EXPERIMENT 1

Aim: To perform AM Modulation and Demodulation using GNU radio.

Software: GNU radio.

Theory:

GNU radio: GNU Radio is a free software development toolkit that provides signal processing blocks to implement software-defined radios and signal processing systems. It can be used with external radio frequency (RF) hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic, and commercial environments to support both wireless communications research and real-world radio systems. The GNU Radio software provides the framework and tools to build and run software radio or just general signal-processing applications. The GNU Radio applications themselves are generally known as "flowgraphs", which are a series of signal processing blocks connected together, thus describing a data flow.

GNU Radio Companion:

The GNU Radio Companion is a graphical UI used to develop GNU Radio applications. This is the front-end to the GNU Radio libraries for signal processing. GRC was developed by Josh Blum during his studies at Johns Hopkins University (2006–2007), then distributed as free software for the *October 2009 Hackfest*. Starting with the 3.2.0 release, GRC was officially bundled with the GNU Radio software distribution.

Amplitude Modulation:

An Amplitude Modulated signal is composed of both low frequency and high frequency components. The amplitude of the high frequency (carrier) of the signal is controlled by the low frequency (modulating) signal. The envelope of the signal is created by the low frequency signal. If the modulating signal is sinusoidal, then the envelope of the modulated Radio Frequency (RF) signal will also be sinusoidal. This would be the case in a common AM radio. The low frequency signal would be an audio signal and the high frequency would be the transmitting frequency of the AM radio station

The mathematical representation for this waveform is as follows:

$$f_{AM}(t) = A\left[1 + \mu\cos(\omega_m t)\right]\cos(\omega_c t)$$

where, A= DC value of the waveform

μ= modulation index

ω_m= modulation frequency (rad/s)

ω_c= carrier frequency (rad/s)

The circuit for generating an AM modulated waveform must produce the product of the carrier and the modulating signal. This can be achieve in many ways, but often is done by biasing a transistor for nonlinear operation (creating the product term) and filtering the output with a tank circuit to remove the higher harmonics introduced. For class B operation, the transistor is biased such that when both the carrier and modulating signals are zero, the DC voltage at the transistor base will be 0.7 V (i.e., the knee voltage of the base emitter junction). If a carrier is added via the coupling capacitor C1 while the modulating signal remains zero the transistor will be turned off for the negative half cycle of the carrier, producing only positive current pulses in the collector. The tank circuit will have large impedance at the carrier frequency, also the fundamental frequency of the current pulses, and low impedance at the higher harmonics of the current pulses. Thus, the voltage produced at the output will be a sinusoid of the carrier frequency. When the modulating signal is added via the coupling capacitor C5 the emitter voltage of the transistor will follow the modulating signal, causing the cutoff voltage of the transistor and also the collector current pulse amplitude to vary with the modulating signal.

The modulation index, μ, can be found with the following equation:

$$\mu = \frac{m_p}{A}$$

After receiving an AM signal, it can be demodulated to recover the low frequency signal. One of the simplest types of AM demodulating circuits is the envelope detector. In order to accurately recover the low frequency signal the envelope detector must satisfy an important condition; the time constant of the envelope detector network must be much longer than the period of the high frequency signal but much shorter than the period of the low frequency signal. Once the signal is demodulated, the high frequency signal is eliminated and what remains is the low frequency component.

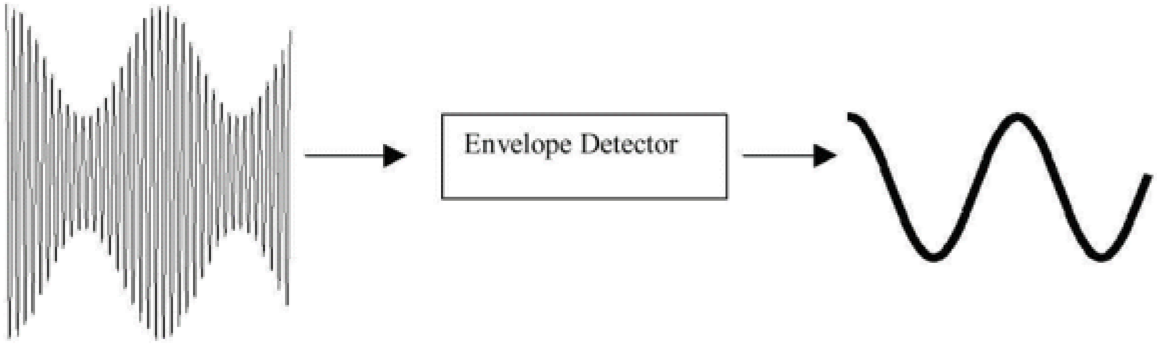


Figure 1.1. Demodulator

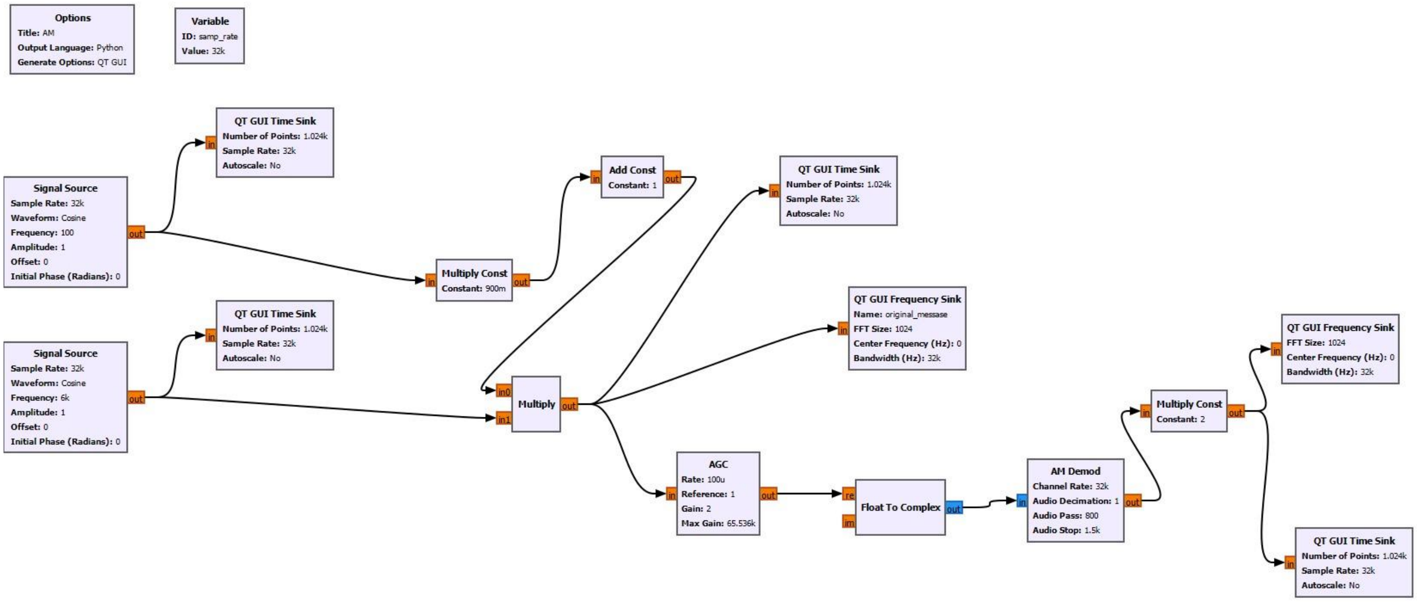
<u>Blocks used in GNU radio</u>:

1. Signal Source: Generates signals such as sine waves, square waves, or other waveforms. Parameters include sample rate, frequency, and waveform type.
2. QT GUI Time Sink: Displays the time-domain representation of the signal. It shows how the signal amplitude varies over time.
3. QT GUI Frequency Sink: Displays the frequency-domain representation of the signal, typically using a Fast Fourier Transform (FFT) to show the signal's spectrum.
4. Throttle: Controls the flow of data to prevent the system from being overwhelmed by too much data. It is often used in simulations to regulate the sample rate.
5. Multiply: Multiplies two signals together, which can be used for modulation or mixing signals.
6. Add: Adds two signals together, useful for combining signals or adding noise.
7. Filter: Applies a filter to the signal, such as a low-pass, high-pass, or band-pass filter, to remove unwanted frequencies.
8. Noise Source: Generates noise, which can be added to a signal to simulate real-world conditions.
9. File Sink: Writes the signal data to a file for later analysis or processing.
10. File Source: Reads signal data from a file, which can be used for playback or further processing.
11. UHD: USRP Source/Sink: Interfaces with Ettus Research Universal Software Radio Peripheral (USRP) hardware for transmitting and receiving radio signals.
12. Message Strobe: Generates messages at regular intervals, which can be used to trigger events or control other blocks.
13. Variable: Defines a variable that can be used to control parameters of other blocks dynamically.
14. AGC (Automatic Gain Control): Automatically adjusts the gain of a signal to maintain a consistent output level.

Procedure:

1. Click on the GNU Radio icon to launch GNU Radio companion.
2. Go to file option then select new→ QT GUI. And save the file.
3. Click on Variable block and set the sample rate 32k (default value 32k).
4. In order to design AM modulator we need various block such as source, multiply, QT GUI Frequency sink and QT GUI Time sink.
5. We need two signal source blocks, one for message signal and another for the carrier signal.
6. Location of signal source block is
   Core →waveform generator→ signal source
   Double click on signal source block (carrier signal block), select output type: float, waveform: cosine, frequency: 6000 (6 kHz), Amplitude: 1, offset: 0 for the carrier signal.
7. Double click on the second signal source block (message signal block), select output type: float, waveform: square, frequency: 100 (100 Hz), Amplitude: 1, offset: 0 for the message signal.
8. We need one multiply block. Location of multiply is
   Core → math operator → multiply
   Set the output type: float
9. Core → Level Controllers → AGC
10. Location of AM Demod is
    Core → Modulators → AM Demod
11. We need QT GUI Frequency sink block to see the output frequency response. Location of QT GUI Frequency sink is
    Core → instrumentation → QT→ QT GUI Frequency sink
    Set the output type: float
12. We need one QT GUI Time sink block to see the output time domain response. Location of QT GUI Time sink is
    Core → instrumentation → QT→ QT GUI Time sink
    Set the output type: float
13. Take care to set the input/output type to float in all the blocks.
14. Now connect the different blocks as shown in Figure 1.2.
15. Click on run option.

Block Diagram:



Options
Title: AM
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 100
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

Multiply Const
Constant: 900m

Add Const
Constant: 1

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 6k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

Multiply

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

QT GUI Frequency Sink
Name: original_message
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k

AGC
Rate: 100u
Reference: 1
Gain: 2
Max Gain: 65.536k

Float To Complex

AM Demod
Channel Rate: 32k
Audio Decimation: 1
Audio Pass: 800
Audio Stop: 1.5k

Multiply Const
Constant: 2

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

Figure 1.2. Block diagram(GNU)

Results:

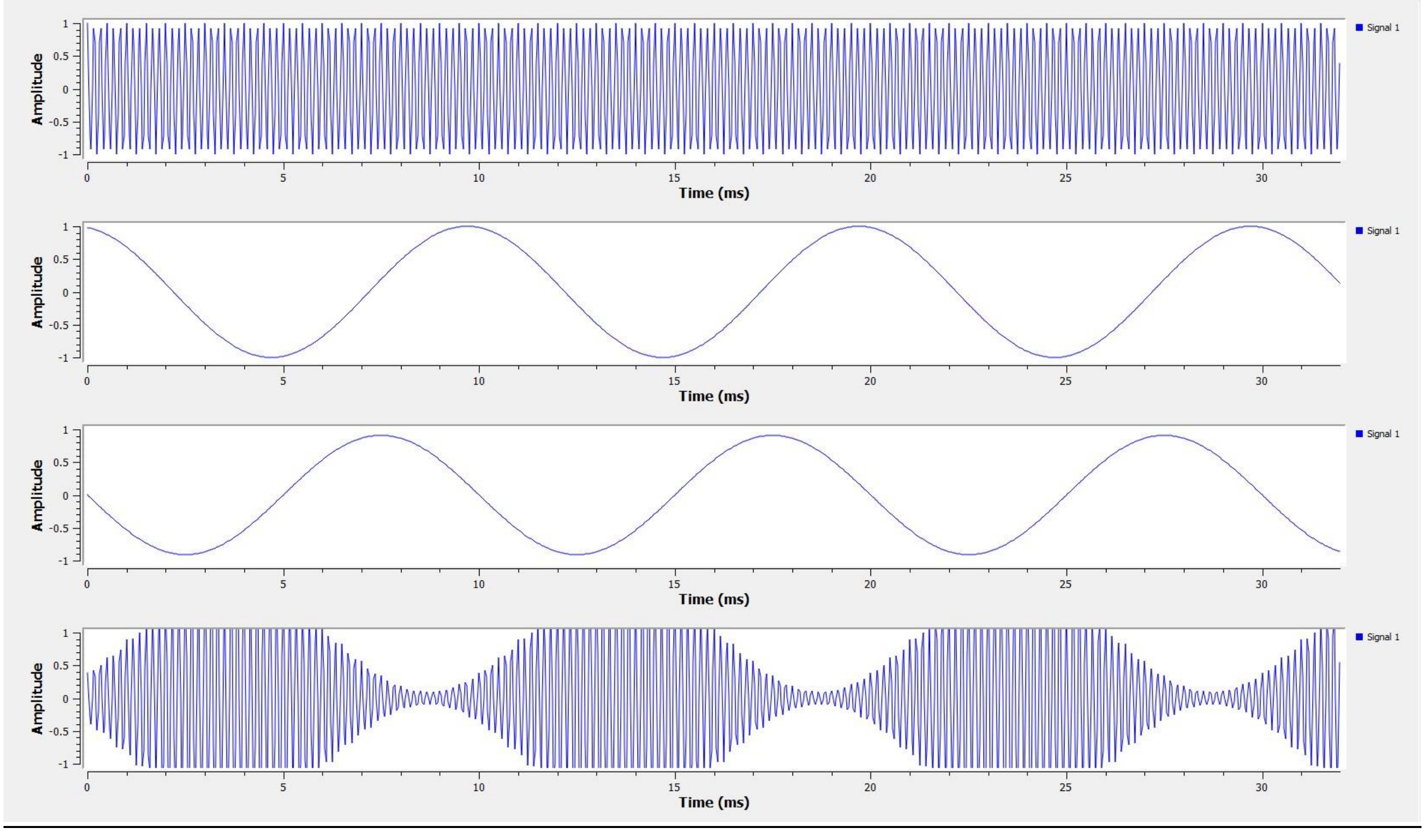


Figure 1.3. Output waveforms

Conclusion:

The experiment in GNU Radio successfully demonstrated AM modulation and demodulation. We modulated a carrier with an input message signal and accurately retrieved the message during demodulation using envelope detection. The results confirmed the effectiveness of AM communication and showcased the practical use of SDR tools for signal processing.

