2 AM Transceiver

In this lab, you will design and implement an AM (Amplitude Modulation) transmitter and receiver using GNU Radio. The lab is divided into three main parts:

1. **Simulation**: Implement and test the AM transceiver in a simulated environment.
2. **Real-World Signal Reception**: Use the USRP (Universal Software Radio Peripheral) to receive and decode a broadcasted AM signal.
3. **Quadrature AM Modulation**: Modify your transmitter and receiver to implement quadrature AM modulation.

2.1 Blocks to Use

Below is a list of GNU Radio blocks that will be helpful for building your AM transceiver. You will need to use them appropriately based on the instructions provided. Refer to the documentation of each block for more details by double-clicking on the block and navigating to the `Documentation` tab.

- `Add`
- `Add Const`
- `Multiply Const`
- `Subtract`
- `Multiply`
- `Phase Shift`
- `Signal Source`
- `Low Pass Filter`
- `Complex to Mag`
- `Complex to Real`
- `Moving Average` (Usage of this block is not intuitive !)

2.2 Transmitter

In this section, you will implement the AM transmitter and observe the signal at different stages of the process. The transmitter follows the superheterodyne architecture, as shown in Fig. 2.1. The superheterodyne architecture consists of two stages of upconversion:

- **First Stage**: You will implement this stage in GNU Radio, focusing on the AM modulation.

- **Second Stage**: This stage shifts the signal to the ISM band and is handled by the USRP.

For the first stage, you will use a simple AM modulation scheme that utilizes only the in-phase component of the signal. The USRP, a software-defined radio, acts as a black box that takes complex samples as input and outputs an RF signal upconverted to a much higher frequency.

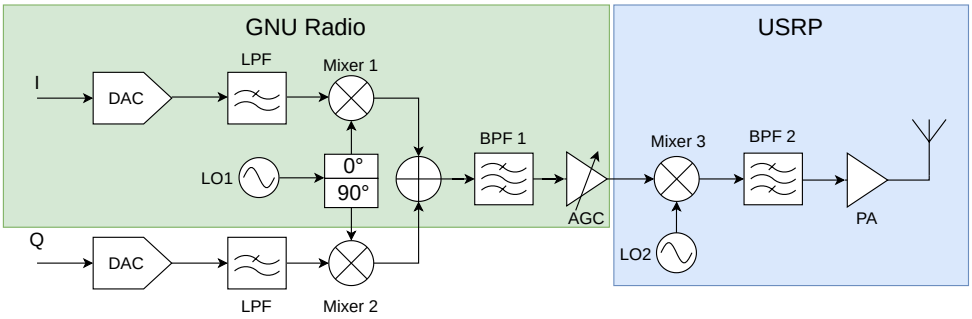

Figure 2.1: Superheterodyne Transmitter

As our objective is to modulate an audio signal in software, we can simplify the architecture in the following ways:

- **DAC Removal**: The digital-to-analog converter (DAC) is unnecessary since we are working with digital signals in GNU Radio.
- **Bandpass Filter Removal**: The bandpass filter after mixing is not required because the mixing process is performed perfectly in software, and we are not implementing single-sideband modulation in this lab.

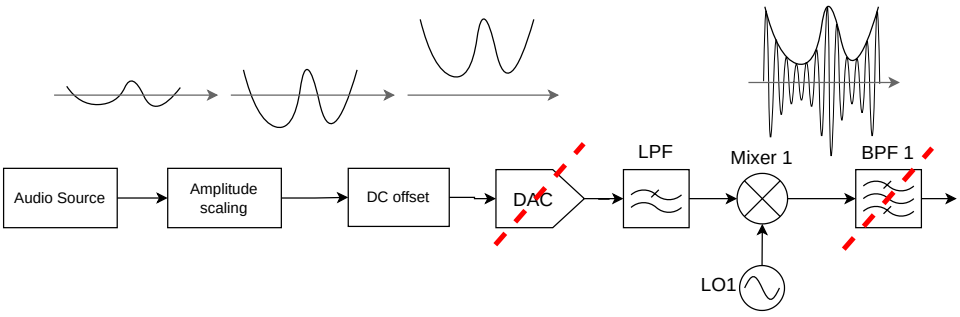

Figure 2.2: Simplified block diagram of the transmitter

You will find a template file named `lab2_AM_trx.grc` on Moodle. Download and open it in `gnuradio-companion`. The flowgraph is preconfigured with the necessary blocks to:

- Visualize the signal at different stages,
- Adjust variables via the GUI,

- Read the input audio file, and
- Output the result to the speakers.

The input signal is an audio file containing a simple sinusoidal signal with a varying frequency.

