Cairo University - Faculty Of Engineering
Computer Engineering Department
Digital Communication - Spring 2025

Digital Communications: Assignment 2

Submitted to

Dr. Mai Badawi
Dr. Hala
Eng. Mohamed Khaled

Submitted by

Akram Hany Sec 1 BN 14
Amir Anwar Sec 1 BN 15

Contents

List of Figures

1 Part 1 Hand Analysis

1.1 Part 1-a: Transmitted baseband waveform

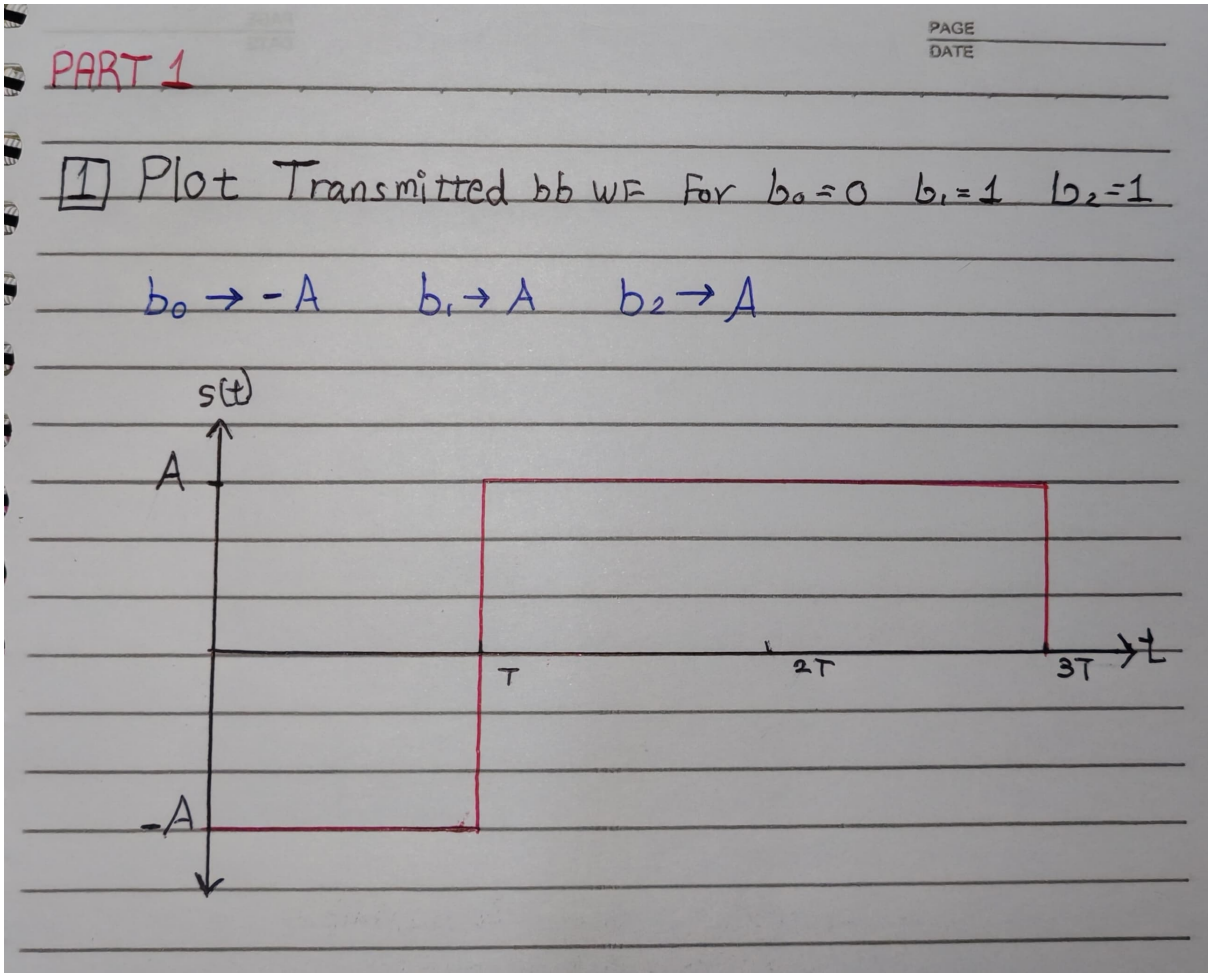


Figure 1: Transmitted baseband waveform

1.2 Part 1-b1: Matched Filter Output

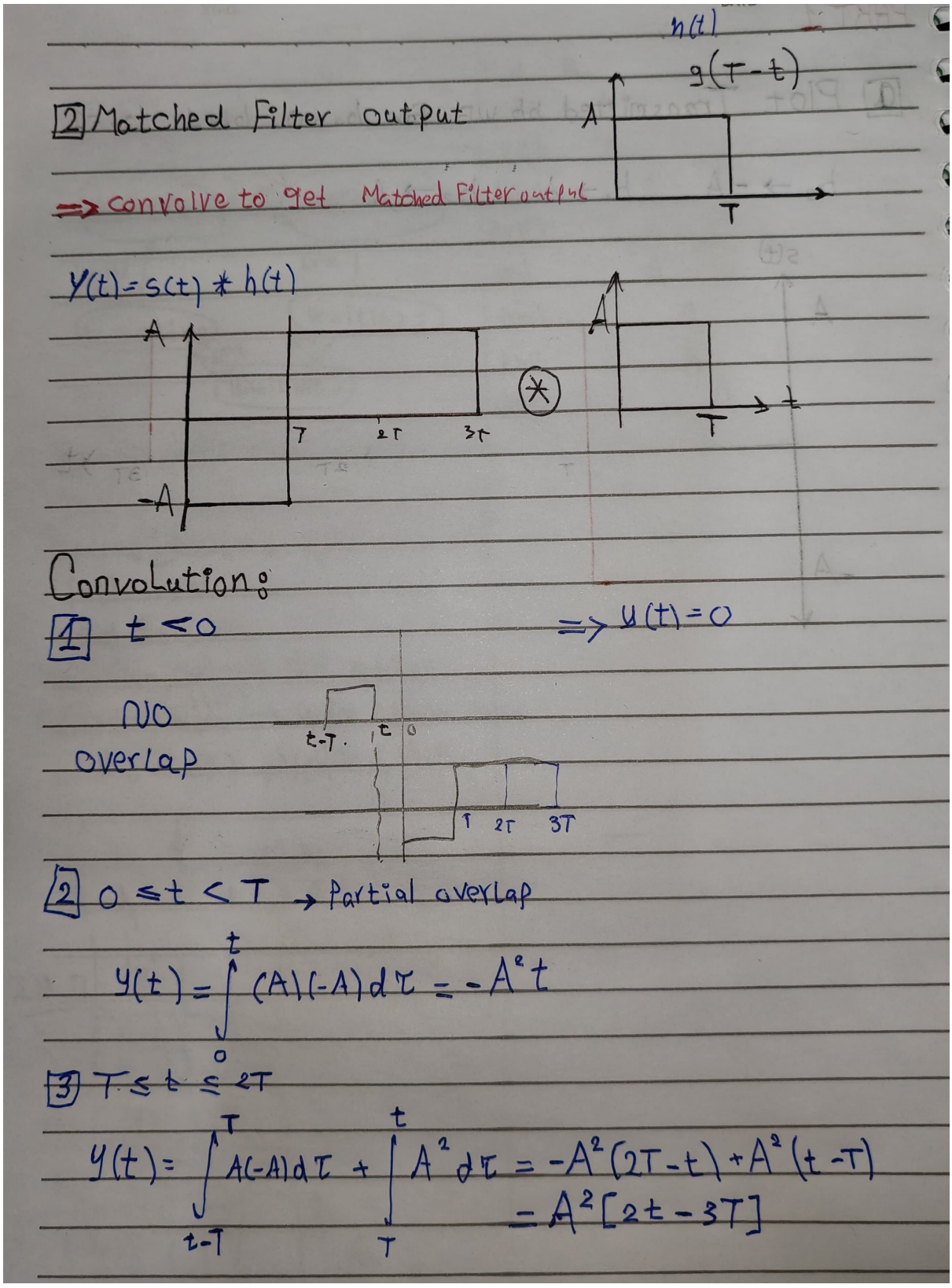


Figure 2: Matched Filter Output