EXPERIMENT 2-A

Aim: To perform ASK modulation and demodulation using GNU radio.

Software: GNU radio.

Theory:

GNU radio: GNU Radio is a free software development toolkit that provides signal processing blocks to implement software-defined radios and signal processing systems. It can be used with external radio frequency (RF) hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic, and commercial environments to support both wireless communications research and real-world radio systems. The GNU Radio software provides the framework and tools to build and run software radio or just general signal-processing applications. The GNU Radio applications themselves are generally known as "flowgraphs", which are a series of signal processing blocks connected together, thus describing a data flow.

GNU Radio Companion:

The GNU Radio Companion is a graphical UI used to develop GNU Radio applications. This is the front-end to the GNU Radio libraries for signal processing. GRC was developed by Josh Blum during his studies at Johns Hopkins University (2006–2007), then distributed as free software for the *October 2009 Hackfest*. Starting with the 3.2.0 release, GRC was officially bundled with the GNU Radio software distribution.

Amplitude Shift Keying:

In digital communication, different modulation techniques are used to transmit data or message to receiver over a communication channel . One such technique is Amplitude Shift Keying (ASK). It is a modulation technique that alters the amplitude of a carrier signal to transmit the information over channel. It is a modulation scheme having wide range of application in real world which includes radio, television, and digital data transmission.

ASK Modulation

Amplitude Shift Keying (ASK) is a digital modulation technique. It transmits the digital information by varying the amplitude of a carrier signal. In ASK, a high-amplitude carrier signal is used to represent a binary '1,' and a low-amplitude carrier signal represents a binary '0.'

It involves the superimposition of a carrier signal and a digital message signal. The carrier signal is often a high-frequency sinusoidal waveform, which serves as the carrier for the digital information. The binary message signal, consisting of '1's and '0's, is used to control the amplitude of the carrier signal. The resultant signal formed after the superimposition of message and carrier is transmitted over the communication channel.

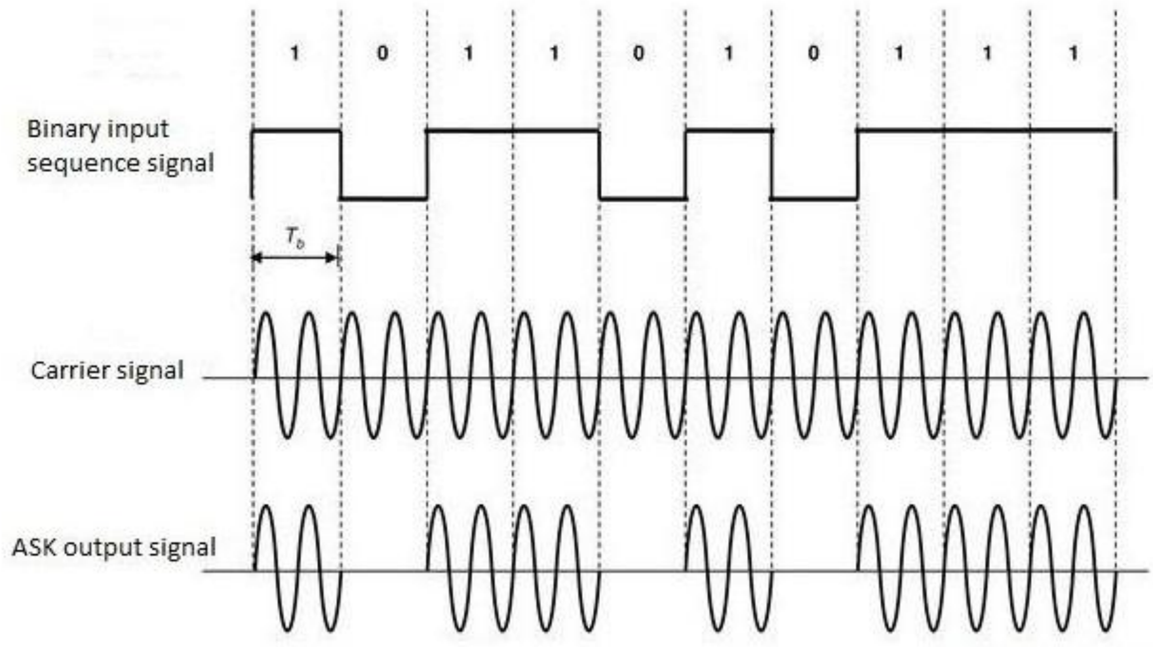


Figure 2.1. ASK waveforms

ASK Demodulation

Demodulation in Amplitude Shift Keying (ASK) involves extracting the original digital message signal from the modulated carrier wave. The process reverses the modulation applied to the carrier signal. The demodulation of ASK can be performed through various methods, such as envelope detection or coherent detection.

Types Of ASK Demodulation

There are mainly two types of demodulation which is performed namely envelope detection and coherent detection technique:

- Envelope detection
- Coherent detection

Envelope detection

Envelop detection is a relatively simpler method as compared to coherent detection which is used for demodulating Amplitude Shift Keying (ASK) signals. It primarily focuses on extracting the envelope of the modulated signal to recover the original digital data.

The steps involved in this technique is as follows:

- The modulated ASK signal, which contains the carrier signal and the digital data, is received by the demodulator. This is called receiving of signal.

- The received signal is then passed through a diode. The diode rectifies the signal, allowing only the positive portion of the waveform (the envelope) to pass through it. This process is called Diode Rectification.
- Following the rectification process, a low-pass filter is applied to the rectified signal. This filter helps remove or discard the high-frequency carrier signal, leaving the varying amplitude that represents the original digital data. This process is called Low-Pass Filtering of the modulated signal.
- Once the signal has been filtered by passing it through the LPF, a threshold detection mechanism is used to interpret the amplitude changes. Comparing the amplitudes to a predefined threshold helps to determine the transmitted digital data. If the amplitude is above the threshold, it might be interpreted as one bit '1', and if it's below the threshold, it might be interpreted as the other bit '0'. This is called Threshold Detection.
- The message signal is recovered from the modulated signal.

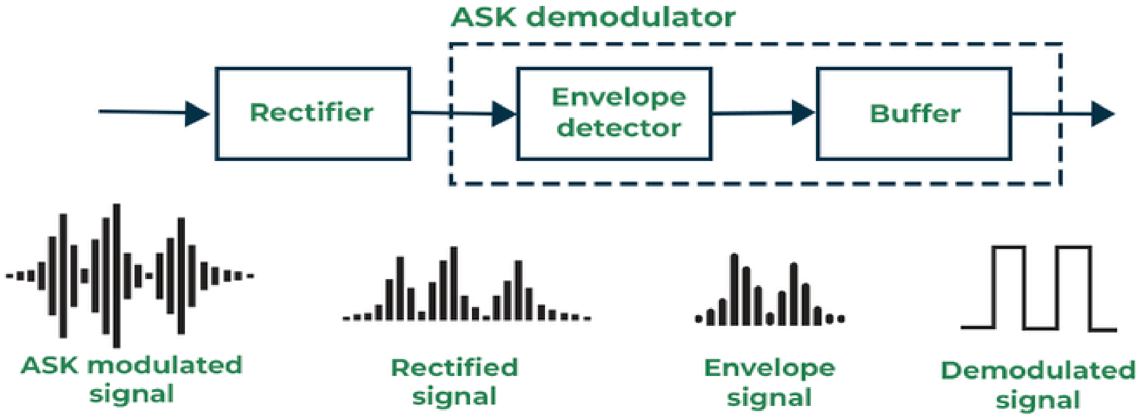


Figure 2.2. Envelope detection

<u>Coherent detection</u>

This is another method which is used for demodulation of ASK signal. It involves phase synchronization with the carrier wave.

The steps involved in this technique is as follows

- The received signal is mixed with a local oscillator, aligning its phase and frequency with the original carrier signal. This process is called Reception.
- Filtering isolates the part of the mixed signal that corresponds to the data signal.
- The demodulator sets an amplitude threshold to distinguish between the '1' and '0' states.
- The amplitude of the received signal is compared against the threshold to determine the transmitted data.

In this way the original message signal is recovered.

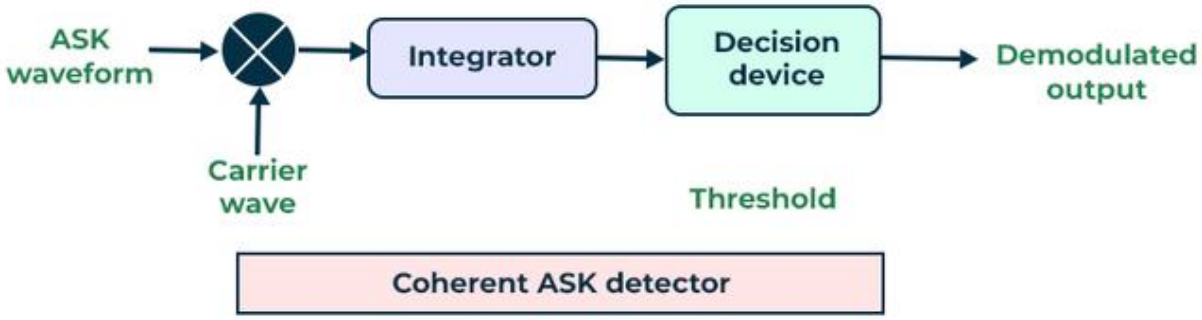


Figure 2.3. Coherent detection

ASK Applications:

Amplitude shift keying applications are mentioned below. They are:

- Low-frequency RF applications
- Home automation devices
- Industrial networks devices
- Wireless base stations
- Tire pressuring monitoring systems

Blocks of GNU radio used:

1. Options Block: Defines global settings for the flowgraph.

Parameters:

Title: The name of the flowgraph (not specified here).

Output Language: Python.

Generate Options: QT GUI for visualizing signals.

2. Variable Block

Variable Name: samp_rate.

Value: 200k.

Purpose: Sets the sample rate for all the blocks in the flowgraph, ensuring consistent signal processing.

3. Signal Sources

Three signal sources are used for different purposes:

Carrier Signal:

Waveform: Cosine.

Frequency: 10 kHz.

Amplitude: 1.

Purpose: Acts as the carrier signal for ASK modulation.

Modulating Signal:

Waveform: Cosine.

Frequency: 500 Hz.

Amplitude: 1.

Purpose: Represents the digital data (modulation signal) to modulate the carrier.

Demodulation Signal:

Waveform: Cosine.

Frequency: 10 kHz (same as the carrier).

Amplitude: 1.

Purpose: Used to demodulate the received ASK signal.

4. Throttle Blocks: Limits the processing speed of the flowgraph to the specified samp_rate (200k) to avoid excessive CPU usage.

5. Multiply Block (Modulation): Multiplies the carrier signal with the modulating signal to perform ASK modulation.

Input 1: Carrier signal.

Input 2: Modulating signal.

6. Float to Complex Block: Converts the real-valued ASK signal to a complex signal for visualization and further processing.

7. QT GUI Frequency Sink: Displays the frequency spectrum of the signals.

Parameters:

FFT Size: 1024.

Center Frequency: 10 kHz (for modulated/demodulated signals).

Bandwidth: 200 kHz (matching samp_rate).

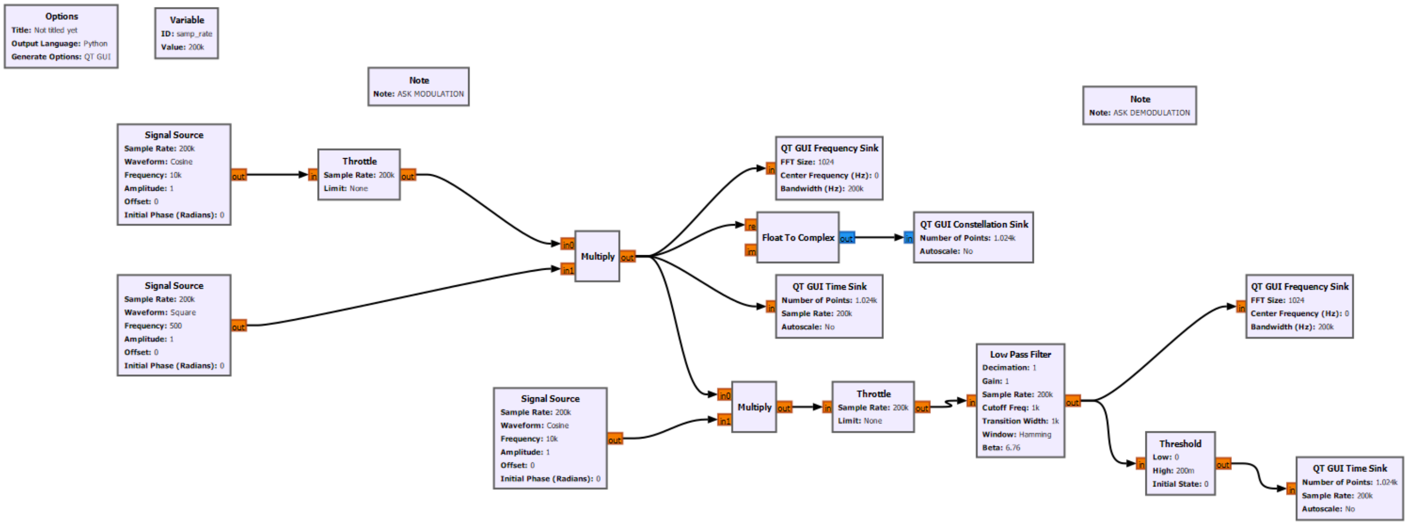
8. QT GUI Constellation Sink: Displays the constellation diagram of the complex signal after modulation or demodulation.

