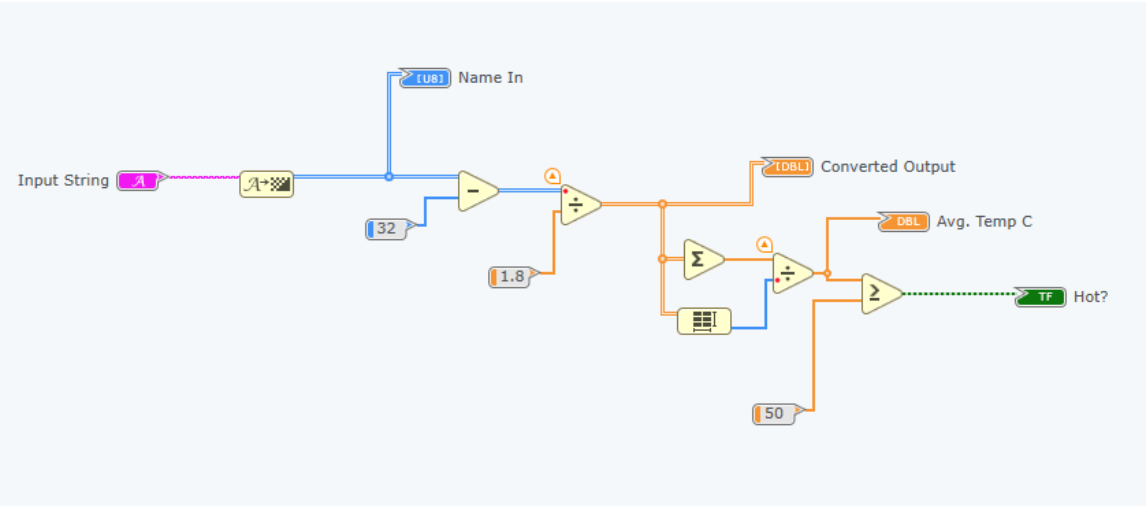
Comms Lab Log Book

By Thomas Nugent and Pu Yang

Lab 1

Exercise 1: Data Types

In this exercise we learnt the basics of LabView including setting up the diagram using modules by searching in the function palette and adding controls, constants and indicators. We also learnt how to set up the panel so we can run the program. We had no problems for this exercise.

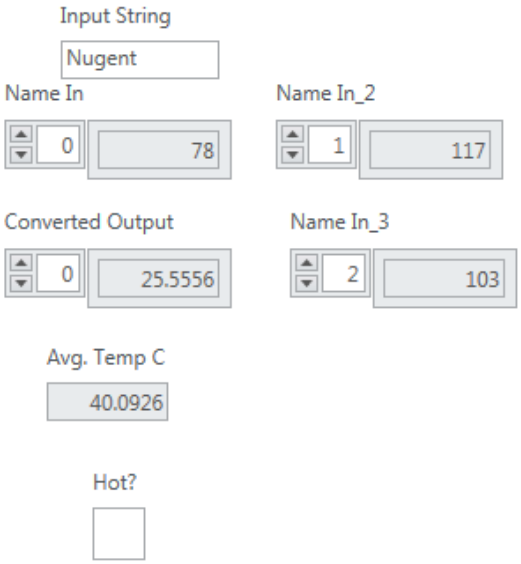


Screenshot 1 - Schematic diagram for the data types exercise.

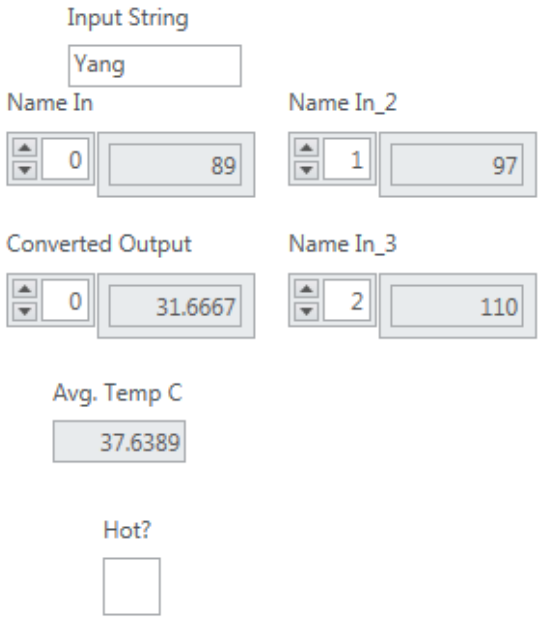
The program works by converting the input string to an integer (ASCII form) and then treating the integers as a temperature in Farenheit, which is then converted to degrees Celsius by the following equation:

```
C = (F - 32) / 1.8
```

These values are then averaged to work out how 'hot' a string is. We tested both of our surnames and got the following results (first 3 characters shown):

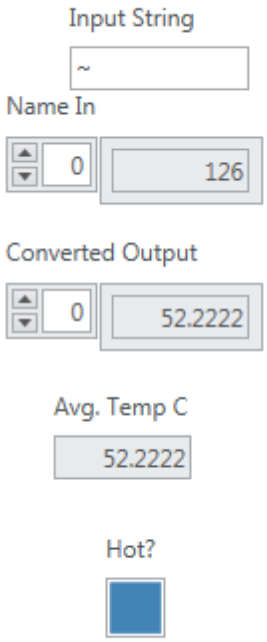


Screenshot 2 - "Nugent" entered into the 'hot' calculator.



Screenshot 3 - "Yang" entered into the 'hot' calculator.

We realised that we would need a string that contained ASCII characters that were a large integer number, and tested our theory by looking up an ASCII table. A high value that we could easily type was "~".



Screenshot 4 - "~" entered into the 'hot' calculator.

Exercise 2: Implementation of the Central Limit Theorem

In this exercise we learnt how to implement the Central Limit Theorem which is stated below. We were introduced to some new modules such as loops, random numbers, comparisons and histograms. We encountered many difficulties with this task because the last part asked us to adjust the probability density function so that it had a mean of 0 and a variance of 1. Several GTAs gave us conflicting answers so we were running around in circles for a short while.

Central Limit Theorem

> When independent random variables are added, their sum tends towards a normal distribution even if the variables themselves are not normally distributed.

For n random variables that are mutually independent and identically distributed, each with mean μ and variance σ^2, the new random variable will have mean and variance as follows:

$$\mathrm{E}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathrm{E}(X_i) = n\mu$$

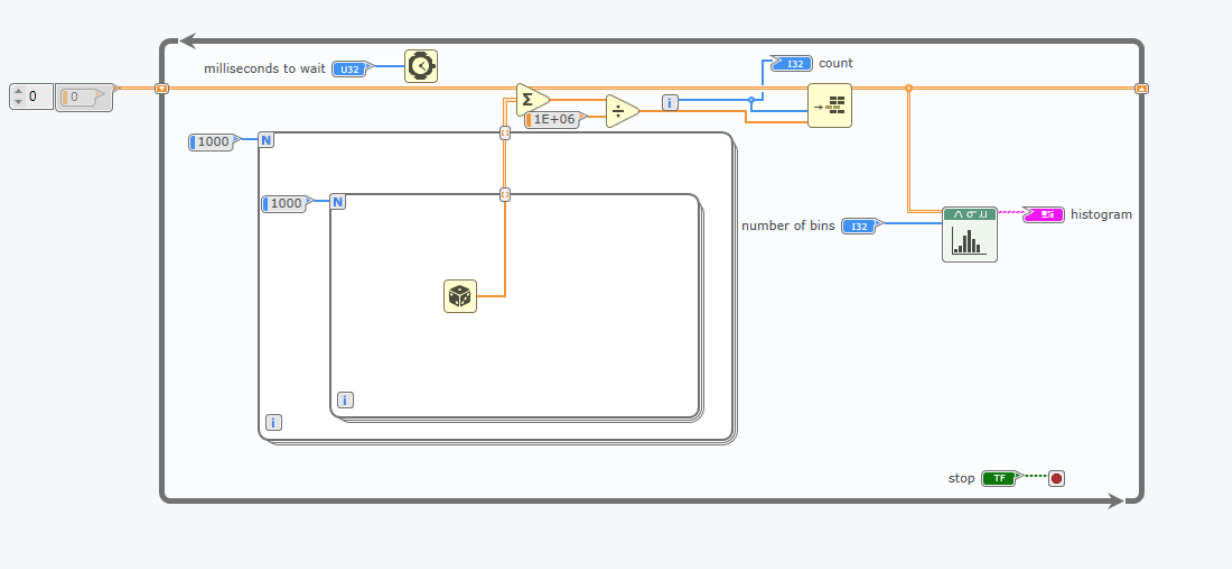
$$\mathrm{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathrm{Var}(X_i) = n\sigma^2\,.$$

As n→∞:

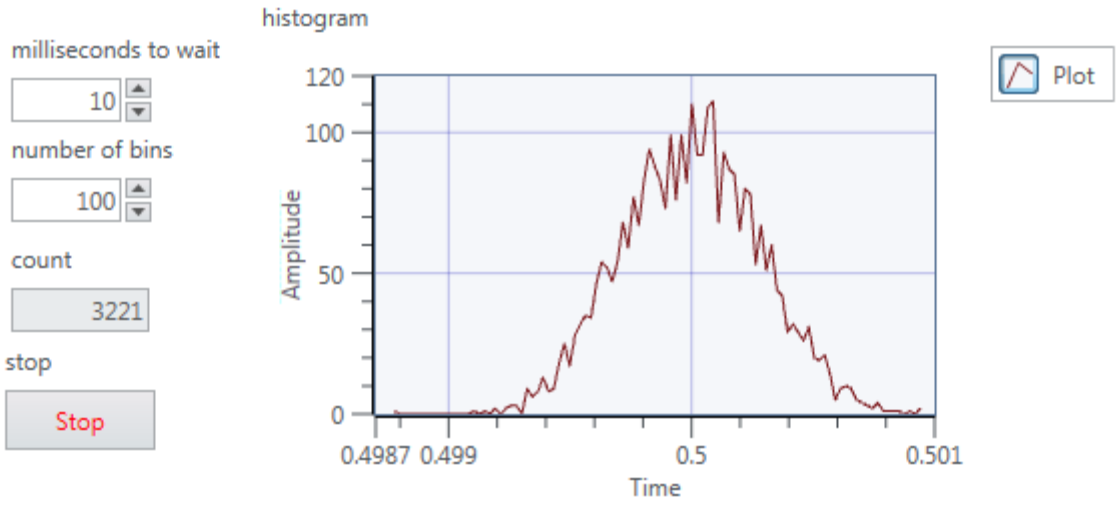
$$X_1 + X_2 + \ldots + X_n \to \mathrm{N}(n\mu, n\sigma^2)\,.$$

Wikipedia / Lecture Notes

To simulate a random variable, we used two *for* loops to create a 2D array that contained random numbers between 0 and 1. We then calculated the mean of this random variable and added this to an array. The loop is then run again (with some milliseconds of delay between iterations) with more i.i.d. RV means added to the array. This array is then plotted in a histogram. Using CLT, this graph converges to a normal distribution with the more loop iterations.



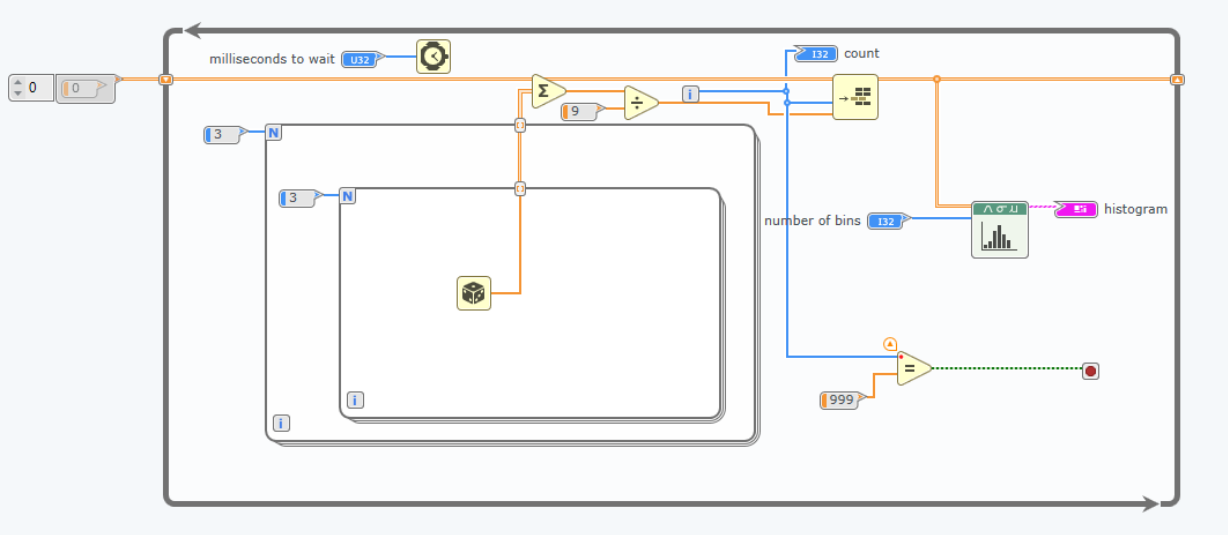
Screenshot 5 - Schematic diagram for the CLT exercise.



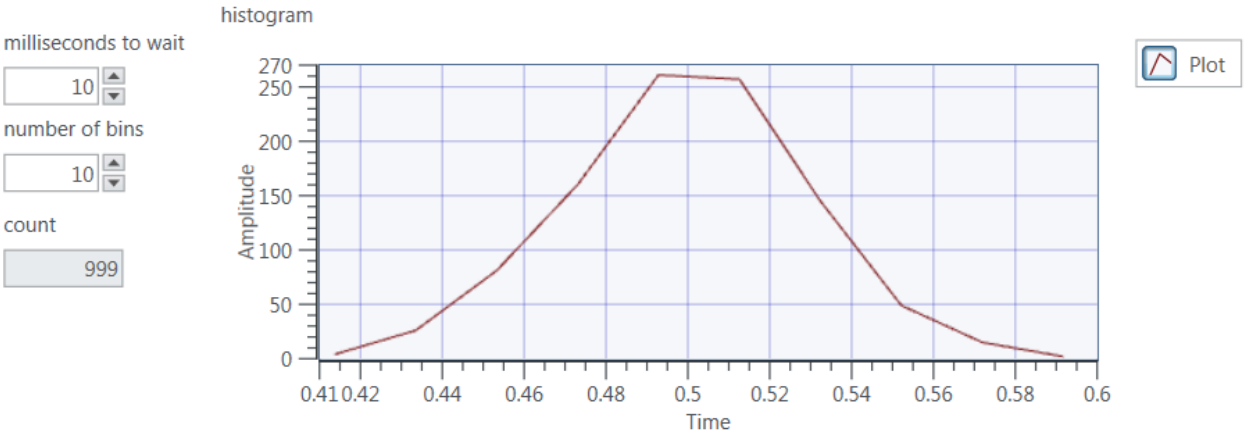
Screenshot 6 - Panel view from the CLT exercise.

Notice that the distribution is not very smooth. This is because the number of bins we have is large meaning the probability of being in any one bin is low. If we decrease the bins to ~10 then the graph is much smoother (this is done later).

In order to stop the program after 1000 iterations we just used a comparison block that is connected to the conditional terminal and a constant 999 (iteration starts at 0).

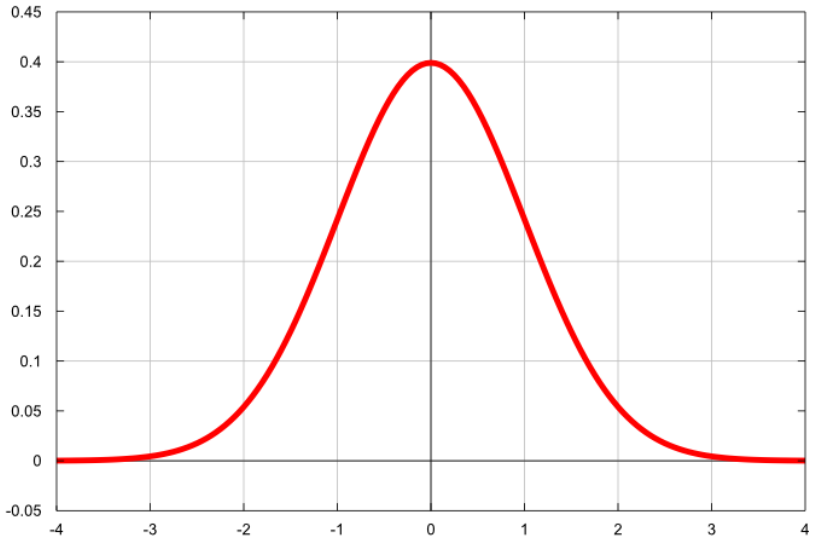


Screenshot 7 - Schematic diagram that stops after 1000 iterations.



Screenshot 8 - Panel view that stops after 1000 iterations.

Our last task was to adjust the random variable so that the normal distribution had 0 mean and unit variance. This would mean it would look something like below:



In order to do this, we decided to adjust the random number after it had came out of the generator. We could have also added the adjustments outside the for loops, the only difference would be a factor of 10.

We know that the random variable is uniformly distributed between 0 and 1. The mean of this is simply the middle which is 0.5, so we minus 0.5 to centre the distribution around 0. The variance of a uniform distribution is given by:

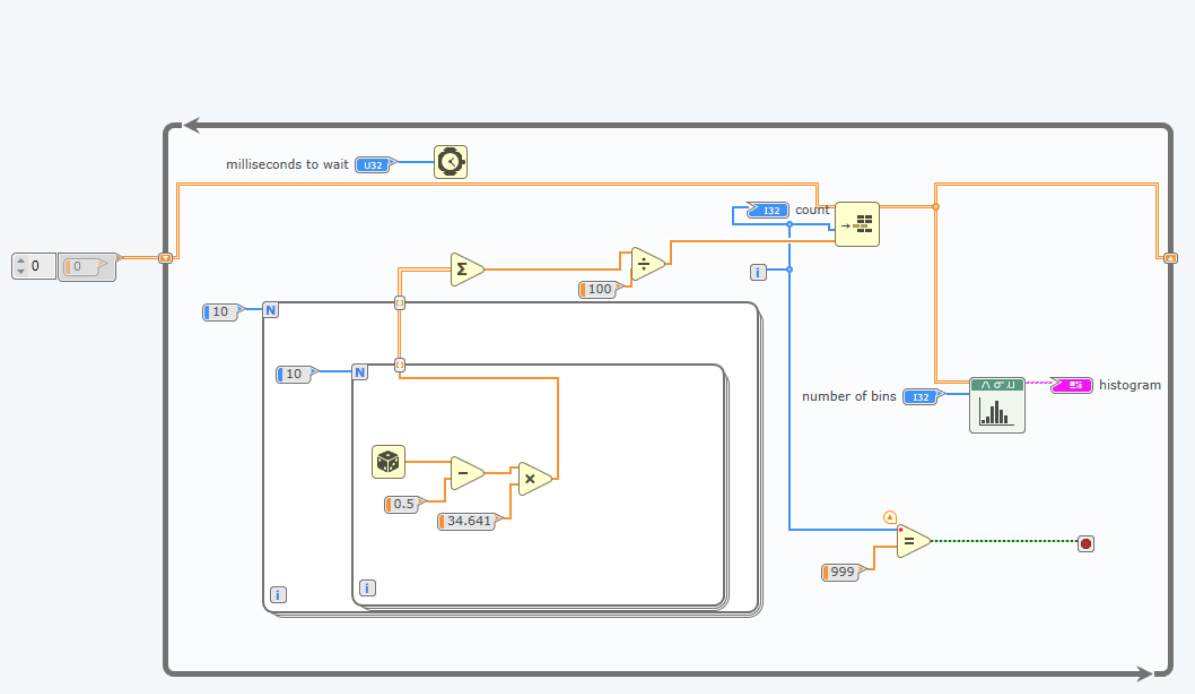
```
X ~ U(a,b)
Var(X) = 1/12 (b - a)^2
```

So for this example the variance is 1/12. If we recall the formula for scaling the variance of a variable:

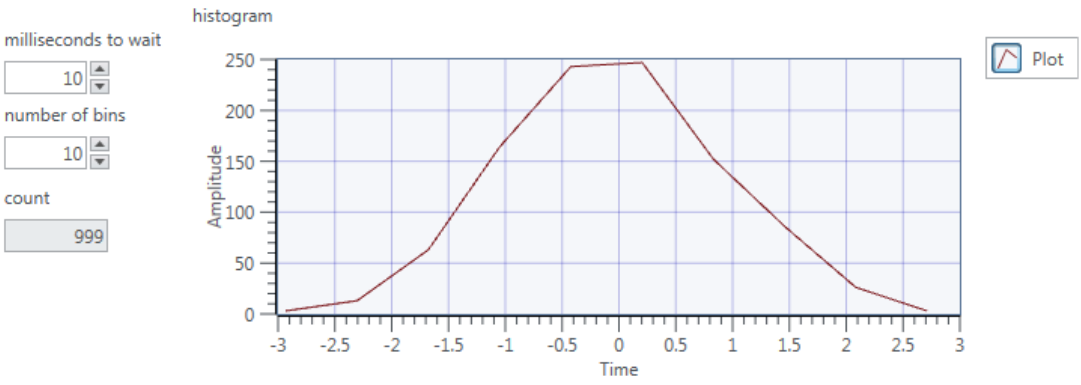
Var(aX) = a^2 Var(X)

we want to multiply by √12 to get a variance of 1. We have to remember however that we also divide by 100 at the end, which does not affect the mean (since it is now 0), so we must also multiply by 10 (√100 = 10). The final multiplier is therefore given by:

√12 x 10 ~ 34.641



Screenshot 9 - Diagram view for the standardised normal distribution.

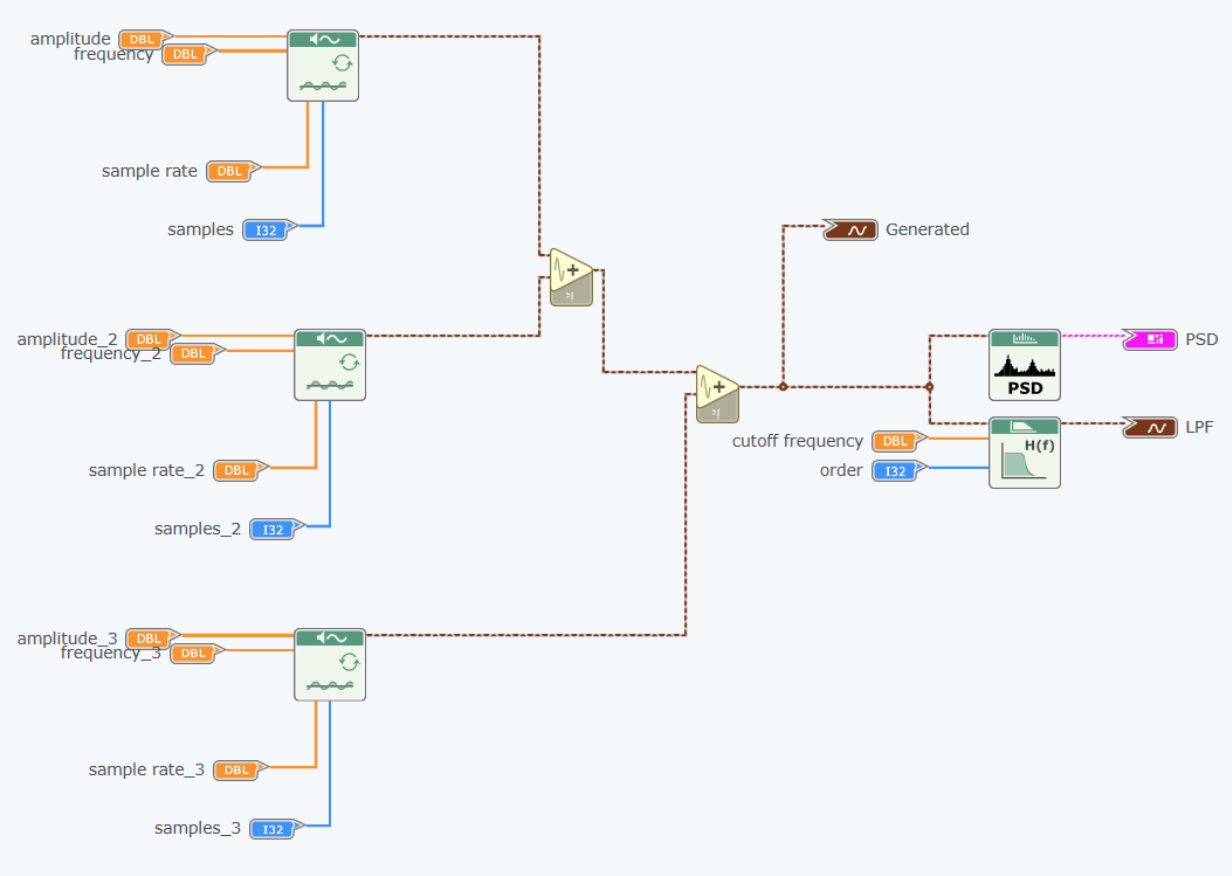


Screenshot 10 - Panel view for the standardised normal distribution.

