Digital Modulation Techniques with Real-World Applications

Name: Dhananjay Sharma (549)

Anuj Shinde (550)

Om Shinde (551)

Pratik Shinde (552)

Aim: To understand the principles of digital modulation techniques**:** Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK).

Software Required: Google Colab

Theory:

In digital communication, information such as text, audio, or video must be transmitted efficiently from one point to another. However, digital signals (composed of 1s and 0s) cannot be directly transmitted over long distances because of attenuation, noise, and bandwidth limitations of the channel.
To overcome this, we use **Modulation**, which involves varying one or more parameters of a high-frequency carrier wave (such as its amplitude, frequency, or phase) according to the instantaneous value of the digital message signal.

Types of Modulation:

1. **Analog Modulation:** AM, FM, PM
2. **Digital Modulation:** ASK, FSK, PSK, QPSK, QAM, etc.

1**) Amplitude Shift Keying (ASK):**
In ASK, the **amplitude** of the carrier wave is varied according to the binary data signal, while the frequency and phase remain constant.
Binary '1' is represented by transmitting the carrier signal, and binary '0' by transmitting no carrier (or a lower amplitude carrier).

Mathematical Expression:

$$s(t) = A_c m(t) \cos(2\pi f_c t)$$

Where:

- $s(t)$: Modulated signal
- A_c: Carrier amplitude
- $m(t)$: Digital message (1 or 0)
- f_c: Carrier frequency
- t: Time

The signal's amplitude switches between a high and low level corresponding to the binary bits. This form of modulation is also called **On-Off Keying (OOK)**.

Applications:

- RFID systems
- Early telephone modems
- Optical fiber communication

2) **Frequency Shift Keying (FSK):**

In FSK, the **frequency** of the carrier wave is varied according to the digital input signal, while amplitude and phase remain constant.

Two distinct carrier frequencies are used:

- f_1 for binary '1'
- f_2 for binary '0'

Mathematical Expression:

$$s(t) = \begin{cases} A_c \cos(2\pi f_1 t), & \text{if } m(t) = 1 \\ A_c \cos(2\pi f_2 t), & \text{if } m(t) = 0 \end{cases}$$

Where:

- A_c: Amplitude of the carrier
- f_1, f_2: Frequencies for binary symbols
- $m(t)$: Binary message signal
- t: Time

Whenever the digital input changes, the frequency of the carrier switches between f_1 and f_2. This produces a waveform with alternating frequencies depending on the transmitted bits.

Applications:

- Bluetooth and paging systems
- Caller ID systems
- Radio and wireless communications

3) **Phase Shift Keying (PSK):**

PSK is a digital modulation technique in which the **phase** of the carrier is changed according to the binary data, while amplitude and frequency remain constant.

Mathematical Representation:

$$s(t) = A_c \cos\left(2\pi f_c t + \phi\right)$$

Where ϕ= 0° for binary 1 and 180° for binary 0 (Binary PSK, BPSK).

Example:

- Binary 1 → Phase = 0°
- Binary 0 → Phase = 180°

Applications:

- Wi-Fi (802.11 standards)
- Satellite communication
- Digital TV and secure data transmission

Procedure:

1) Generate a binary data sequence (e.g., [1, 0, 1, 1, 0]) in Python.
2) Define sampling frequency, carrier frequency, and time vector using NumPy.
3) ASK Simulation: Multiply binary data with the carrier (cos(2πf_ct)), plot the waveform using Matplotlib, and observe amplitude variations.
4) FSK Simulation: Assign two carrier frequencies for 0 and 1, generate the waveform, plot, and observe frequency shifts.
5) PSK Simulation: Apply phase shifts to the carrier (0° for 1, 180° for 0), generate waveform, plot, and observe phase changes.
6) Compare ASK, FSK, and PSK waveforms to analyse differences in amplitude, frequency, and phase.

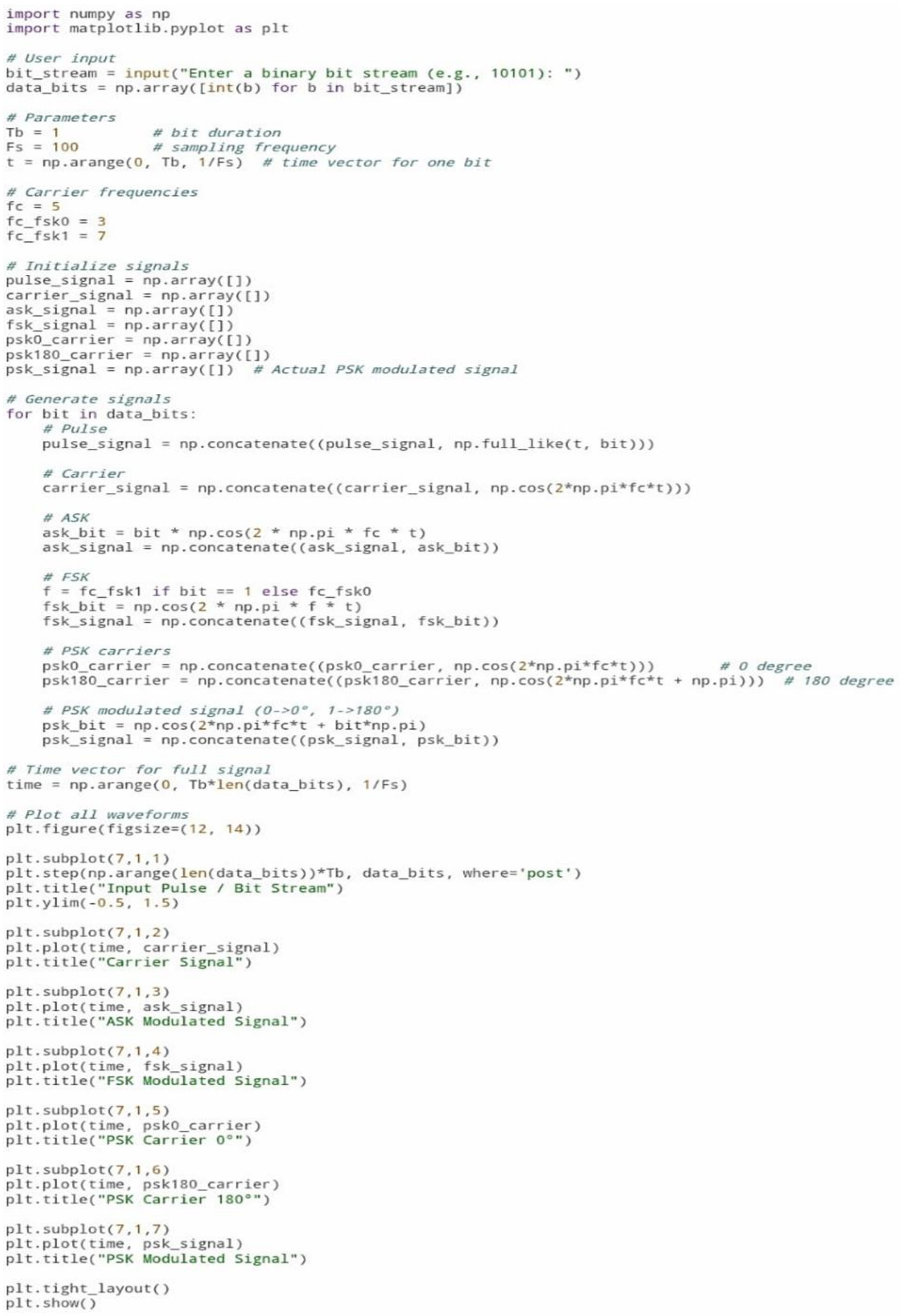
Observation:

1. ASK: Amplitude clearly changes according to binary data.
2. FSK: Frequency shifts are distinct and observable.
3. PSK: Phase reversals occur precisely at binary transitions.

Conclusion:

1. Modulation enables efficient transmission of digital data over analog channels.
2. ASK is simple but easily affected by noise.
3. FSK offers better noise immunity but uses more bandwidth.
4. PSK is highly reliable and preferred in modern systems.
5. Understanding these techniques is key to digital communication design.

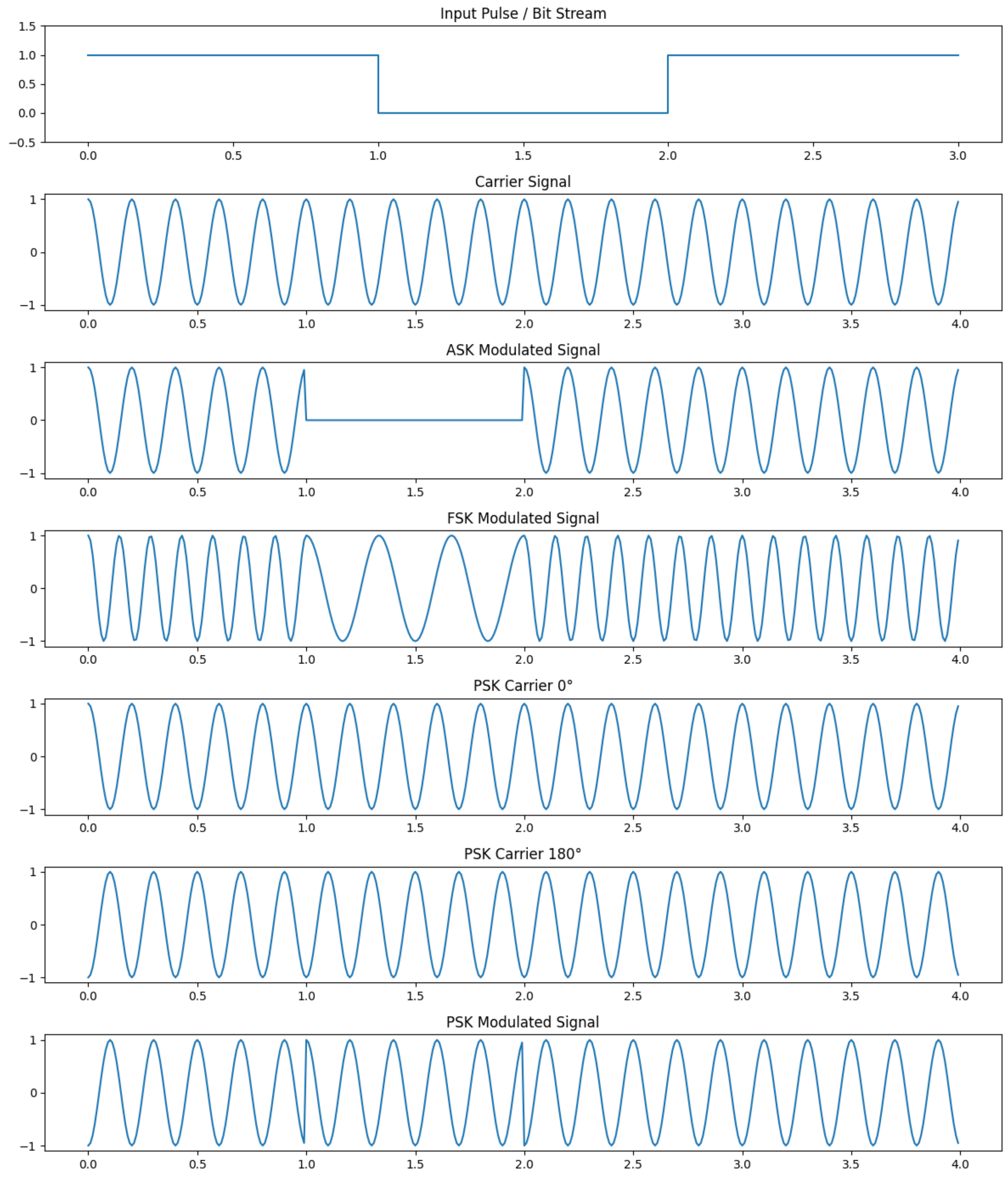
Modulation



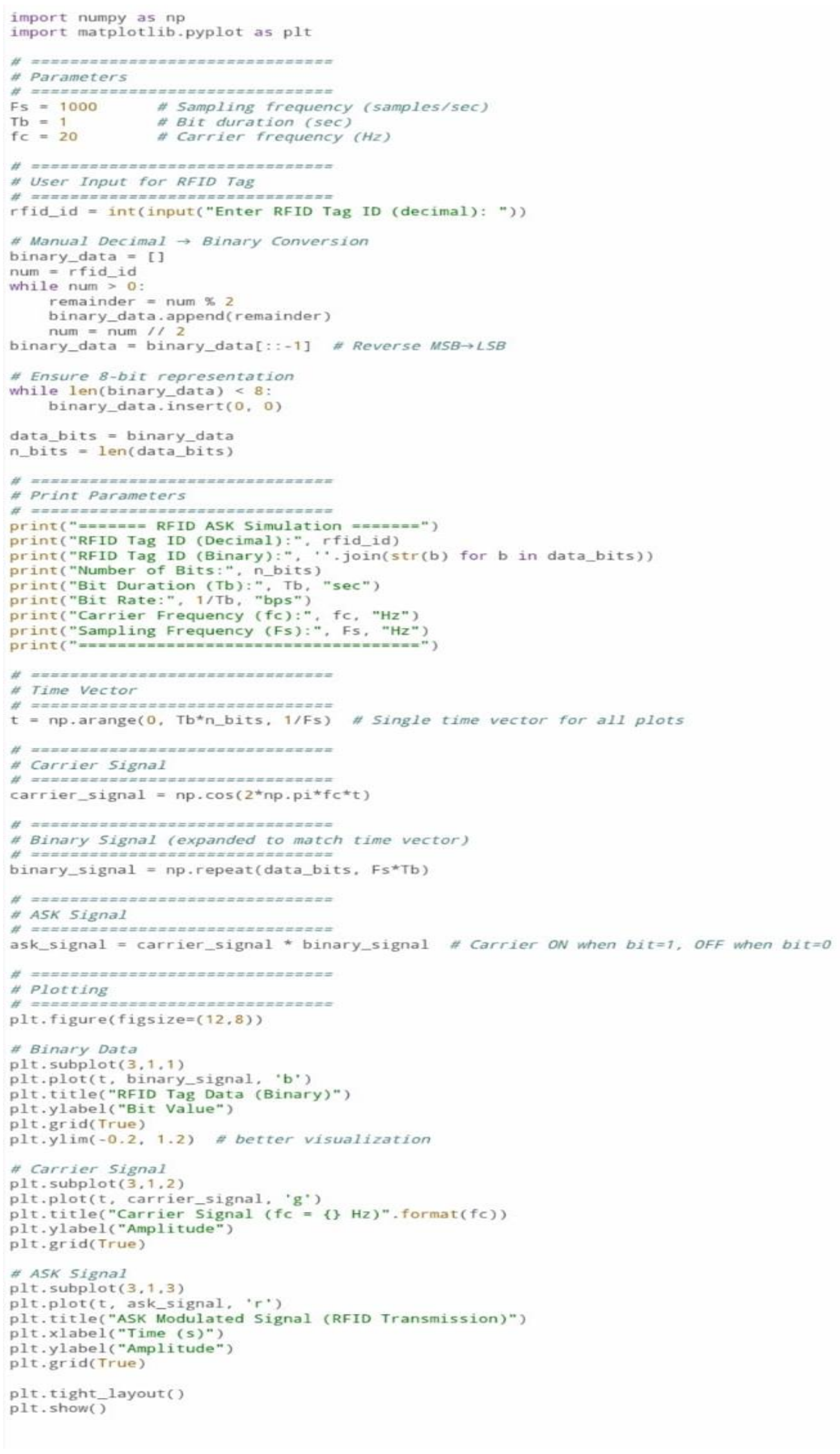
```python
import numpy as np
import matplotlib.pyplot as plt

# User input
bit_stream = input("Enter a binary bit stream (e.g., 10101): ")
data_bits = np.array([int(b) for b in bit_stream])

# Parameters
Tb = 1            # bit duration
Fs = 100          # sampling frequency
t = np.arange(0, Tb, 1/Fs)  # time vector for one bit

# Carrier frequencies
fc = 5
fc_fsk0 = 3
fc_fsk1 = 7

# Initialize signals
pulse_signal = np.array([])
carrier_signal = np.array([])
ask_signal = np.array([])
fsk_signal = np.array([])
psk0_carrier = np.array([])
psk180_carrier = np.array([])
psk_signal = np.array([])  # Actual PSK modulated signal

# Generate signals
for bit in data_bits:
    # Pulse
    pulse_signal = np.concatenate((pulse_signal, np.full_like(t, bit)))

    # Carrier
    carrier_signal = np.concatenate((carrier_signal, np.cos(2*np.pi*fc*t)))