9. QT GUI Time Sink: Visualizes the time-domain waveform of the signals.

Procedure:

1. Click on the GNU Radio icon to launch GNU Radio companion.
2. Go to file option then select new→ QT GUI. And save the file.
3. Click on Variable block and set the sample rate 200k (default value 32k).
4. In order to design ASK modulator we need various block such as source, multiply, QT GUI Frequency sink and QT GUI Time sink.
5. We need two signal source blocks, one for message signal and another for the carrier signal.
6. Location of signal source block is
   Core →waveform generator→ signal source
   Double click on signal source block (carrier signal block), select output type: float, waveform: cosine, frequency: 16000 (16 kHz), Amplitude: 1, offset: 0 for the carrier signal.
7. Double click on the second signal source block (message signal block), select output type: float, waveform: square, frequency: 500 (500 Hz), Amplitude: 1, offset: 0 for the message signal.
8. We need one throttle block. Location of throttle is
   Core → mise → throttle
9. We need one multiply block. Location of multiply is
   Core → math operator → multiply
   Set the output type: float
10. We need QT GUI Frequency sink block to see the output frequency response. Location of QT GUI Frequency sink is
    Core → instrumentation → QT→ QT GUI Frequency sink
    Set the output type: float
11. We need one QT GUI Time sink block to see the output time domain response. Location of QT GUI Time sink is
    Core → instrumentation → QT→ QT GUI Time sink
    Set the output type: float
12. Take care to set the input/output type to float in all the blocks.
13. Now connect the different blocks as shown in Figure 2.4.
14. Click on run option.

Block Diagram:



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 200k

Note
Note: ASK MODULATION

Signal Source
Sample Rate: 200k
Waveform: Cosine
Frequency: 10k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Throttle
Sample Rate: 200k
Limit: None

Signal Source
Sample Rate: 200k
Waveform: Square
Frequency: 500
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Multiply

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

Float To Complex

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 200k
Autoscale: No

Signal Source
Sample Rate: 200k
Waveform: Cosine
Frequency: 10k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Multiply

Throttle
Sample Rate: 200k
Limit: None

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 200k
Cutoff Freq: 1k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Note
Note: ASK DEMODULATION

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 200k

Threshold
Low: 0
High: 200m
Initial State: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 200k
Autoscale: No

Figure 2.4. Block diagram(GNU)

Results:

Time domain waveforms:

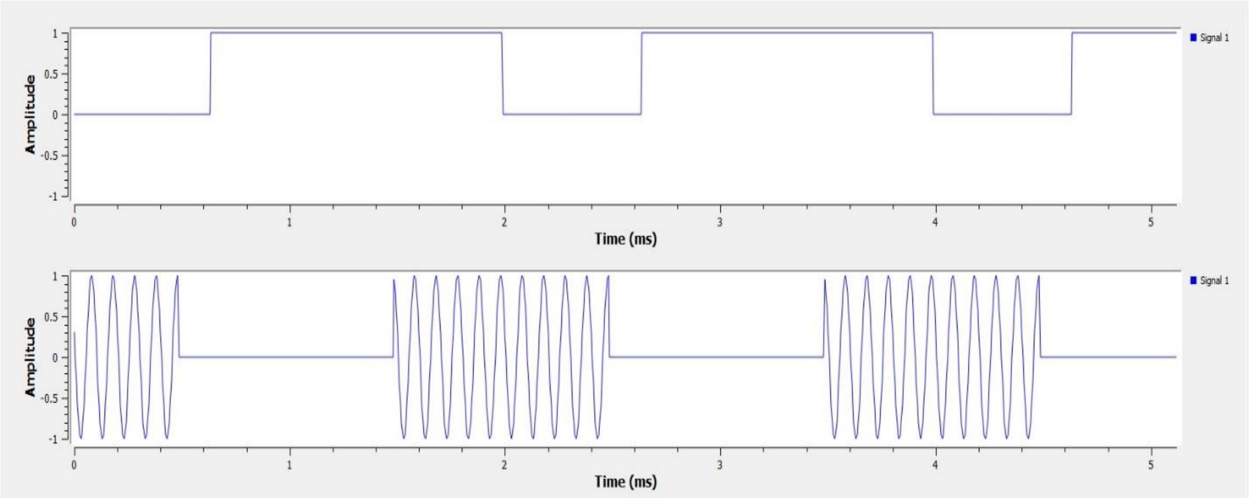


Figure 2.5. (a)Demodulated waveform

(b)Modulated waveform

Frequency domain waveforms:

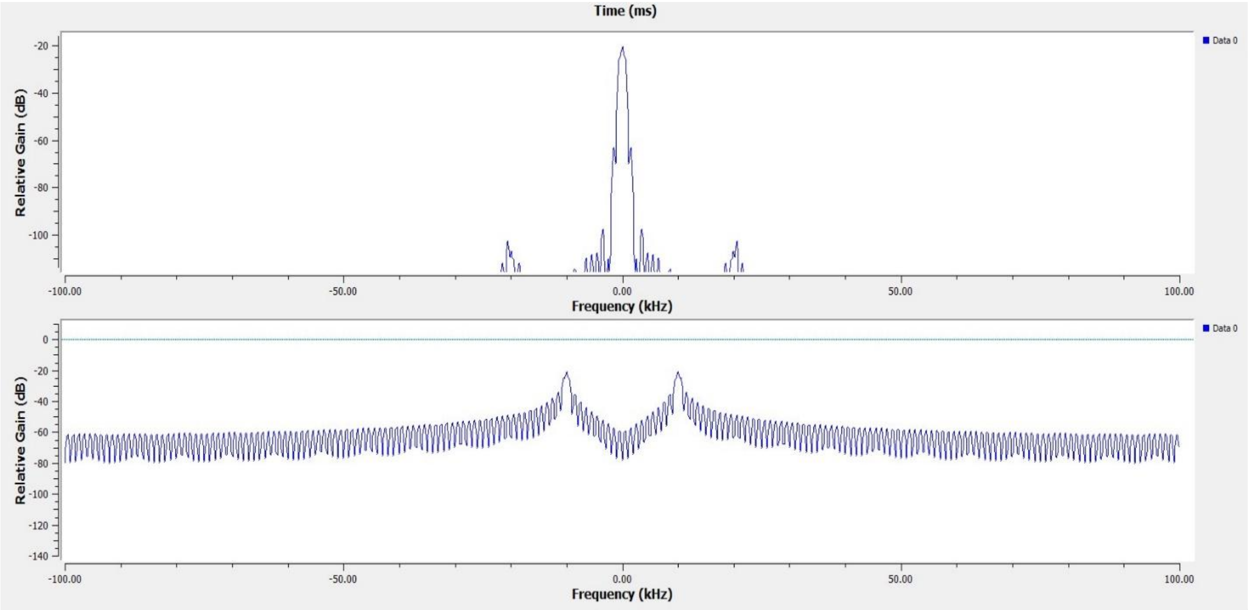


Figure 2.6. (a)Demodulated waveform

(b)Modulated waveform

Constellation diagram:

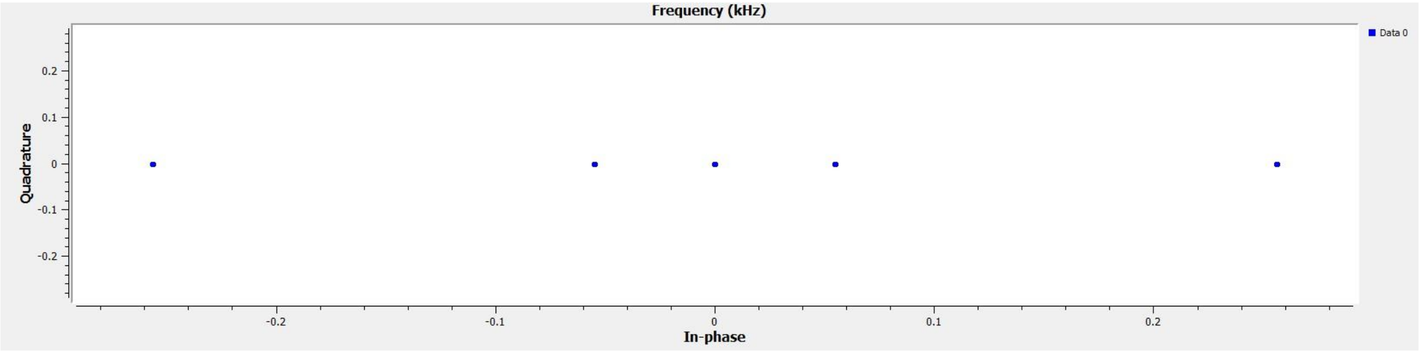


Figure 2.7. Constellation diagram

Conclusion:

In the ASK Modulation and Demodulation experiment using GNU Radio, we successfully implemented and analyzed Amplitude Shift Keying (ASK), a fundamental digital modulation scheme. The experiment demonstrated how binary data can be transmitted by varying the amplitude of a carrier signal and later retrieved through demodulation. Through GNU Radio, we visualized the ASK waveform, spectrum, and constellation diagrams, which provided insight into the signal behavior during transmission and reception.

EXPERIMENT 2-B

Aim: To perform QAM using GNU radio.

Software: GNU radio.

Theory:

Quadrature amplitude modulation (QAM):

Quadrature amplitude modulation (QAM) is the name of a family of digital modulation methods and a related family of analog modulation methods widely used in modern telecommunications to transmit information. It conveys two analog message signals, or two digital bit streams, by changing (modulating) the amplitudes of two carrier waves, using the amplitude-shift keying (ASK) digital modulation scheme or amplitude modulation (AM) analog modulation scheme. The two carrier waves are of the same frequency and are out of phase with each other by 90°, a condition known as orthogonality or quadrature. The transmitted signal is created by adding the two carrier waves together. At the receiver, the two waves can be coherently separated (demodulated) because of their orthogonality. Another key property is that the modulations are low-frequency/low-bandwidth waveforms compared to the carrier frequency, which is known as the narrowband assumption.

Phase modulation (analog PM) and phase-shift keying (digital PSK) can be regarded as a special case of QAM, where the amplitude of the transmitted signal is a constant, but its phase varies. This can also be extended to frequency modulation (FM) and frequency-shift keying (FSK), for these can be regarded as a special case of phase modulation.

QAM is used extensively as a modulation scheme for digital communications systems, such as in 802.11 Wi-Fi standards. Arbitrarily high spectral efficiencies can be achieved with QAM by setting a suitable constellation size, limited only by the noise level and linearity of the communications channel. QAM is being used in optical fiber systems as bit rates increase; QAM16 and QAM64 can be optically emulated with a three-path interferometer.

As in many digital modulation schemes, the constellation diagram is useful for QAM. In QAM, the constellation points are usually arranged in a square grid with equal vertical and horizontal spacing, although other configurations are possible (e.g. a hexagonal or triangular grid). In digital telecommunications the data is usually binary, so the number of points in the grid is typically a power of 2 (2, 4, 8, …), corresponding to the number of bits per symbol. The simplest and most commonly used QAM constellations consist of points arranged in a square, i.e. 16-QAM, 64-QAM and 256-QAM (even powers of two). Non-square constellations, such as Cross-QAM, can offer greater efficiency but are rarely used because of the cost of increased modem complexity. By moving to a higher-order constellation, it is possible to transmit more bits per symbol. However, if the mean energy of the constellation is to remain the same (by way of making a fair comparison), the points must be closer together and are thus more susceptible to noise and other corruption; this results in a higher bit error rate and so higher-order QAM can deliver more data less reliably than lower-order QAM, for constant mean constellation energy. Using higher-order QAM without increasing the bit error rate requires a higher signal-to-noise ratio (SNR) by increasing signal energy, reducing noise, or both. If data rates beyond those offered by 8-PSK are required, it is more usual to move to QAM since it achieves a greater distance between adjacent points in the I-Q plane by distributing the points more evenly. The complicating factor is that the points are no longer all the same amplitude and so the demodulator must now correctly detect both phase and amplitude, rather than just phase. 64-QAM and 256-QAM are often used in digital cable television and cable modem applications. In the United States, 64-QAM and 256-QAM are the mandated modulation schemes for digital cable (see QAM tuner) as standardized by the SCTE in the standard ANSI/SCTE 07 2013. In the UK, 64-QAM is used for digital terrestrial television (Freeview) whilst 256-QAM is used for Freeview-HD.

Blocks used in GNU radio:

1. **Options Block:** Sets general properties for the flowgraph.

Parameters:

Title: The name of the flowgraph(QAM-16).

Output Language: Python.

Generate Options: Specifies the GUI type (QT GUI in this case).

2. **Variable Blocks:** Define key parameters used in the flowgraph:

samp_rate: Sets the sample rate (1M samples per second).

mod_order: Determines the number of bits per symbol.

4. **Random Source:** Generates random bytes (representing digital data to be transmitted).

Parameters:

Minimum/Maximum: Range of random values (0 to 255).

Num Samples: Number of samples (1k).

Repeat: Ensures data repeats for continuous transmission.

5. **Packed to Unpacked:** Converts bytes into bits for modulation.

Parameters:

Bits per Chunk: Sets the number of bits per symbol (4 for QPSK).

Endianness: Specifies the bit order (MSB first).

6. **Chunks to Symbols**: Maps input bits to complex symbols based on the selected modulation scheme.

7. **Noise Source:** Adds Gaussian noise to the modulated signal to simulate channel interference.

Parameters:

Noise Type: Gaussian.

Amplitude: Controls noise strength (currently 0, i.e., no noise).

8. **Signal Source:** Generates a carrier signal (cosine waveform) for modulation.

Parameters:

Sample Rate: Matches the samp_rate (1M).

Waveform: Cosine wave.

Frequency: Carrier frequency (0 Hz in this setup).

Amplitude: Signal amplitude (1).

9. **Multiply:** Modulates the carrier with the digital signal.

10. **Add:** Combines the modulated signal with noise.

11. **Throttle:** Limits the processing speed to the specified sample rate (1M) to avoid excessive CPU usage.

12. **QT GUI Constellation Sink:** Displays the received signal's constellation diagram.

Parameters:

Number of Points: Total displayed points (1,024).

Autoscale: Disabled (fixed scale).

13. **QT GUI Time Sink:** Visualizes the time-domain waveform of the signal.