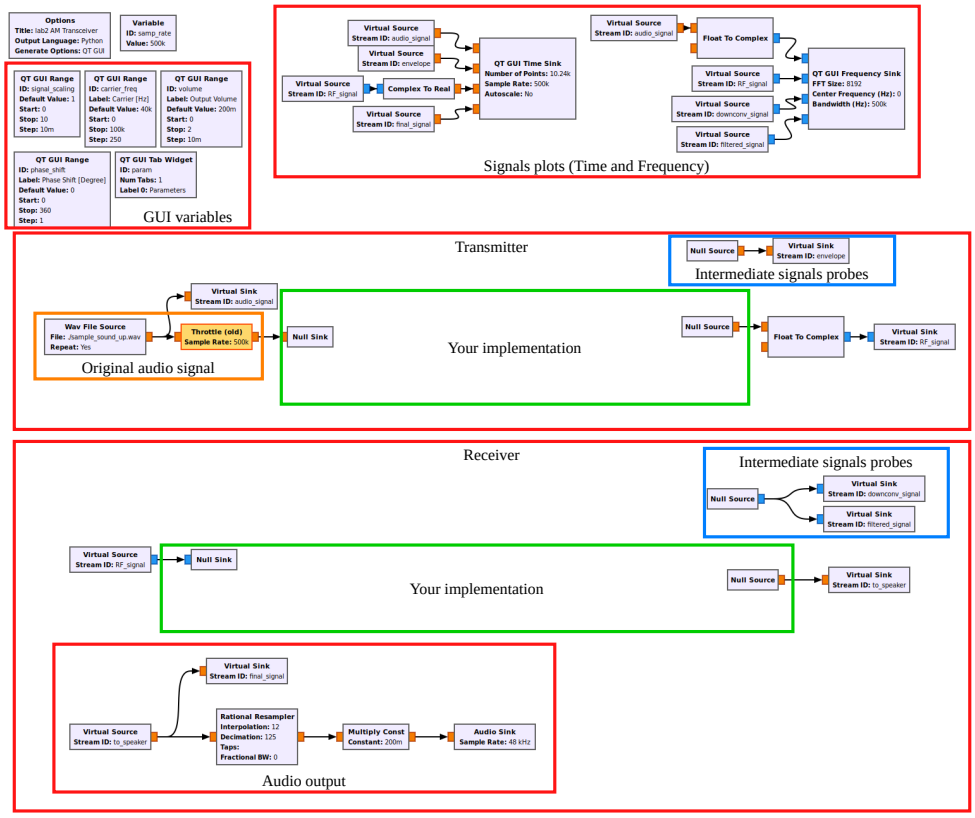

Figure 2.3: Starting point

Your Tasks

Based on the transmitter diagram shown in Fig. 2.2, you will implement the following steps in the flowgraph:

- **Create the AM Signal Envelope**:
  - Add the necessary operations to generate the envelope of the AM signal. Use the parameter *signal_scaling* for amplitude scaling.
  - Verify the signal's appearance in both the time and frequency domains using the *Virtual Sink* blocks with the ID: `envelope`.
- **Add a Low-Pass Filter**:
  - Insert a low-pass filter to remove high-frequency components from the signal.
  - Choose the cutoff frequency based on the requirement to transmit an audible audio signal.

- Set the transition width to 10% of the cutoff frequency.

- **Mix the Signal with a Carrier**:
  - Mix the filtered signal with a cosine wave at the carrier frequency specified by the variable *carrier_freq*.
  - Connect the mixed signal to the real input of the *Float to Complex* block already present in the flowgraph.

Note: The low-pass filter introduces a delay to the signal. Depending on where you connect the *Virtual Sink* `envelope`, the signal may appear out of sync in the time-domain plot.

2.3 Receiver

Next, we will implement the receiver part of the system, which will demodulate the signal and recover the original audio.

Your Task

Implement each step described below to build your AM receiver.

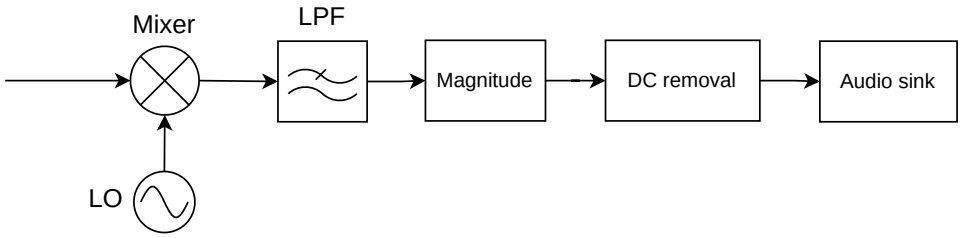

Figure 2.4: AM receiver

Downconversion

The first step in the receiver is to downconvert the signal to baseband. This is achieved by mixing the received signal with a complex exponential at the negative carrier frequency, $-carrier\_freq$.

Important Note Unlike the upconversion process, where a real cosine was used, downconversion requires a complex exponential, $e^{-j2\pi f_c t}$.

Question: What would be the problem if you used a real sine for the mixing?

In GNU Radio, *Signal Source* block set to a cosine (or sine) will generate a complex exponential if the output type complex is selected.
You can use the *Virtual sink* with ID: `downconv_signal` to visualize the signal at this stage.