    # ASK
    ask_bit = bit * np.cos(2 * np.pi * fc * t)
    ask_signal = np.concatenate((ask_signal, ask_bit))

    # FSK
    f = fc_fsk1 if bit == 1 else fc_fsk0
    fsk_bit = np.cos(2 * np.pi * f * t)
    fsk_signal = np.concatenate((fsk_signal, fsk_bit))

    # PSK carriers
    psk0_carrier = np.concatenate((psk0_carrier, np.cos(2*np.pi*fc*t)))        # 0 degree
    psk180_carrier = np.concatenate((psk180_carrier, np.cos(2*np.pi*fc*t + np.pi)))  # 180 degree

    # PSK modulated signal (0->0°, 1->180°)
    psk_bit = np.cos(2*np.pi*fc*t + bit*np.pi)
    psk_signal = np.concatenate((psk_signal, psk_bit))

# Time vector for full signal
time = np.arange(0, Tb*len(data_bits), 1/Fs)

# Plot all waveforms
plt.figure(figsize=(12, 14))

plt.subplot(7,1,1)
plt.step(np.arange(len(data_bits))*Tb, data_bits, where='post')
plt.title("Input Pulse / Bit Stream")
plt.ylim(-0.5, 1.5)

plt.subplot(7,1,2)
plt.plot(time, carrier_signal)
plt.title("Carrier Signal")

plt.subplot(7,1,3)
plt.plot(time, ask_signal)
plt.title("ASK Modulated Signal")

plt.subplot(7,1,4)
plt.plot(time, fsk_signal)
plt.title("FSK Modulated Signal")

plt.subplot(7,1,5)
plt.plot(time, psk0_carrier)
plt.title("PSK Carrier 0°")

plt.subplot(7,1,6)
plt.plot(time, psk180_carrier)
plt.title("PSK Carrier 180°")

plt.subplot(7,1,7)
plt.plot(time, psk_signal)
plt.title("PSK Modulated Signal")

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

Output