Parameters:

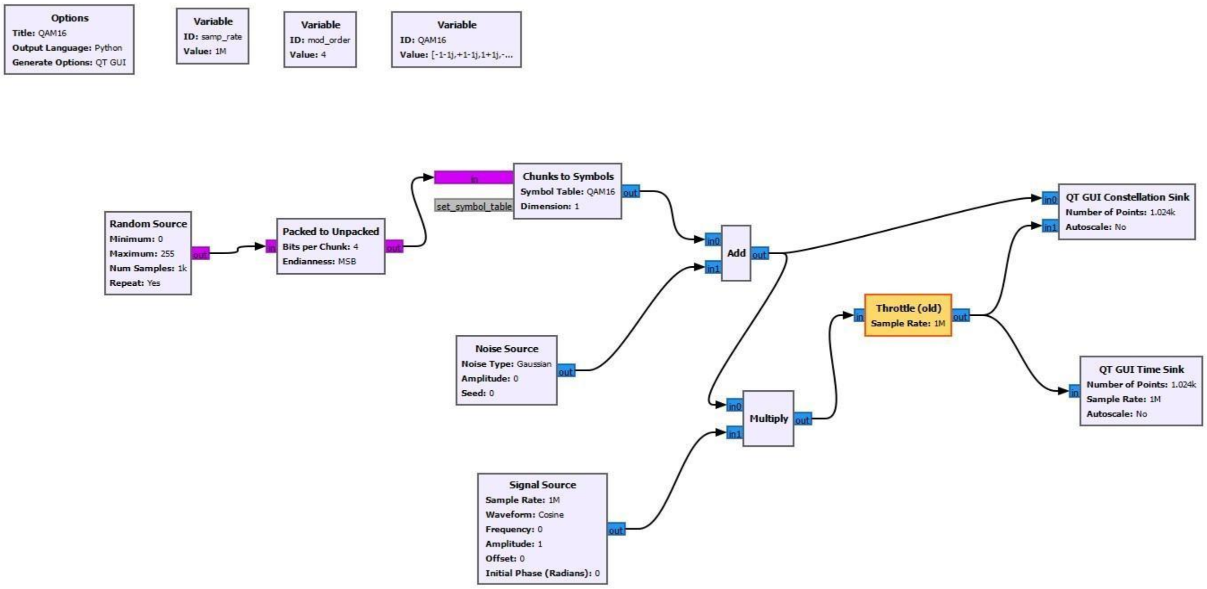
Number of Points: 1,024 samples.

Sample Rate: Matches the samp_rate.

Procedure:

1. Click on the GNU Radio icon to launch GNU Radio companion.
2. Go to file option then select new→ QT GUI. And save the file.
3. Click on Variable block and set the sample rate 1M (default value 32k).
4. In order to design QAM we need various block such as random source, signal source, packed to unpacked, chunks to symbols, noise source, multiply, add, throttle, QT GUI Time sink and QT GUI Constellation sink.
5. Location of random source block is
   Core →waveform generator→ random source
   Minimum=0, Maximum=255, Num of samples=1k
6. For packed to unpacked
   Core → byte operator→ packed to unpacked
7. For Chunks to symbols
   Core → Symbol coding→ Chunks to symbols
8. Location of noise source block is
   Core →waveform generator→ noise source
9. We need one add block. Location of add is
   Core → math operator→ add
10. We need one throttle block. Location of throttle is
    Core → mise → throttle
11. We need one multiply block. Location of multiply is
    Core → math operator → multiply
    Set the output type: float
12. We need QT GUI Constellation sink block to see the constellation. Location of QT GUI Constellation sink is
    Core → instrumentation → QT→ QT GUI Constellation sink
    Set the output type: float
13. We need one QT GUI Time sink block to see the output time domain response. Location of QT GUI Time sink is
    Core → instrumentation → QT→ QT GUI Time sink
    Set the output type: float
14. Take care to set the input/output type to float in all the blocks.
15. Now connect the different blocks as shown in Figure 2.1.
16. Click on run option.

Block Diagram:



Options
Title: QAM16
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

Variable
ID: mod_order
Value: 4

Variable
ID: QAM16
Value: [-1-1j,+1-1j,1+1j,-...

Random Source
Minimum: 0
Maximum: 255
Num Samples: 1k
Repeat: Yes

Packed to Unpacked
Bits per Chunk: 4
Endianness: MSB

Chunks to Symbols
Symbol Table: QAM16
Dimension: 1

Noise Source
Noise Type: Gaussian
Amplitude: 0
Seed: 0

Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: 0
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Add

Multiply

Throttle (old)
Sample Rate: 1M

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 1M
Autoscale: No

Figure 2.1. Block diagram(GNU)

Results:

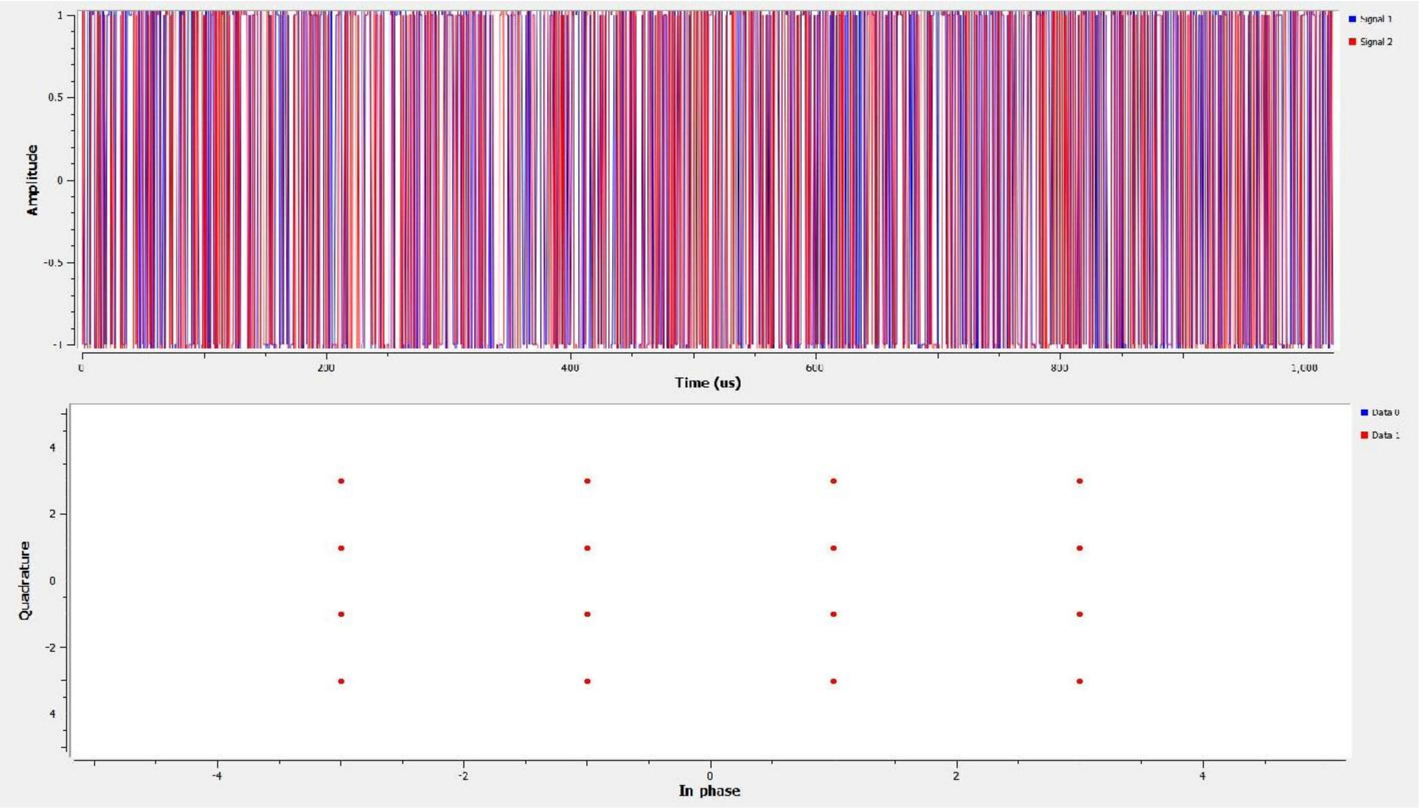


Figure 2.2. (a)Time domain

(b)Constellation diagram

Conclusion:

In this experiment, we implemented and analyzed Quadrature Amplitude Modulation (QAM) using GNU Radio, a powerful software-defined radio (SDR) tool. The process demonstrated the key concepts of digital modulation and provided insights into real-world communication systems.

EXPERIMENT 3

Aim: To perform audio transmission and reception using GNU radio.

Software: GNU radio.

Theory:

In the present era, all the communication processes are done on hardware but the combination of the proper hardware can create errors in various signal processing. The cost of the hardware is very high and also not easily portable. So SDR is the effective solution for the high cost and hardware-based radios that are less flexible. The modulation schemes are implemented to show the effectiveness of SDR based on mathematical models. Good performance has been achieved in wireless communication through the combination of USRP and GNU Radio.

NI USRP-2922

The USRP-2922 is a tunable RF transceiver with a high-speed analog-to-digital converter and digital-to-analog converter for streaming baseband I and Q signals to a host PC over 1 Gigabit Ethernet.

Specifications of USRP 2922:

Frequency range	400 MHz to 4.4 GHz
Frequency step	< 1 kHz
Frequency accuracy	2.5 ppm
Gain step	0.5 dB
DAC	2 channels, 400 MS/s, 16 bit
Analog-to-digital converter (ADC)	2 channels, 100 MS/s, 14 bit

Blocks of GNU radio used:

WAV File Source: Reads audio data from a .wav file and outputs it as a stream.

File: Path to the .wav file.

Repeat: If set to "Yes", it continuously loops the audio.

UHD: USRP Source: Interfaces with a USRP (Universal Software Radio Peripheral) to receive radio signals.

Sync: Synchronization mode (e.g., "PC Clock" or "Unknown PPS").

Sample Rate: Sets the sampling rate in samples per second.

Center Frequency (Hz): The frequency to tune the USRP’s receiver.

Gain Value: Amplification applied to the received signal.

Antenna: The antenna port used (e.g., RX2).

Multiply Const: Scales the amplitude of the input signal by a constant value.

Constant: The scaling factor (e.g., 400m).

WBFM Transmit: Modulates an input audio signal into a wideband FM signal.

Audio Rate: The sample rate of the input audio signal.

Quadrature Rate: The rate for quadrature sampling of the FM signal.

Max Deviation: The maximum frequency deviation for FM modulation.

Preemphasis High Corner Freq: Frequency for preemphasis filtering.

Decimating FIR Filter: Filters the input signal and reduces the sample rate (decimation).

Decimation: The factor by which the input sample rate is reduced.

Taps: The filter coefficients.

WBFM Receive: Demodulates a wideband FM signal back to audio.

Quadrature Rate: Sample rate of the incoming FM signal.

Audio Decimation: The decimation factor applied to convert the FM signal to audio.

AGC (Automatic Gain Control): Dynamically adjusts the gain of the signal to maintain a consistent amplitude.

Rate: The speed at which gain is adjusted.

Reference: Target signal level.

Max Gain: The maximum allowable gain.

Complex to Float / Float to Complex: Converts between real (float) and complex (I/Q) signal representations.

Usage: Necessary when transitioning between complex signal processing and real (audio) signals.

UHD: USRP Sink: Sends a processed signal to a USRP for transmission.

Sync: Synchronization mode (e.g., "PC Clock").

Sample Rate : Sampling rate for transmitted data.

Center Frequency (Hz): Transmission frequency.

Gain Value: Signal amplification for transmission.

Antenna: Port used (e.g., TVRX).

Audio Sink: The output audio signal to the computer's sound card for playback.

Sample Sink: The sample rate of the audio stream (e.g., 48 kHz).

QT GUI Sink: Visualizes the output signal in the time or frequency domain using a graphical display.

FFT Size: Number of points used in the FFT computation.

Center Frequency: Frequency at the center of the spectrum.

Bandwidth: Frequency range displayed.

Procedure:

1. Download the NI Package Manager and make sure USRP 2922 is connected in NI configuration utility.
2. Click on the GNU radio icon to launch GNU Radio companion.
3. Open a new flowgraph (Ctrl+N) and save the file.
4. Now connect the different blocks as shown in figure 3.1.
5. We need one WAV file source block. Location is Core → File operators → wav file source

6. We need one multiply constant block. Location of multiply constant is

   Core → Math operators → Multiply Const

7. Location of WBFM transmit is

   Core → Modulators → WBFM transmit

8. For UHD USRP sink block follow

   Core → UHD → USRP sink

9. For UHD USRP source block

   Core → UHD → USRP source

10. For decimating FIR filter

    Core→ Filters → Decimating FIR filter

11. We need AGC block

    Core → Level Controllers → AGC

12. We need QT GUI Frequency sink block to see the output frequency response.
    Location of QT GUI Frequency sink
    Core→ QT → QT GUI Frequency sink

Set the output type: float

13. We need one QT GUI Time sink block to see the output time

    Core → QT → QT GUI Time sink

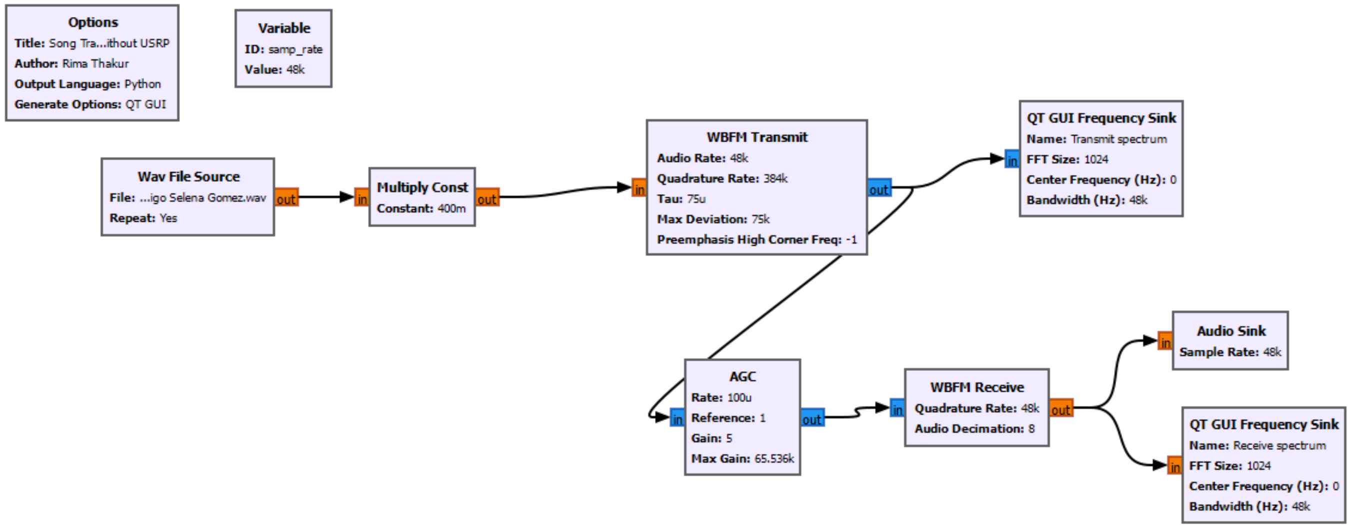
Set the output type: float

14. Give a song in .wav file source. Make sure the song is in .wav format.

15. Click on run option.

Block Diagram:

Without USRP:



Options
Title: Song Tra...ithout USRP
Author: Rima Thakur
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 48k

Wav File Source
File: ...igo Selena Gomez.wav
Repeat: Yes

Multiply Const
Constant: 400m

WBFM Transmit
Audio Rate: 48k
Quadrature Rate: 384k
Tau: 75u
Max Deviation: 75k
Preemphasis High Corner Freq: -1

QT GUI Frequency Sink
Name: Transmit spectrum
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

AGC
Rate: 100u
Reference: 1
Gain: 5
Max Gain: 65.536k

WBFM Receive
Quadrature Rate: 48k
Audio Decimation: 8

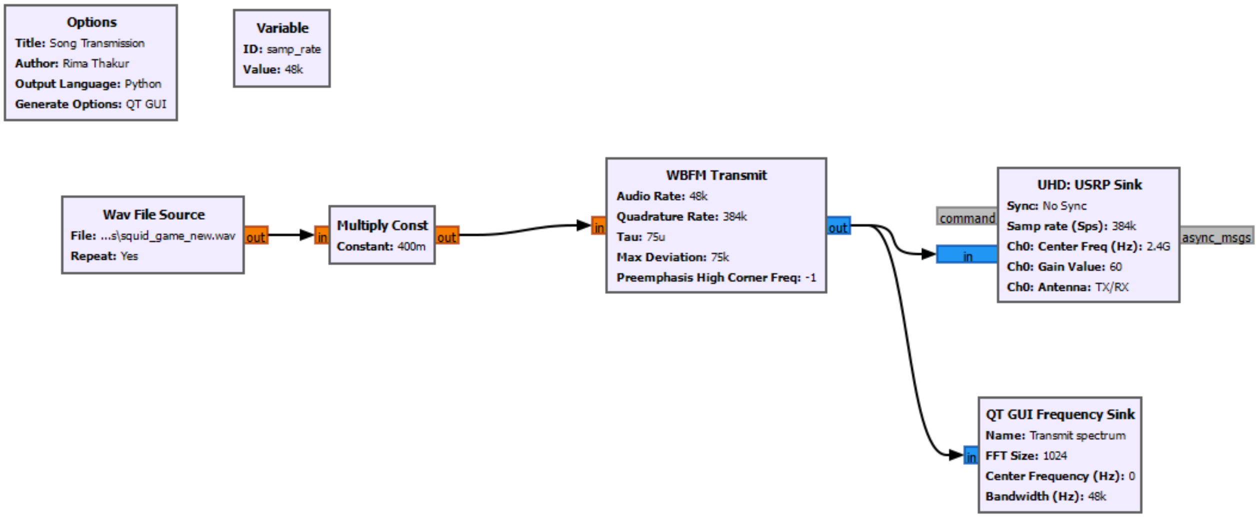
Audio Sink
Sample Rate: 48k

QT GUI Frequency Sink
Name: Receive spectrum
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

Fig.3.1: Without USRP Block diagram

With USRP:

Transmitter:



Options
Title: Song Transmission
Author: Rima Thakur
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 48k

Wav File Source
File: ...s\squid_game_new.wav
Repeat: Yes

Multiply Const
Constant: 400m

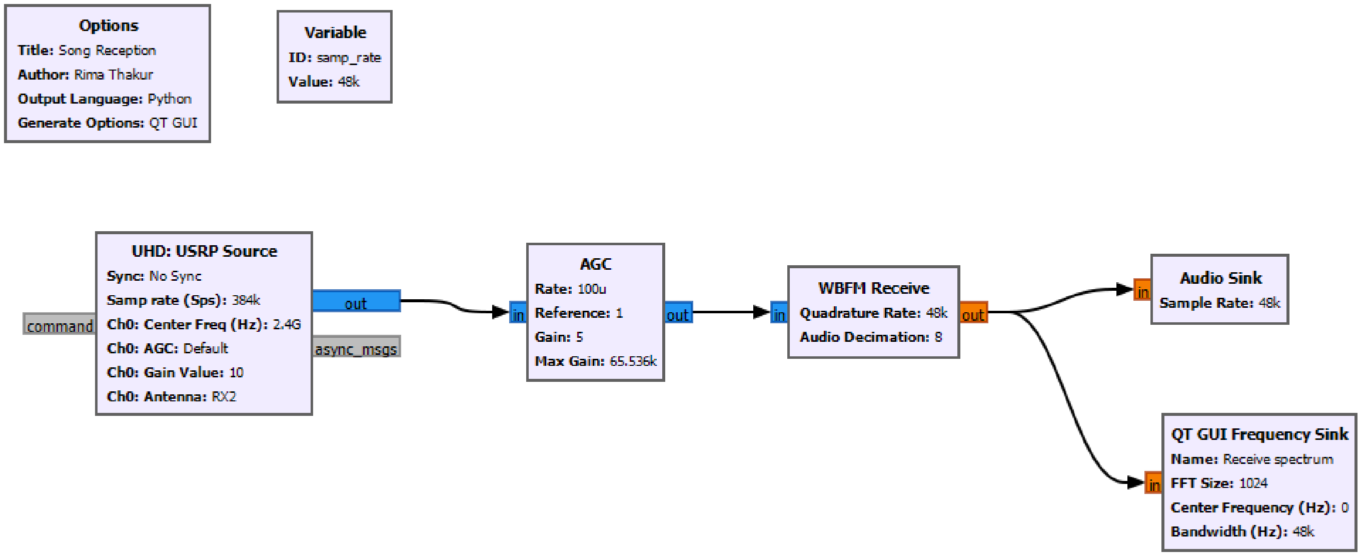
WBFM Transmit
Audio Rate: 48k
Quadrature Rate: 384k
Tau: 75u
Max Deviation: 75k
Preemphasis High Corner Freq: -1

UHD: USRP Sink
Sync: No Sync
Samp rate (Sps): 384k
Ch0: Center Freq (Hz): 2.4G
Ch0: Gain Value: 60
Ch0: Antenna: TX/RX

QT GUI Frequency Sink
Name: Transmit spectrum
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

Fig.3.2: With USRP Transmitter

Receiver:



Options
Title: Song Reception
Author: Rima Thakur
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 48k

UHD: USRP Source
Sync: No Sync
Samp rate (Sps): 384k
Ch0: Center Freq (Hz): 2.4G
Ch0: AGC: Default
Ch0: Gain Value: 10
Ch0: Antenna: RX2

AGC
Rate: 100u
Reference: 1
Gain: 5
Max Gain: 65.536k

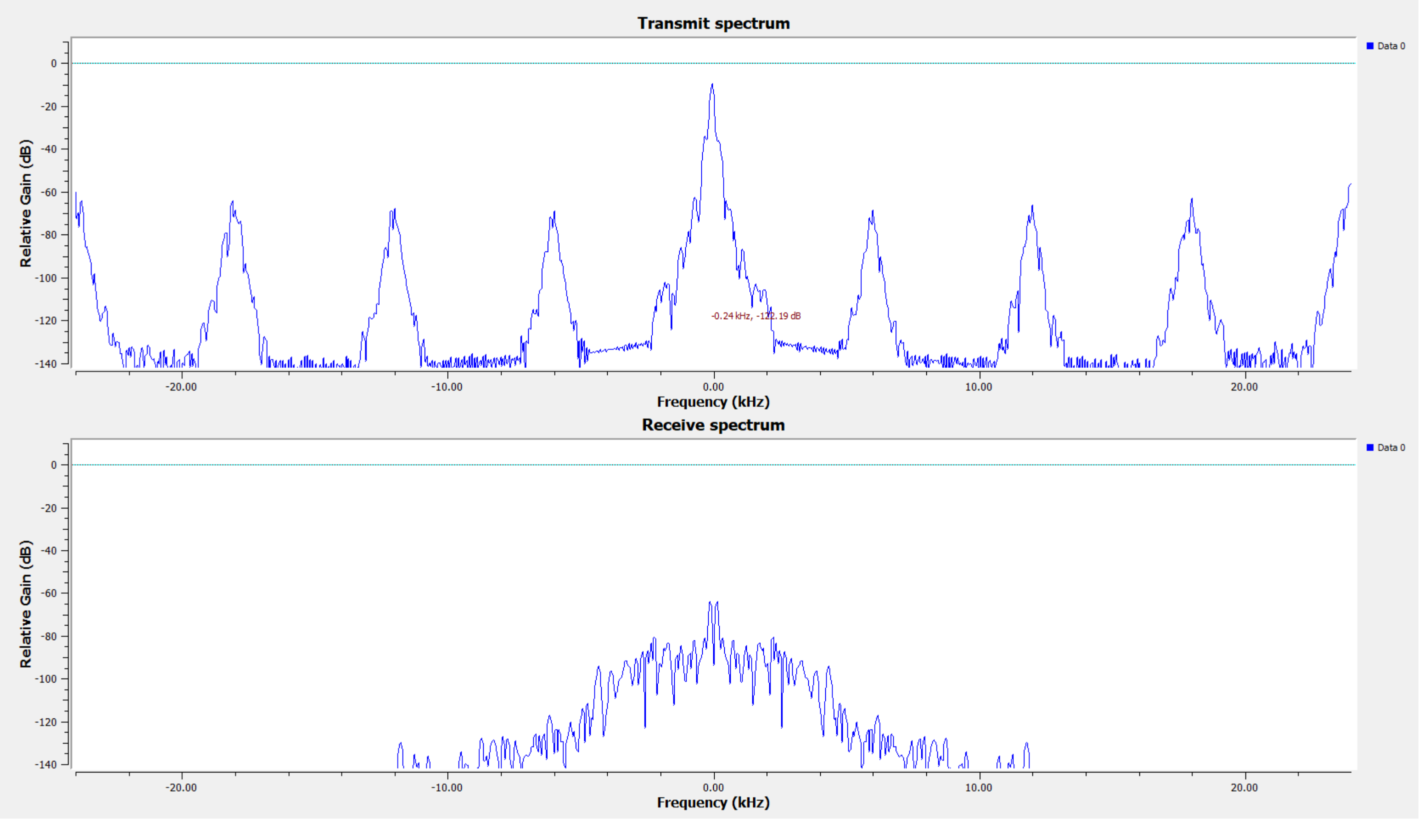
WBFM Receive
Quadrature Rate: 48k
Audio Decimation: 8

Audio Sink
Sample Rate: 48k

QT GUI Frequency Sink
Name: Receive spectrum
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

Fig.3.3: With USRP Receiver

Results:



Transmit spectrum

Receive spectrum

Fig.3.4: Waveforms

Conclusions:

The experiment using the USRP 2922 and GNU Radio with WBFM modulation successfully demonstrated audio transmission and reception. The system provided clear audio quality with effective modulation and demodulation. The USRP 2922 enabled stable transmission, while GNU Radio offered flexibility for real-time processing. The key outcomes included successful signal handling, low latency, and reliable performance.

EXPERIMENT 4

Aim: To perform real time audio transmission and reception using GNU radio.

Software: GNU radio.

Theory:

Real-time audio transmission and reception refer to the process of sending audio signals wirelessly from a transmitter to a receiver without noticeable delay. This is commonly used in radio broadcasting, VoIP, live communication, and wireless audio systems. The primary goal is to ensure that the transmitted audio is received, processed, and played back in real time, allowing seamless communication.

Blocks of GNU radio used:

1) Audio Source

Captures audio from the system.

Sample Rate: 64k.

2) Multiply Const

Multiplies the input audio by 400m (scales up the amplitude for better transmission).

3) QT GUI Time Sink

Shows the audio waveform before modulation.

Number of Points: 1.024k.

Sample Rate: 64k.

4) QT GUI Frequency Sink

Shows the frequency content of the input audio.

FFT Size: 1024.

Center Frequency: 0 Hz.

Bandwidth: 64k.

5) WBFM Transmit

Modulates the input audio into a Wideband FM signal.

Key Parameters:

Audio Rate: 64k.

Quadrature Rate: 1.024M → Intermediate sampling rate.

Tau: 75u → Pre-emphasis filter parameter.

Max Deviation: 75kHz → Peak frequency deviation.

Preemphasis High Corner Freq: -1.

6) Complex to Float

Converts the modulated signal from complex format to real (float) format for visualization.

7) UHD: USRP Sink

Transmits the FM-modulated signal through the USRP.

Key Parameters:

Sync: No Sync (free-running mode).

Sample Rate: 64k.

Center Frequency: 3.5 GHz.

Gain: 60 dB.

Antenna: TX/RX.

Bandwidth: 40M.

8) QT GUI Time Sink

Displays the modulated signal before transmission.

Number of Points: 1.024k.

Sample Rate: 64k.

9) QT GUI Frequency Sink

Shows the spectrum of the FM-modulated signal before transmission.

FFT Size: 1024.

Center Frequency: 0 Hz.

Bandwidth: 64k.

10) UHD: USRP Source

This block receives signals from the USRP hardware.

Key Parameters:

Sync: "No Sync" → The receiver runs independently without external synchronization.

Sample Rate : 64k → Samples per second.

Center Frequency (Hz): 3.5 GHz → The RF frequency the USRP is tuned to.

Gain Value: 60 dB → Amplifies the received signal.

Antenna: RX2 → The antenna port used for reception.

Bandwidth (Hz): 40M → Specifies the bandwidth of the received signal.

11) Decimating FIR Filter

A Finite Impulse Response (FIR) filter that also performs decimation (reducing the sample rate).

Decimation factor: 1 (meaning no reduction in sample rate here, but can be used for filtering purposes).

Taps: 13.9264u (filter coefficients used to shape the signal).

12) AGC (Automatic Gain Control)

Automatically adjusts the signal gain to maintain a stable amplitude.

Key Parameters:

Rate: 100u → Speed at which gain is adjusted.

Reference: 1 → Target signal power.

Gain: 1 → Initial gain value.

Max Gain: 65.536k → The upper limit of gain.

13) WBFM Receive

Converts the received FM signal into an audio signal.

Key Parameters:

Quadrature Rate: 1.024M → Intermediate frequency sample rate.

Audio Decimation: 16 → Reduces sample rate for audio output.

14) Audio Sink (Output to Speakers)

Sends the final audio signal to the computer's speakers.