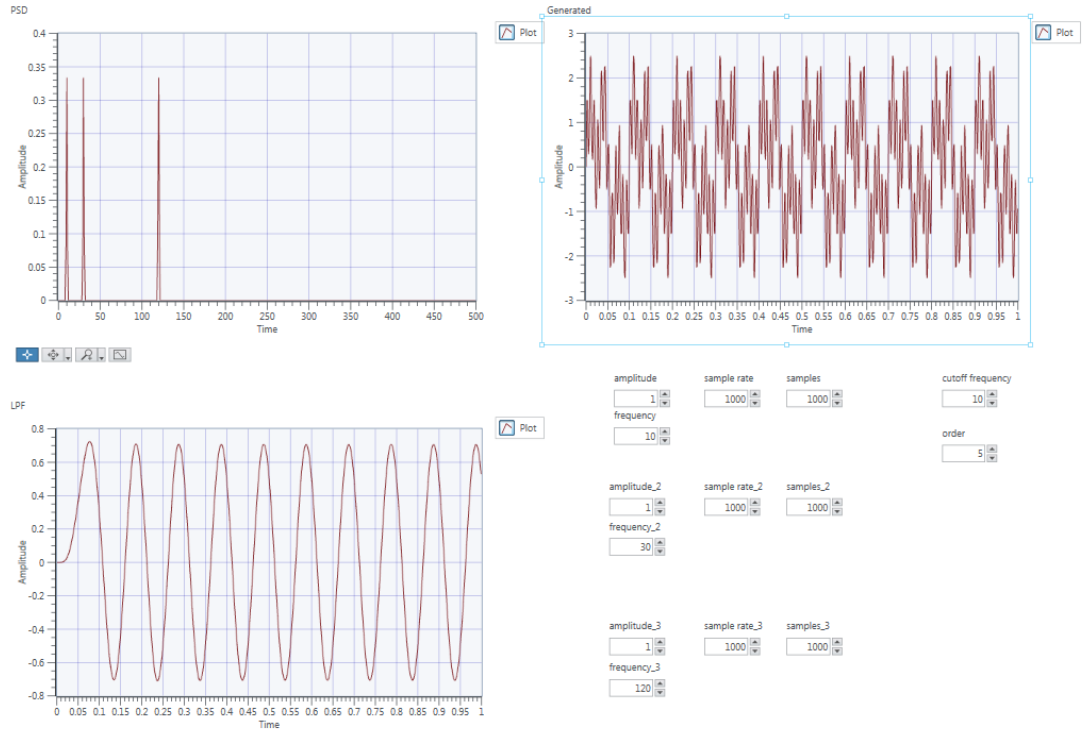
Exercise 3: Waveforms

In this exercise we learnt how to generate waveforms with given input and plot these in the time and frequency domain. We also explored the use of filters.

We first created 3 sine waveforms with frequencies 10Hz, 30Hz and 120Hz and added them together. Using the low-pass filter module with a cutoff frequency of 10Hz eliminated the higher frequency waveforms leaving us with the 10Hz sine wave on the output as shown below in screenshot 12.

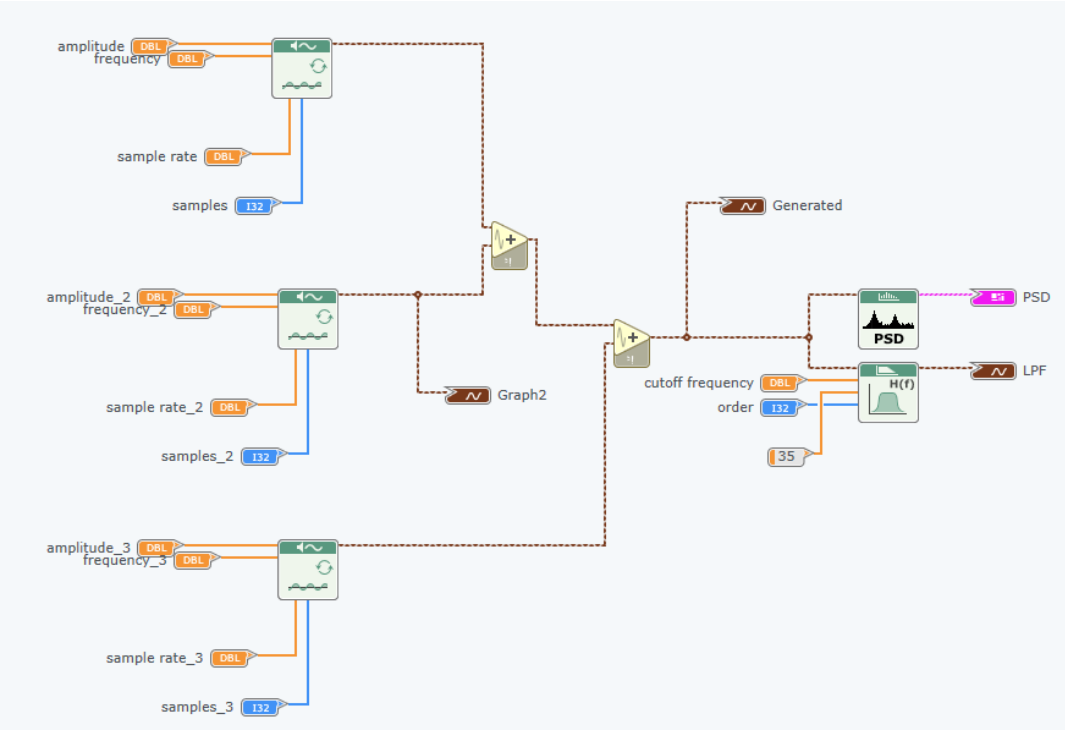


Screenshot 11 - Initial diagram view for ex3.

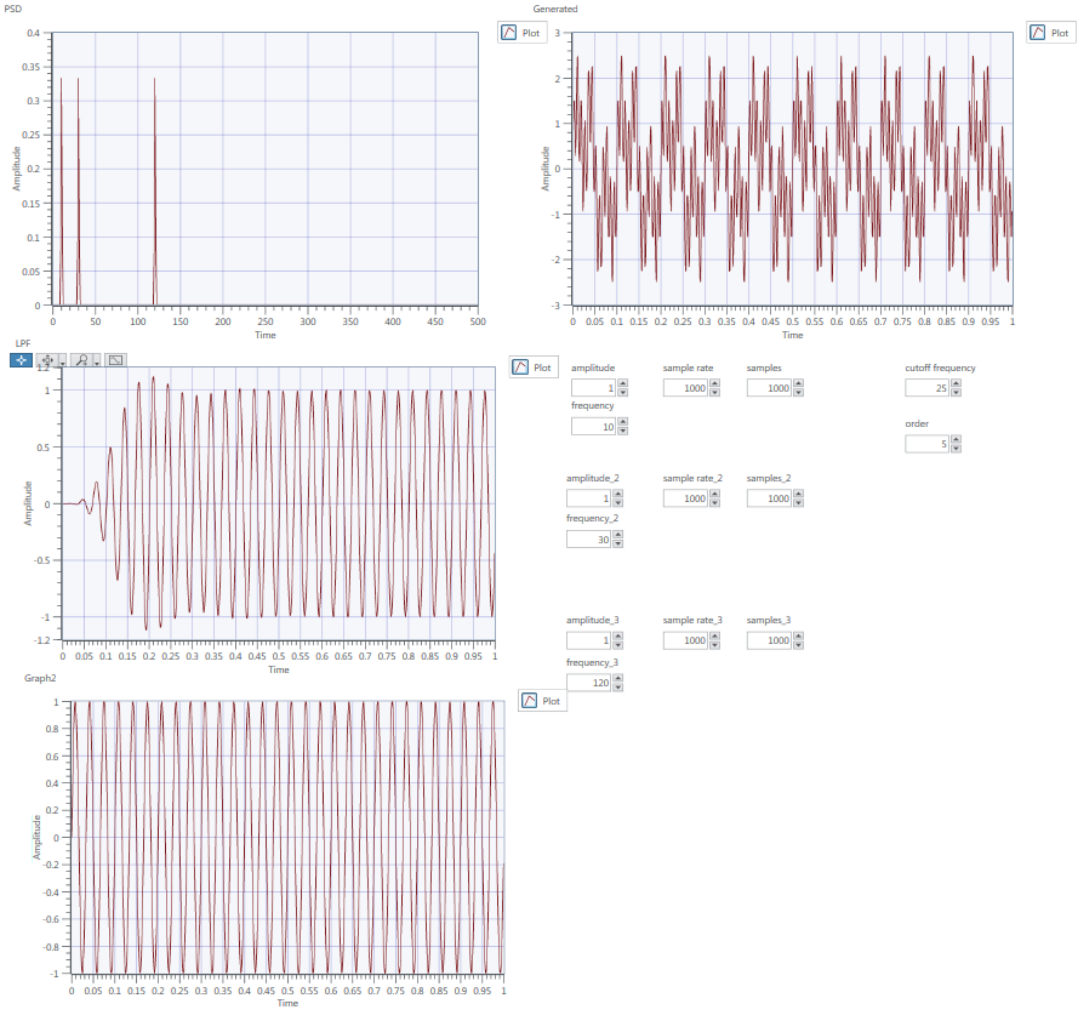


Screenshot 12 - Initial panel view for ex3.

Next we needed to modify the filter parameters so that we only saw a 30Hz sine wave on the output. To do this we changed the filter block to be a band-pass filter and set the upper limit as 35Hz. The lower cut off frequency was then given by an indicator, set to be 25Hz. The diagram and panel are shown below.



Screenshot 13 - Final diagram for ex3.



Screenshot 14 - Final panel for ex3.

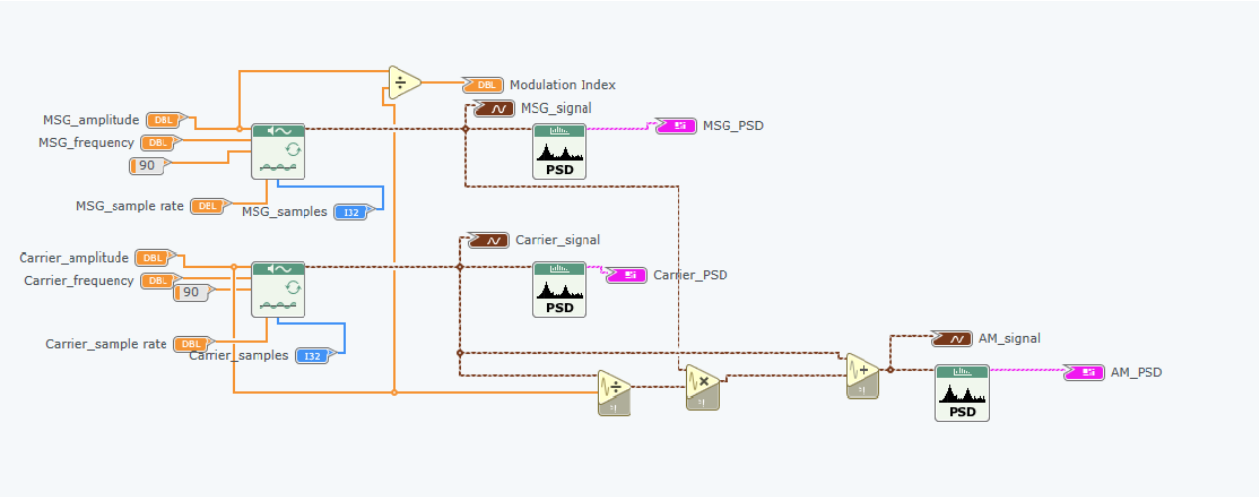
Lab 2

Exercise 1 - AM Modulator

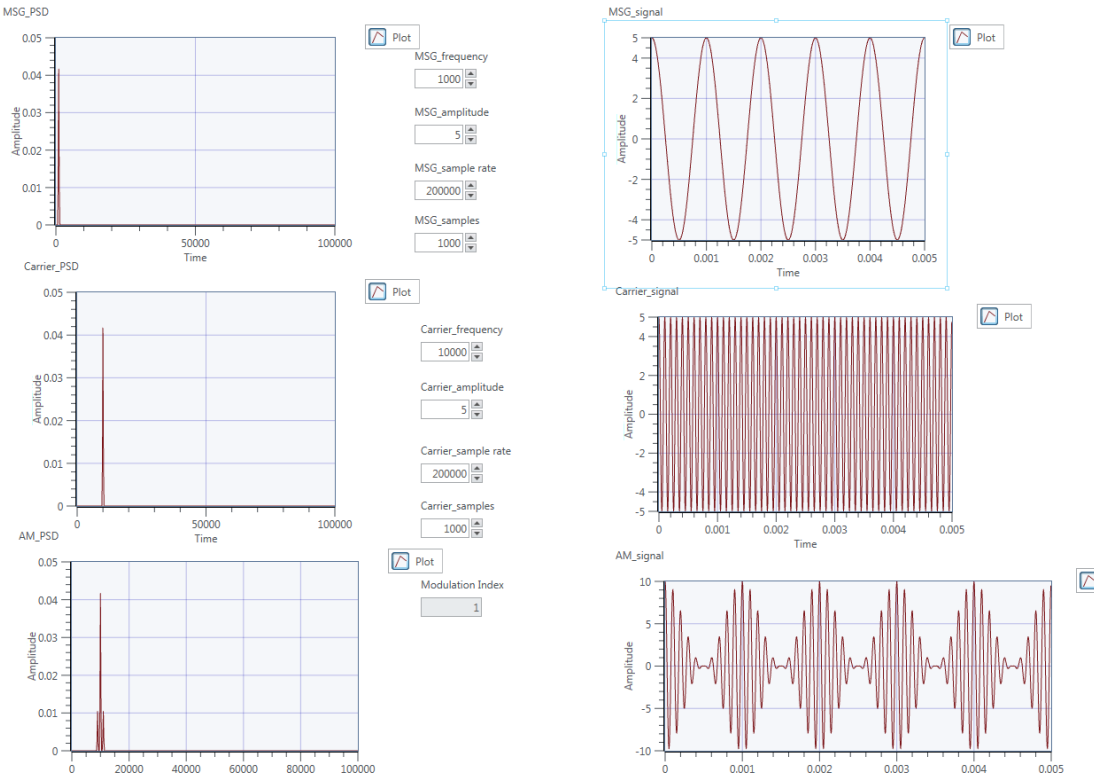
In this part of the lab we created an AM modulator. Recall the equation for AM:

$$s(t) = [A_c + A_m \cos(2\pi f_m t)]\cos(2\pi f_c t)$$

We expanded this so it was simpler to implement. We also calculated the modulation index and set an indicator for it. The implementation is given below, which involved just a few multipliers and dividers and an adder:



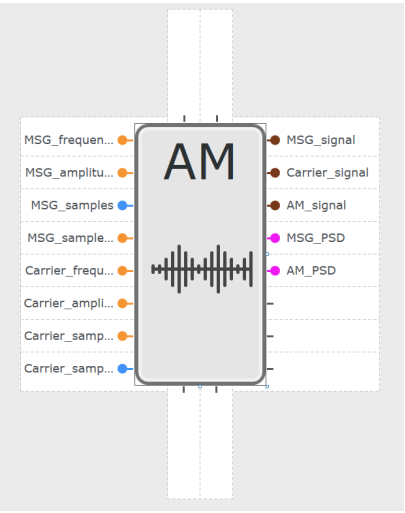
Screenshot 1 - Diagram for ex1.



Screenshot 2 - Panel for ex1.

We can clearly see the higher frequency of the carrier wave, and how modulating the signal affects the frequency domain. The PSD has one major peak at 10000Hz (the carrier frequency) and two smaller peaks 1000Hz (the message frequency) either side of the major peak. The AM signal is shown.

We also learnt how to create sub VIs, which allows you to import the module into any future VIs. The icon we made is shown below. We added the inputs on the left and outputs on the right and also added a nice image in the centre which clearly explains the functionality of the module. We were able to select the icon and increase the size so we could fit more inputs/outputs on either side.

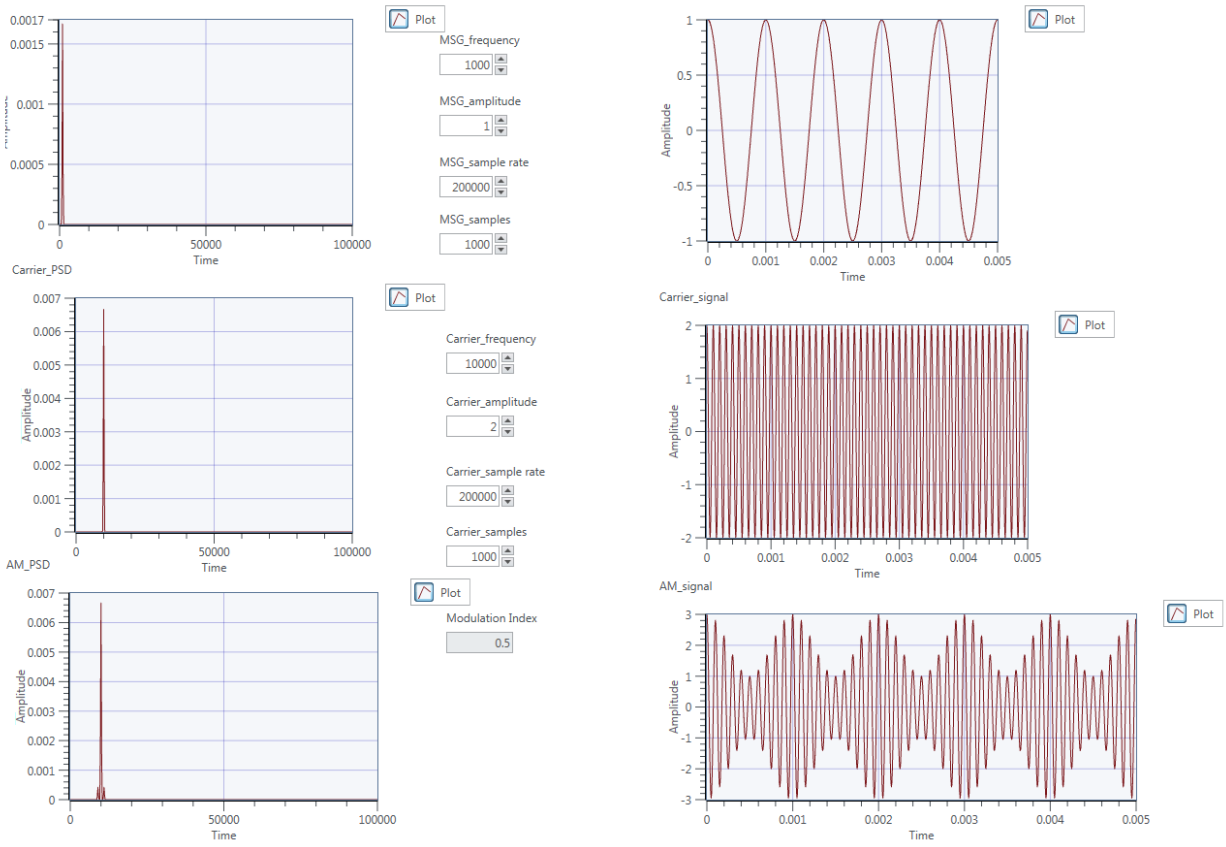


Screenshot 3 - Sub VI icon for amplitude modulation.

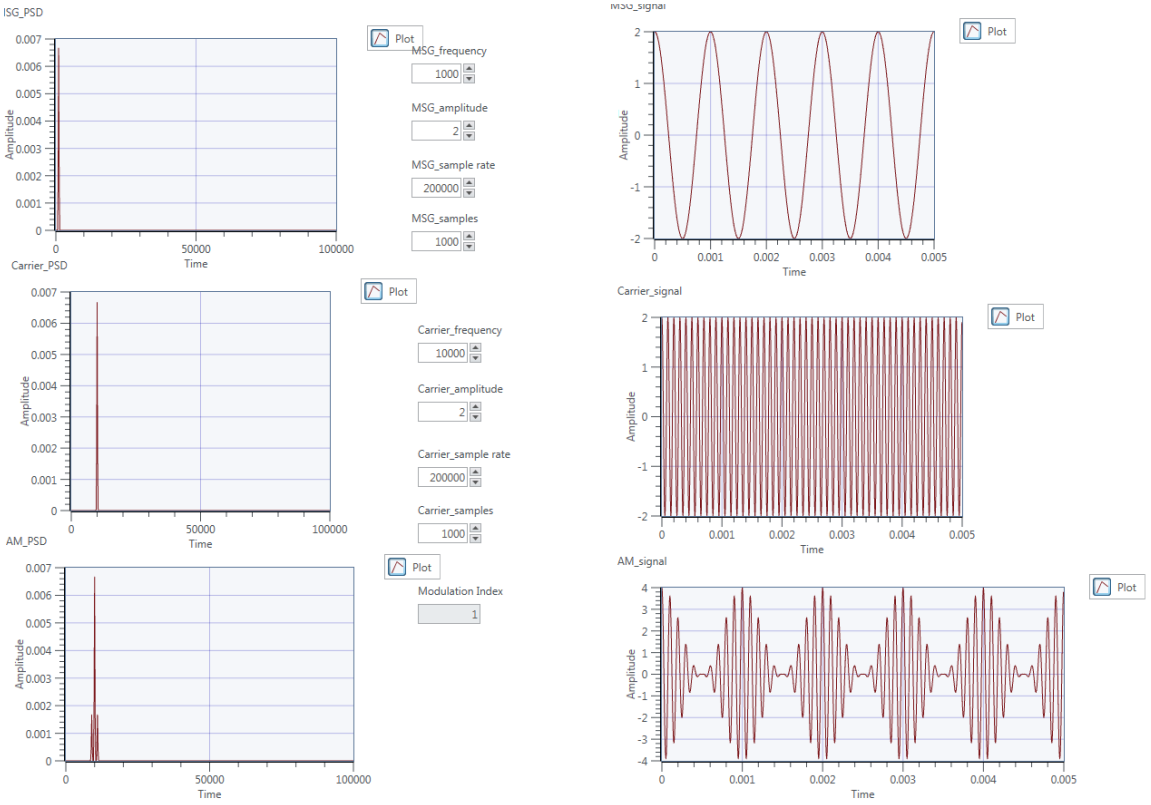
Next we wanted to observe what happens when you change the modulation index, which is given by the following equation:

μ = Am / Ac

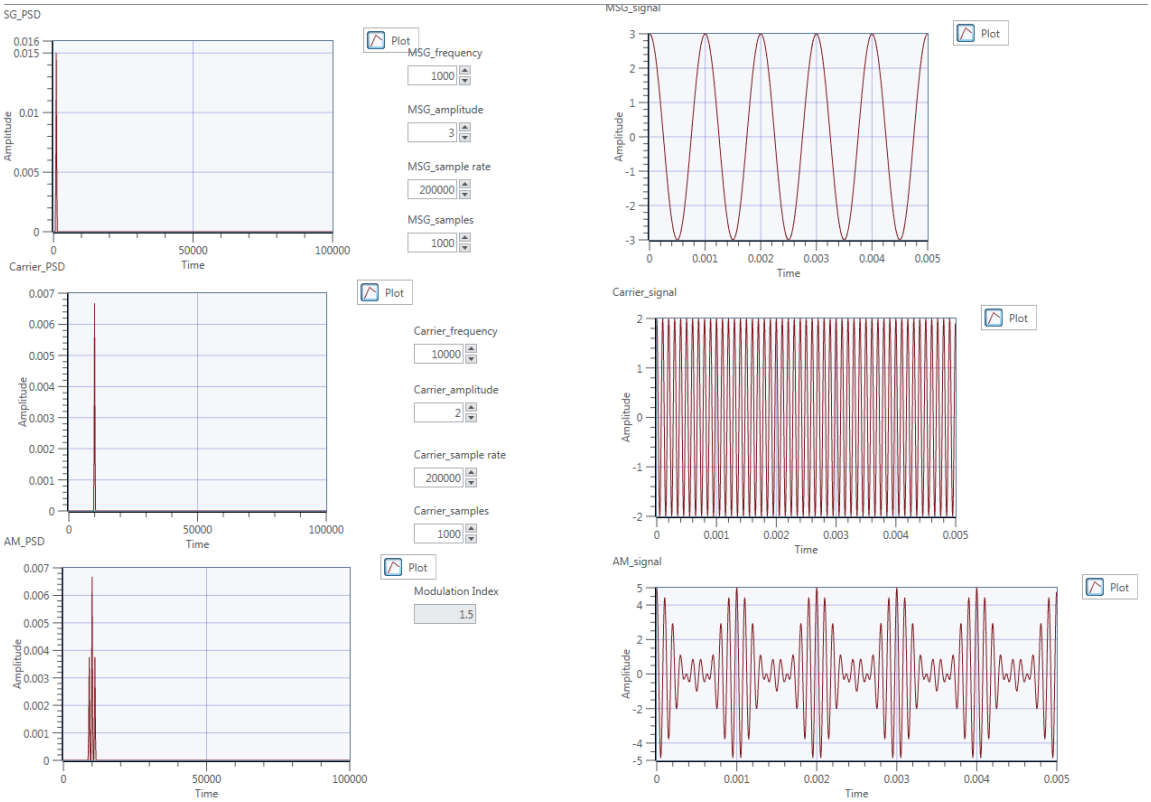
We set Ac to 2, and then set Am to 1, 2 and 3 to get a corresponding modulation index of 0.5, 1 and 1.5 respectively. The spectra are shown below:



Screenshot 4 - Modulation index of 0.5; Am = 1.