```
Enter a binary bit stream (e.g., 10101): 1011
```



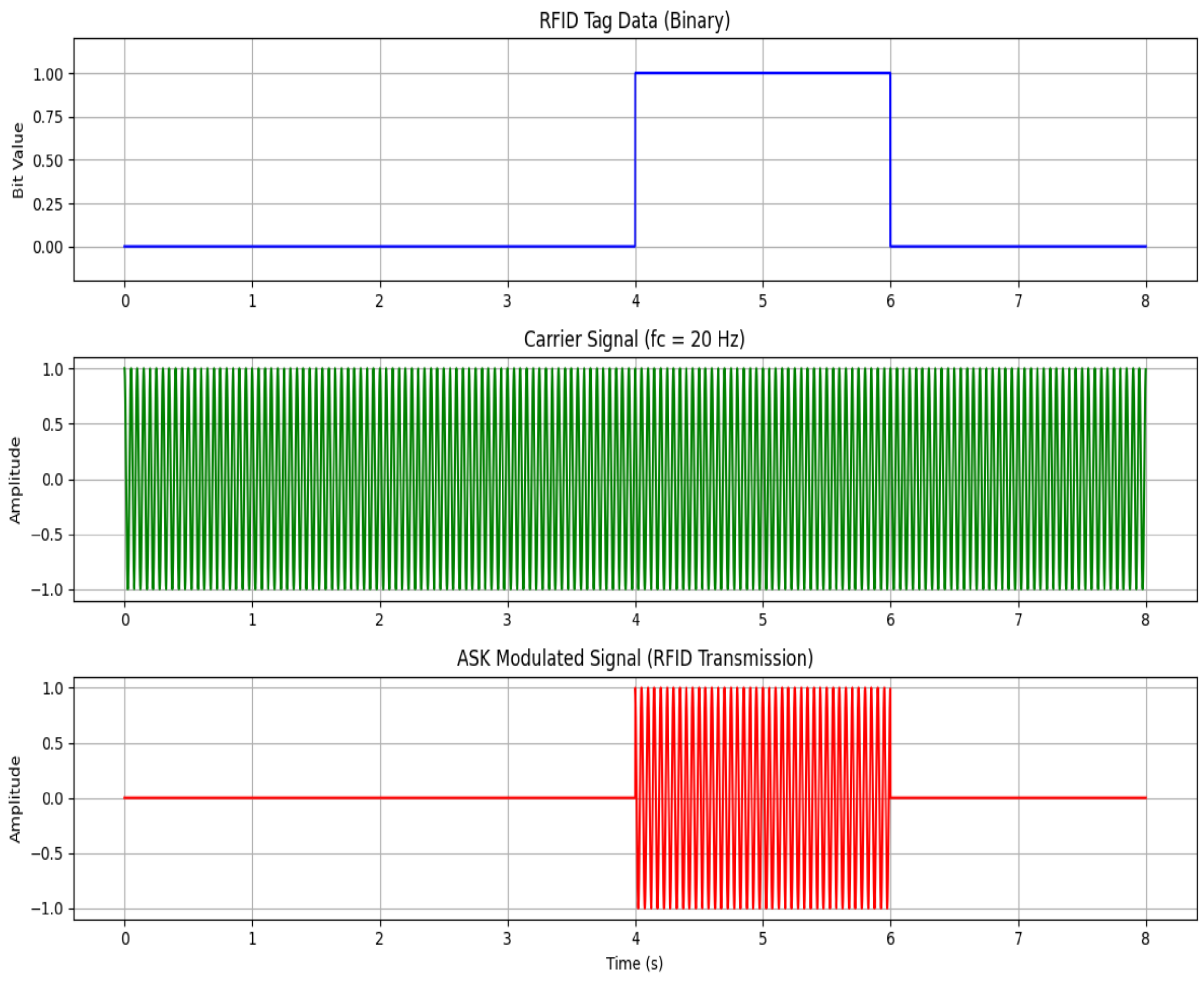
ASK Application - RFID



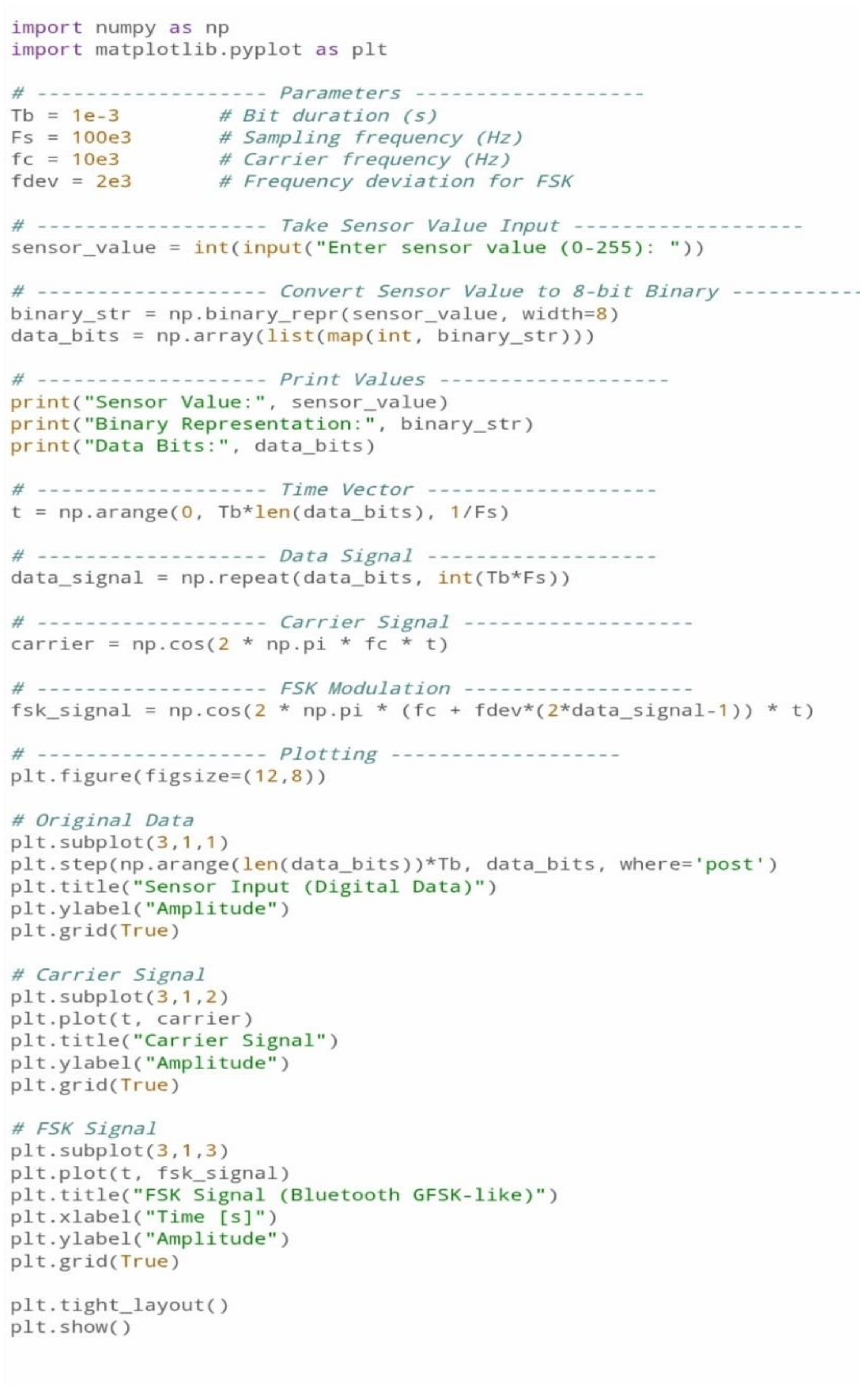
```python
import numpy as np
import matplotlib.pyplot as plt