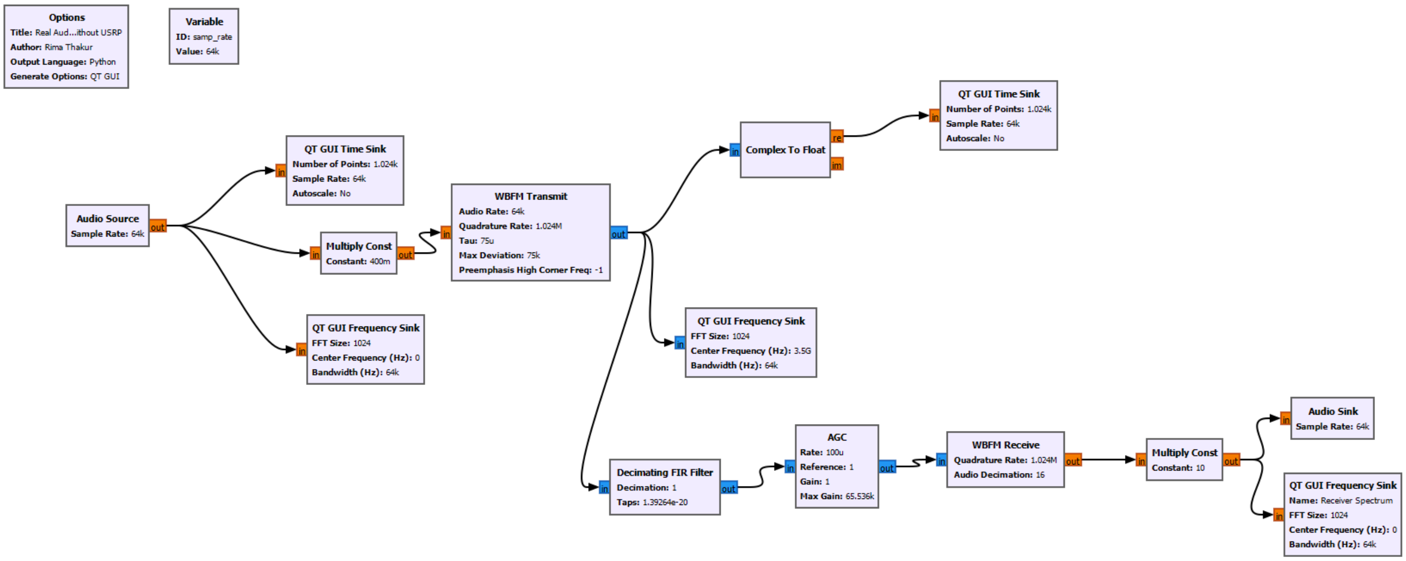
Sample Rate: 64k.

Procedure:

1. Download the NI Package Manager and make sure USRP 2922 is connected in NI configuration utility.
2. Click on the GNU Radio icon to launch GNU Radio companion.
3. Go to file option then select new→ QT GUI. And save the file.
4. Now connect the different blocks as shown in Figure 4.1. and 4.2 for transmission and reception respectively.
5. We need one audio source block. Location is
   Core → Audio source.
6. We need one multiply constant block. Location of multiply constant is
   Core → math operator → multiply constant
   Set the output type: float
7. Location of WBFM transmit is
   Core → Modulators → WBFM transmit
8. For UHD USRP sink follow
   Core → UHD→ USRP sink
9. For UHD USRP source follow
   Core → UHD→ USRP source
10. Core → Filters → Decimating FIR filter
11. Core → Level Controllers → AGC
12. We need QT GUI Frequency sink block to see the output frequency response. Location of QT GUI Frequency sink is
    Core → instrumentation → QT→ QT GUI Frequency sink
    Set the output type: float
13. We need one QT GUI Time sink block to see the output time domain response. Location of QT GUI Time sink is
    Core → instrumentation → QT→ QT GUI Time sink
    Set the output type: float
14. We need one audio sink block. Location is
    Core → Audio sink
15. Click on run option.

Block Diagram:

Without USRP:



Options
Title: Real Aud...ithout USRP
Author: Rima Thakur
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 64k

Audio Source
Sample Rate: 64k

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 64k
Autoscale: No

Multiply Const
Constant: 400m

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 64k

WBFM Transmit
Audio Rate: 64k
Quadrature Rate: 1.024M
Tau: 75u
Max Deviation: 75k
Preemphasis High Corner Freq: -1

Complex To Float

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 64k
Autoscale: No

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 3.5G
Bandwidth (Hz): 64k

Decimating FIR Filter
Decimation: 1
Taps: 1.39264e-20

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 65.536k

WBFM Receive
Quadrature Rate: 1.024M
Audio Decimation: 16

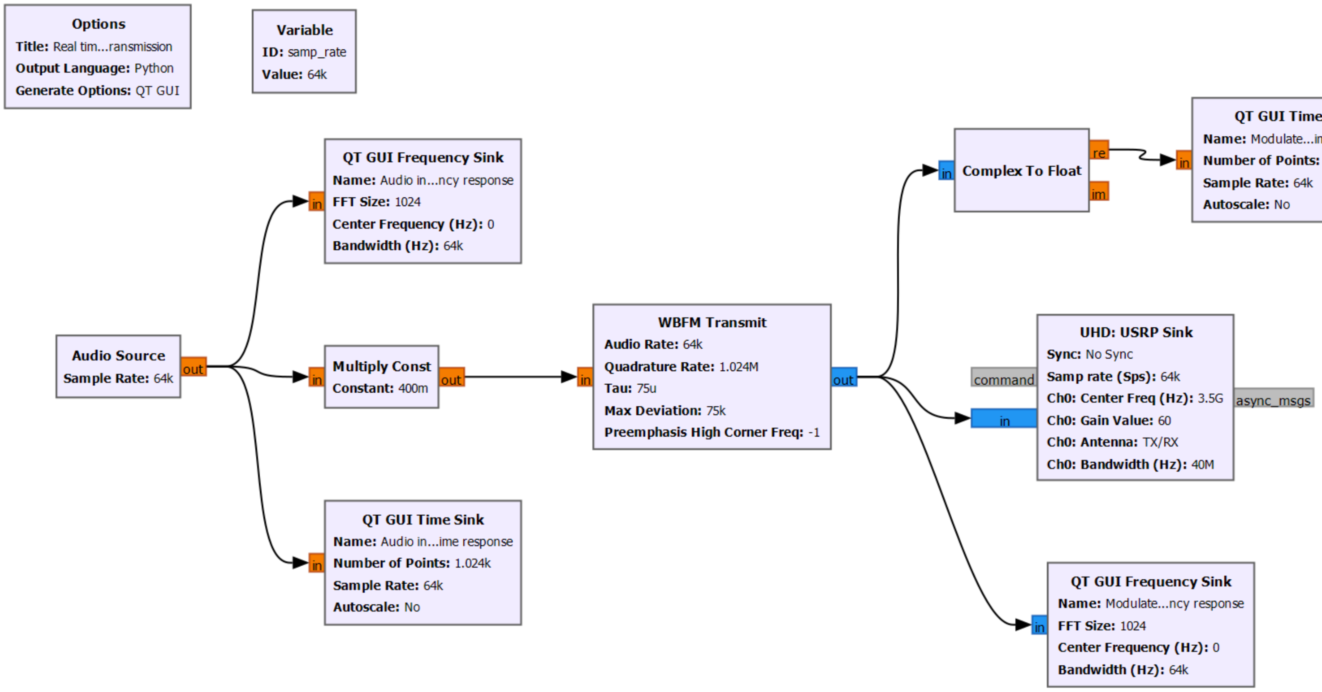
Multiply Const
Constant: 10

Audio Sink
Sample Rate: 64k

QT GUI Frequency Sink
Name: Receiver Spectrum
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 64k

Figure 4.1. Block diagram

With USRP:



Options
Title: Real tim...ransmission
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 64k

Audio Source
Sample Rate: 64k

QT GUI Frequency Sink
Name: Audio in...ncy response
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 64k

Multiply Const
Constant: 400m

QT GUI Time Sink
Name: Audio in...ime response
Number of Points: 1.024k
Sample Rate: 64k
Autoscale: No

WBFM Transmit
Audio Rate: 64k
Quadrature Rate: 1.024M
Tau: 75u
Max Deviation: 75k
Preemphasis High Corner Freq: -1

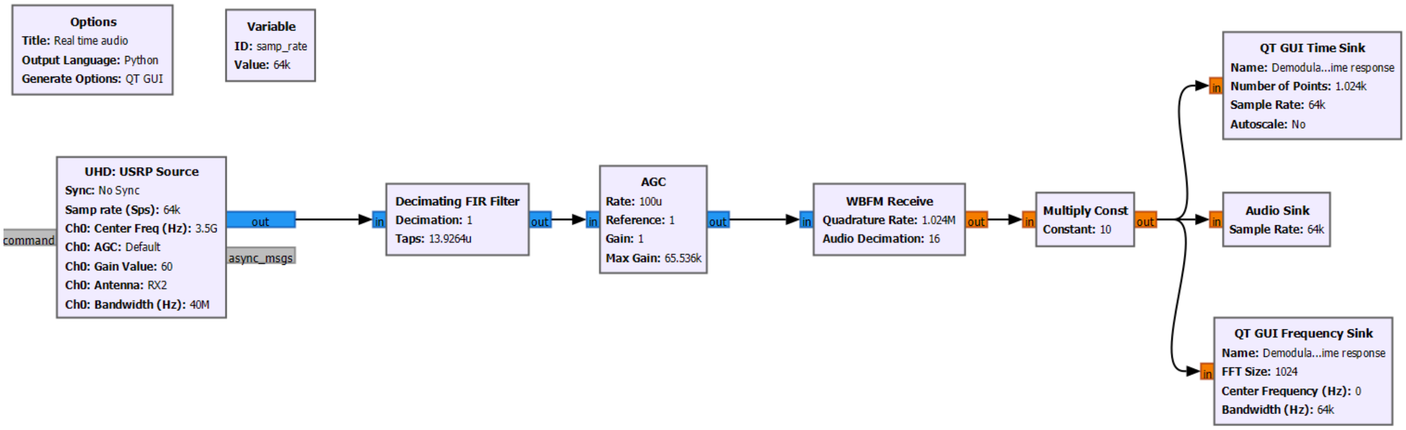
Complex To Float

QT GUI Time Sink
Name: Modulate...ime response
Number of Points: 1.024k
Sample Rate: 64k
Autoscale: No

UHD: USRP Sink
Sync: No Sync
Samp rate (Sps): 64k
Ch0: Center Freq (Hz): 3.5G
Ch0: Gain Value: 60
Ch0: Antenna: TX/RX
Ch0: Bandwidth (Hz): 40M

QT GUI Frequency Sink
Name: Modulate...ncy response
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 64k

Figure 4.2. Transmitter



Options
Title: Real time audio
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 64k

UHD: USRP Source
Sync: No Sync
Samp rate (Sps): 64k
Ch0: Center Freq (Hz): 3.5G
Ch0: AGC: Default
Ch0: Gain Value: 60
Ch0: Antenna: RX2
Ch0: Bandwidth (Hz): 40M

Decimating FIR Filter
Decimation: 1
Taps: 13.9264u

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 65.536k

WBFM Receive
Quadrature Rate: 1.024M
Audio Decimation: 16

Multiply Const
Constant: 10

QT GUI Time Sink
Name: Demodula...ime response
Number of Points: 1.024k
Sample Rate: 64k
Autoscale: No

Audio Sink
Sample Rate: 64k

QT GUI Frequency Sink
Name: Demodula...ime response
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 64k

Figure 4.3. Receiver

Results:

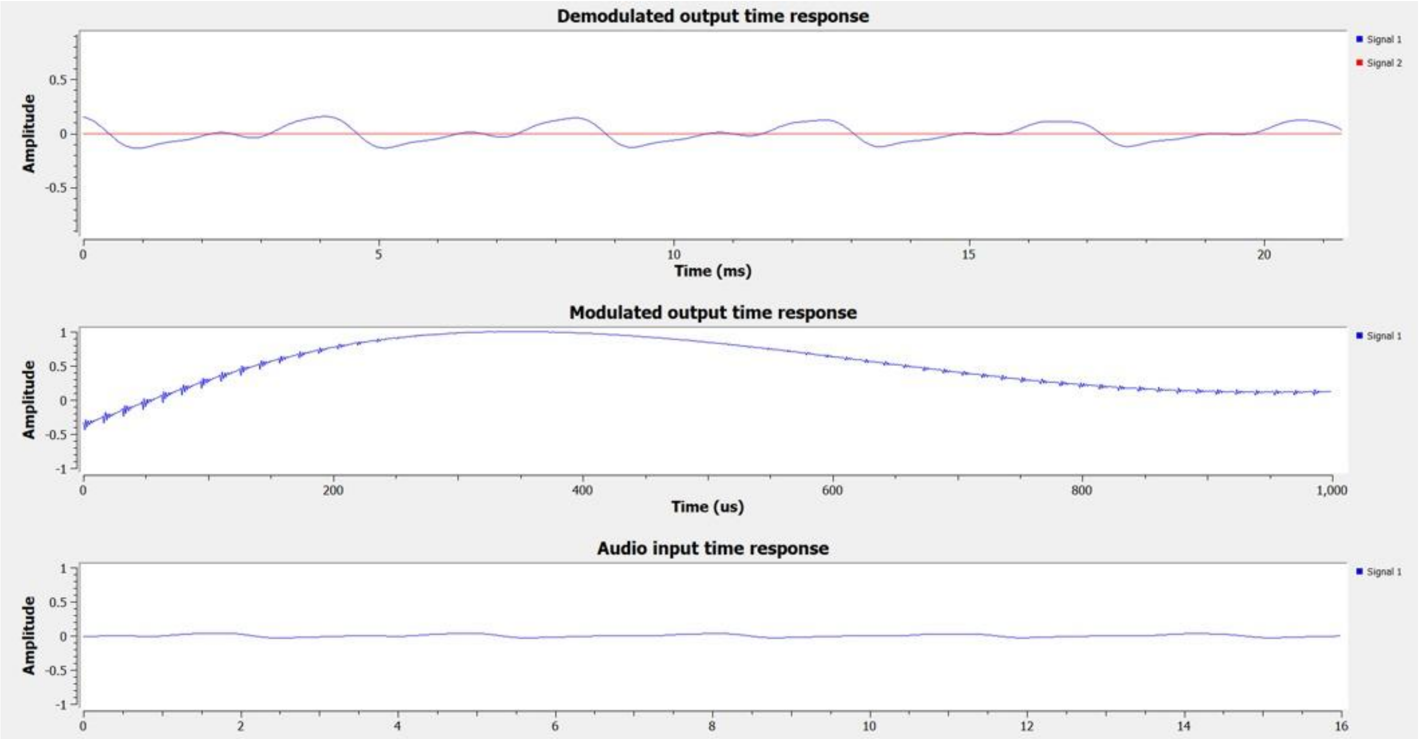


Figure 4.4. Waveforms

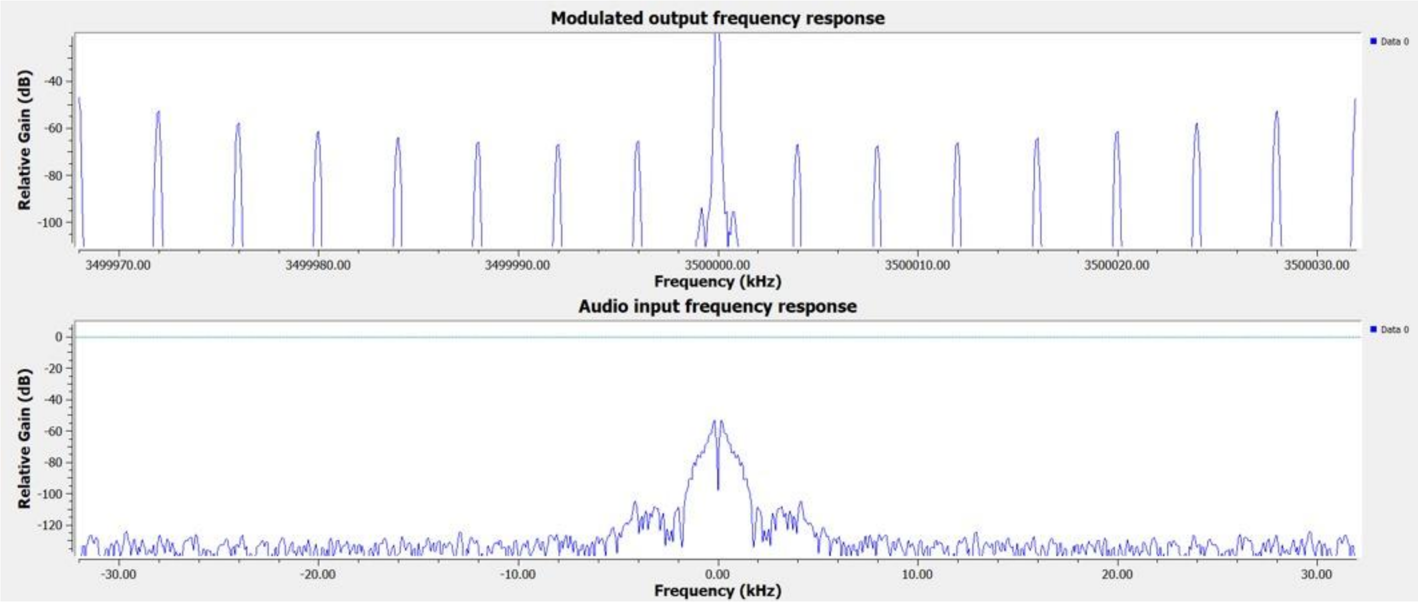


Figure 4.5. Waveforms

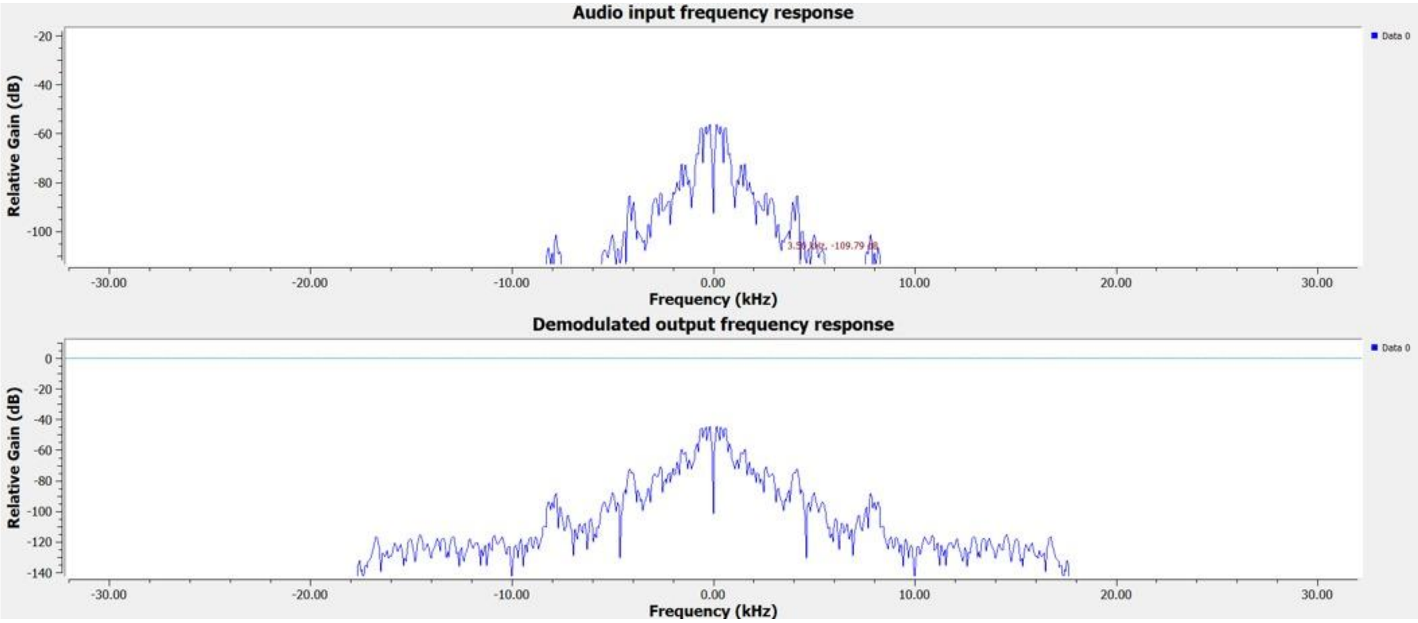


Figure 4.6. Waveforms

Conclusion:

This experiment demonstrates how audio signals can be modulated, transmitted, received, and demodulated in real time, providing insights into signal processing, radio frequency communication, and SDR technology.

Observations:

1. We observed that tap value in decimating FIR filter play an important role for better reconstruction of the real time audio signal.
2. Ensure that $\frac{Quadrature\ rate}{Decimation} = sampling\ rate$ for clarity of voice in WBFM receive block.
3. The sampling rate must be high enough for reconstruction.

EXPERIMENT 5

Aim: To perform image transmission and reception using GNU radio.

Software: GNU radio.

Theory:

GNU Radio can be used for image transmission and reception by first encoding the image data, then transmitting it over the air using a radio waveform, and finally decoding the received signal at the receiver end.

ODFM:

Orthogonal frequency-division multiplexing is a method of data transmission where a single information stream is split among several closely spaced narrowband subchannel frequencies instead of a single Wideband channel frequency. OFDM builds on simpler frequency-division multiplexing (FDM). In FDM, the total data stream is divided into several subchannels, but the frequencies of the subchannels are spaced farther apart so they do not overlap or interfere. With OFDM, the subchannel frequencies are close together and overlapping but are still orthogonal, or separate, in that they are carefully chosen and modulated so that the interference between the subchannels is canceled out.

File Source: Reads the image file (img1.raw) to transmit.

Stream to Tagged Stream: Converts continuous stream into tagged packets of 60 bytes using the packet_len tag key.

OFDM Transmitter

This block modulates the data using OFDM.

- FFT Length: 16
- Cyclic Prefix: 4
- Header Modulation: BPSK
- Payload Modulation: BPSK
- Tags are added for OFDM frame structure.
- Occupied Carriers, Pilot Carriers, Pilot Symbols: Define the OFDM spectrum usage.

Multiply Const: Multiplies the OFDM signal by a constant (50m), possibly to adjust amplitude before transmission.

Complex to Float: Converts complex samples into real and imaginary parts for visualization.

USRP Sink: Sends the modulated signal to the USRP for transmission.

  - Freq: 2.4 GHz (ISM band)
  - Gain: 150
  - Antenna: TX/RX
  - Sample Rate: 3M

USRP Source: Receives the signal from USRP at the same sample rate and frequency.

OFDM Receiver:

- Demodulates the received OFDM signal.
- Matches FFT length and cyclic prefix with transmitter.
- Uses the packet_len tag key to reassemble data.

UChar to Float: Converts received unsigned character data to float format (for visualization).

File Sink: Saves the final received image as received_image2.raw.

Procedure:

1.Open GNU radio.

2.Connect the block as per block diagram.

3.Give .raw file generated using a python code for an jpg image.

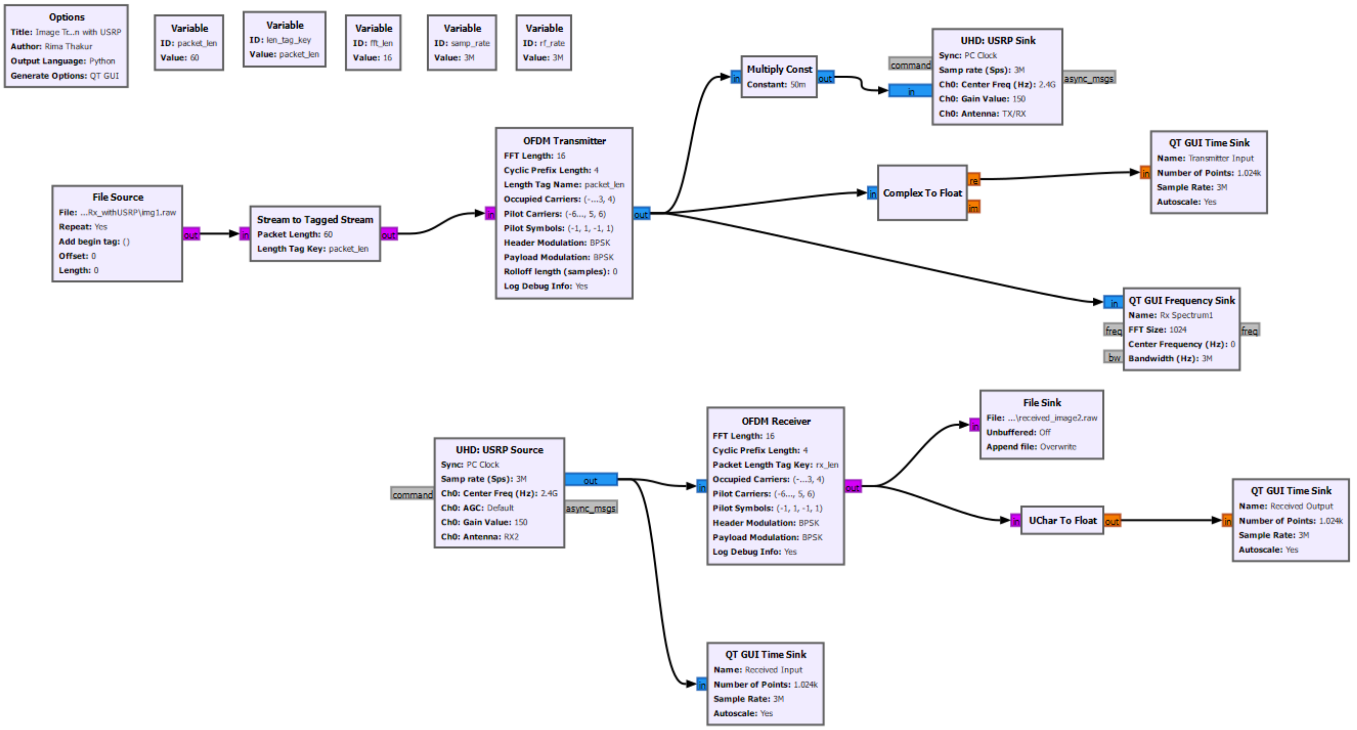
4.Run the grc file.

5.Convert the .raw file back to jpg using python code given.

6.View the reconstructed image.

Block Diagram:

With USRP:



Options
Title: Image Tr...n with USRP
Author: Rima Thakur
Output Language: Python
Generate Options: QT GUI

Variable
ID: packet_len
Value: 60

Variable
ID: len_tag_key
Value: packet_len

Variable
ID: fft_len
Value: 16

Variable
ID: samp_rate
Value: 3M

Variable
ID: rf_rate
Value: 3M

File Source
File: ...Rx_withUSRP\img1.raw
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Stream to Tagged Stream
Packet Length: 60
Length Tag Key: packet_len

OFDM Transmitter
FFT Length: 16
Cyclic Prefix Length: 4
Length Tag Name: packet_len
Occupied Carriers: (-...3, 4)
Pilot Carriers: (-6..., 5, 6)
Pilot Symbols: (-1, 1, -1, 1)
Header Modulation: BPSK
Payload Modulation: BPSK
Rolloff length (samples): 0
Log Debug Info: Yes

Multiply Const
Constant: 50m

UHD: USRP Sink
Sync: PC Clock
Samp rate (Sps): 3M
Ch0: Center Freq (Hz): 2.4G
Ch0: Gain Value: 150
Ch0: Antenna: TX/RX

Complex To Float

QT GUI Time Sink
Name: Transmitter Input
Number of Points: 1.024k
Sample Rate: 3M
Autoscale: Yes

QT GUI Frequency Sink
Name: Rx Spectrum1
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 3M

UHD: USRP Source
Sync: PC Clock
Samp rate (Sps): 3M
Ch0: Center Freq (Hz): 2.4G
Ch0: AGC: Default
Ch0: Gain Value: 150
Ch0: Antenna: RX2

OFDM Receiver
FFT Length: 16
Cyclic Prefix Length: 4
Packet Length Tag Key: rx_len
Occupied Carriers: (-...3, 4)
Pilot Carriers: (-6..., 5, 6)
Pilot Symbols: (-1, 1, -1, 1)
Header Modulation: BPSK
Payload Modulation: BPSK
Log Debug Info: Yes