Screenshot 5 - Modulation index of 1; Am = 2.

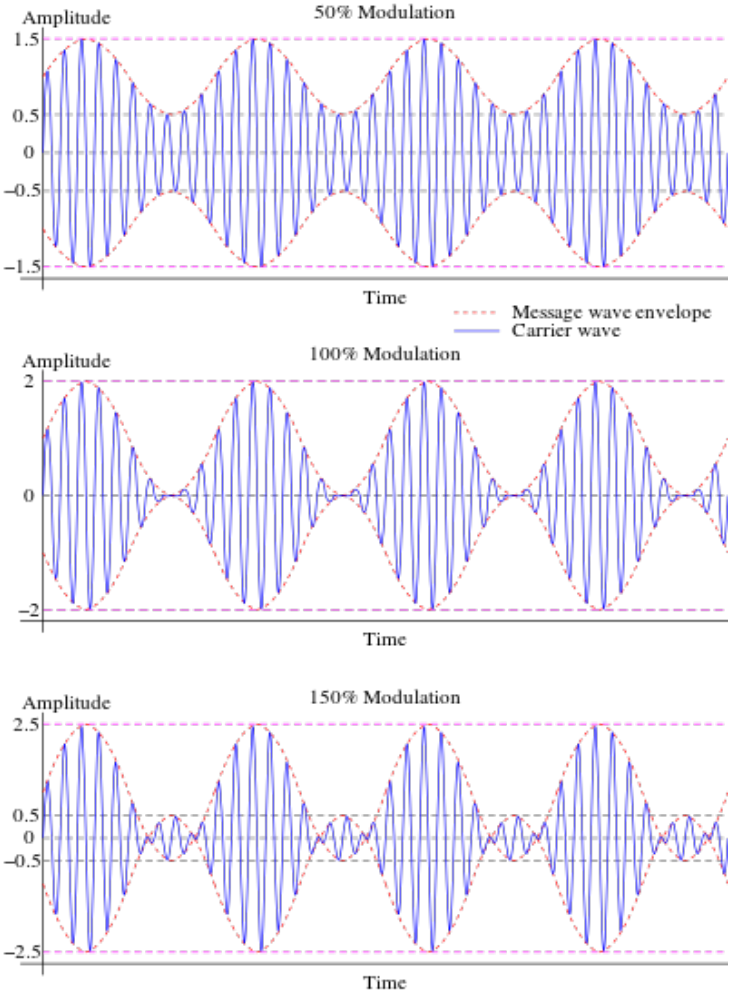


Screenshot 6 - Modulation index of 1.5; Am = 3.

After reviewing the resulting graphs we can see that increasing the modulation index does not affect the location of the peaks in the PSD, but the amplitude of the

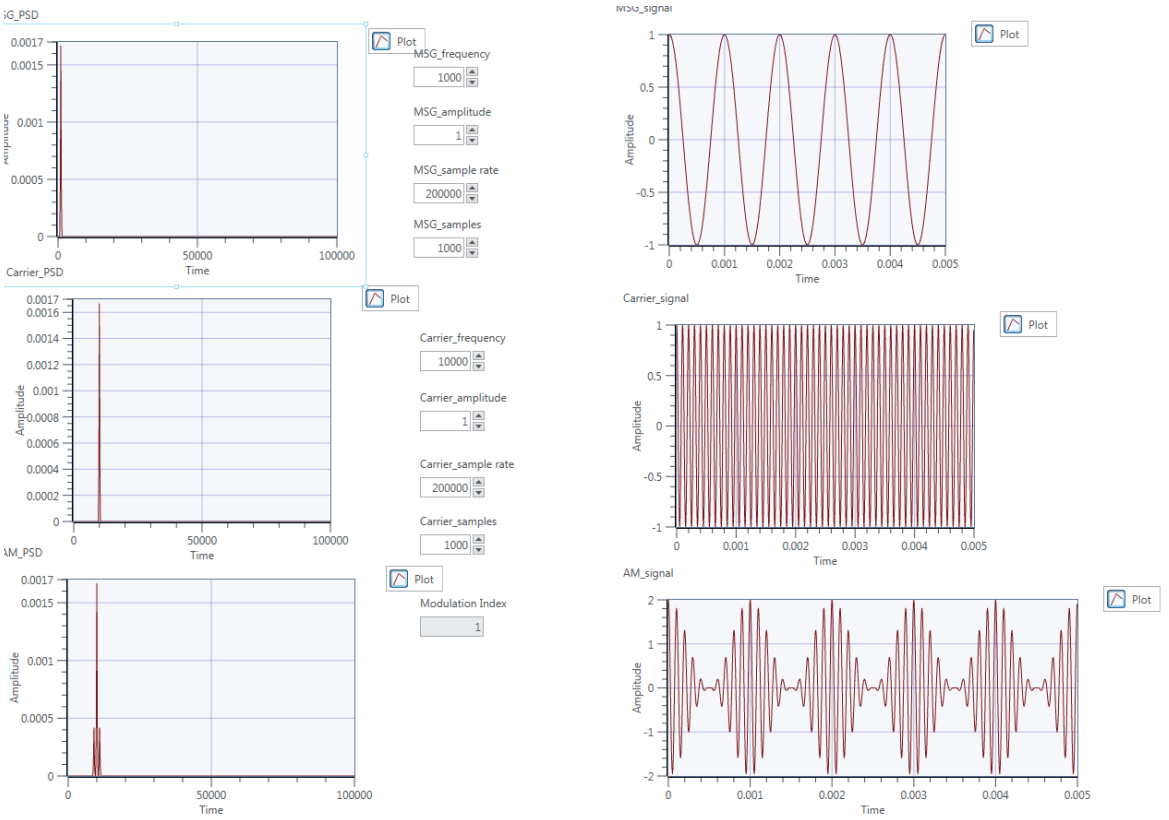
sideband peaks is increasing. This makes sense since we are increasing the amplitude of the message signal and therefore increasing the energy of the sidebands.

We can also see some strange effects happening the the signal in time domain. Initially it seems we are decreasing the envelope the higher the modulation index, but then with a modulation index of 1.5 it seems to increase again. Upon further research we can really understand what is happening, as it is hard to see from our diagram. The modulation index needs to stay between 0 and 1, and if it goes beyond 1 the signal becomes *over-modulated* and the signal is distorted. It then becomes impossible to use an *envelope detector* to demodulate the signal. You can however still use *coherent detection*. We can see this effect clearly in the diagram below:

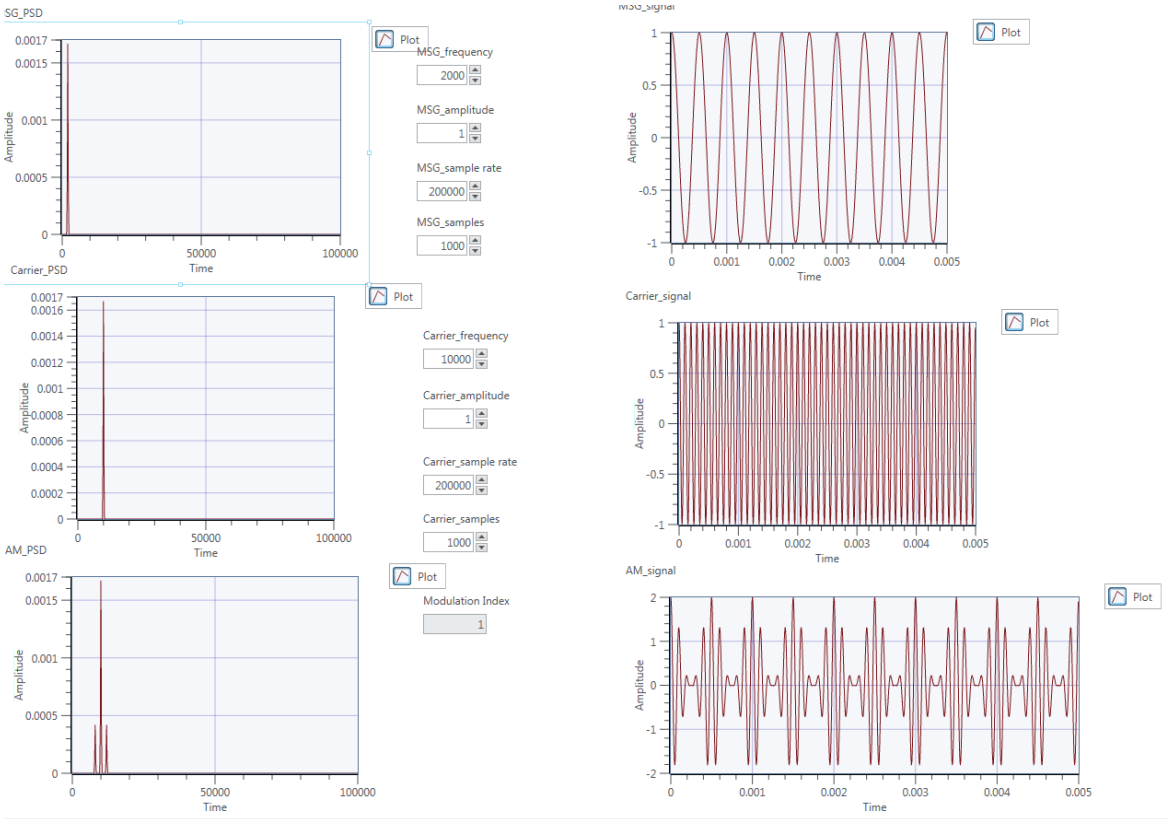


Screenshot 7 - Diagram showing the effects of increasing modulation index (Diagram from Wikipedia).

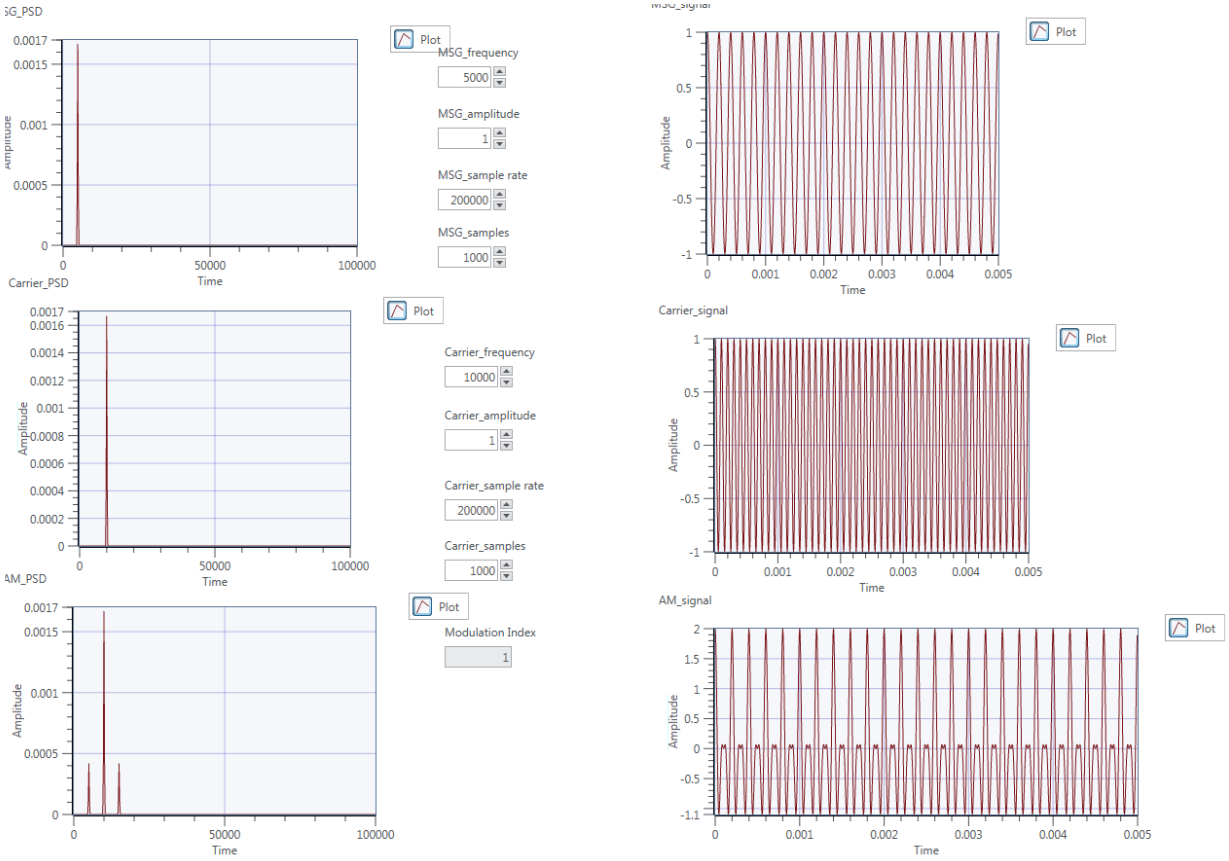
Next we wanted to observe the effect of changing the message frequency, whilst keeping the modulation index at 1 with both Am and Ac set to 1. We changed the message signal frequency to *1k, 2k and 5k* and the corresponding plots are shown below:



Screenshot 8 - Panel view with the message frequency set to *1k*.



Screenshot 9 - Panel view with the message frequency set to *2k*.



Screenshot 10 - Panel view with the message frequency set to *5k*.

Here we observe that the sideband peaks on the PSD are getting further away from the main peak at 10kHz the more we increase the frequency. The amplitude of the peaks is not changing because we are not changing the message or carrier amplitudes.

We see a more interesting affect in the *time domain* representations. As we increase the frequency we are getting closer and closer to the carrier frequency and we see some aliasing occur. At this point we cannot accurately reconstruct the original message wave.

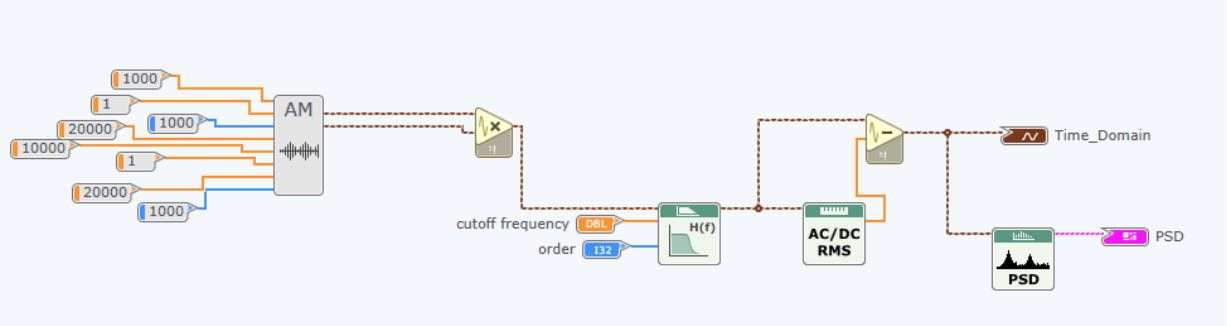
Exercise 2 - AM Demodulators

In this lab we are creating AM demodulators using two techniques, *Coherent Demodulation* and *Envelope Detection*.

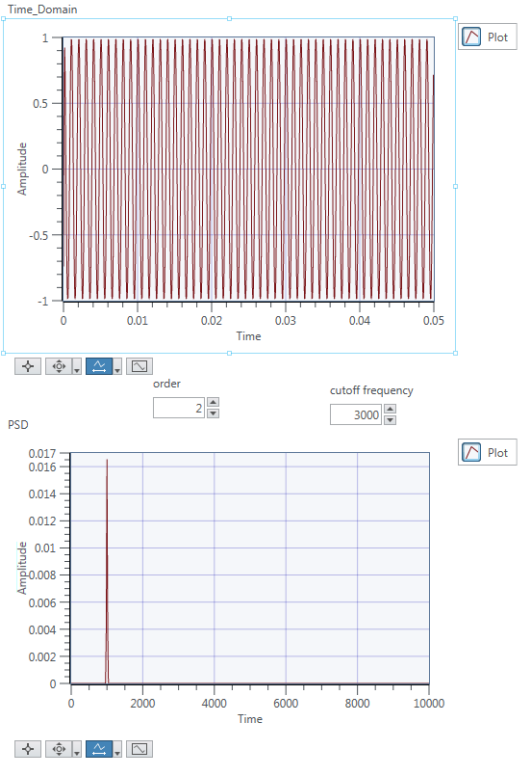
Exercise 2a - Coherent Detection

This demodulation technique uses a cosine wave of the same frequency of the carrier. By multiplying by the carrier, we move the frequency spectrum back to the

origin and also moves the same spectrum to a higher frequency. We then use a LPF to remove the higher frequency components and finally remove the DC offset to get left with the message signal.



Screenshot 11 - The diagram view of the coherent demodulator.



Screenshot 12 - The panel view of the coherent demodulator.

Here we can see a peak in the PSD at 1000Hz, which was our original message signal frequency. We set a cutoff frequency of ~3000, so that is above the 1000Hz we want, but still below the higher frequency component at 20000Hz.

Mathematical Proof:

The following is proof that the demodulation scheme works.

$$s(t) = [A_c + A_m \cos(2\pi f_m t)] \cos(2\pi f_c t)$$

$$s(t) \times \cos(2\pi f_c t) = \cos^2(2\pi f_c t)[A_c + A_m \cos(2\pi f_m t)]$$

If we recall the double angle cosine formula:

cos^2(x) = 1/2 + 1/2cos(2x)

$$= [\frac{1}{2} + \frac{1}{2}\cos(4\pi f_c t)][A_c + A_m \cos(2\pi f_m t)]$$

$$= \frac{A_c}{2} + \frac{A_c}{2}\cos(4\pi f_c t) + \frac{A_m}{2}\cos(2\pi f_m t) + \frac{A_m}{2}\cos(2\pi f_m t)\cos(4\pi f_c t)$$

We then use a low pass filter to remove the 4πfc frequency components.

$$= \frac{A_c}{2} + \frac{A_m}{2}\cos(2\pi f_m t)$$

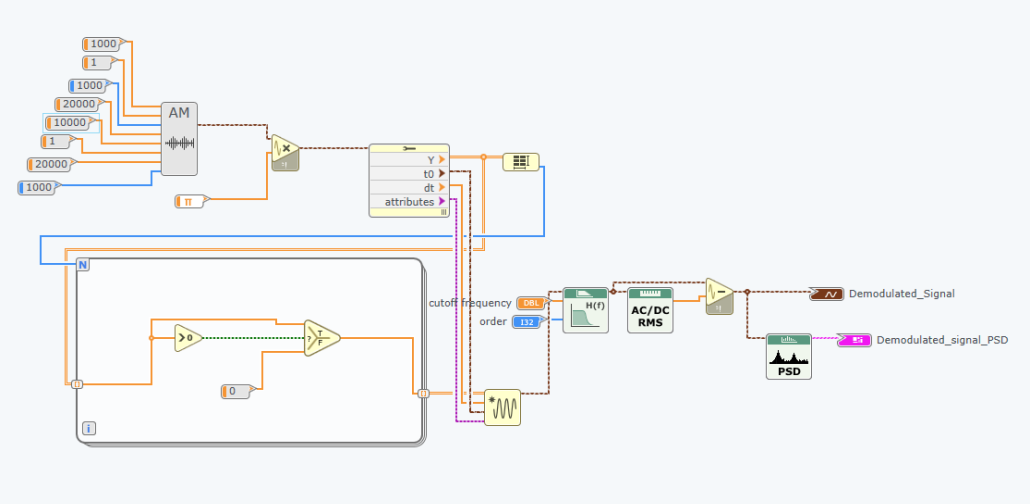
And remove the DC component as this gets rid of the Ac/2 term, leaving us with just the message signal (although it is scaled).

$$= \frac{A_m}{2}\cos(2\pi f_m t)$$

In order to get the correct amplitude of the message signal we must either double the message signal amplitude before modulating, or when demodulating multiply by 2cos(2πfct).

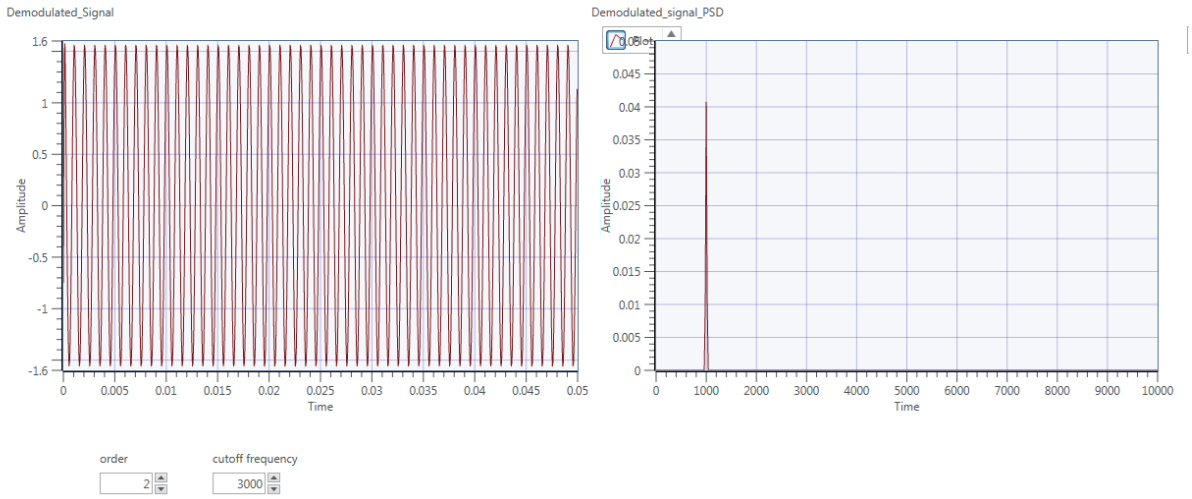
Exercise 2b - Envelope Detection

This demodulation technique relies on the envelope of the received signal, meaning that it does not work in all cases.



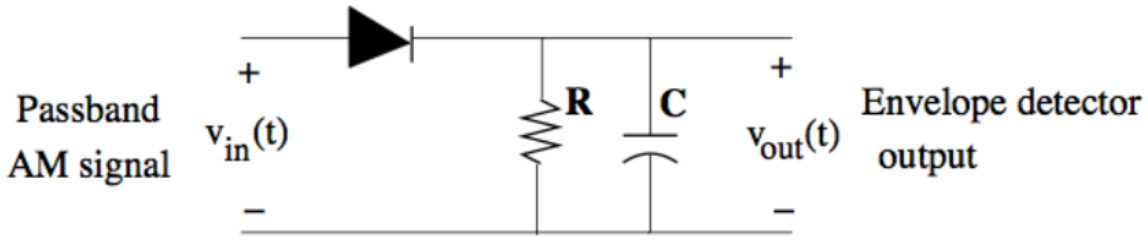
Screenshot 13 - The diagram view of the envelope detector.

The diagram works by converting the waveform values to an array so we can do some comparisons on each value. We then use a for loop which iterates through each of the values in the array and checks if it is greater than 0. If it is, then we output the array value, otherwise we output 0. This ensures we eliminate all the negative parts of the wave. This is then built back into a waveform using the same parameters from the earlier array function. Finally we use a LPF to remove the high frequency component leaving us with the message signal in the form of an envelope. We then take away the DC offset to get the message signal. Below we see the expected peak at 1000Hz:

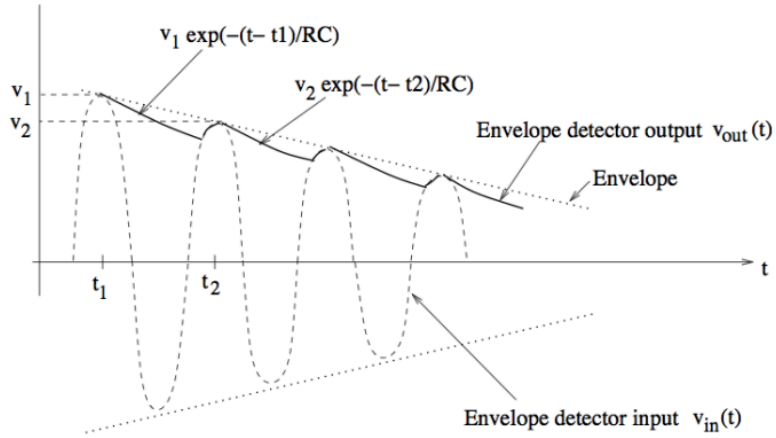


Screenshot 14 - The panel view of the envelope detector.