# ================================
# Parameters
# ================================
Fs = 1000       # Sampling frequency (samples/sec)
Tb = 1          # Bit duration (sec)
fc = 20         # Carrier frequency (Hz)

# ================================
# User Input for RFID Tag
# ================================
rfid_id = int(input("Enter RFID Tag ID (decimal): "))

# Manual Decimal → Binary Conversion
binary_data = []
num = rfid_id
while num > 0:
    remainder = num % 2
    binary_data.append(remainder)
    num = num // 2
binary_data = binary_data[::-1]  # Reverse MSB→LSB

# Ensure 8-bit representation
while len(binary_data) < 8:
    binary_data.insert(0, 0)

data_bits = binary_data
n_bits = len(data_bits)

# ================================
# Print Parameters
# ================================
print("======= RFID ASK Simulation =======")
print("RFID Tag ID (Decimal):", rfid_id)
print("RFID Tag ID (Binary):", ''.join(str(b) for b in data_bits))
print("Number of Bits:", n_bits)
print("Bit Duration (Tb):", Tb, "sec")
print("Bit Rate:", 1/Tb, "bps")
print("Carrier Frequency (fc):", fc, "Hz")
print("Sampling Frequency (Fs):", Fs, "Hz")
print("===================================")

# ================================
# Time Vector
# ================================
t = np.arange(0, Tb*n_bits, 1/Fs)  # Single time vector for all plots

# ================================
# Carrier Signal
# ================================
carrier_signal = np.cos(2*np.pi*fc*t)

# ================================
# Binary Signal (expanded to match time vector)
# ================================
binary_signal = np.repeat(data_bits, Fs*Tb)

# ================================
# ASK Signal
# ================================
ask_signal = carrier_signal * binary_signal  # Carrier ON when bit=1, OFF when bit=0

# ================================
# Plotting
# ================================
plt.figure(figsize=(12,8))

# Binary Data
plt.subplot(3,1,1)
plt.plot(t, binary_signal, 'b')
plt.title("RFID Tag Data (Binary)")
plt.ylabel("Bit Value")
plt.grid(True)
plt.ylim(-0.2, 1.2)  # better visualization

# Carrier Signal
plt.subplot(3,1,2)
plt.plot(t, carrier_signal, 'g')
plt.title("Carrier Signal (fc = {} Hz)".format(fc))
plt.ylabel("Amplitude")
plt.grid(True)

# ASK Signal
plt.subplot(3,1,3)
plt.plot(t, ask_signal, 'r')
plt.title("ASK Modulated Signal (RFID Transmission)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)

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

Output

```
Enter RFID Tag ID (decimal): 12
======= RFID ASK Simulation =======
RFID Tag ID (Decimal): 12
RFID Tag ID (Binary): 00001100
Number of Bits: 8
Bit Duration (Tb): 1 sec
Bit Rate: 1.0 bps
Carrier Frequency (fc): 20 Hz
Sampling Frequency (Fs): 1000 Hz
===================================
```