High Frequency Image Filtering

By examining the spectrum of the downconverted signal, you will observe that the signal is centered around 0 Hz, but a copy of the signal appears at twice the negative carrier frequency. This is known as the **high-frequency image**, a consequence of the downconversion of a real signal.

To remove this image, apply a low-pass filter with a cutoff frequency set to the maximum frequency of the audio signal. This will preserve the baseband signal while eliminating the unwanted high-frequency components.

You can visualize the filtered signal using the *Virtual Sink* with the ID: `filtered_signal`.

Removing DC Offset

As the transmitted signal is modulated only in the amplitude of the carrier, we only care about the envelope of the signal, i.e only the magnitude of the complex signal.
Convert the complex signal to its magnitude using the *Complex to Mag* block.
If you connect the signal to the *Virtual sink* with ID: to_speaker, you can hear the audio signal. You will notice that the signal is not centered around 0 but around a certain value, indicating a DC offset. You may be able to hard-code an offset value for your simulated transmission, however, in the next section, you will receive a real radio signal and you will need to find a way to remove the DC offset in a more general way.

Question: Can you implement a general way to remove the DC offset from the signal?

Testing the System

You will now verify that your system correctly modulate and demodulate the audio signal.
Ask the TA to lend you a set of speakers and connect them to your computer.

Important Make sure the speakers are recognized by the computer (i.e., you see the speaker icon  on the top right corner of the screen). If the speakers are not recognized, logging out and in again should fix it.
After restarting your flowgraph, you should be able to hear a clear audio signal. If this is the case, congratulations, you can now proceed to using it with a real radio signal.
If you don't hear a clear signal, try to identify where the problem is by looking at the different stages of the signal in the GUI. Both time domain and frequency domain can help you identify the source of the issue.

2.4 USRP

You will now incorporate a radio in your system.

Important **Make a copy of your flowgraph** such that you can freely modify it for this task.

Cleanup Flowgraph
You now only need to use the receiver part of the flowgraph. To keep things cleaner, you can delete the Tx part of the flowgraph. The *QT GUI Sink* blocks will complain that some inputs are missing, just replace the *virtual Source* blocks corresponding to elements of the transmitter by a *Null Sink* instead.

USRP Configuration

- First, you need a USRP (Universal Software Radio Peripheral). Ask the TA to provide you with one as well as an Ethernet cable, power supply and an antenna.
- Connect the USRP to the computer using the Ethernet cable. You need to connect to one of the two ethernet ports at the **bottom** of the computer, one of them will work but it's not the same on every computer.
  Try one and try communicating with the USRP by opening a terminal and running the command `ping 192.168.2.10`. If you get an answer, perfect, else switch the ethernet port.
- In gnuradio-companion, add a *UHD: USRP Source* block to the flowgraph. This block allows you to control the USRP and receive signals from it.
  - Set the *Center Frequency* (under RF options) to $868.5\,\mathrm{MHz}$. (GNU radio accept parameters given in exponential notation ,e.g., $868.5\mathrm{e}6$)
  - Set the *Device Address* to "addr=192.168.2.10" (with the quotation marks)
  - Set the *Sync* option to No Sync
- Connect the output of the USRP to a *AGC* block which stands for adaptive gain control and will make sure the signal is scaled to an average power of 1.
- Finally use the output of the AGC instead of the *Virtual Source* in the downconversion.

Frequency Tuning
You should now be able to receive an AM signal broadcasted by the TA!

- You may need to adjust the carrier frequency slightly to achieve the best reception. This is because each radio has a unique reference oscillator, leading to small variations in frequency.
- Observe the spectrum of the downconverted signal to identify the broadcasted AM signal and fine-tune the frequency accordingly.

2.5 Quadrature AM

In this final task, you will modify the transmitter and receiver to use a quadrature AM modulation.

Important Go back to the copy you made before including the USRP

The quadrature AM modulation is a more efficient way to transmit the signal as it uses both the in-phase and quadrature components of the signal. As a sine and cosine are orthogonal you can transmit twice the information in the same bandwidth!

Transmitter

To implement quadrature AM modulation, modify the transmitter by adding the ”Q” branch as shown in the superheterodyne architecture in Fig. 2.1.
Use copy and paste to quickly add the necessary blocks to the flowgraph.
This modification will enable quadrature AM modulation, allowing you to transmit both the in-phase (I) and quadrature (Q) components of the signal.
In the new *Audio Source* block created for the new branch, load the audio file:
`./sample_sound_up2.wav`.

Receiver

The change to the receiver is minimal, you only need to replace the *Complex to Mag* by a *Complex to Real* block (or *Complex to Imag* depending on which stream you want to listen to).
However, it is more interesting to be able to choose the signal you want to listen to in the GUI.
To this end, you can add a *Phase Shift* block just before the *Complex to Real*.
Set the phase shift to the GUI variable “phase_shift” and the units to “Degrees”.

Question: Can you choose which stream to listen? What phase shifts correspond to each stream and what happens for other values?

Question: What happens if the carrier frequency of the downconversion is not perfect? (e.g. $0.02\,\mathrm{Hz}$ off).