Envelope detection works by using a diode to keep the positive part of the wave, and then passing the signal through a low pass filter to remove the high frequency carrier component.



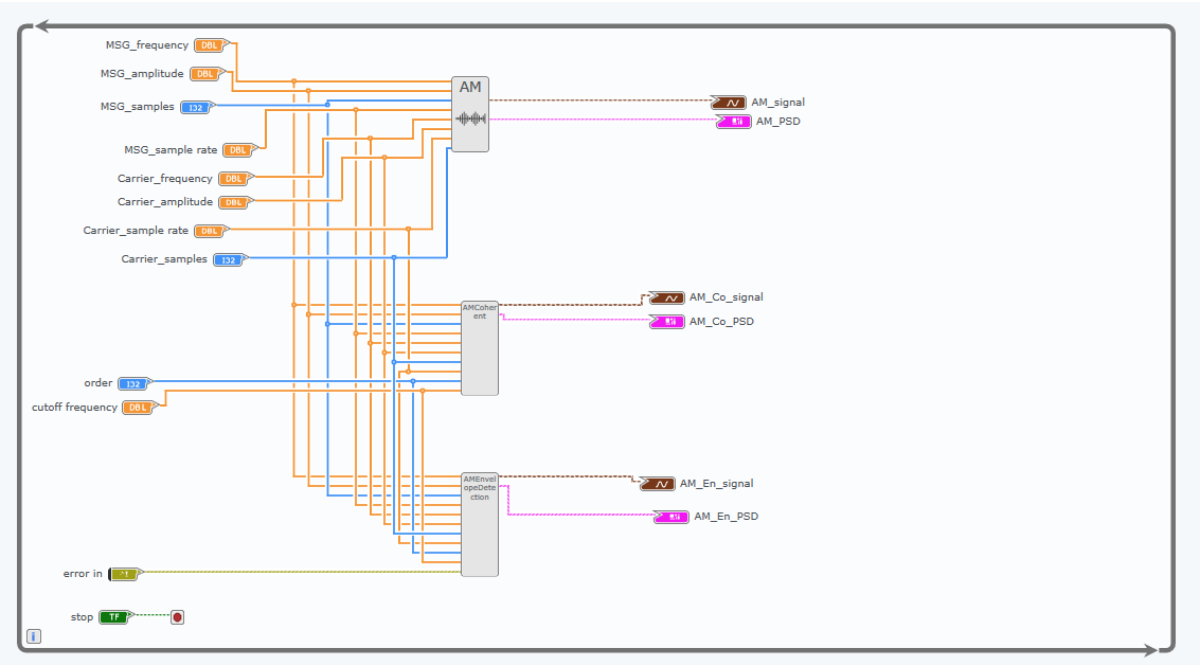
The capacitor is continually charged and discharged so that the output is the envelope of the signal. Values of the components must be chosen so that the capacitor does not discharge too slowly or too quickly. The input is usually scaled first by π because the LPF will reduce the amplitude of the signal.



After this point we have an envelope representing the message signal, but we must remove the DC offset to get the original message signal.

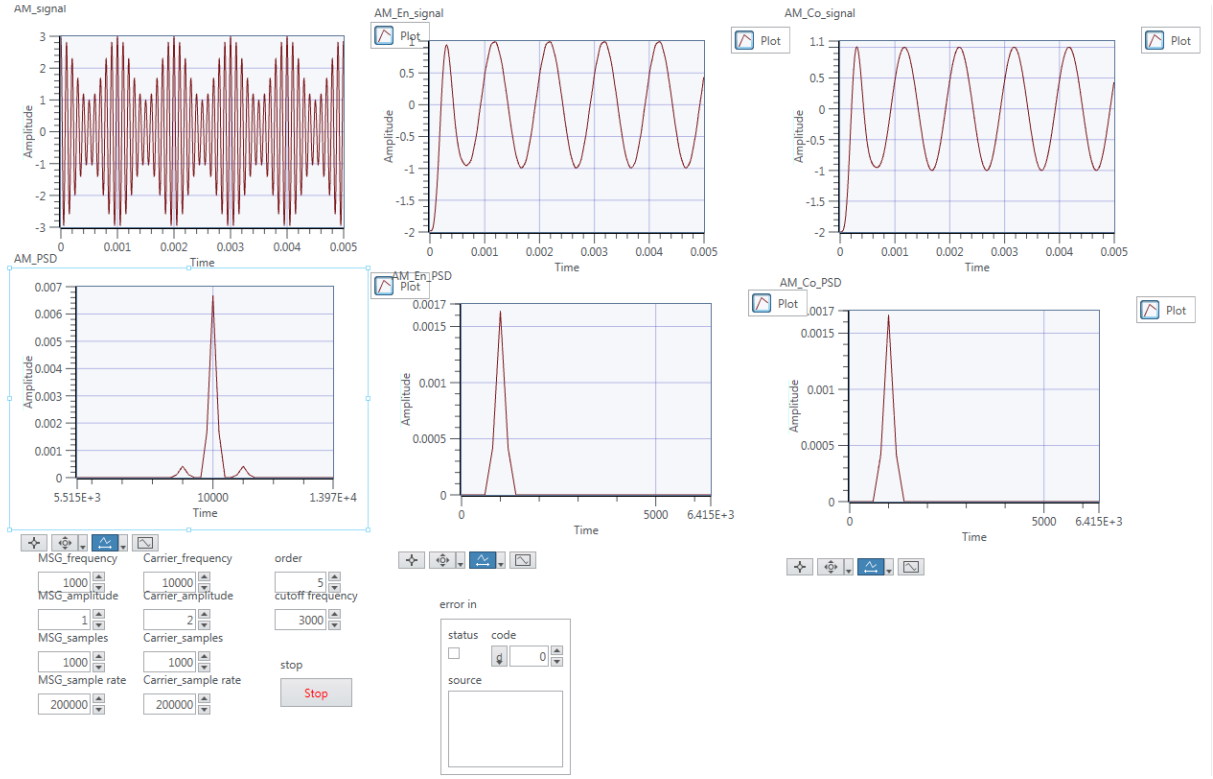
Exercise 3 - AM Simulation

In this exercise we used the VIs created earlier for AM modulation and demodulation and made a top level so we could observe the whole AM process. We used a while loop so that we could observe the effects of changing the parameters without having the stop and restart the process. The diagram is shown below:



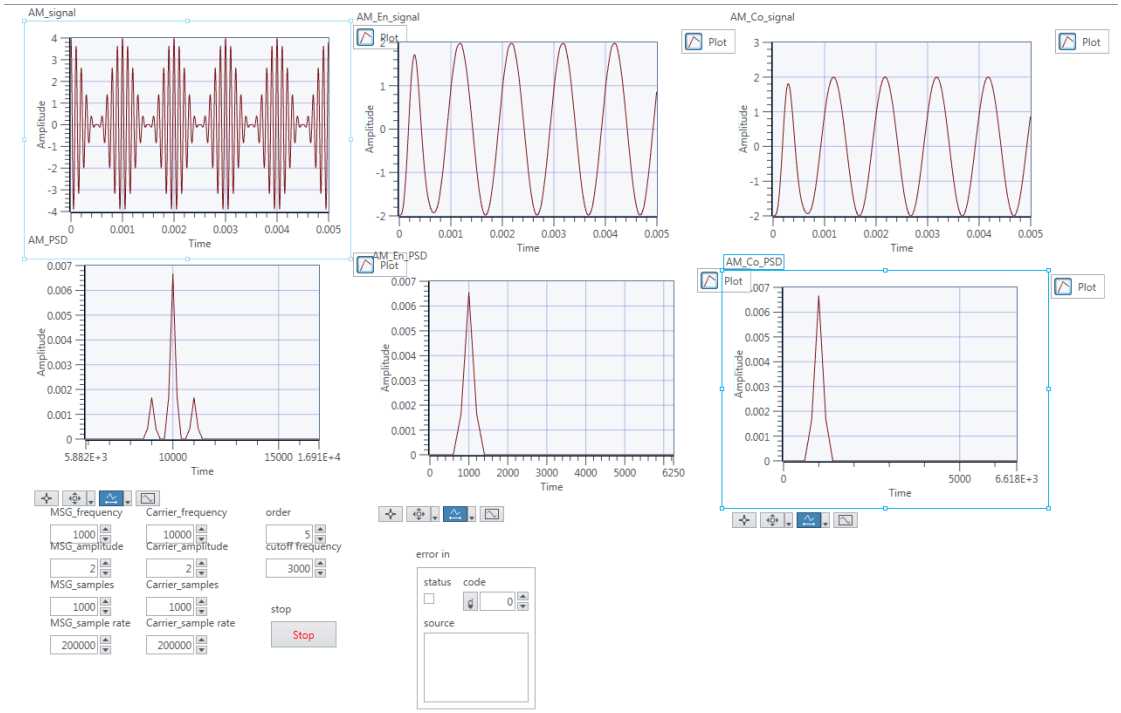
Screenshot 15 - The diagram view for exercise 3.

We then changed the amplitude of the message signal from 1 - 4 in steps of 1 and witnessed the effect on both modulation techniques. First is with a message amplitude of 1:



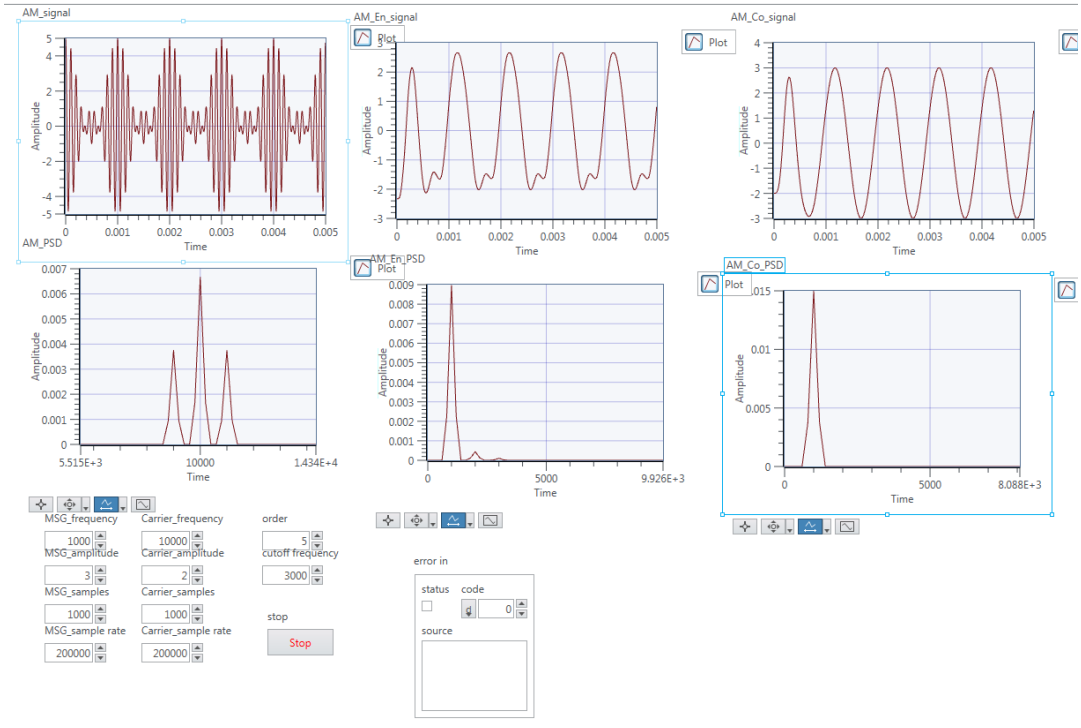
Screenshot 16 - The panel view with message amplitude set to 1.

Here we see the major peak and sidebands in the AM signal, and the corresponding demodulation PSDs. Here we notice no difference between the techniques. Next we tried an amplitude of 2:



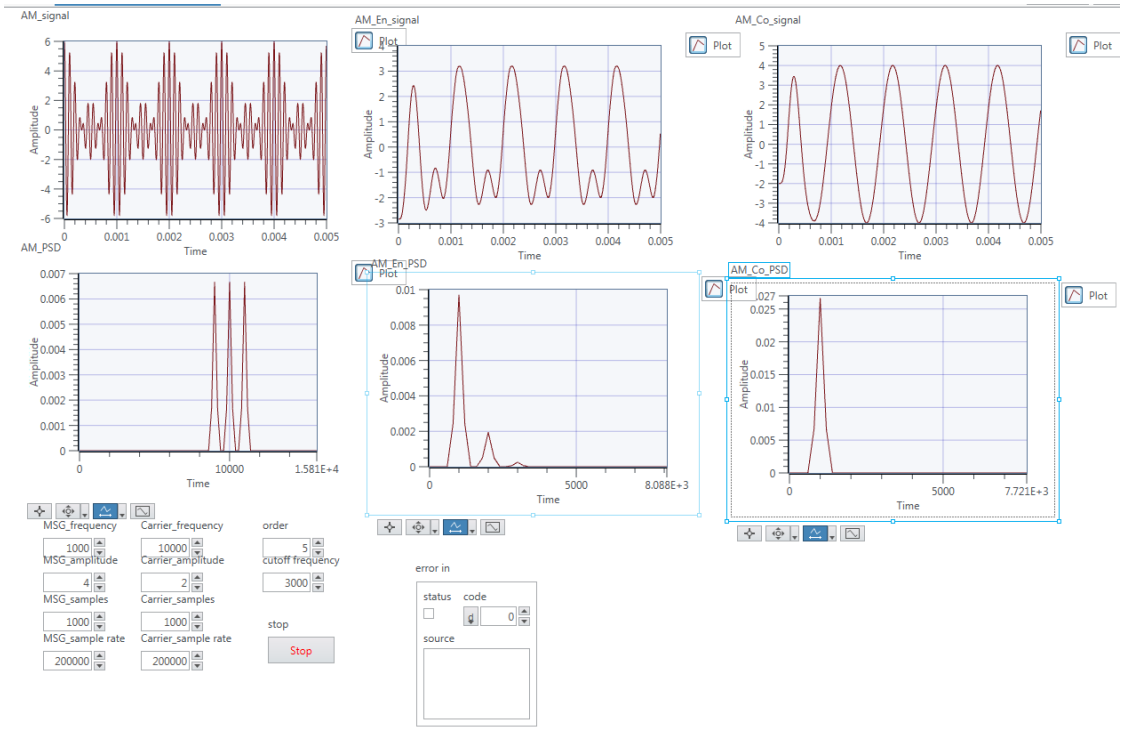
Screenshot 17 - The panel view with message amplitude set to 2.

Here we can see the envelope of the signal is much smaller, and the sidebands on the PSD have a higher amplitude because they are carrying more energy. Both techniques are still able to demodulate effectively producing identical outputs. Notice the PSD of the demodulated signal has a higher amplitude which is because the original message signal has a higher amplitude. Next we tried an amplitude of 3:



Screenshot 18 - The panel view with amplitude = 1.5.

Here we start to see the envelope detection technique break down. This is because we now have a modulation index of 1.5, and you can see in the AM time domain graph that there is no clear envelope. This causes *over modulation* and the signal is distorted. As we can see the envelope detection does no longer produce just the message frequency, but also some minor higher frequencies. Next we tried an amplitude of 4:



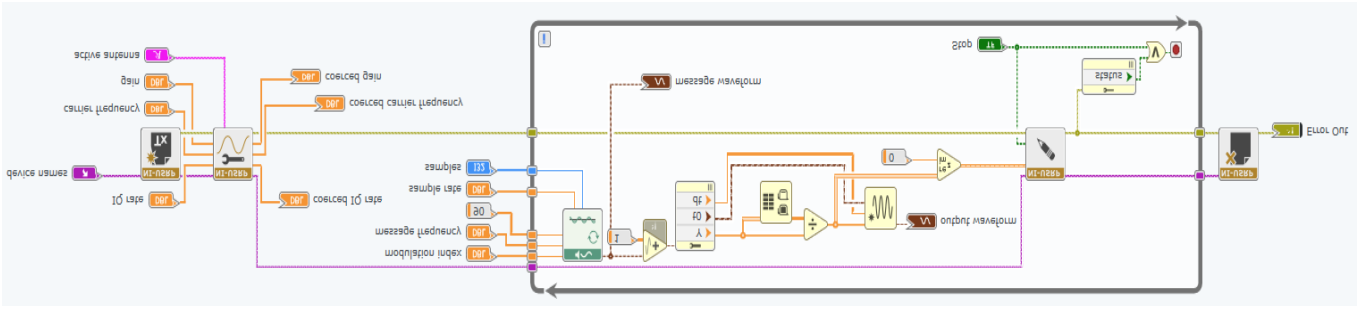
Screenshot 19 - The panel view with amplitude = 4.

Here we see the effect of *over modulation* even more. We now have a modulation index of 2, which is causing the minor peaks in the demodulated signal to have higher amplitude, meaning they are more present in the signal. The demodulated signal is even further away from being like the original message signal. Again we can see that coherent detection has no problem dealing with this change.

From this we can conclude that *Envelope Detection* will only work when the modulation index is between 0 and 1. If the modulation index increases beyond this then the message signal cannot be reconstructed accurately. *Coherent Detection* on the other hand will always work, no matter the modulation index.

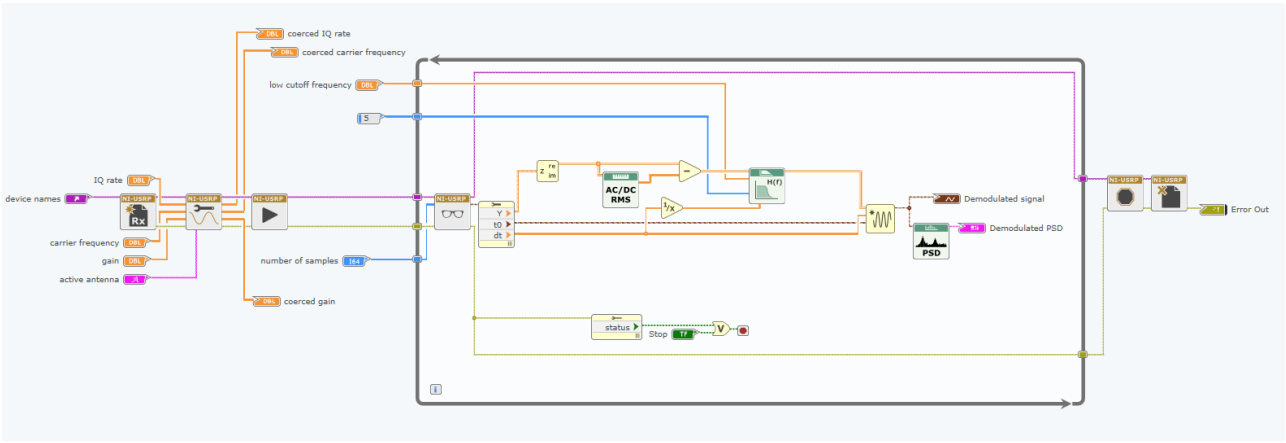
Exercise 4 - AM Communications Via USRP

In this exercise we use the USRP device to actually transmit an AM signal. The USRP modules do all the modulation and demodulation for us, so all we need to do is provide it with complex data. Below is the diagram for the Tx module:



Screenshot 20 - The diagram for the USRP Tx module.

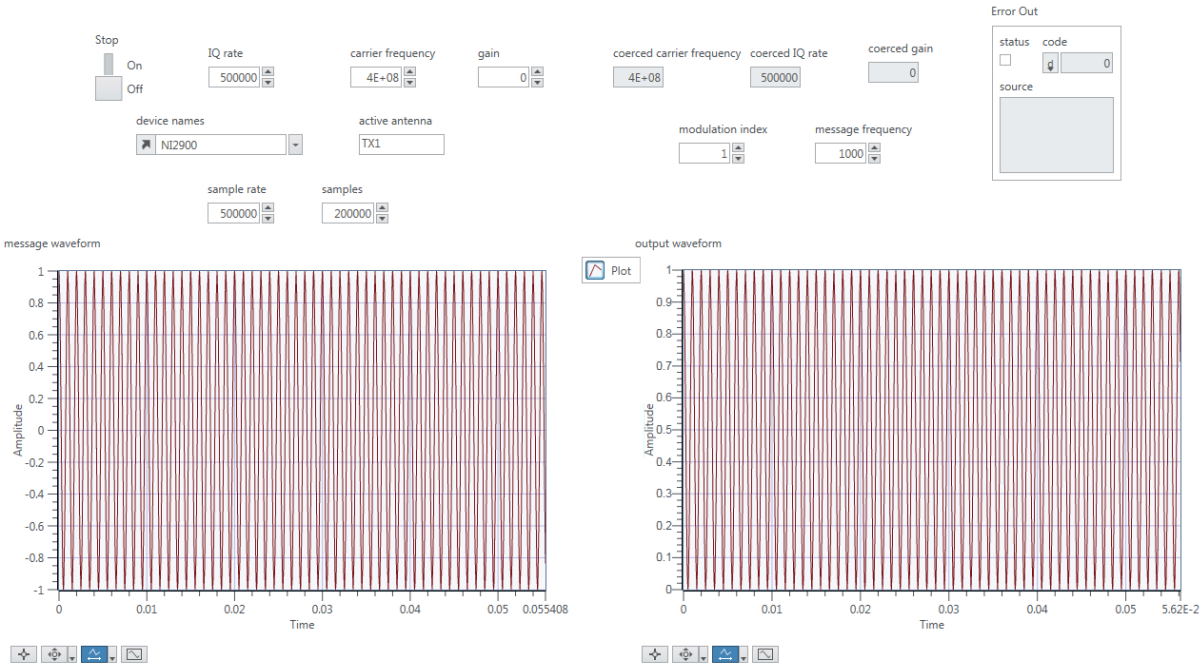
The two modules on the far left open up a USRP Tx session and set the relevant parameters for transmitting. All the main processing is done in a while loop which can be stopped by us on the panel or automatically stopped when the USRP encounters an error. Initially we didn't realise this was the case and became confused when our program stopped execution randomly. Upon further inspection we noticed it was because of these errors which if you read, may help you to stop this happening again (e.g. reducing IQ rate or increasing number of samples). In the while loop we first add one to the signal so that it is all positive. Then we convert it into an array so we can do calculations on the data. The array is normalised by dividing by the highest value and then finally converted to complex by combining this data with a constant of 0. The two modules on the right transmit the data and close the USRP session respectively. Below is a diagram of the receiver:



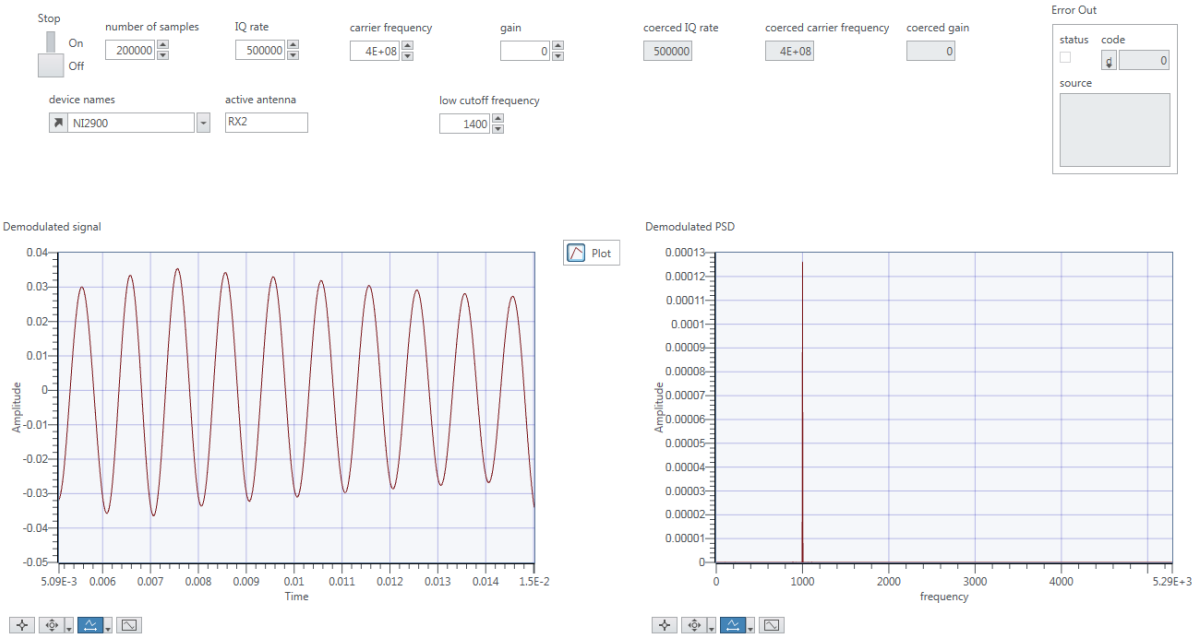
Screenshot 21 - The diagram for the USRP Rx module.