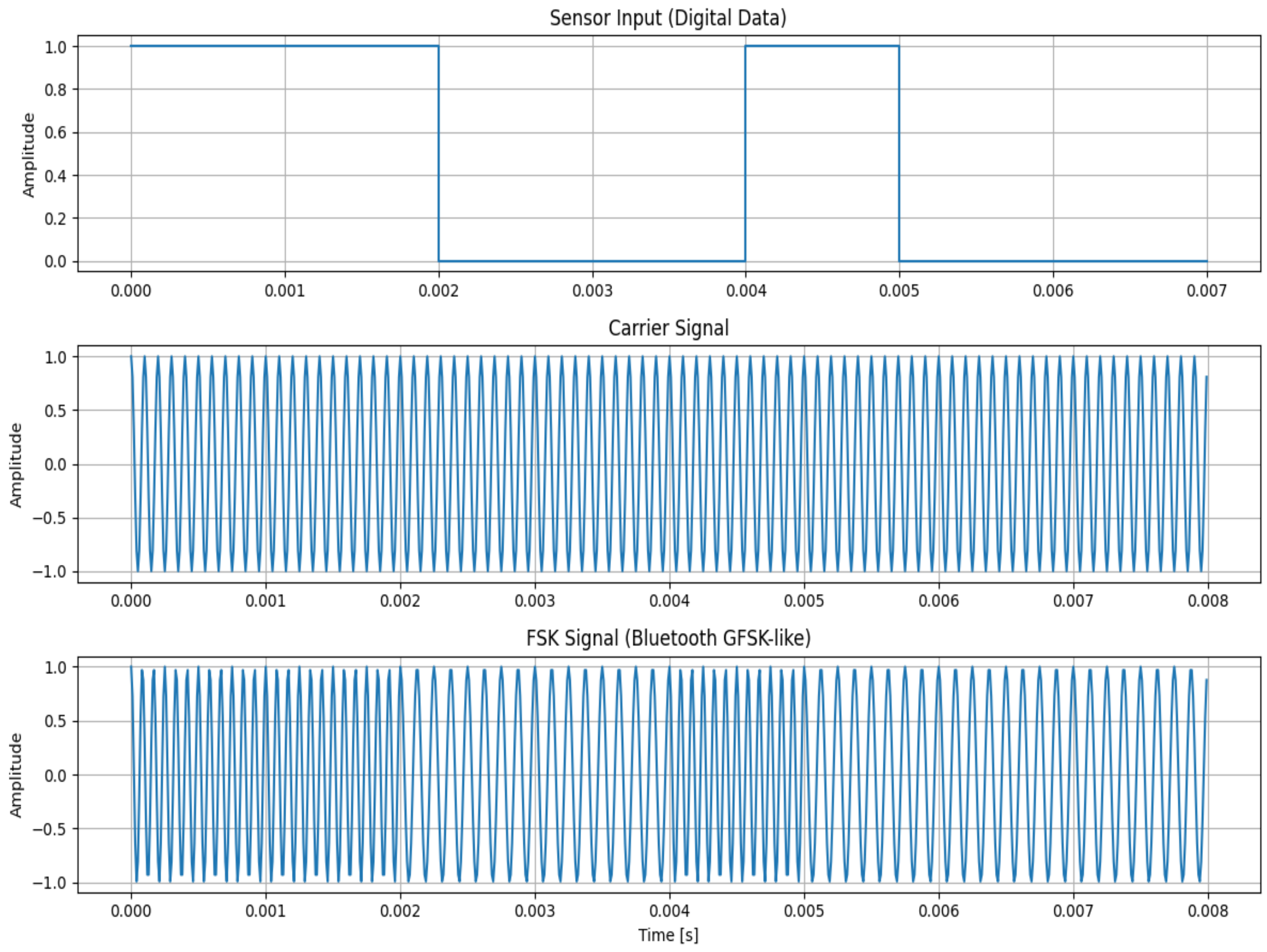
FSK Application - Bluetooth



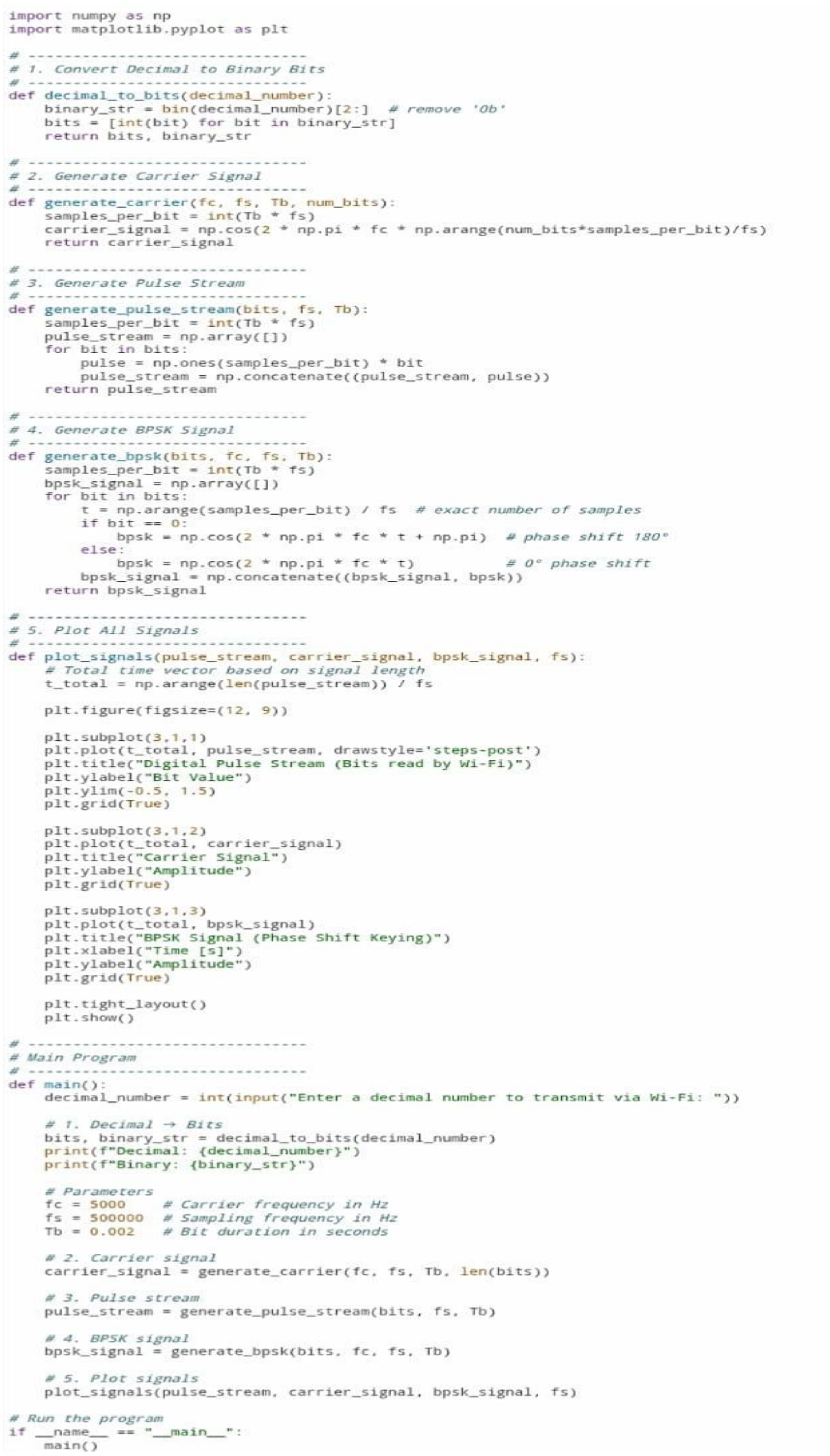
```python
import numpy as np
import matplotlib.pyplot as plt

# ------------------- Parameters -------------------
Tb = 1e-3          # Bit duration (s)
Fs = 100e3         # Sampling frequency (Hz)
fc = 10e3          # Carrier frequency (Hz)
fdev = 2e3         # Frequency deviation for FSK

# ------------------- Take Sensor Value Input -------------------
sensor_value = int(input("Enter sensor value (0-255): "))

# ------------------- Convert Sensor Value to 8-bit Binary ----------
binary_str = np.binary_repr(sensor_value, width=8)
data_bits = np.array(list(map(int, binary_str)))

# ------------------- Print Values -------------------
print("Sensor Value:", sensor_value)
print("Binary Representation:", binary_str)
print("Data Bits:", data_bits)

# ------------------- Time Vector -------------------
t = np.arange(0, Tb*len(data_bits), 1/Fs)

# ------------------- Data Signal -------------------
data_signal = np.repeat(data_bits, int(Tb*Fs))

# ------------------- Carrier Signal -------------------
carrier = np.cos(2 * np.pi * fc * t)

# ------------------- FSK Modulation -------------------
fsk_signal = np.cos(2 * np.pi * (fc + fdev*(2*data_signal-1)) * t)

# ------------------- Plotting -------------------
plt.figure(figsize=(12,8))

# Original Data
plt.subplot(3,1,1)
plt.step(np.arange(len(data_bits))*Tb, data_bits, where='post')
plt.title("Sensor Input (Digital Data)")
plt.ylabel("Amplitude")
plt.grid(True)

# Carrier Signal
plt.subplot(3,1,2)
plt.plot(t, carrier)
plt.title("Carrier Signal")
plt.ylabel("Amplitude")
plt.grid(True)

# FSK Signal
plt.subplot(3,1,3)
plt.plot(t, fsk_signal)
plt.title("FSK Signal (Bluetooth GFSK-like)")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid(True)

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

Output

