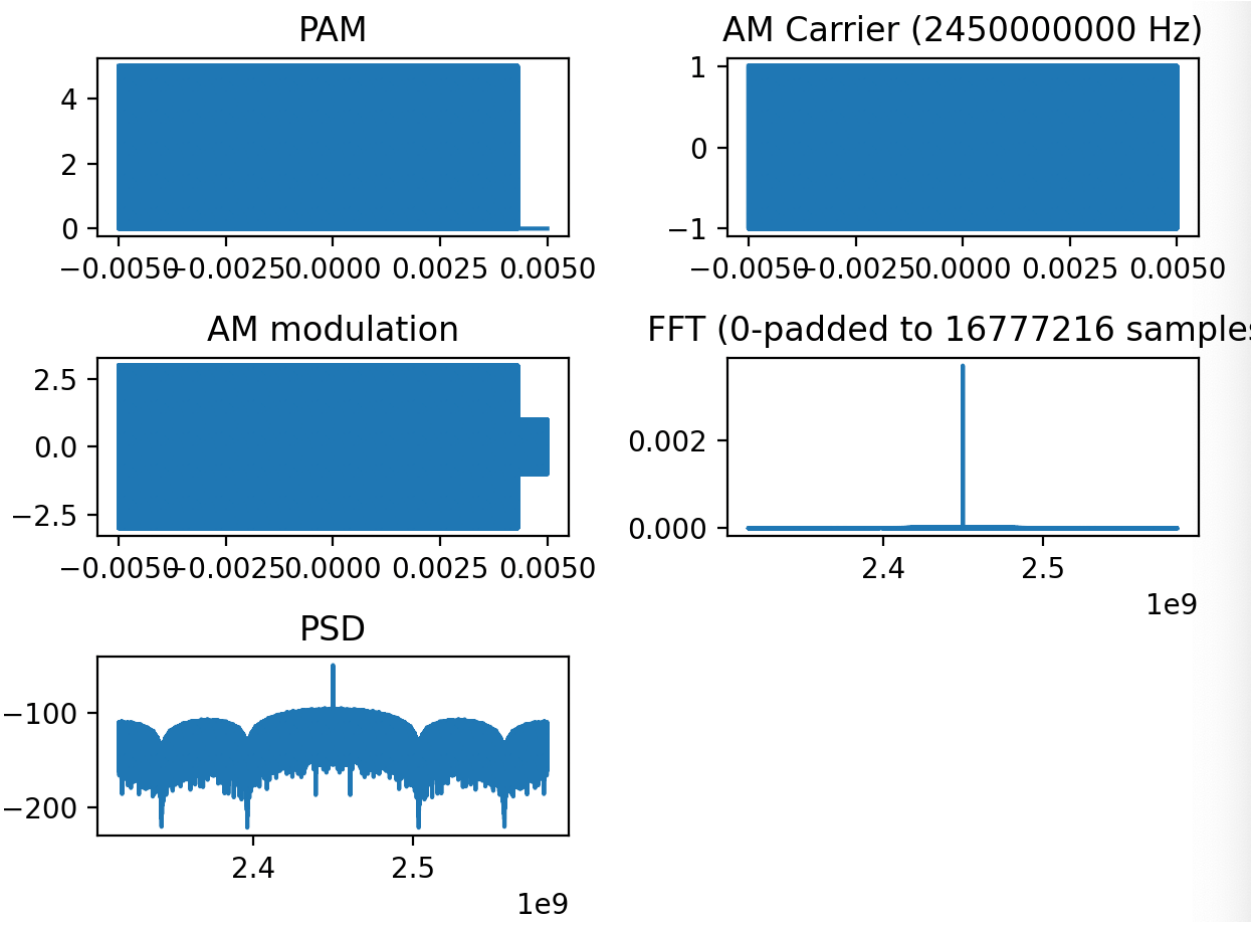
Christopher Austin O'Connell
GTID: 902706132
ECE6390
Homework 6: Pulse Shaping

The following parameters have been used for the transmit:

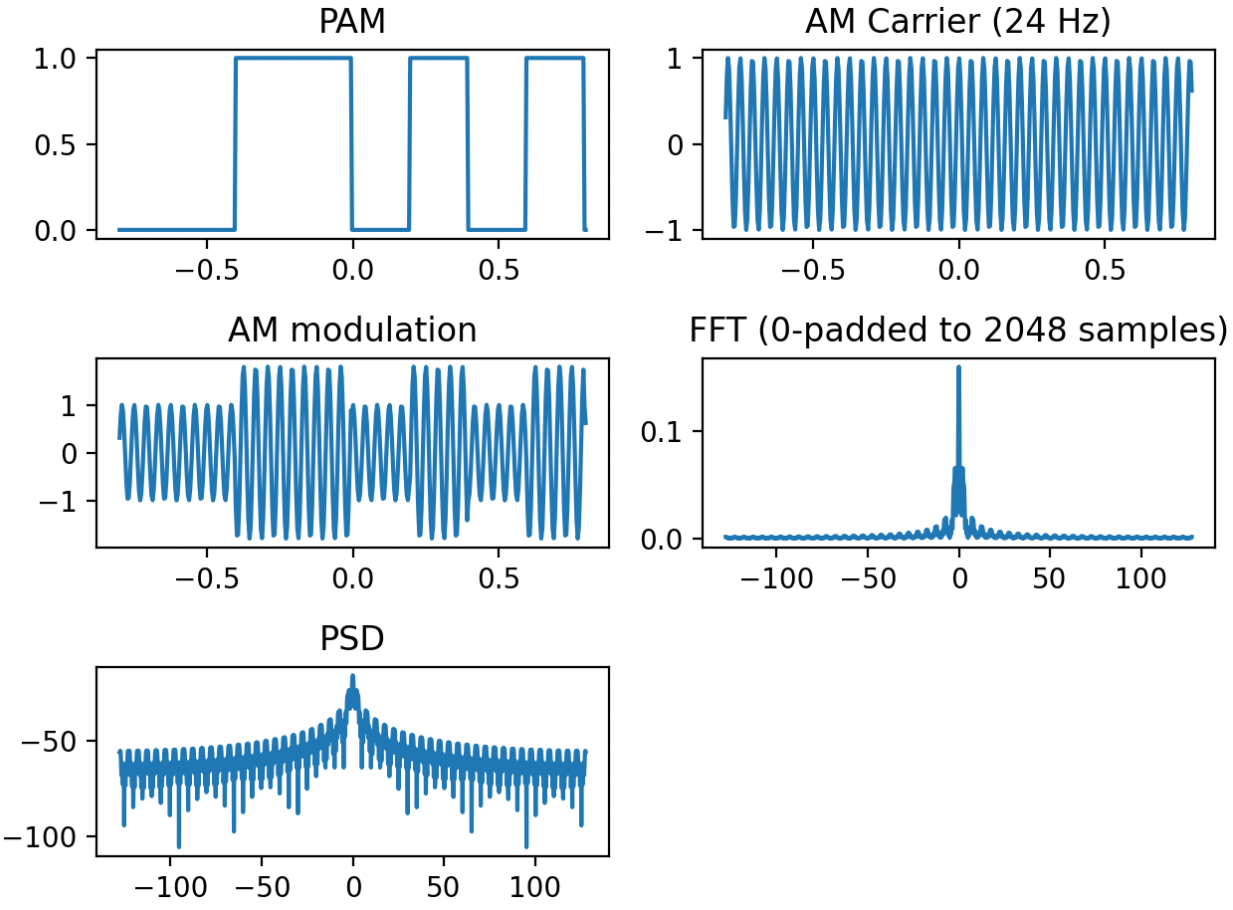
- Pulse Amplitude Modulation parameters
  - Symbol rate of 1 bit per symbol:
    - 1 => 1
    - 0 => 0
  - Pulse width: 20 nanoseconds (corresponds to 50MHz)
  - Pulse repetition interval: 20 nanoseconds
    - Each bit corresponds to a different amplitude, so 100% duty cycle is applicable here to maximize throughput
  - PAM sample rate: 2^27, or 134,217,728 Hz
    - This was selected as the first power of 2 that exceeds the Nyquist Shannon theorem for a frequency higher than 50 MHz. 50MHz is the highest frequency at baseband, corresponding to +/- 50MHz that will then we upconverted to our desired FCC range of 2.4-2.5GHz (100MHz channel bandwidth).
- Amplitude Modulation parameters
  - Carrier frequency: 2.45 GHz
    - Midpoint between 2.4-2.5 GHz (selected because the baseband signal scales from +/- 50MHz
  - Modulation Index = 0.8
    - Arbitrarily selected value close to 1.0 to have a noticeable carrier envelope