The surrounding modules in this diagram work the same way as in the Tx module. They open the session, set parameters and close the session when needed. Again the main processing is done in a while loop. The module on the left of the while loop takes one block of message samples at a time and allows it to be processed. First we convert this waveform to an array, and then convert the complex array into real and imaginary. We can discard the imaginary part, since the receiver just set this to 0. We then remove the DC offset that was added before transmitting and pass this through a LPF to remove the high frequency carrier. This array data is then converted back into a waveform and displayed in the time and frequency domain.

The corresponding Tx and Rx panels are shown below. Notice we manage to demodulate the message signal perfectly with no other frequencies present.

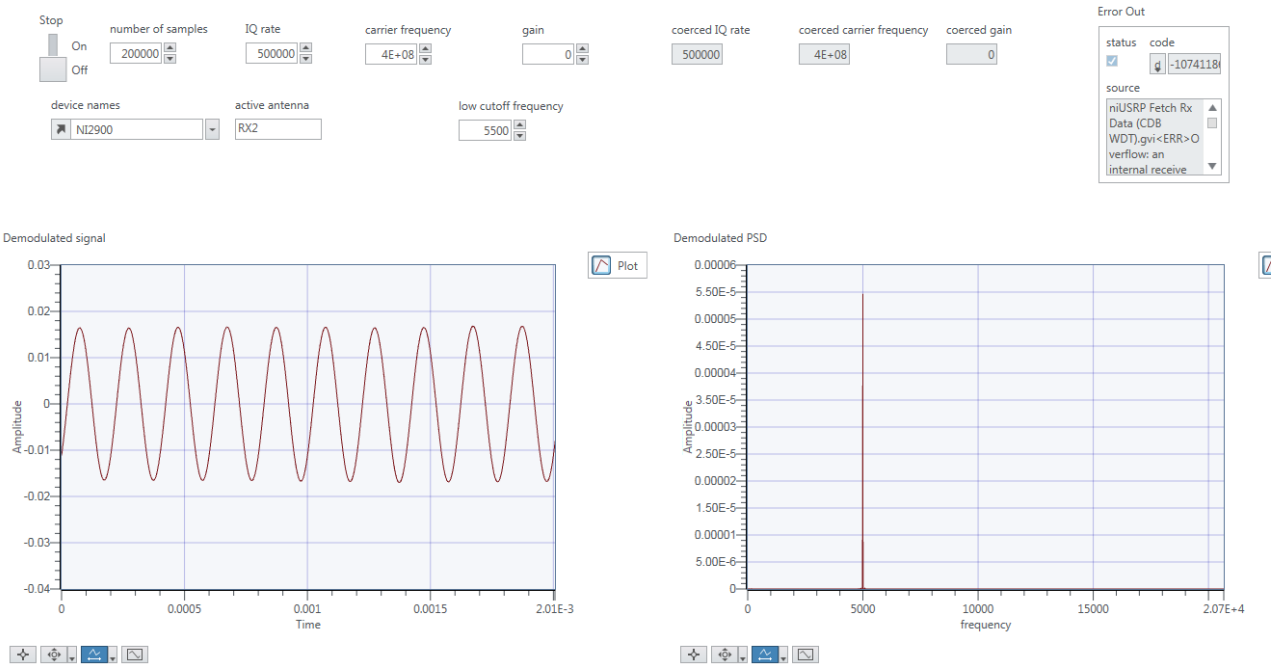


Screenshot 22 - The panel view for the Tx module.



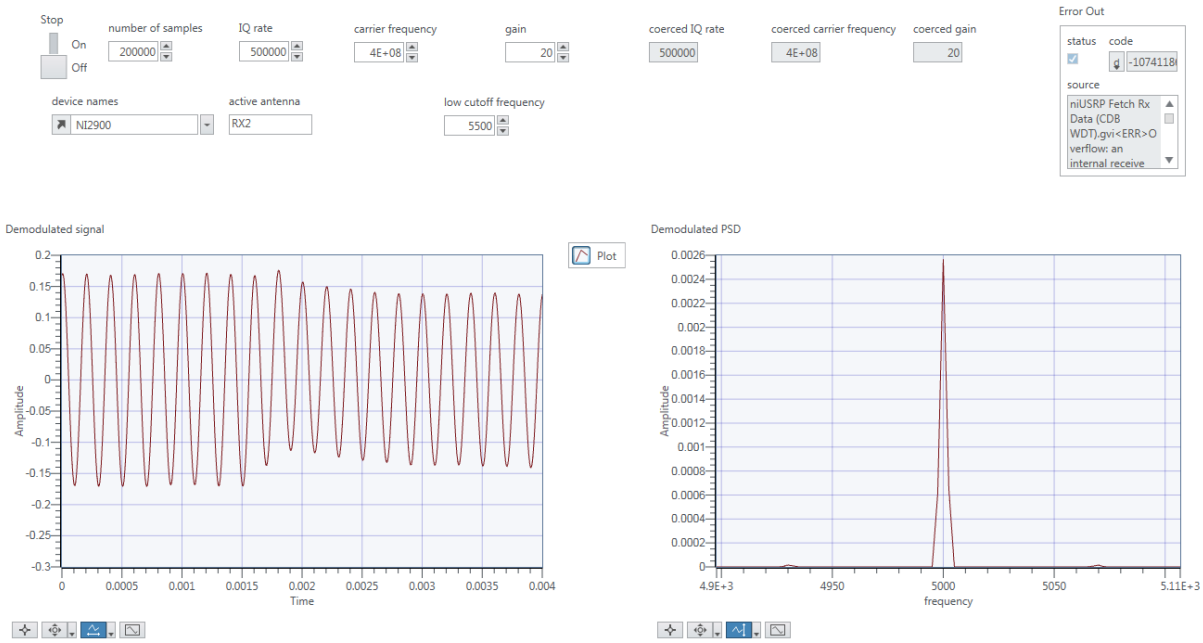
Screenshot 23 - The panel view for the Rx module.

Next we tried to transmit a 5kHz message signal with modulation index 1. We had to change the low cut off frequency so that it did not cut off the 5kHz signal.



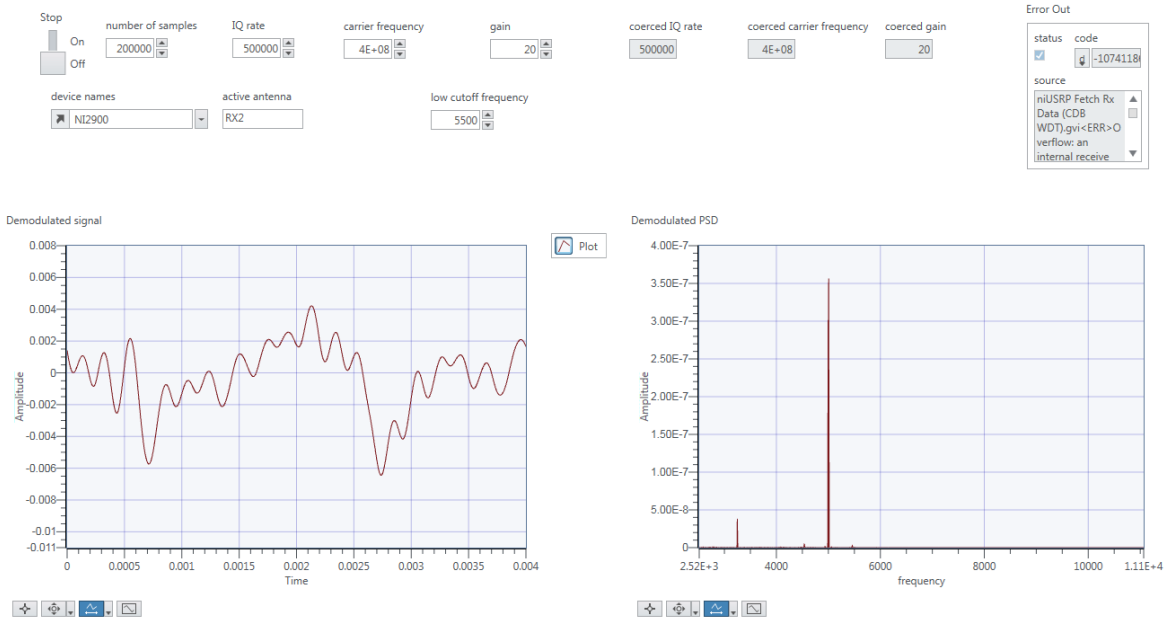
Screenshot 24 - The panel view for transmitting a 5kHz message signal.

Here we notice the signal is picked up perfectly, with little noise. Next we wanted to observe the effect of noise on the signal. To do this we increased the receivers gain to 20dB so that some of the weaker signals would be amplified enough to be seen. We then changed the modulation index and saw how this affects a signal. We started with a modulation index of 1, shown below:



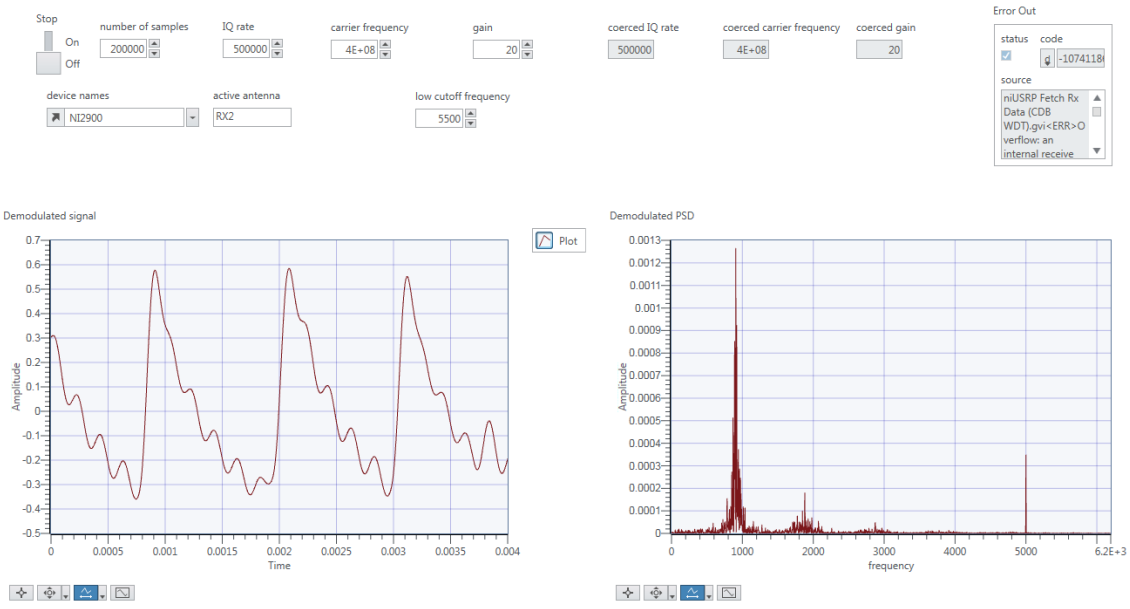
Screenshot 25 - The panel view for a modulation index of 1.

Here we can see an almost perfect demodulated signal. Next we reduced the modulation index to 0.4:



Screenshot 26 - The panel view for a modulation index of 0.4.

Here we can start to see noise affecting the signal. We predominantly pick up the 5kHz signal, but there are also some other minor peaks at other frequencies. If we reduce the modulation index even further, to 0.1, the effect becomes even more drastic:



Screenshot 25 - The panel view for a modulation index of 0.1.

At this point the 5kHz signal peak is now very small compared to the other frequencies we are picking up as noise.

Lab 3

Exercise 1 - FM Modulator

In this exercise we were implementing frequency modulation (FM), which is given by the general equation:

$$g(t) = A_c \cos\left(2\pi f_c t + \theta_m(t)\right), \text{ where } \theta_m(t) = 2\pi\Delta_f \int m(\tau)d\tau,$$

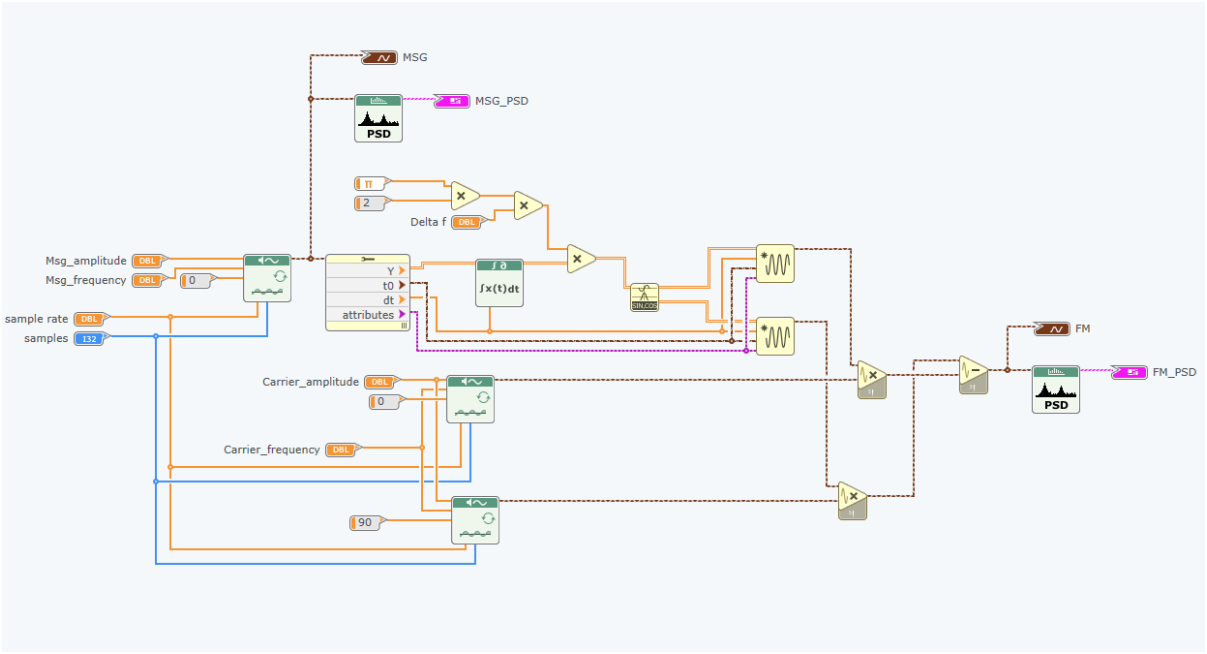
In this lab we were using the equivalent form:

$$s(t) = A_c \cos\left(2\pi f_c t\right)\cos\left(\theta_m(t)\right) - A_c \sin\left(2\pi f_c t\right)\sin\left(\theta_m(t)\right)$$

The modulation index of an FM signal is given by:

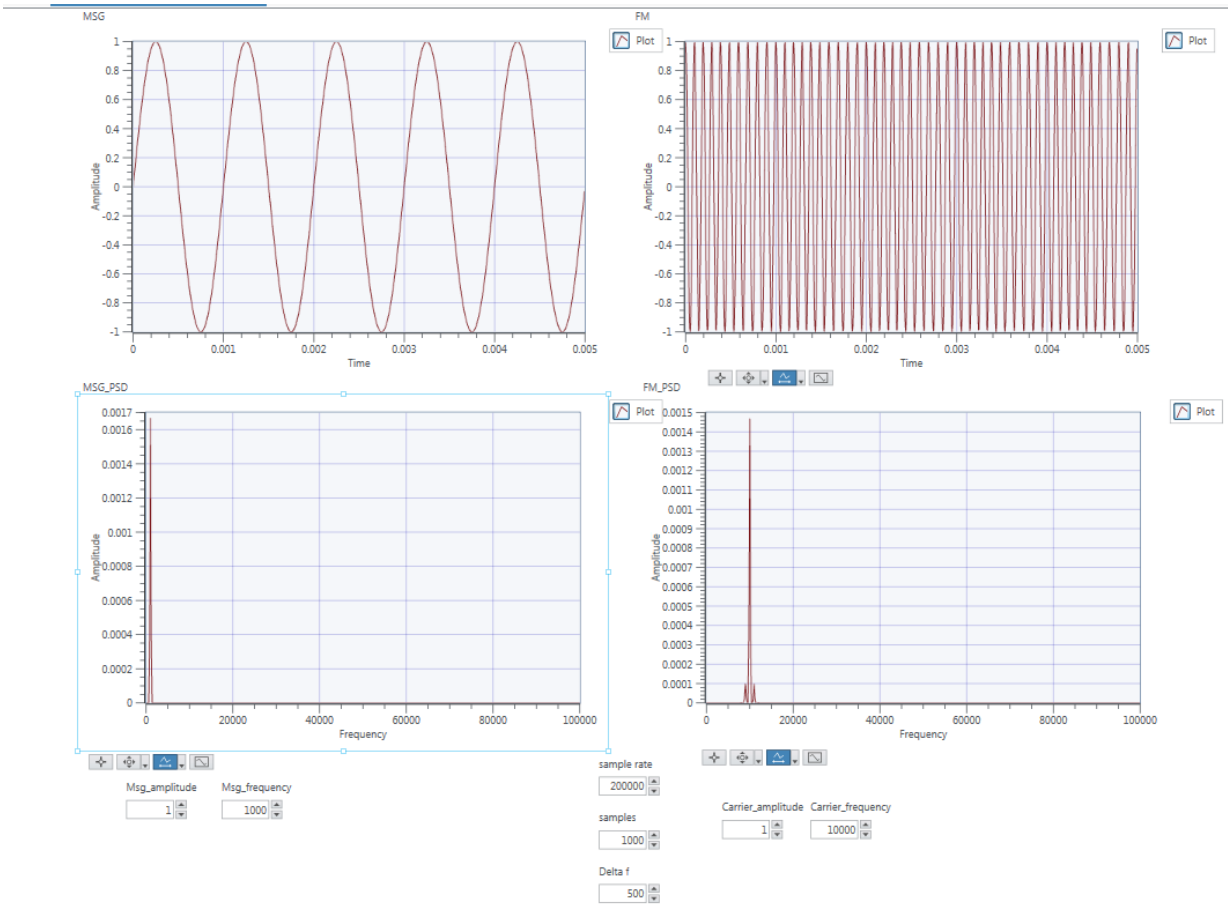
$$\mu = \frac{\Delta_f}{f_m}$$

The finished diagram is below:



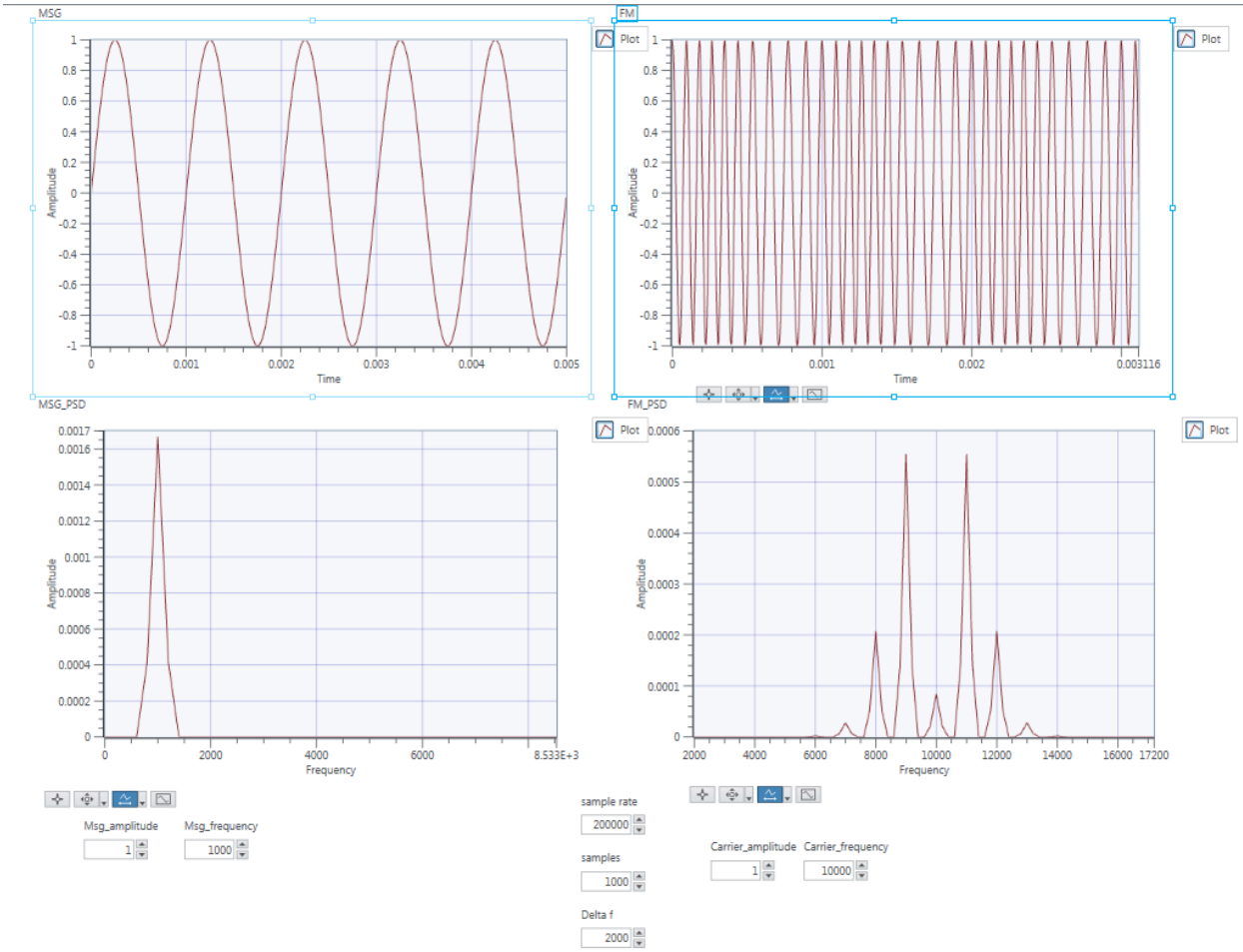
Screenshot 1 - Diagram view for exercise 1.

In this VI we converted the message waveform to an array so we could integrate it and add in the delta f part to create theta(t). We also used a sine/cosine function to get cos(theta(t)) and sin(theta(t)), which could then be multiplied by the carrier signal. We used a message signal of 1kHz and carrier of 10kHz, with delta f set to 500 initially. The results are shown below:



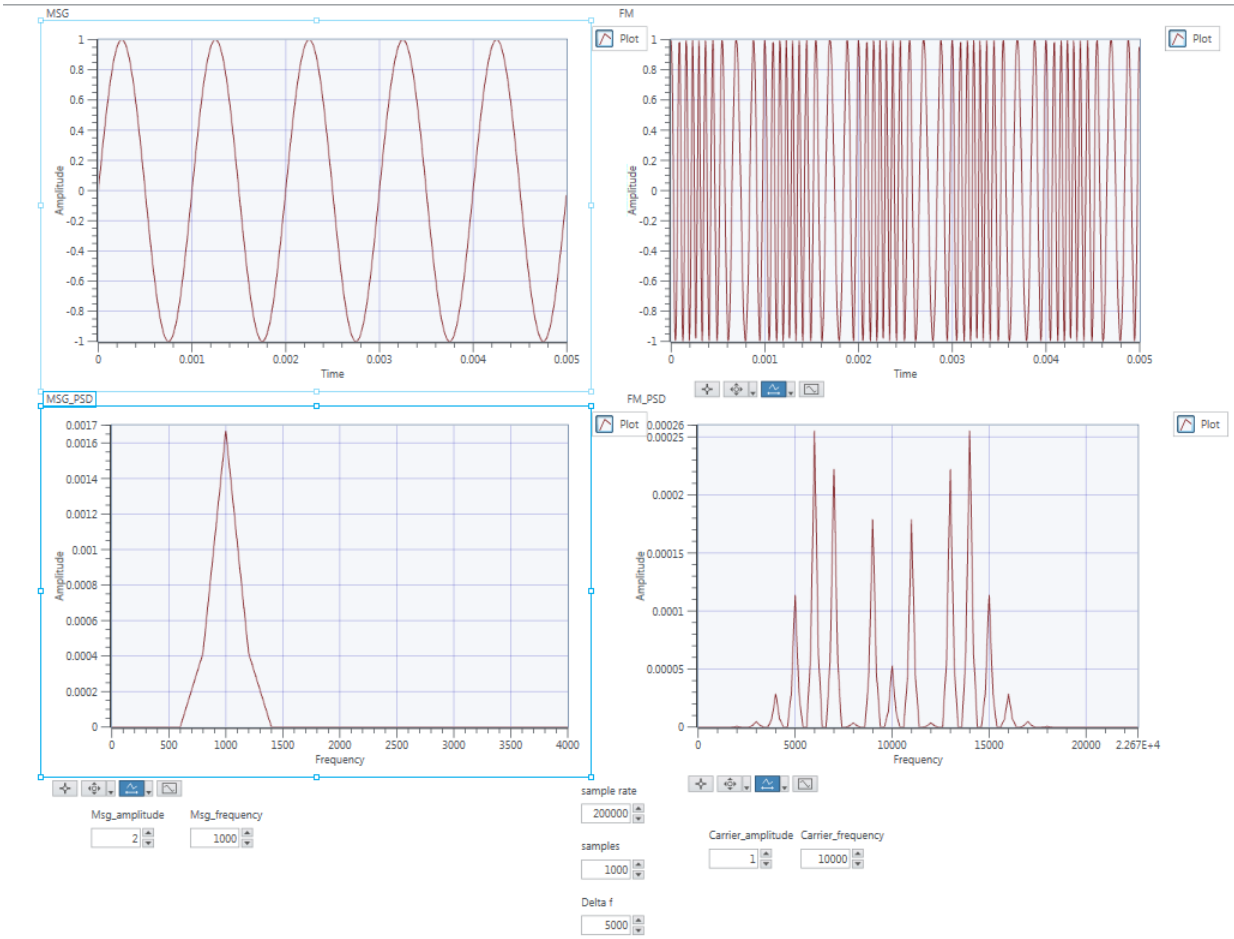
Screenshot 2 - Panel view with delta f = 500.

