Digital Communication Using GNU Radio

Prof Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

Week-02

Lecture-07

The next task is for us to come back from the pass band to the base band. So, we have to do the same thing for the base band. So, we have to do the same thing for the base band. So, we have to do the same thing for the base band. So, we have to do the same thing for the base band. So, we have to do the same thing for the base band.

So, we have to do the same thing for the base band. So, we have to do the same thing for the base band. So, we have to do the same thing for the base band. So, we have to do the same thing for the base band.

So, we have to do the same thing for the base band. So, clearly from this particular result, if we somehow eliminate the frequency components near

$2f_c$,

we will be able to recover the original

$s_c(t)$.

This is possible because in general

$2f_c$

is a far away frequency and using an appropriately designed low pass filter can be used to cut these signals off and get only the component of signals that are close to DC which is what you want, you want

$s_c(t)$.

In a very similar fashion, multiplying

$s_p(t)$

by

$\sin(2\pi f_c t)$

instead of the cos will give you a similar expression except that

$s_s(t)$

will now come out and the other components related to

$s_c(t)$and $s_s(t)$

will once again be close to

$2f_c$.

Therefore, this technique allows you to recover

$s_c(t)$and $s_s(t)$

from the pass band signal without any loss of information.

To look at this up and down conversion, we can now pictorially depict how we can construct

$s_p(t)$

from

$s_c(t)$and $s_s(t)$

that is multiply by

$\sqrt{2}\cos(2\pi f_c t)$

in the case of

$s_c(t)$

multiply by

$-\sqrt{2}\sin(2\pi f_c t)$

in the case of

$s_s(t)$

and add these two to get

$s_p(t)$.

Now, to get back your

$s_c(t)$and $s_s(t)$,

you then take

$s_p(t)$

add a low pass filter that cuts off the

$2f_c$

frequencies, you get

$s_c(t)$.

In a similar manner, you multiply by

$-\sqrt{2}\sin(2\pi f_c t)$

add a low pass filter that cuts off your

$2f_c$

frequencies and you end up with

$s_s(t)$.

We will now take a detour and demonstrate how you can perform this up and down conversion for a pair of real baseband signals using GNU radio. As you would recall, the simplified structures shown here can be used directly in order to produce our upconverted pass band signal and then downconvert it to obtain our original baseband signals as well.

We will now construct a very simple example that takes two baseband real valued signals, upconverts them to passband and then downconverts them back to baseband and obtains the two real signals once again. For our example, we will produce two signals namely a sinc and a sinc square. Let us begin by first importing numpy so that we can use the numpy python functions conveniently. So control f or command f and type import, we grab the import block, double click it and we type import numpy. Next we

create an array consisting of the time values at which the sinc has to be evaluated.

We press control f or command f and type variable. We grab the variable block, put it here, double click it. The name will be t underscore val and the value will be numpy a range from minus 512 to 512 upon 32. This will allow us to produce a sinc that is 32 milliseconds long and has spacing between its successive zero crossings and the peak value and the successive zero crossings of exactly one millisecond. Let us add our common elements namely a vector source, time sink, frequency sink, control f or command f, vector source and we have our vector source over here.

We double click the vector source, change the type to float and the vector that we want is numpy.sinc of t_val. We add our throttle, command f or control f and throttle. We double click the throttle to convert the type to float. We then add a time sink, so control f or command f and time sink.

Double click, convert it to float. We add a grid, we also auto scale and we add a frequency sink, so control f or command f, f, r, e, q. Grab the frequency sink, double click it, convert it to float. We want a rectangular window. We scroll down a bit.

We want a grid, we want auto scale. We say ok. We then connect the vector source to the throttle, the throttle to the sink as well as the frequency sink and we then run our flow graph. So, we get a nice sinc and its spectrum. This is familiar.

Let us now construct our second signal that is sinc square. To do that, we will just take the vector source that we already have, hit control c to copy, hit control v to paste. If you are using a mac, please use command c and command v. We double click the vector source and we replace or we add a star star 2 that corresponds to a squared of every element in the array. So, this will yield us sinc square for the same t_val.

Let us now increase the number of signals in the time sink and frequency sink. We double click this time sink and say 2 inputs. We double click the frequency sink and say 2 inputs. Connect the second input to the second signal to the second inputs and executing this will yield a sinc squared which arguably is narrower and it has a slightly wider spectrum because its spectrum can be obtained by convolving the rect with the rect because it will be triangular in the log domain it appears in this fashion.

Great. Our next task is to up convert this by multiplying the first signal by

$\cos\left(2\pi f_c t\right)$

and the second signal by

$\sin(2\pi f_c t)$.

Let us first add a range slider for fc. So, we press control f or command f type range grab this range and double click it. We will change the id to fc the type will be float. Let us keep the default value as 3000 and let us keep 3000 as the start value and let us say 8000 as the end value step 1.

Our next task will be to produce

$\cos(2\pi f_c t)$ and $\sin(2\pi f_c t)$.

Let us do that a little bit below on the flow graph. We press control f or command f and say signal source we grab the signal source and place it here. We double click the signal source the frequency is going to be fc and it is going to be float. We can then create a copy of this by hitting control c or command c and control v or command v dragging it below and changing this to sine.

We now have 2 signal sources. Let us move this a little bit. We now have our 2 signal sources that will produce a sine and cos. We will multiply the first vector source with the cos and the second vector source with the sine. We will temporarily ignore the root 2 factor, but you can always add it right over here by changing the amplitude of the signal source to root 2.

Let us grab multiply blocks by hitting control f or command f and typing multiply. We drag this multiply block, we double click it make it float. We can copy this multiply block and paste it again. The first signal gets multiplied by the cos. The second signal gets multiplied by the sine.