Results:
Peak power outside FCC band: 62 dB less than peak value in band.
Bit rate: 50 Mbps
Note: Adhering to digital PAM chipset limitation of 4Tb of time support, the above PAM implementation could be updated to use a symbol rate of 4 bits per symbol, which would increase the overall bitrate to 200 Mbps

Graphs of results:



And here’s some downsampled graphs to show off visual clarity of the PAM and AM implementation:



Source code:

```
import numpy as np
import matplotlib.pyplot as plt

def pam(data):
    pulse_width = 0.00000002 # 20 ns limit
    Tb = 0.00000002 # PRI
    bitrate = 1/Tb

    print(f"bitrate: {int(bitrate)} bps")

    Fs = 2**28 # 1e9 # 100MHz sample rate
    Ts = 1 / Fs # Ts = period (time between each sample)

    samples_per_bit = int(Tb / Ts)
    samples_per_pulse = int(pulse_width / Ts)
    print(f"Samples per bit: {samples_per_bit}. Samples per pulse:
{samples_per_pulse}")

    duration = len(data) * Tb
    t = np.arange(-duration/2, duration/2, Ts) # nof_samples

    # Create PAM signal (single bit per symbol)
    # 0 -> 0
    # 1 -> 1
    pam_signal = np.zeros(len(t))
    for i, bit in enumerate(data):
        start = i * samples_per_bit
        end = start + samples_per_pulse
        pam_signal[start:end] = 1 if bit == 1 else 0

    return t, pam_signal, Fs, duration

# The binary sequence to modulate
# data = np.array([1, 0, 1, 0, 1])
data = np.random.randint(2, size=500000)

# PAM
t, pam_signal, Fs, duration = pam(data)

plt.subplot(3, 2, 1)
plt.plot(t, pam_signal)
plt.title("PAM")

# AM upconvert to 2.45 GHz center frequency
```

```
carrier_amp = 1
carrier_freq = 2.45e9 # 2.45 GHz
modulation_index = 0.4 # ratio of carrier to sideband amplitude (aka carrier envelope)
carrier = carrier_amp * np.cos(2*np.pi*carrier_freq*t)
am_signal = (1 + modulation_index * pam_signal) * carrier

plt.subplot(3, 2, 2)
plt.plot(t, carrier)
plt.title(f"AM Carrier ({int(carrier_freq)} Hz)")

plt.subplot(3, 2, 3)
plt.plot(t, am_signal)
plt.title("AM modulation")

output_signal = pam_signal

# Zero pad before taking FFT
next_power_of_2 = int(np.ceil(np.log2(len(output_signal)))) + 2
target_length = 2**next_power_of_2
deficit = target_length - len(output_signal)
padded_output_signal = np.pad(output_signal, (0, deficit), mode='constant',
constant_values=0)
print(f"Length of padded output: {len(padded_output_signal)}")

# FFT
output_fft = np.fft.fft(padded_output_signal)

# frequency plot points for graphing FFT and PSD
f_axis = carrier_freq + np.arange(-Fs/2, Fs/2, Fs/len(padded_output_signal))

# FFT shifted and scaled for visualization
output_fft_scaled = abs(np.fft.fftshift(output_fft))*(duration)/len(output_fft)

plt.subplot(3, 2, 4)
plt.plot(f_axis, output_fft_scaled)
plt.title(f"FFT (0-padded to {len(padded_output_signal)} samples)")

# PSD
psd = 10*np.log10(output_fft_scaled**2)
plt.subplot(3, 2, 5)
plt.plot(f_axis, psd)
plt.title("PSD")

plt.tight_layout()
plt.show()
```