasonable length which is enough 8192 is good for us.

Next we will get the interpolating FIR filter. So, control F or command F and type enter and get the interpolating FIR filter double click it set it to float to float the interpolation will be 64 or we will write samp_rate upon 1000 which is basically we want 1000 you know want 1 symbol every millisecond so we set it to samp_rate upon 1000 taps rrc_taps. What this will do is that it will essentially place 64 zeros in between each of these samp symbols that are being generated and then filter them with this rrc_taps so that you get a nice root raised cosine response. Connect this over here we will set the throttle so control F or command F we will grab a throttle we will double click the throttle and set it to a float and finally, control F or command F and type time and get a time sink we will double click the time sink set it to a float we will say rrc and number of points let us make it 8192 and let us make it yes and yes for grid and auto scale and let us connect these and inspect what we get. So, we end up getting some waveform which seems to indicate that there is some data that is going on we cannot really make out the values well the values seem to be between 0 and near 3 which makes sense but exactly figuring out the 0 crossings etc. is not really possible because that makes sense only when you do a raised cosine. As a quick example let us just double click this and then make it 2 inputs and pass the same sequence to the same filter and inspect what happens with a raised cosine so you can click here hit control C or command C control V or command V to paste double click it and set the interpolation to 1 and connect this FIR filter to this FIR filter and over here we can visualize and we will be able to see our actual data in this particular signal how if you run this of course, there is a gain issue to address the gain issue we will again close this say control F or command F say import we will import numpy type import numpy change the rrc_taps to be a numpy dot array which is convenient for doing arithmetic.

So, we are set just enclosing this within a numpy dot array now these rrc_taps we are going to divide by samp weight upon 1000 notice the double divide for integer division. Now let us stop and let us see if we can make something out over here. So, let us see it appears that over here there is 0 around the 40th millisecond. So, now let us go to the 41st millisecond at the 41st millisecond the value seems to be around 3 at the 42nd millisecond the value seems to be 3 again at the 43rd millisecond the value seems to fall down to 1 and at the 44th millisecond it seems to be 3. So, it does appear that when we have an actual raised cosine we are able to see the actual data with that is embedded within this root raised cosine signal.

Our next task is to decimate after putting the raised root raised cosine and recover the data for this we are going to get a decimating FIR filter. So, control F for command F

decimating FIR filter will double click this decimating IR filter set the decimation to 64 or rather samp_rate upon 1000 taps are rrc_taps the same ones we are choosing they are symmetric we do not really mind say ok we will set it to float real taps we will get a separate time sink. So, we will say control F for command F and QT-GUI time sink we will double click it we will set it to recovered data we will require a grid we will use auto scale let us also set it to a stem plot for convenience and we will also make it float and connect this output over here and it is interpolating filter output to this and let us try executing this. Now, let us look at what this recovered data is of course, there is a scaling what does that scaling in the decimating filter will divide by 64 or samp_rate upon 1000.

Now, let us actually inspect this. So, as we can see the recovered data seems to make sense because it takes values 0 1 2 and 3 in a random sequence indicating that this is likely related to the input random source. To check we can just get the random source also over here we double click this GUI time sink set the number of inputs to 2 we then get an int to float. So, control F for command F say int to float we connect this random source over here and before we connect this particular int to float directly there is a delay because this interpolating FIR filter is going to cause some amount of delay this decimating FIR filter is going to cause some amount of delay. So, we are going to add control F for command F a delay block and we will double click this delay block set it to float and we will set the delay to 128. The reason we choose the delay as a 128 is because this is the effective delay between the received and the transmitted signals due to the causal nature of this particular RRC filter.

So, now, we connect this over here and connect this over here we can hit play and we see that the input and output match exactly. The final step for us is to ensure that the data rather the spectrum usage is as promised for example, in this case we have said that we are using about 40 percent of excess bandwidth. Let us do control F for command F and grab an FREQ QT-GUI frequency sink we will double click it we will set the type to float we will set the window type to rectangular we will also set a grid and auto scale and let us connect the output of the interpolating FIR filter over here and see what the actual excess bandwidth usage is. So, there is an extra peak that can be attributed to the fact that our symbols are between 0 and 3 as opposed to between you know something which has a middle midpoint around 0. If you really wish to address that you can choose a different set of symbols like -3 -1 1 3 these do not suffer from that problem because this will give you 0 average DC.

So, in this case you will now see that the spectrum usage is around you know -0.6 to around 0.6 which indicates like when you compare 0.5 to 0.6 that is about a 20 percent 20 percent over here corresponding to 40 percent.

Let us go to the one extreme let us set this to not 0.4 but 0.0001 which is very much like

a sinc in which case you will see a very flat spectrum ranging between -1/2 and 1/2 kilohertz which is exactly the amount you need in order for you to get the in order for you to get the you know 1 kilo sample per second symbol. Let us go back to our original 0 1 2 3 and let us go to the other extreme where we set the excess bandwidth to 0.9 in which case you will see that you end up using almost the whole bandwidth from about -1 kilohertz to about 1 kilohertz. Let us actually start you know let us say max hold or let us do an average. We will start averaging it so you can see that the pulse is being used from about -1 to 1 kilohertz -0.

9 to 0.9 kilohertz which is strongly indicative of the fact that we are using a lot of excess bandwidth. The waveform also looks very much like a sink related one which sorry the you know root raised cosine 1 where you can almost directly make out the original sequence and at around the midpoint let us say around 0.5 you will get about 50 percent bandwidth usage so it will go till around 0.7 or 0.8 kilohertz so it is about -0.7 to 0.7 kilohertz. So, in this manner we were able to put together a baseband waveform that utilizes the root raised cosine and this enables us to design pulse shaping that honors the bandwidth constraints much much better than using a rectangular pulse. In this lecture we have put together a simulation of the root raised cosine. We have seen how using the root raised cosine both at the transmitter and the receiver allows you to have an ISI-free communication that is you get the raised cosine that satisfies the Nyquist ISI-free signaling criterion. This of course is satisfied by the sinc as well but the excess bandwidth that the raised cosine allows you to trade off some properties particularly the

$1/t^3$

decay that makes the raised cosine much more robust to jitter. The root raised cosine at both the transmitter and receiver is a very practical approach that is often employed in many practical communication systems.

Of course in this simulation we have ignored the presence of the channel

$g_c(t)$.

In future lectures after we cover demodulation we will carefully look at a way by which

$g_c(t)$

that is the channel is also present and can be compensated for by using equalization. Thank you.