```
Enter sensor value (0-255): 200
Sensor Value: 200
Binary Representation: 11001000
Data Bits: [1 1 0 0 1 0 0 0]
```



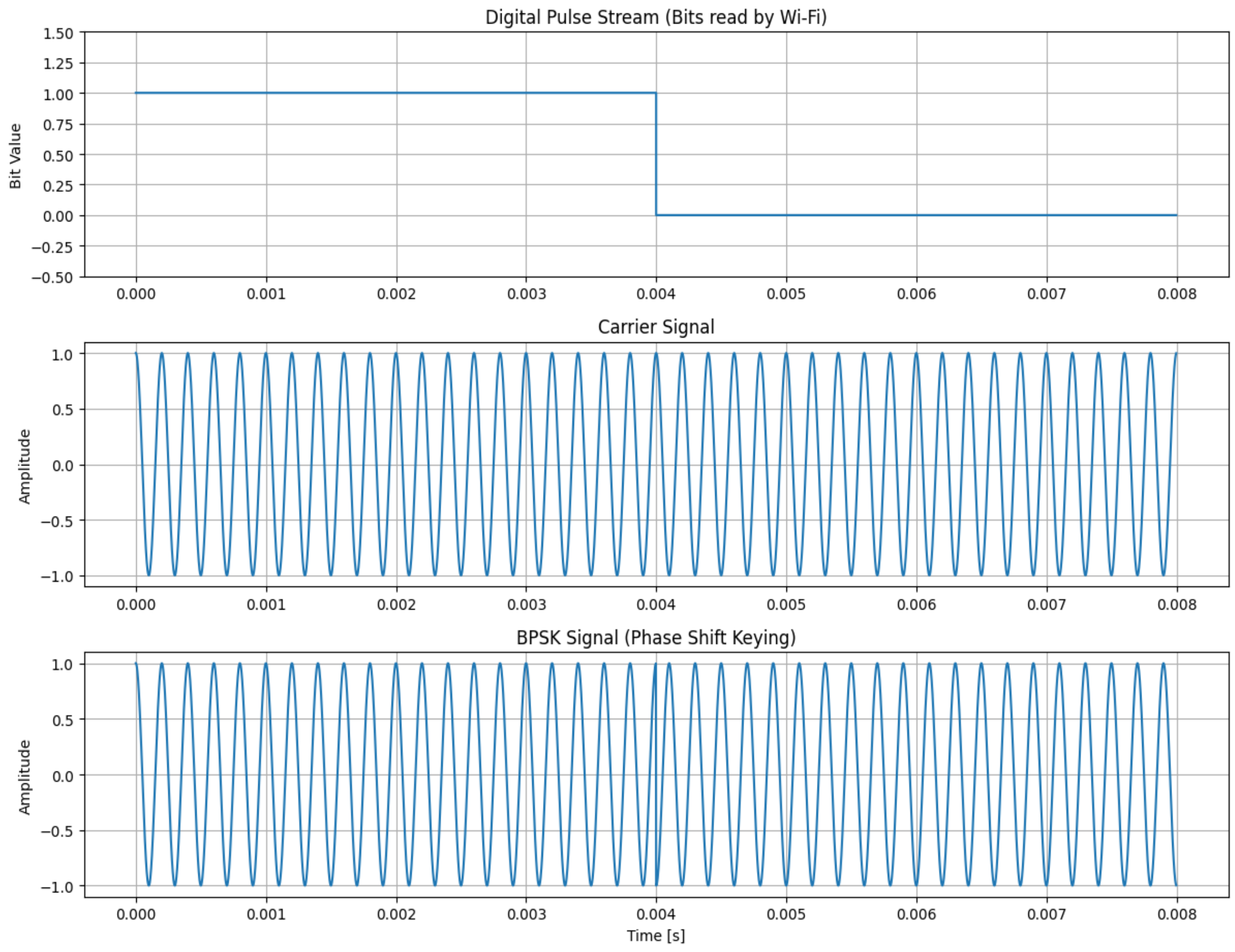
PSK Application - Wi-fi



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

# ------------------------------
# 1. Convert Decimal to Binary Bits
# ------------------------------
def decimal_to_bits(decimal_number):
    binary_str = bin(decimal_number)[2:]  # remove '0b'
    bits = [int(bit) for bit in binary_str]
    return bits, binary_str

# ------------------------------
# 2. Generate Carrier Signal
# ------------------------------
def generate_carrier(fc, fs, Tb, num_bits):
    samples_per_bit = int(Tb * fs)
    carrier_signal = np.cos(2 * np.pi * fc * np.arange(num_bits*samples_per_bit)/fs)
    return carrier_signal

# ------------------------------
# 3. Generate Pulse Stream
# ------------------------------
def generate_pulse_stream(bits, fs, Tb):
    samples_per_bit = int(Tb * fs)
    pulse_stream = np.array([])
    for bit in bits:
        pulse = np.ones(samples_per_bit) * bit
        pulse_stream = np.concatenate((pulse_stream, pulse))
    return pulse_stream

# ------------------------------
# 4. Generate BPSK Signal
# ------------------------------
def generate_bpsk(bits, fc, fs, Tb):
    samples_per_bit = int(Tb * fs)
    bpsk_signal = np.array([])
    for bit in bits:
        t = np.arange(samples_per_bit) / fs  # exact number of samples
        if bit == 0:
            bpsk = np.cos(2 * np.pi * fc * t + np.pi)  # phase shift 180°
        else:
            bpsk = np.cos(2 * np.pi * fc * t)          # 0° phase shift
        bpsk_signal = np.concatenate((bpsk_signal, bpsk))
    return bpsk_signal

# ------------------------------
# 5. Plot All Signals
# ------------------------------
def plot_signals(pulse_stream, carrier_signal, bpsk_signal, fs):
    # Total time vector based on signal length
    t_total = np.arange(len(pulse_stream)) / fs

    plt.figure(figsize=(12, 9))

    plt.subplot(3,1,1)
    plt.plot(t_total, pulse_stream, drawstyle='steps-post')
    plt.title("Digital Pulse Stream (Bits read by Wi-Fi)")
    plt.ylabel("Bit Value")
    plt.ylim(-0.5, 1.5)
    plt.grid(True)

    plt.subplot(3,1,2)
    plt.plot(t_total, carrier_signal)
    plt.title("Carrier Signal")
    plt.ylabel("Amplitude")
    plt.grid(True)

    plt.subplot(3,1,3)
    plt.plot(t_total, bpsk_signal)
    plt.title("BPSK Signal (Phase Shift Keying)")
    plt.xlabel("Time [s]")
    plt.ylabel("Amplitude")
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# ------------------------------
# Main Program
# ------------------------------
def main():
    decimal_number = int(input("Enter a decimal number to transmit via Wi-Fi: "))

    # 1. Decimal → Bits
    bits, binary_str = decimal_to_bits(decimal_number)
    print(f"Decimal: {decimal_number}")
    print(f"Binary: {binary_str}")

    # Parameters
    fc = 5000      # Carrier frequency in Hz
    fs = 500000    # Sampling frequency in Hz
    Tb = 0.002     # Bit duration in seconds

    # 2. Carrier signal
    carrier_signal = generate_carrier(fc, fs, Tb, len(bits))

    # 3. Pulse stream
    pulse_stream = generate_pulse_stream(bits, fs, Tb)

    # 4. BPSK signal
    bpsk_signal = generate_bpsk(bits, fc, fs, Tb)

    # 5. Plot signals
    plot_signals(pulse_stream, carrier_signal, bpsk_signal, fs)

# Run the program
if __name__ == "__main__":
    main()
```

Output

```
Enter a decimal number to transmit via Wi-Fi: 12

Decimal: 12

Binary: 1100
```


Digital Pulse Stream (Bits read by Wi-Fi)
Bit Value
Carrier Signal
Amplitude
BPSK Signal (Phase Shift Keying)
Amplitude
Time [s]