Negative sine, so let us make a small change. We change the amplitude of the sine to minus 1 and these are then added up control f or command f and add we get the add block. We double click this. We change it to float. We then connect these two and we then observe the spectrum of the resulting signal by adding a separate frequency sink.

So control f or command f and freq grab a QT-GUI frequency sink, double click change it to float, change it to a rectangular window. We will add a grid, we will add auto scale. We will say ok, connect the out to the in, run the flow graph. You can see that the combination of the two spectra appear at fc which we have chosen as 3000 Hz.

Let us change it to 4000 Hz. You will see that the spectra have translated to be around 4000 Hz but the values are still between 4.5 and minus 4.5 for the blue curve which is faithful because the original baseband signal was between minus half kHz and half kHz.

The second one is a little wider because of the convolution but it will largely be between minus 1 kHz and 1 kHz which is what you see roughly over here.

It is between 3 kHz and 5 kHz. This spectrum is a magnitude spectrum and you can clearly see that it has similar characteristics on the positive and negative frequencies and it is the spectrum of a real signal whose baseband equivalent is a complex signal that embeds these two signals namely the sinc and the sinc square. Let us now work on getting back our original signal from this complex baseband rather this passband signal that contains the complex baseband signals. We close this. What we will now do is to multiply this particular signal by cos, multiply it by minus sign and again grabbing the resulting signals and putting them in a sink. However before that we must remember that we need the low pass filter to cancel the components along 2f_c.

Let us now build this particular system. So, we can use the same signal source. We can just take a multiply block. I am going to hit Ctrl C and Ctrl V to copy this multiply block. Take this signal, add it to the input 1.

Take the cosine, add it over here. Of course you could use separate signal sources as well but here to reduce the number of blocks I am keeping it simple and you reusing the same signal source. We then take this multiply block and create a second multiply block by hitting Ctrl C and Ctrl V or Cmd C and Cmd V. We connect the same output signal to the input and connect the minus sign over here. We then have to add two low pass filters that cut off the frequencies around 2f_c. To do this we use the inbuilt low pass filter in GNU radio.

We hit Ctrl F or Cmd F and say low and we take the low pass filter, place it over here. We double click the low pass filter and say it is float to float interpolating it does not matter because we are just setting the interpolation to 1. We will now set the cutoff frequency to FC. It is safe to set it to FC because any frequencies above FC are cut off and our original signal is actually just at baseband and it is actually the components that we obtain near 2f_c that we want to remove.

So we press OK. We then copy this and paste it again to get a second filter. Let us make some space for these filters. We connect the first output to one filter, second output to another filter and we will grab another QT-GUI frequency sink. We already have this one over here. So we can say Ctrl C and Ctrl V to get a two input GUI frequency sink.

Connect these outputs over here. And we need to set the transition width. Apologies. We will set the transition width to 1000 Hz in both the filters. Let us now run this flow graph. If you now observe the QT-GUI frequency sink, both the syncs give you roughly the same type of signal.

Of course, there are some artifacts over here because of the filtering but this is well below about minus 90 dB indicating that it is very very minimal and you largely get the characteristic that you wanted. In order to ensure that this is indeed the case, let us add a time sink as well. We will just copy this time sink. Place it over here.

Connect this output here. Connect this output over here. In fact, we will temporarily remove this time sink or we will disable it by right clicking and saying disable. Let us run the flow graph now. You can see that the only time sink that we have is this one and that displays the signal and the other spectra are as we expect. Let us actually view all the four signals at once.

Let us remove this QT-GUI time sink and let us just add four signals to this QT-GUI time sink which are the third being this one and the fourth being this one. Therefore, we would expect the signal 0 and 2 to be similar, 1 and 3 to be similar. That is the first and the third and second and the fourth should be similar. If we now look at what we have over here, the signal 1 and the signal 3 do look similar like a sinc. The signal 2 and signal 4 do look similar like a sinc square.

If you now compare the amplitude, it is 1 over here and about 0.5 over here. The reason for this is the root 2 that we chose to ignore. Let us address that by changing the amplitude of our cosine and sine to root 2 or maybe 1.414 to approximate root 2 minus 1.414 and run our flow graph. You can see that the signal 1 and signal 3 are very similar. Signal 2 and signal 4 are very similar meaning that barring the delay that is produced by the low pass filter, you are able to recover the signals with reasonable accuracy and that confirms that this approach of converting this baseband signals to passband and converting them back to baseband is able to preserve all the characteristics of your signal, the band limited baseband signals and you are not wasting any spectral redundancy because you are sending two real signals that occupy the same bandwidth within fc plus w and fc minus w. So to summarize for complex baseband signals, in particular for digital communications, we will henceforth directly design complex signals because there is no real constraint for us to use pairs of real signals. It is very natural for us to just design and construct complex baseband signals in the case of digital communication.

So we will be moving to designing complex baseband signals that convey information effectively while taking into account constraints such as bandwidth, power usage and so on and thus to emphasize we will be considering only complex baseband signals. As a summary, real signals there is a redundancy in the spectrum. This as you have seen is a natural consequence of the fact that

$$S_c(f) = S_c^*(-f)$$

that is there is a conjugate event symmetry in the spectrum. The complex baseband signals utilize the full bandwidth effectively because they break this redundancy by allowing you to transmit two real signals concurrently over the same bandwidth. By then up converting this complex baseband signal, we then obtain real passband signals that capture all this information and are ready for transmission.

During reception, these very real passband signals can then be used to recover the complex baseband signals. Thank you.