Here because delta f is very small, it is difficult to see the frequency variation in the time domain plots. However, we can clearly see in the PSD that there is a major peak at 10kHz with sidebands at 9kHz and 11kHz. This looks similar to AM, but with the lower sideband out of phase by 180 degrees. The modulation index in this case is 0.5. Next we tried a value for delta f of 2000:



Screenshot 3 - Panel view with delta f = 2000.

Here it is slightly easier to see the frequency variation in the time domain signal. Now we have a modulation index of 2, and as this increases we see more sidebands at 8kHz, 9kHz, 11kHz and 12kHz, each spaced 1kHz apart. The magnitude of the peak at 10kHz has also dramatically been reduced. Next we changed delta f to 5000:

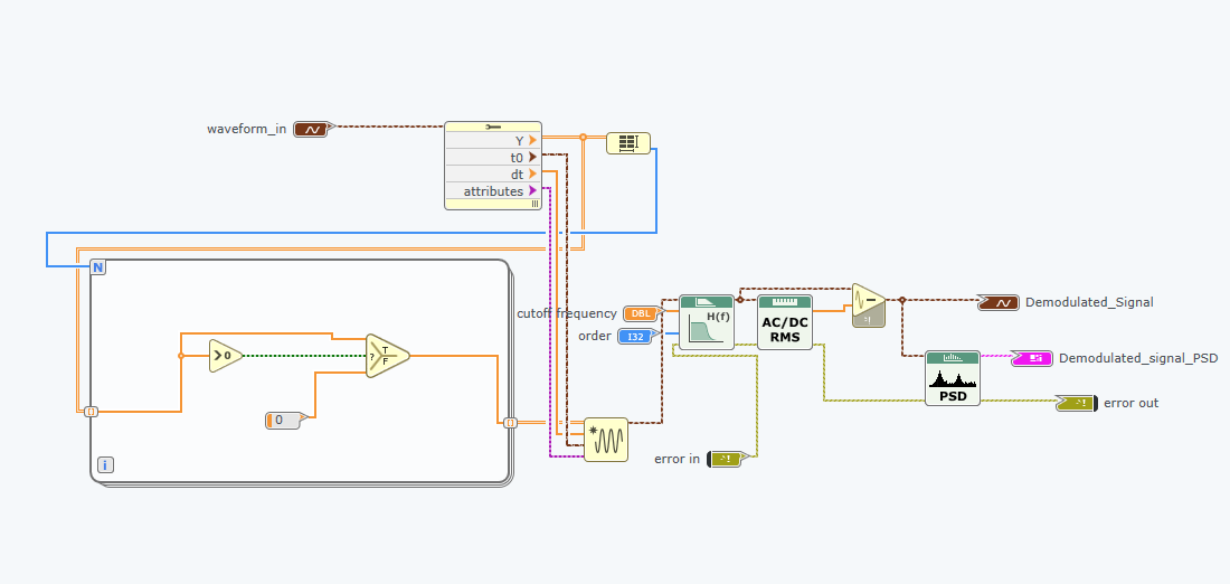


Screenshot 4 - Panel view with delta f = 5000.

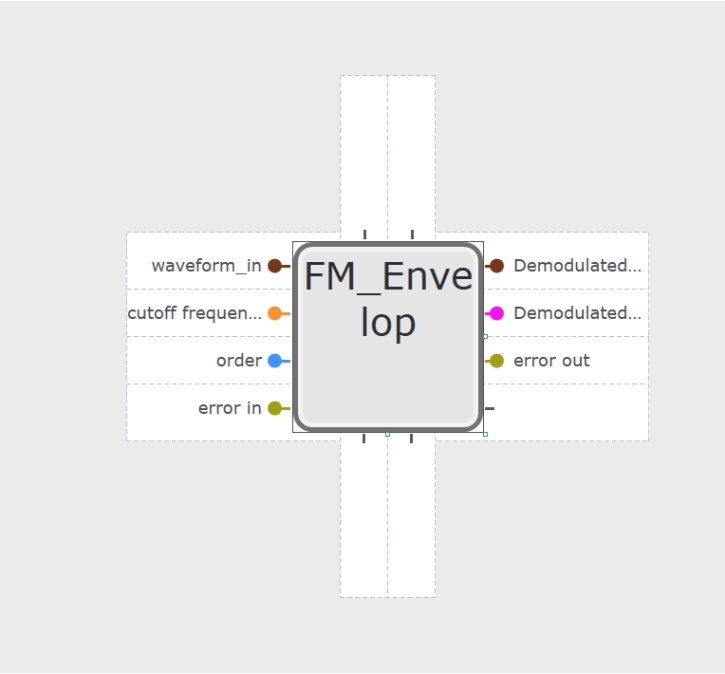
Now the frequency variation is really clear in the time domain. The PSD of the signal is also interesting, with even more sidebands at frequencies spaced 1kHz apart from ~3kHz - 18kHz. We can see that as we increase delta f, the bandwidth of the signal increases (Carson's Rule).

Exercise 2 - FM Demodulator

In this exercise we built an FM demodulator that relied on the envelope detector previously built. In the previous lab we had built an envelope detector that took in the various parameters of the signal and then demodulated it. This time we wanted a subVI which would take in a waveform instead. We went back and redesigned the envelope detector to allow waveform input.

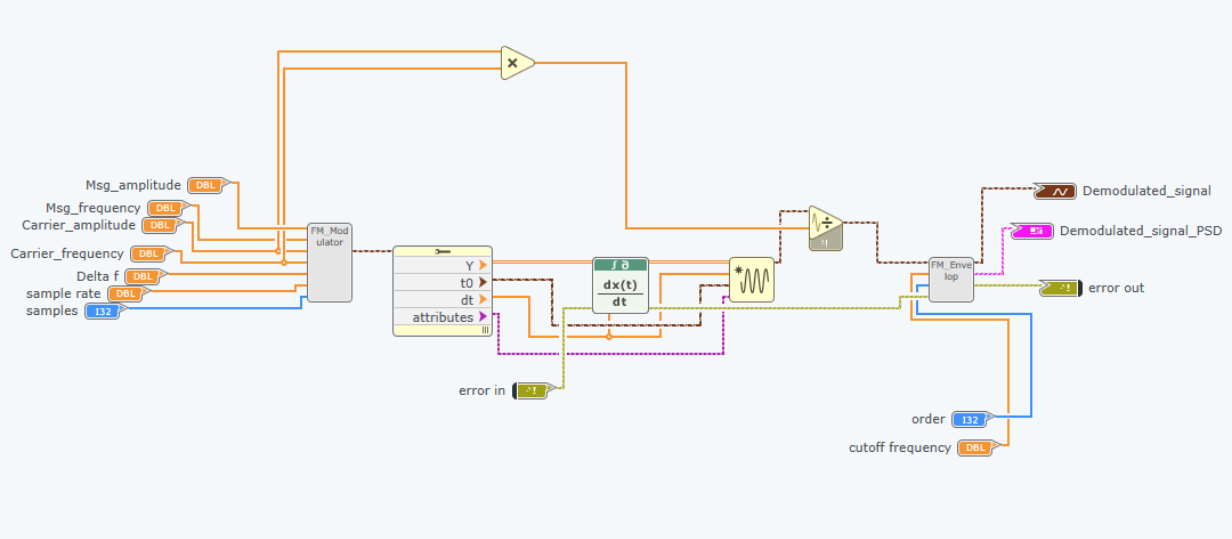


Screenshot 5 - Diagram view of the modified envelope detector.



Screenshot 6 - Icon for the sub VI, with modified input.

This then allowed us to add this to our design. The demodulator works by first converting the waveform to an array so we can differentiate it. Once the waveform has been differentiated we convert the array back into a waveform and pass it through an envelope detector.



Screenshot 7 – Diagram view.

Mathematical Proof

First recall the equation for FM:

$$\boldsymbol{g(t) = A_c \cos\left(2\pi f_c t + \theta_m(t)\right)}, \text{ where } \boldsymbol{\theta_m(t) = 2\pi\Delta_f \int m(\tau)d\tau},$$

Differentiating this signal gives us:

$$g(t) = A_c \cos(2\pi f_c t + \theta_m(t))$$

$$g'(t) = -A_c \sin(2\pi f_c t + \theta_m(t)) \times (2\pi f_c + \theta_m'(t))$$

Then theta(t) differentiated is:

$$\theta_m'(t) = 2\pi\Delta f \; m(t)$$

Giving the full equation:

$$g'(t) = -A_c \sin(2\pi f_c t + \theta_m(t)) \times (2\pi f_c + 2\pi\Delta f \; m(t))$$

$$g'(t) = -2\pi A_c [f_c + \Delta f \; m(t)] \sin(2\pi f_c t + \theta_m(t))$$

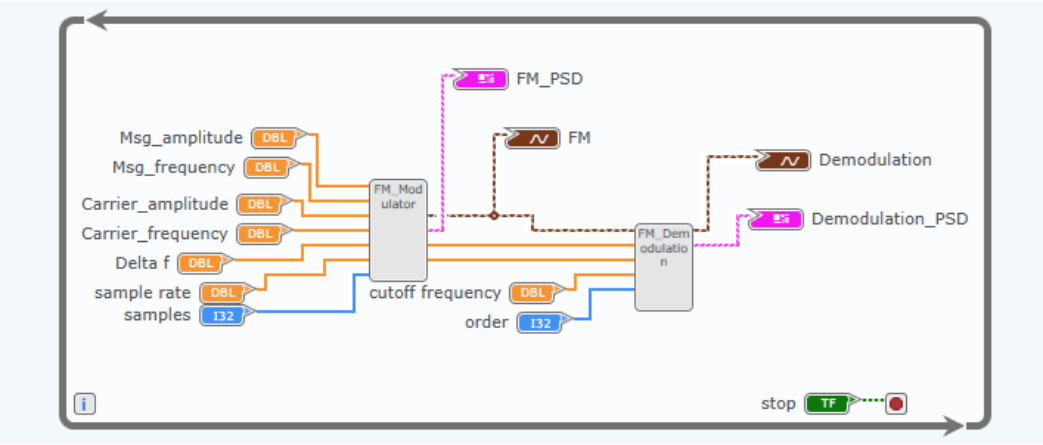
This is now an AM and FM modulated signal. Therefore we can use envelope detection to get an output waveform proportional to the message signal:

$$= -2\pi A_c [f_c + \Delta f \; m(t)]$$

We can remove the DC component and adjust the amplitude to get the desired message signal.

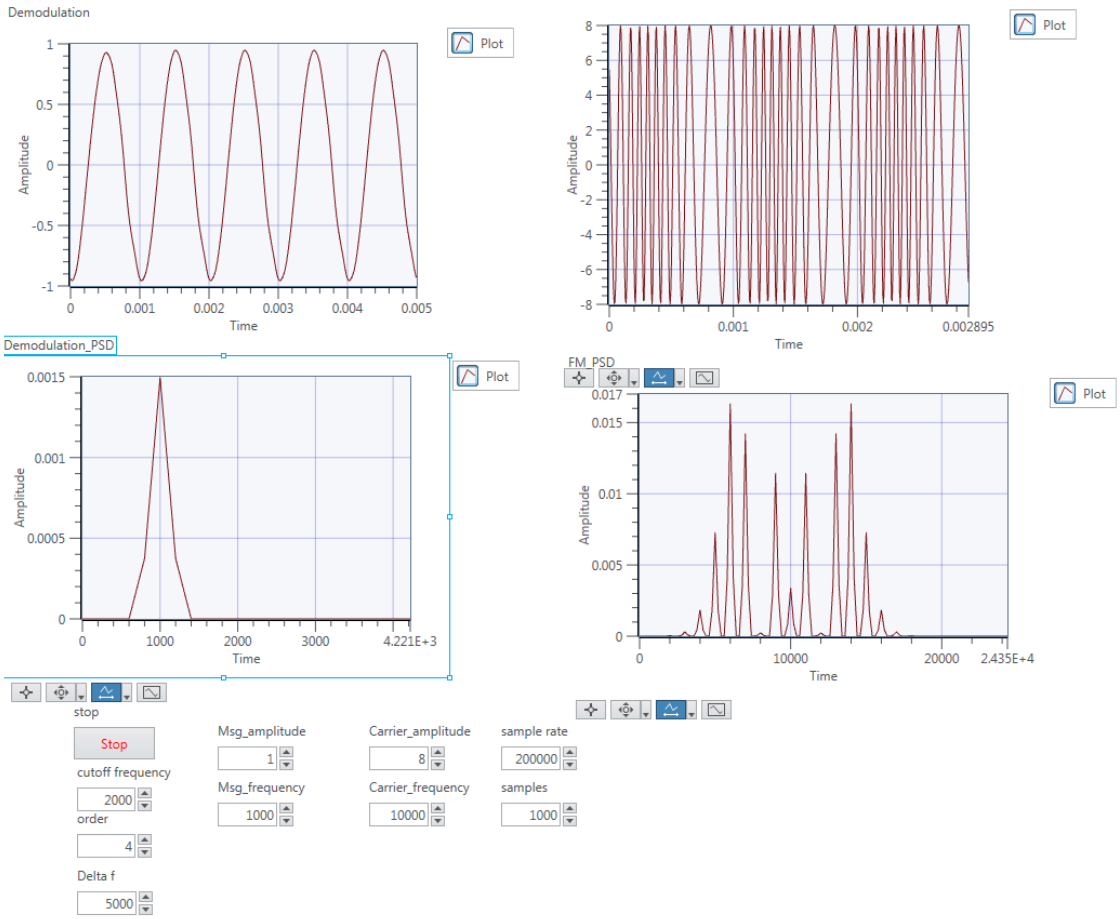
Exercise 3 - FM Simulation

In this exercise we used the modules we had designed previously and put them into one top level design so that we could witness the whole frequency modulation process.



Screenshot 8 - The diagram view for exercise 3.

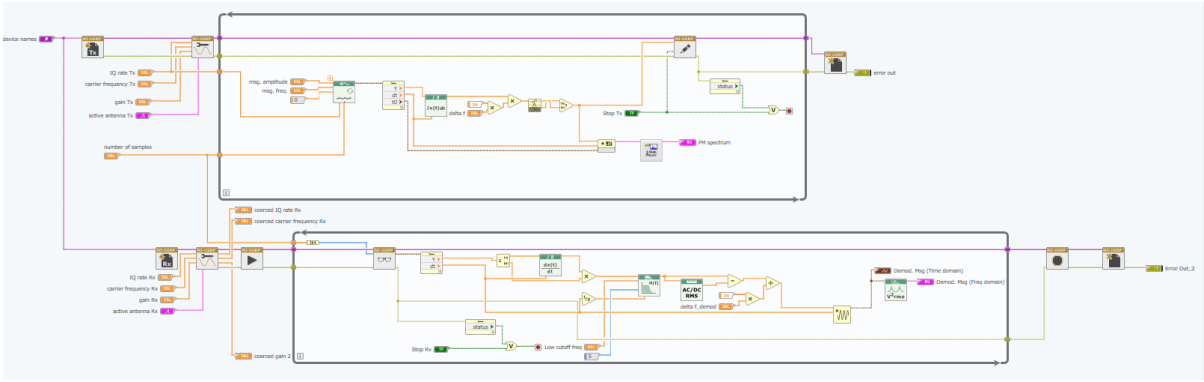
Below is the panel view, showing the time domain and frequency domain representations of the FM signal and demodulated signal.



Screenshot 9 - The panel view for exercise 3.

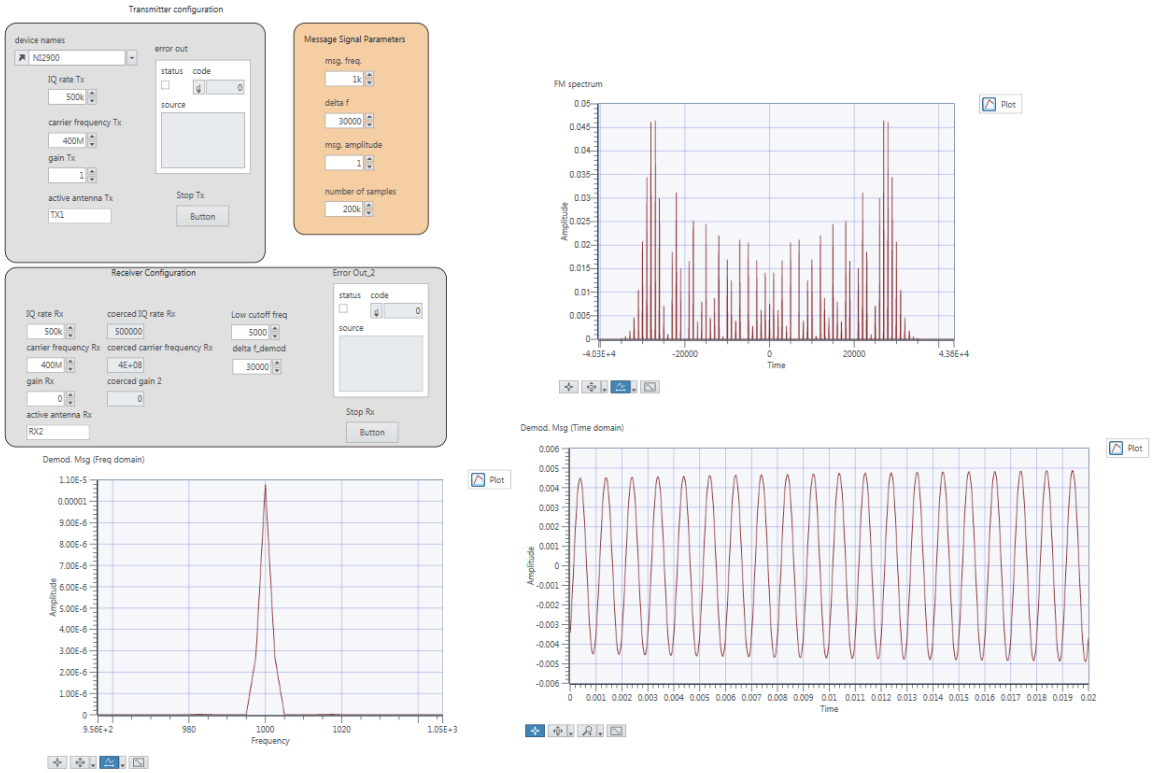
Exercise 4 - FM Communications Via USRP

In this exercise we used the template FM-TxRx.gvi to build an FM transmitter and receiver for the USRP device.



Screenshot 10 - The diagram view for exercise 4.

If we compare this diagram to what we have done in the previous exercises we can see it works exactly the same, except we convert the values to complex to work with the USRP. The panel view with graphs is shown below.

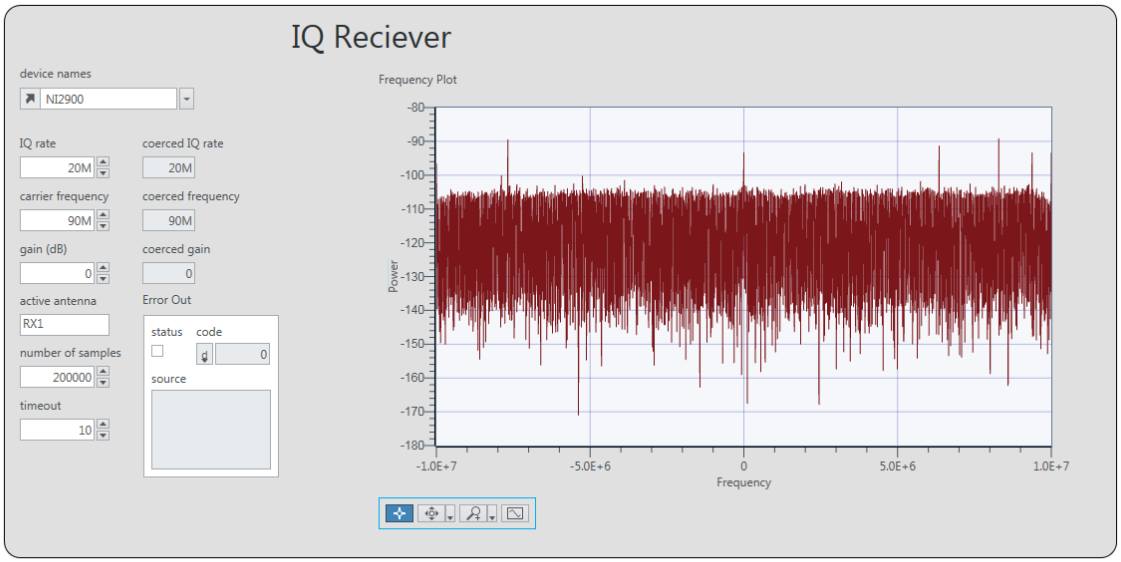


Screenshot 11 - The panel view for exercise 4.

Here we can easily see the bandwidth of the signal, which is approximately 35000Hz x 2 = 70kHz

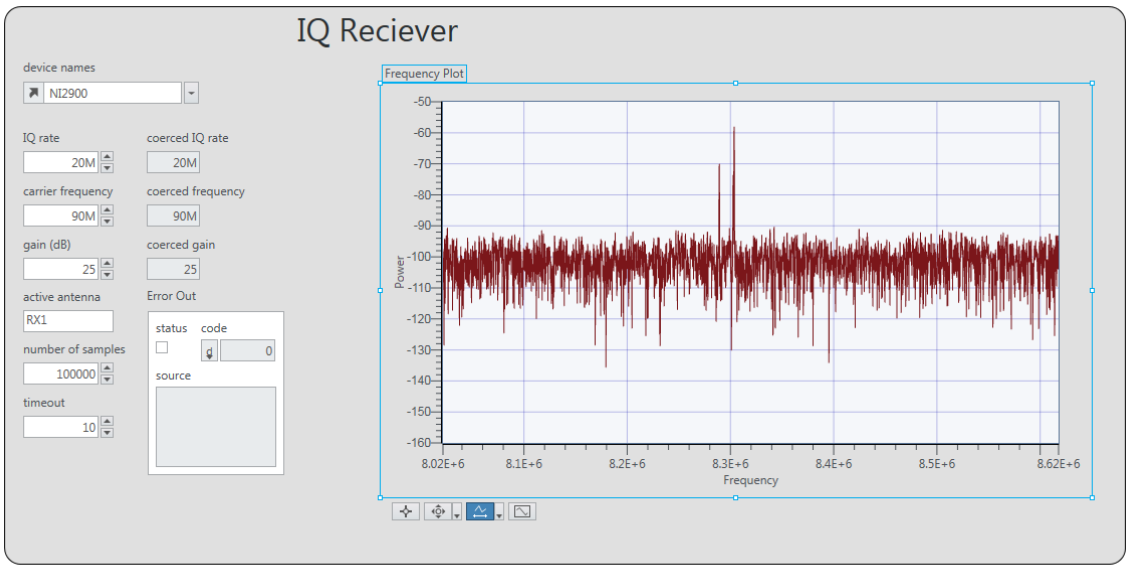
Exercise 5 - Demodulating and Listening to FM Radio Using the USRP

In this exercise we are using the USRP to tune into local radio stations. The first part involves us using the *Find Radio Station.gvi* to get a frequency spectrum including local radio stations. This is shown below:



Screenshot 12 - The panel view showing local radio stations.