File Sink
File: ...\received_image2.raw
Unbuffered: Off
Append file: Overwrite

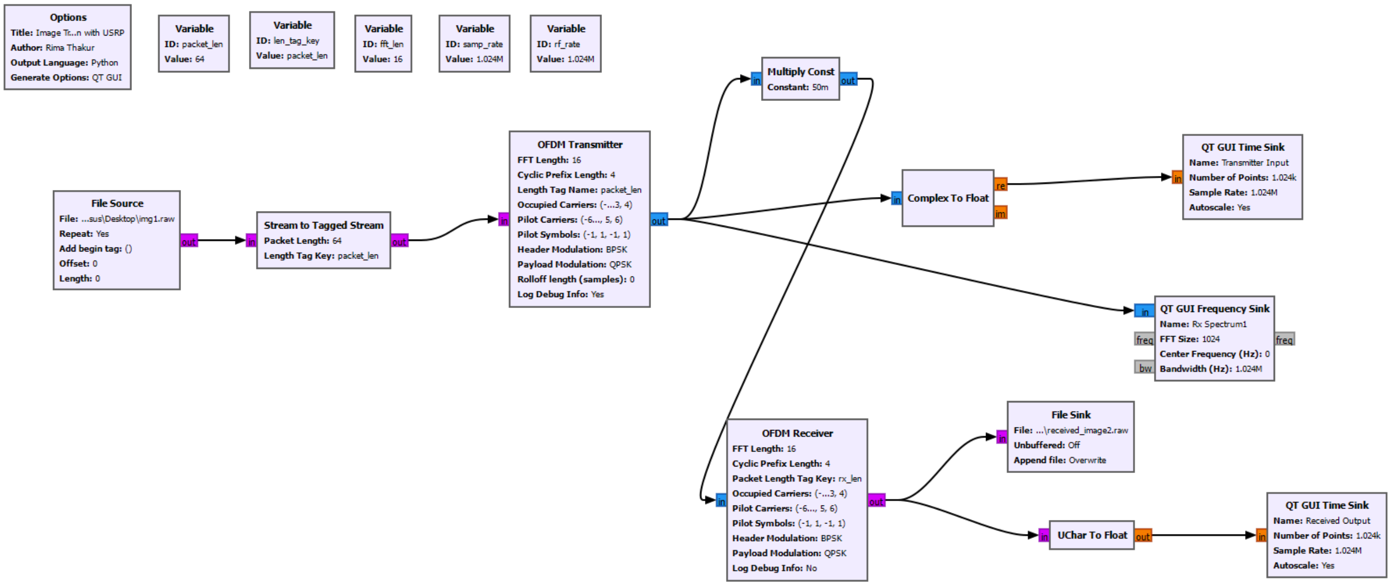
UChar To Float

QT GUI Time Sink
Name: Received Output
Number of Points: 1.024k
Sample Rate: 3M
Autoscale: Yes

QT GUI Time Sink
Name: Received Input
Number of Points: 1.024k
Sample Rate: 3M
Autoscale: Yes

Fig 5.1. Block diagram with USRP

Without USRP:



Options
Title: Image Tr...n with USRP
Author: Rima Thakur
Output Language: Python
Generate Options: QT GUI

Variable
ID: packet_len
Value: 64

Variable
ID: len_tag_key
Value: packet_len

Variable
ID: fft_len
Value: 16

Variable
ID: samp_rate
Value: 1.024M

Variable
ID: rf_rate
Value: 1.024M

File Source
File: ...sus\Desktop\img1.raw
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Stream to Tagged Stream
Packet Length: 64
Length Tag Key: packet_len

OFDM Transmitter
FFT Length: 16
Cyclic Prefix Length: 4
Length Tag Name: packet_len
Occupied Carriers: (-...3, 4)
Pilot Carriers: (-6..., 5, 6)
Pilot Symbols: (-1, 1, -1, 1)
Header Modulation: BPSK
Payload Modulation: QPSK
Rolloff length (samples): 0
Log Debug Info: Yes

Multiply Const
Constant: 50m

Complex To Float

QT GUI Time Sink
Name: Transmitter Input
Number of Points: 1.024k
Sample Rate: 1.024M
Autoscale: Yes

QT GUI Frequency Sink
Name: Rx Spectrum1
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 1.024M

OFDM Receiver
FFT Length: 16
Cyclic Prefix Length: 4
Packet Length Tag Key: rx_len
Occupied Carriers: (-...3, 4)
Pilot Carriers: (-6..., 5, 6)
Pilot Symbols: (-1, 1, -1, 1)
Header Modulation: BPSK
Payload Modulation: QPSK
Log Debug Info: No

File Sink
File: ...\received_image2.raw
Unbuffered: Off
Append file: Overwrite

UChar To Float

QT GUI Time Sink
Name: Received Output
Number of Points: 1.024k
Sample Rate: 1.024M
Autoscale: Yes

Fig 5.2. Block diagram without USRP

Results:

Transmitted image:

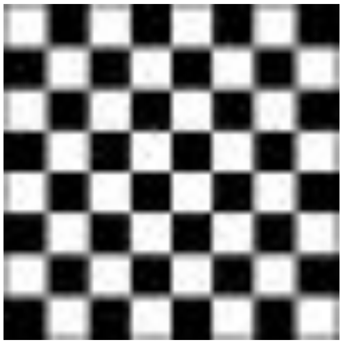


Fig 5.3. Transmitted image of size 50x50

Reconstructed image:

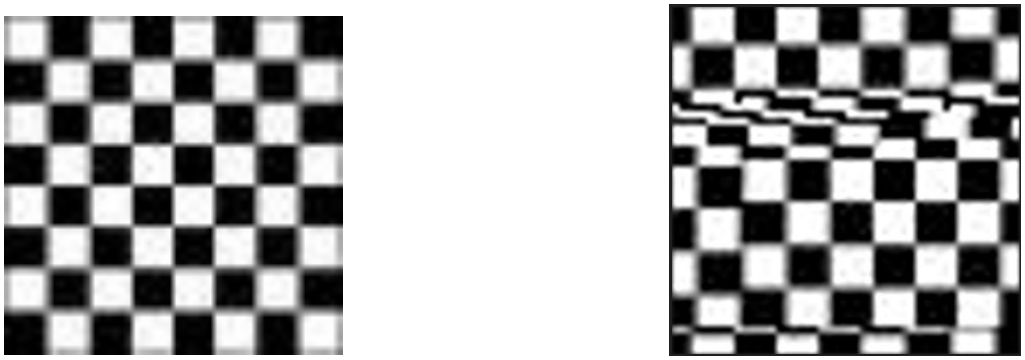


Fig 5.4. (a) Reconstructed image without USRP, (b) Reconstructed image with USRP

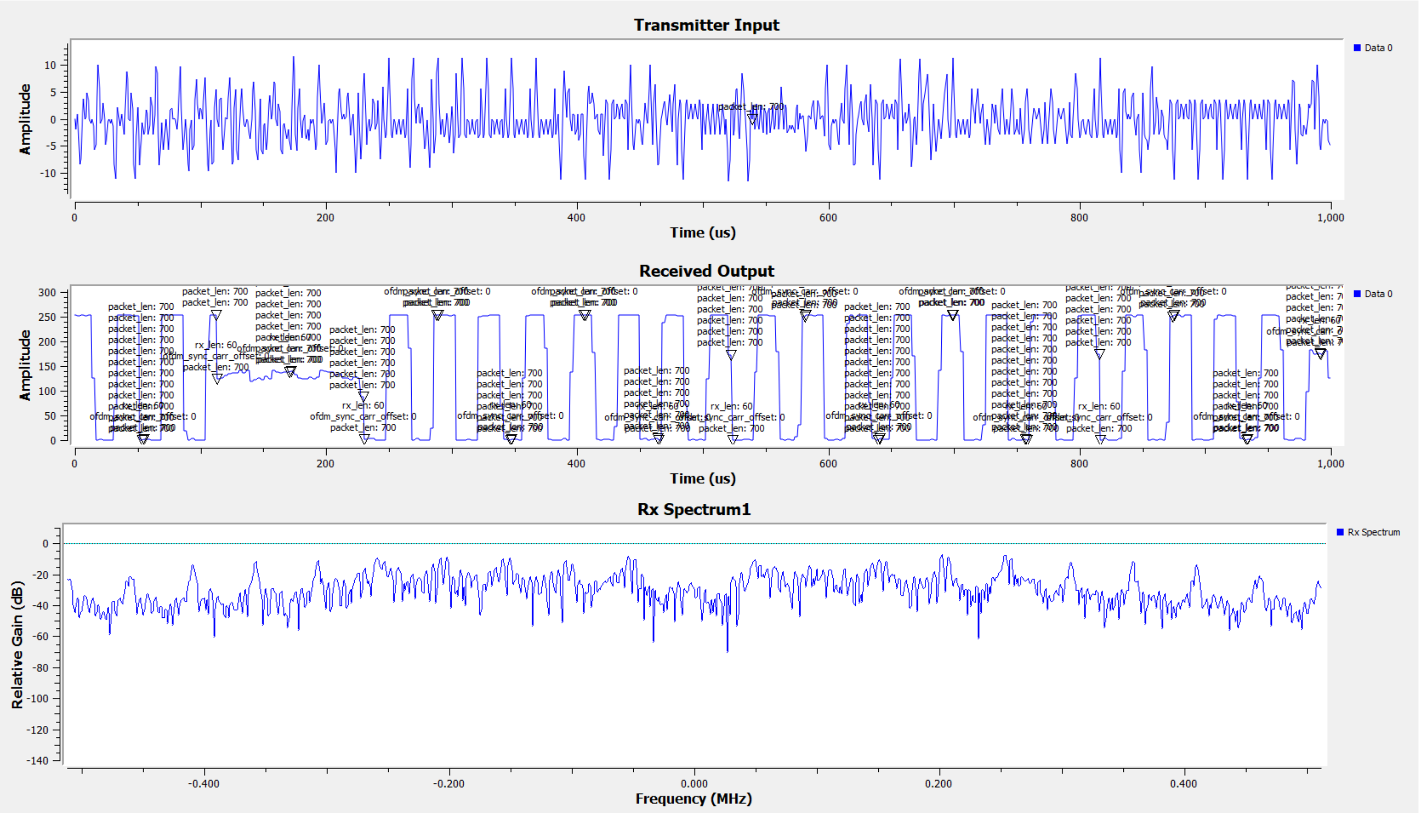
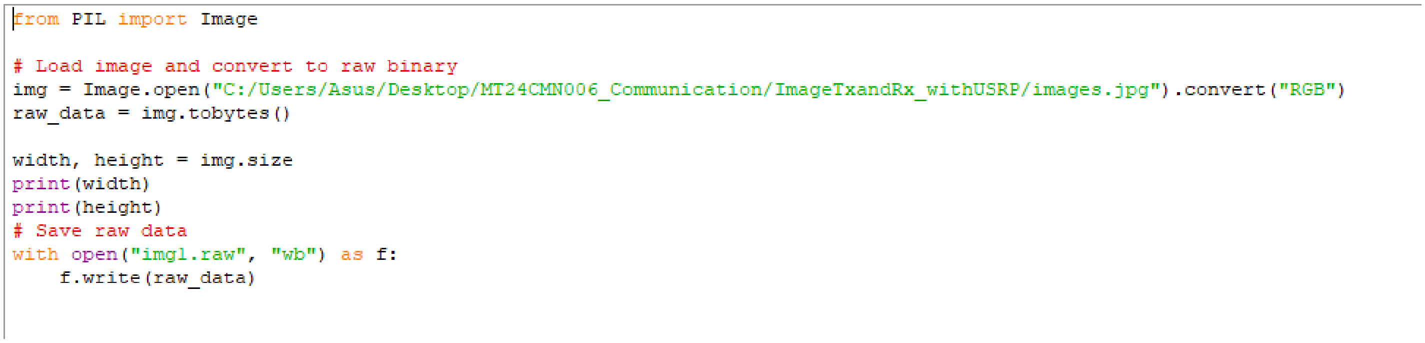


Fig 5.5. a) Waveforms

Observations:

1.We used a separate python code to produce .raw file of .jpg image we are transmitting as we observed that .raw image extension is easier to process by file source and file sink and reconstruction happens for .raw only using USRP.



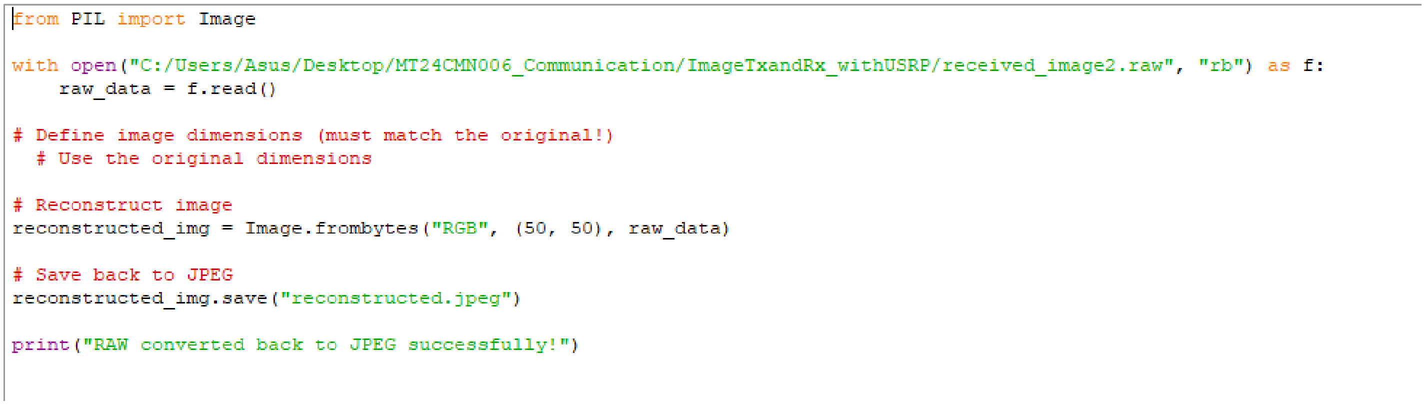
```
from PIL import Image

# Load image and convert to raw binary
img = Image.open("C:/Users/Asus/Desktop/MT24CMN006_Communication/ImageTxandRx_withUSRP/images.jpg").convert("RGB")
raw_data = img.tobytes()

width, height = img.size
print(width)
print(height)
# Save raw data
with open("img1.raw", "wb") as f:
    f.write(raw_data)
```

2.The packet length plays an important role in reconstruction. Select an appropriate packet length.

3.The .raw reconstructed file can be converted back to jpg using a python code:



```
from PIL import Image

with open("C:/Users/Asus/Desktop/MT24CMN006_Communication/ImageTxandRx_withUSRP/received_image2.raw", "rb") as f:
    raw_data = f.read()

# Define image dimensions (must match the original!)
  # Use the original dimensions

# Reconstruct image
reconstructed_img = Image.frombytes("RGB", (50, 50), raw_data)

# Save back to JPEG
reconstructed_img.save("reconstructed.jpeg")

print("RAW converted back to JPEG successfully!")
```

Conclusion:

This GNU Radio flowgraph successfully demonstrates an end-to-end OFDM-based image transmission and reception system using USRP, showcasing how digital data can be reliably packetized, modulated, transmitted over-the-air, and reconstructed—making it a practical example for learning SDR, OFDM, and real-time wireless communication.