Here we can see that each peak is a radio station. We zoomed into one of these peaks below:



Screenshot 13 - Zoomed in section, showing a local radio station.

Here we can see one peak 8.3MHz. We tried to use the *FM Music.gvi* module to tune into this radio station but we only heard a lot of noise. Instead we used the given frequency of 98.8Mhz for Radio 1. This actually worked for us, and we were able to hear Drake being played on the radio despite it being very noisy.

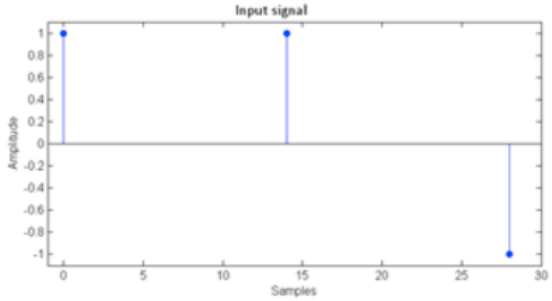
Lab 4

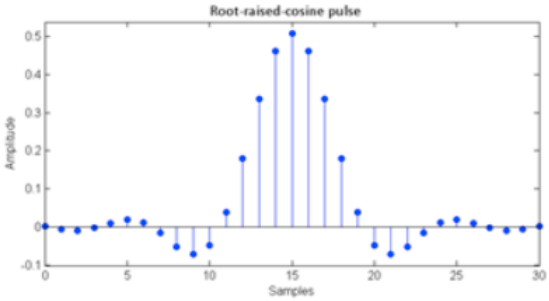
Exercise 1 – BPSK Transmitter

For this exercise we would be implementing binary phase shift keying, which encodes information in the wave using the phase. An equation for the wave is displayed below:

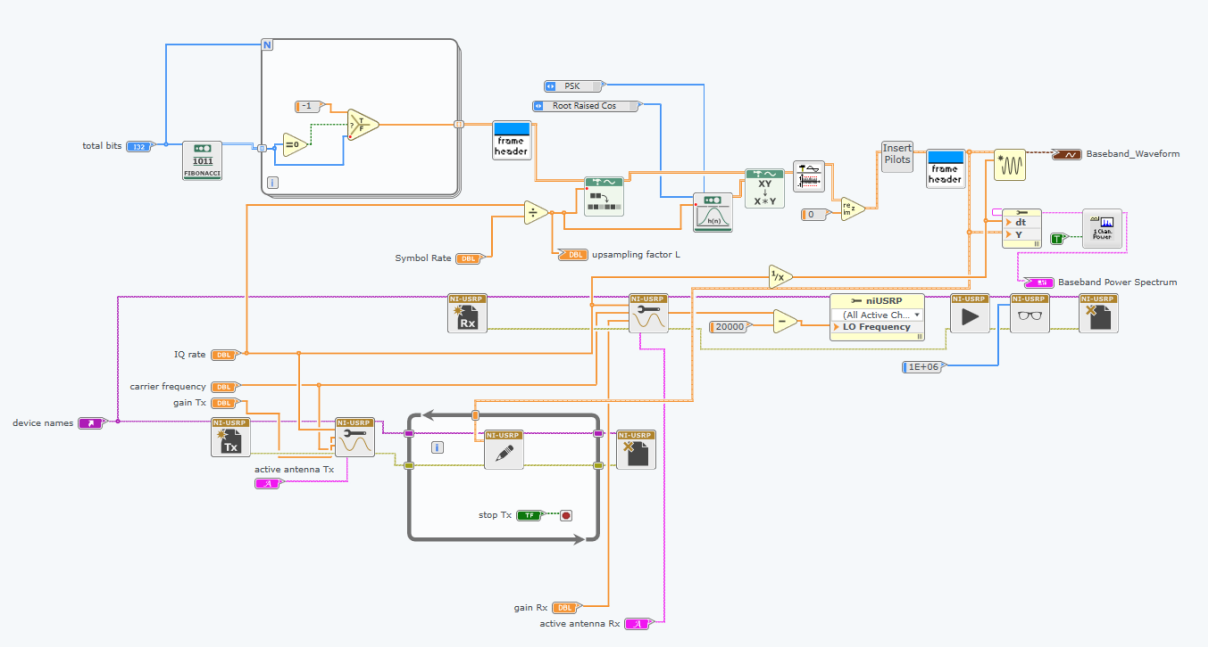
$$Ag_{TX}(t)\cos(2\pi f_c t + \theta).$$

We generated some random bits which were then translated into symbols with 1 being represented by a +1 and 0 by a -1. The **addFrameHeader** function provided will add a specific 26-bit sequence to the samples that the receiver will look for to know when the sequence starts. We then need to replace these symbols with pulses so we use the **up-sampling function** which adds L-1 zeroes after the symbol. We then have a **generate filter co-efficients** function so we can create a filter that can change our pulses to root-raised cosines. By **convolving** the filter co-efficients with the symbol sequence, we effectively apply this filter to get symbols represented by a root-raised cosine pulse. This pulse has a rapid spectral roll-off so that the signal will not cause interference with other signals at neighbouring frequencies. The output can be seen below:





We then use the **Quick Scale 1D** function to normalise our output to a maximum amplitude of 1. We then use the **Insert Pilots** function which adds a sequence of symbols that the receiver looks for to estimate the phase difference and correct it at the receiver. Finally, we convert these symbols to complex so that the USRP can transmit them, and send it to the receiver. The resulting diagram for this exercise can be viewed below:

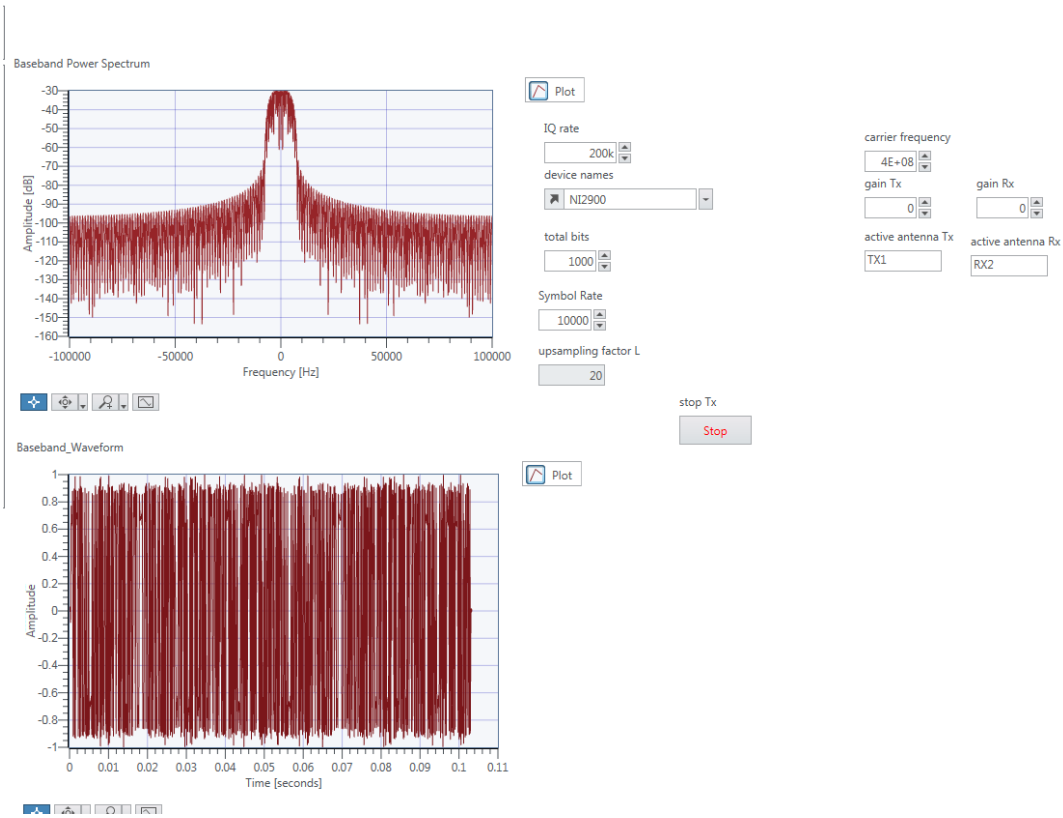


Screenshot 1 – Diagram view for ex1.

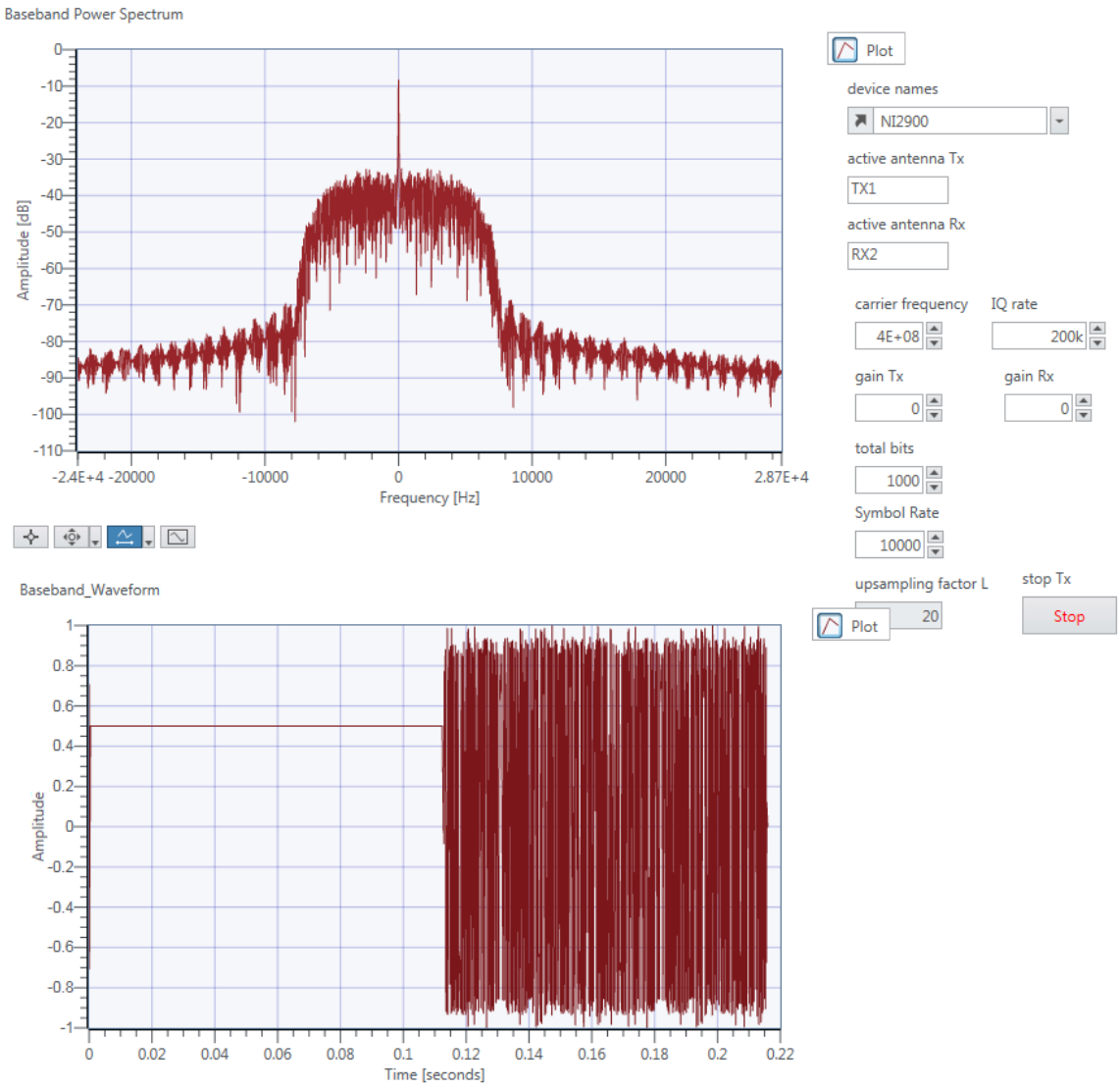
Given that the equation is as follows, we can work out the value for the sampling factor, L:

L = T / Tx = (1 / symbol rate) / (1 / IQ rate) = IQ rate / symbol rate

200000 / 10000 = 20

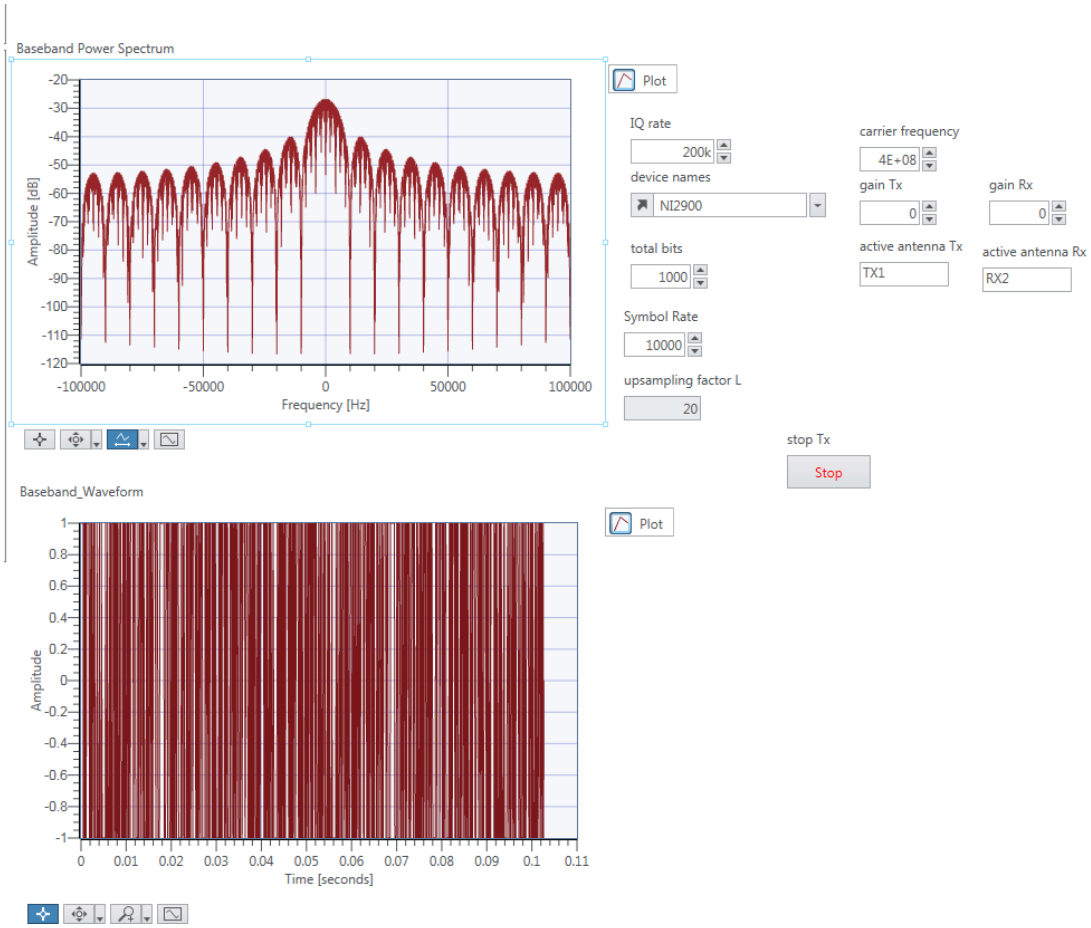


Screenshot 2 – Panel view for ex1.

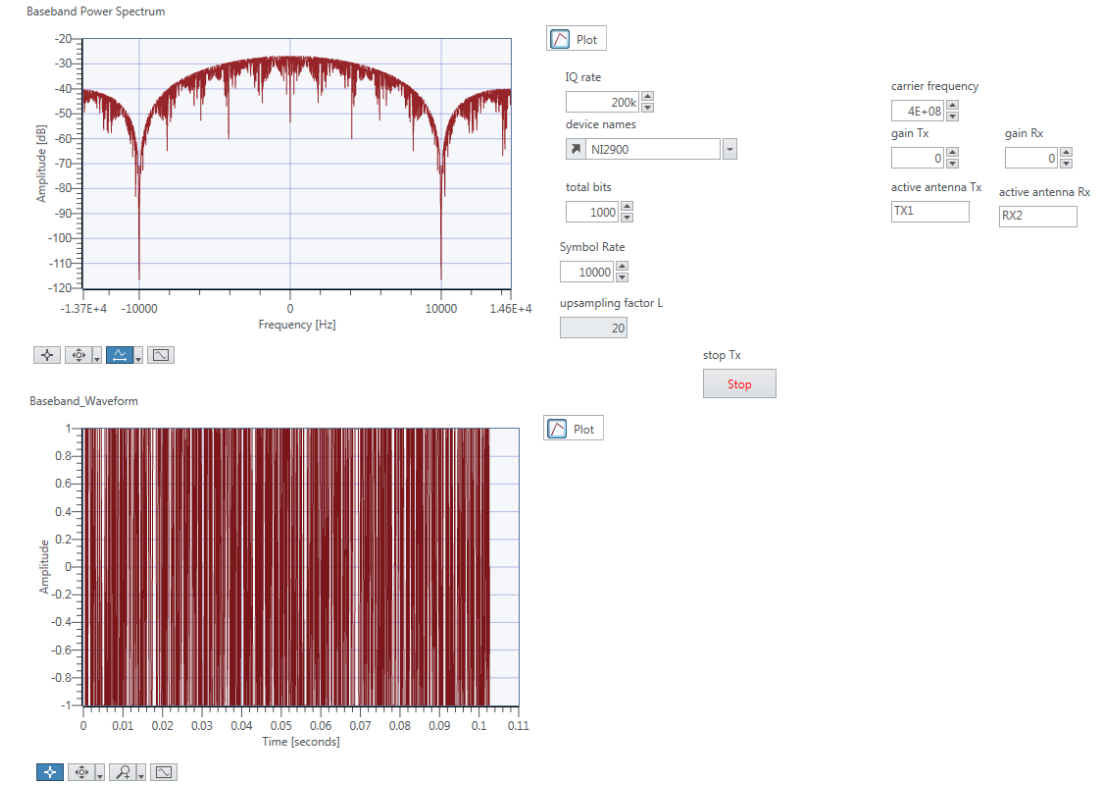


Screenshot 3 – Panel view for ex1 zoomed.

Here we can see the main lobe in the centre with a bandwidth of approximately 8500 / 9000Hz. The spectral roll-off is large so it is hard to make out the other lobes. Next we changed the filter co-efficients to none to create rectangular pulses and we ran the transmitter again.



Screenshot 4 – Panel view with ‘none’ as filter co-efficients.



Screenshot 5 – Panel view with ‘none’ as filter co-efficients zoomed.

Here we can see that the spectral roll-off for the ‘none’ filter co-efficients is much less than for the ‘root-raised cosine’ filter. We can also see the bandwidth of the main lobe is exactly 10000Hz.

Exercise 2 – BPSK Receiver

The BPSK signal arriving at the receiver has the form of a pulse train, each pulse given by the equation:

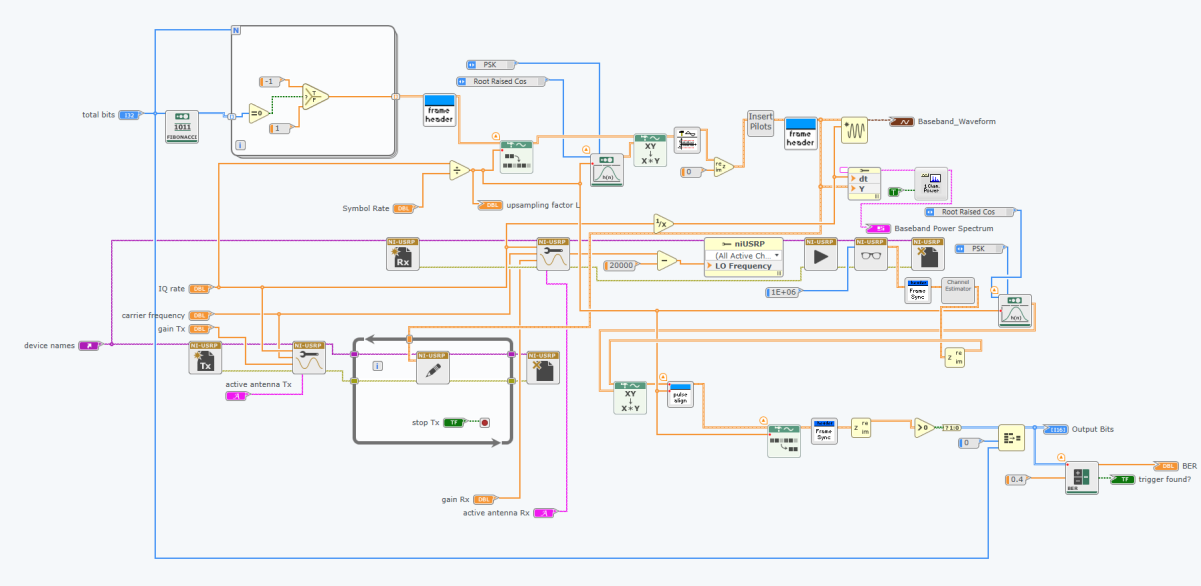
$$r(t) = \pm D g_{TX}(t) \cos(2\pi f_c t + \varphi),$$

Since most of the work done to demodulate this signal is done in the USRP device, the output data we receive is then given by:

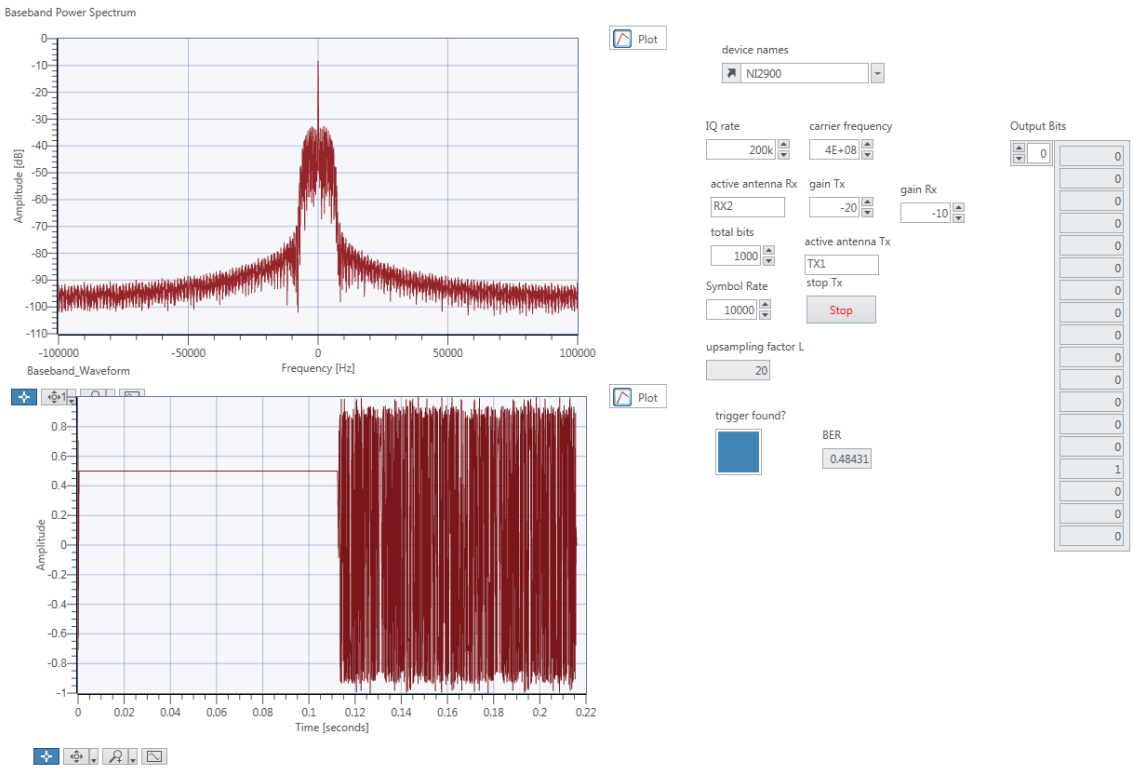
$$\tilde{r}[n] = \pm \frac{D}{2} g_{TX}[n] e^{j\varphi}$$

This is approximately the same as the message signal, except with an amplitude and phase difference.

First we use the **FrameSync** function to find the start of the sequence and discard everything else. We then use the **ChannelEstimator** function to eliminate the phase offset caused by the drift and skew of the oscillators at the Rx and Tx. We then convert this result to real values since the USRP works with complex data. We then use a **Generate Filter Coefficients** function to create the co-efficients for a filter, using the same parameters as in the transmitter (Note the number of samples, M is equal to L from the transmitter). We then use the **convolution** function to apply this filter to the incoming data, which gives optimum performance in the presence of noise. We then use the **PulseAlign** function which tries to align the pulses which may be delayed due to propagation delays or distortion. We then use the **Decimate** function to sample this waveform. We use another **FrameSync** function and extract the real part of the output. We then determine if this represents a 1 or a 0 by using the **Greater Than 0** function to determine which symbol it is (remember a +1 represents a 1 and a -1 represents a 0). The results of this comparison are stored in an array. We put this array into the **MT Calculate** function to determine the BER (bit error rate) which tells us how much error there is in our data. The diagram and panel views for this exercise is shown below. Note we observed a BER of ~0.48.



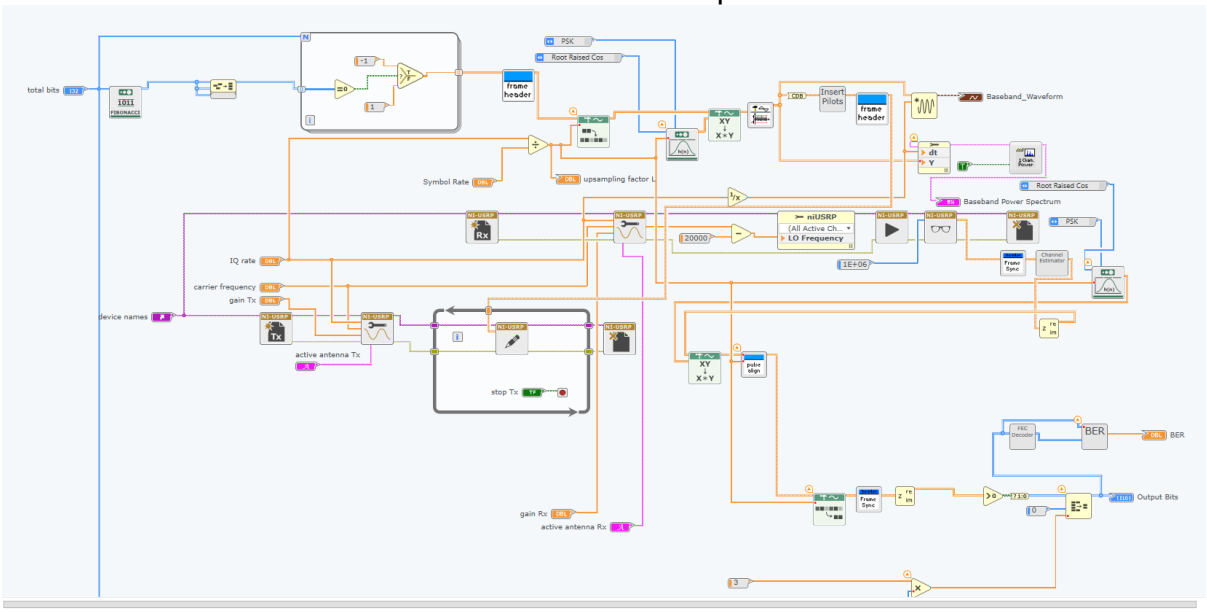
Screenshot 6 – The diagram view for exercise 2 (including transmitter).



Screenshot 7 – The panel view for exercise 2.

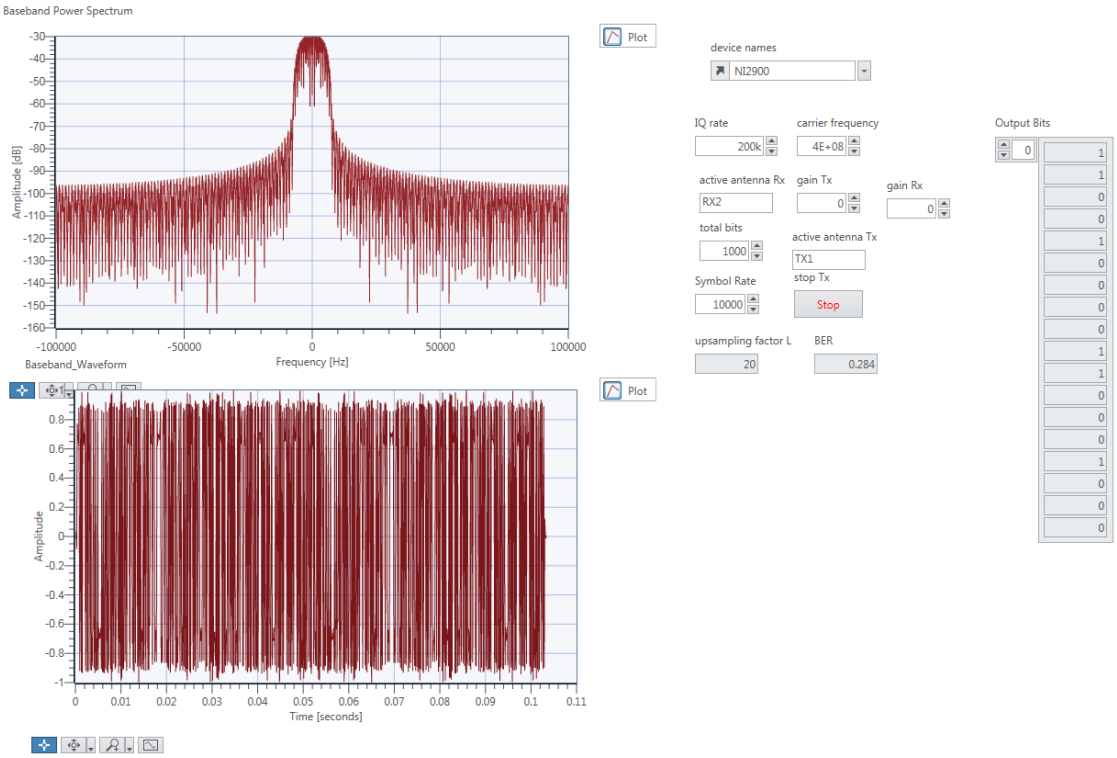
Exercise 3 – Error Correction Coding

In this exercise we perform error correcting by transmitting the same bit 3 times. We can then work out from these 3 bits what the most probable transmitted bit was.

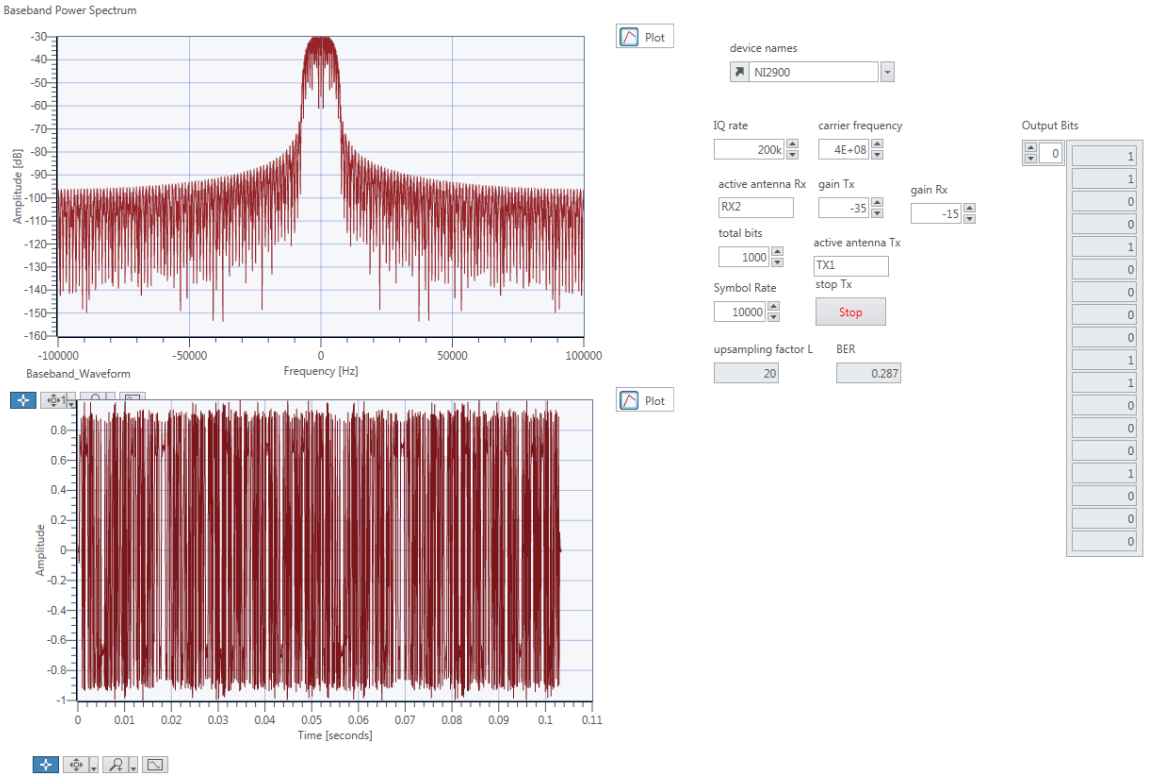


Screenshot 8 – The panel view for exercise 3.

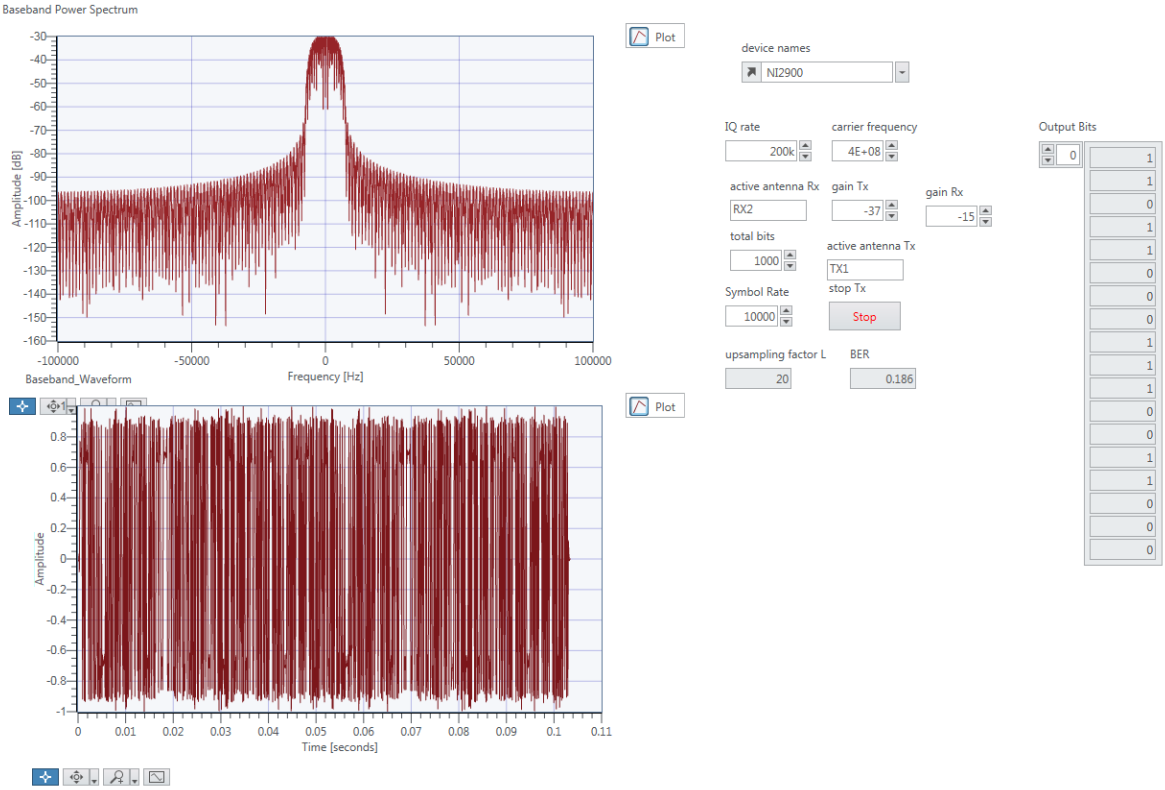
In this part, we first transmitted the bit 3 times by using an array.
Figure below shows the BER value change when Tx Gain (0, -35,-37,-40) and Rx Gain (0,-15).



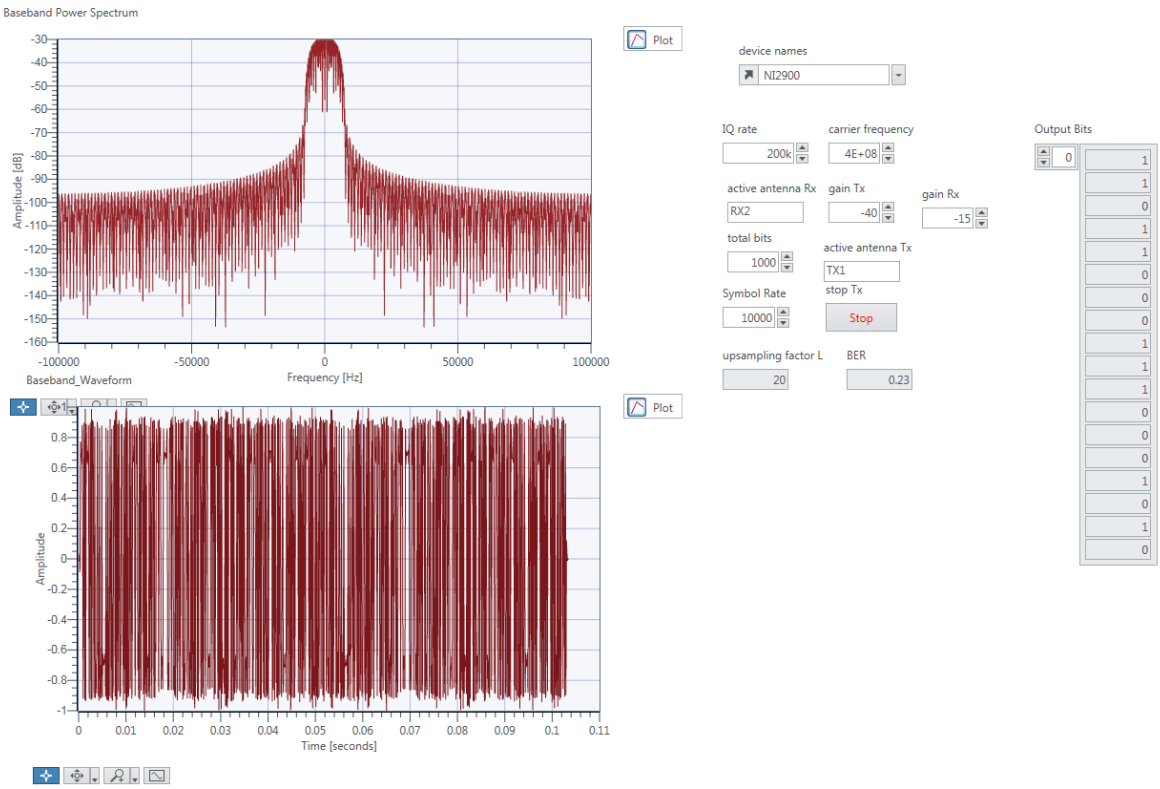
Screenshot 9 – BER 0.284.



Screenshot 10 – BER 0.287.



Screenshot 11 – BER 0.186.



Screenshot 12 – BER 0.23.

The average BER value is:

(0.274 + 0.287 + 0.186 + 0.23) / 4 = 0.24425

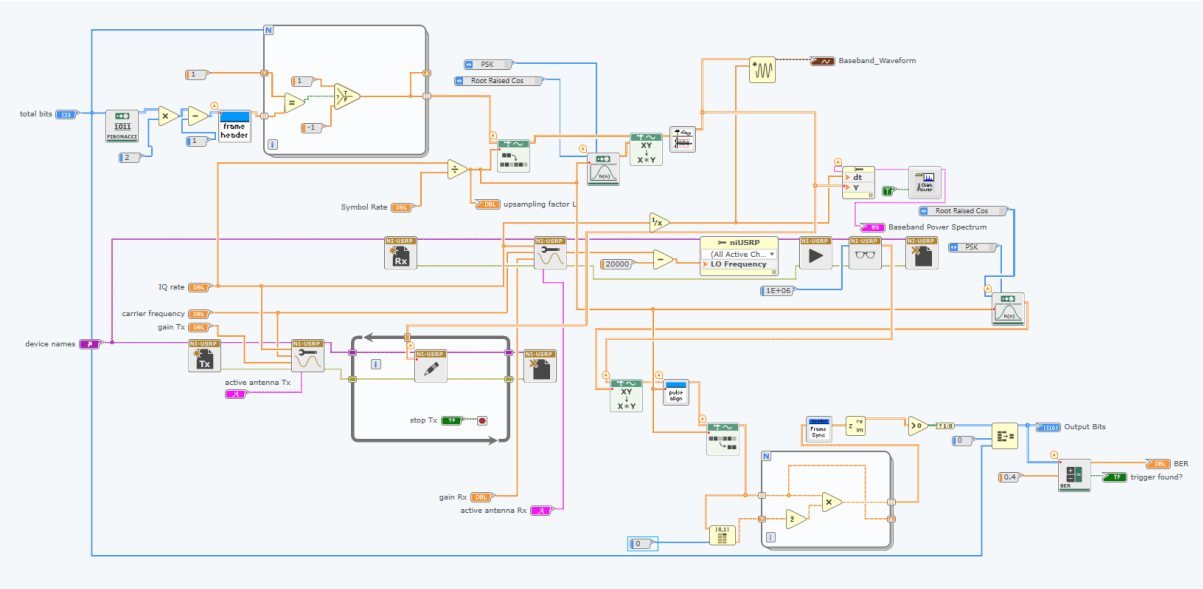
BER value in exercise 2 is 0.48431. The bit error rate (BER) is the number of bit errors per unit time. So we can get the result that FEC has less error during transmission. The disadvantage is due to we send a bit 3 times, it adds latency to a connection. Also FEC generated redundant data to its messages, so the cost of retransmissions are expensive; and hard to broadcast to multiple receivers.

Exercise 4 – Differential Phase Shift Keying

In this exercise, we implemented DPSK, which transmits the difference between two adjacent bits rather than the bits themselves. The table below shows how the difference is obtained for possible pairs of symbols.

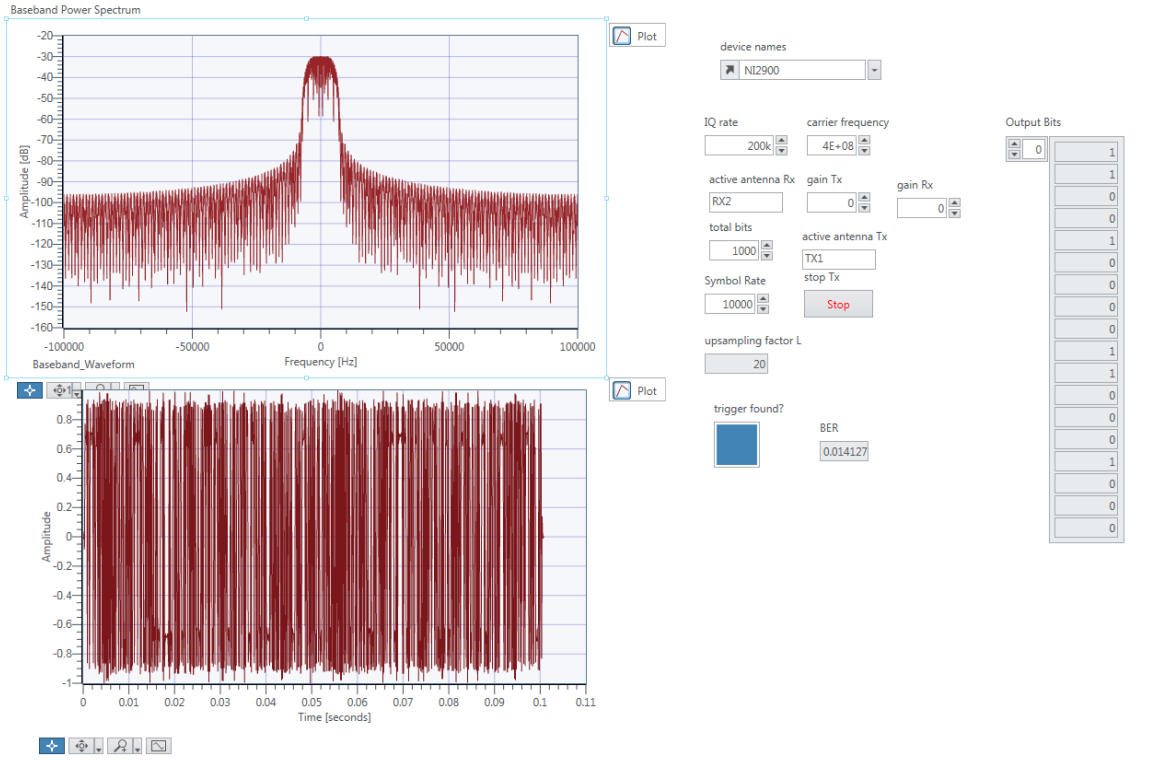
Information symbols $\{a_n\}$		1	-1	-1	1	-1	-1	1	1
$\{b_n\}$		1	1	-1	1	1	-1	1	1
Differentially encoded sequence $\{b_n\}$	1	1	-1	1	1	-1	1	1	1
Output of lowpass filter (polarity)		+	-	-	+	-	-	+	+
Decision		1	-1	-1	1	-1	-1	1	1

Note: The symbol 1 at the beginning of the encoded sequence is the reference symbol

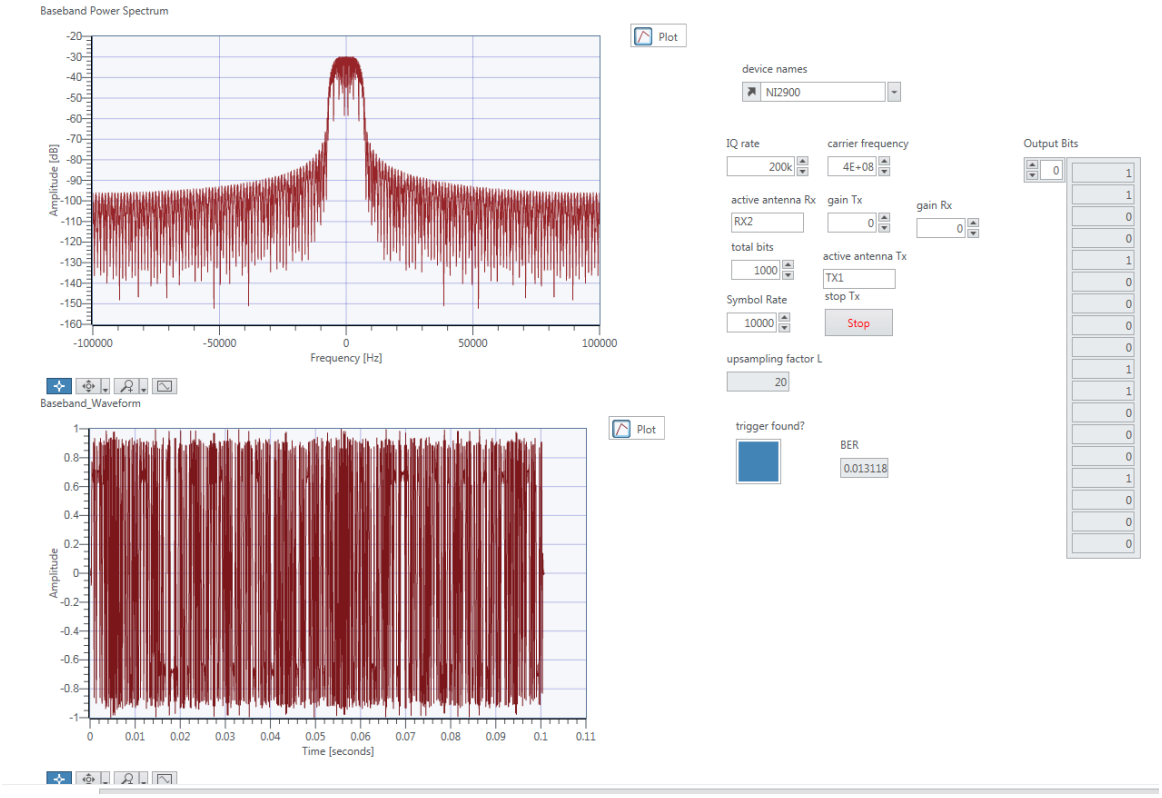


Screenshot 13 – Diagram view for ex4.

After we generate the bit, we multiple it by 2 then minus 1, in order to get 1 or -1(information symbols).



Screenshot 14 – Panel view for ex4.



Screenshot 15 – Diagram view for ex4 (2).

The figure above show that the BER value is quite small compare to FEC and other kind of transmission method. BPSK has a large BER but is simpler to implement.