1.3 Part 1-b2: Matched Filter Output continued

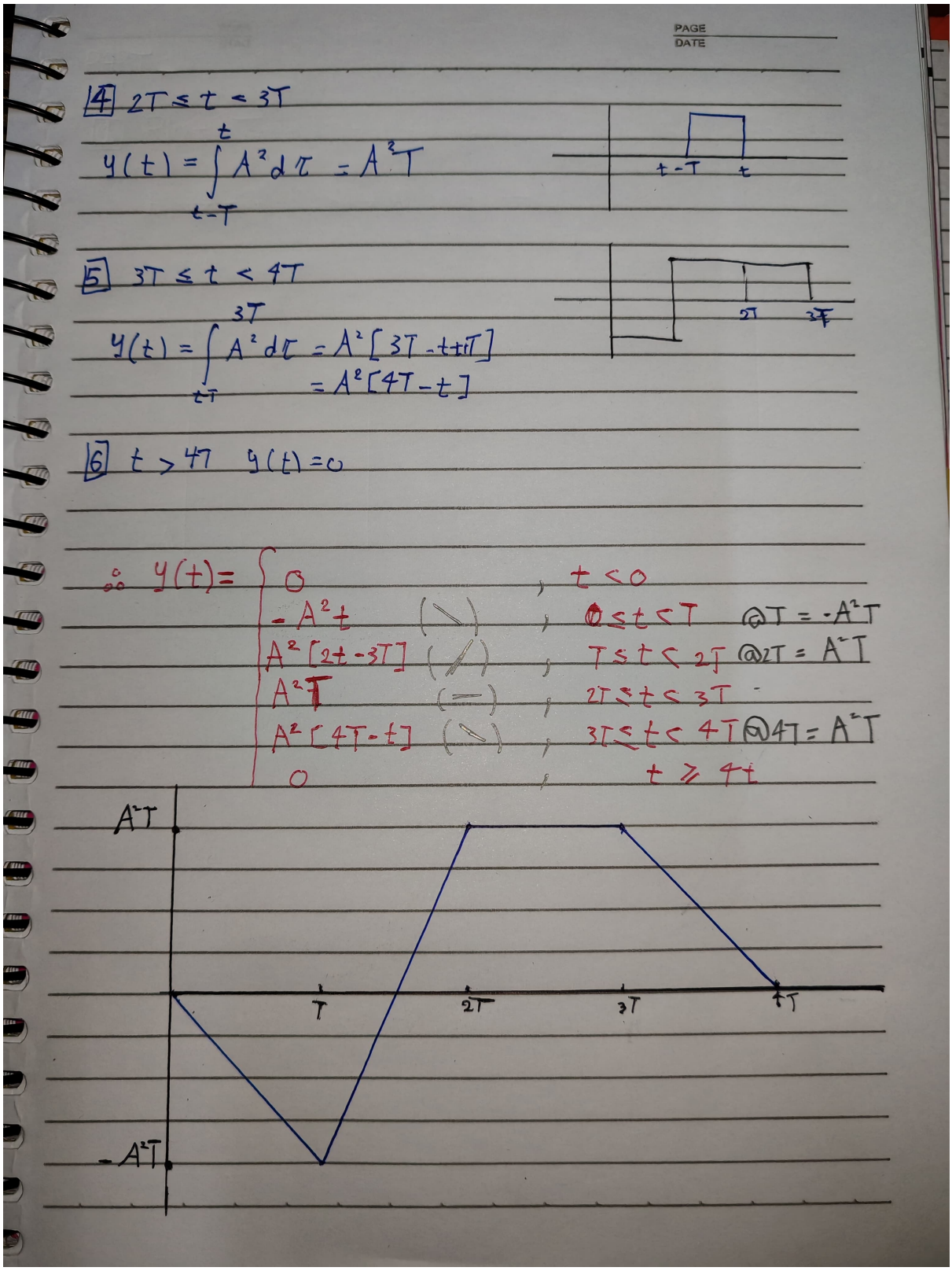


Figure 3: Matched Filter Output continued

1.4 Part 1-c: Sampling instances

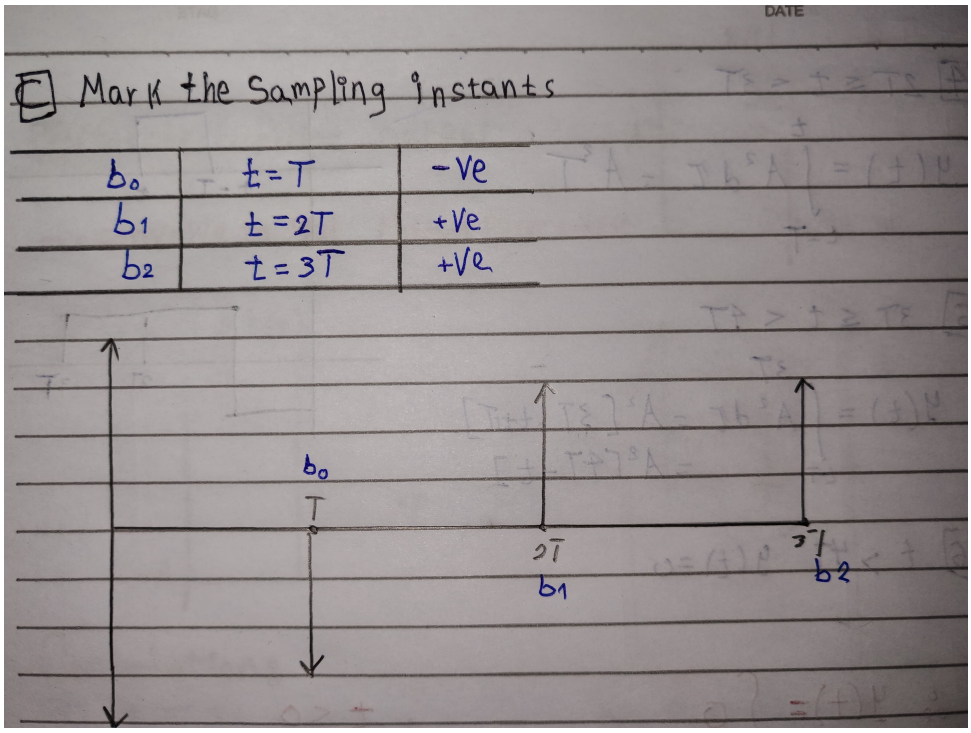


Figure 4: Sampling instances

1.5 Part 1-d: Transmitter Block Diagram

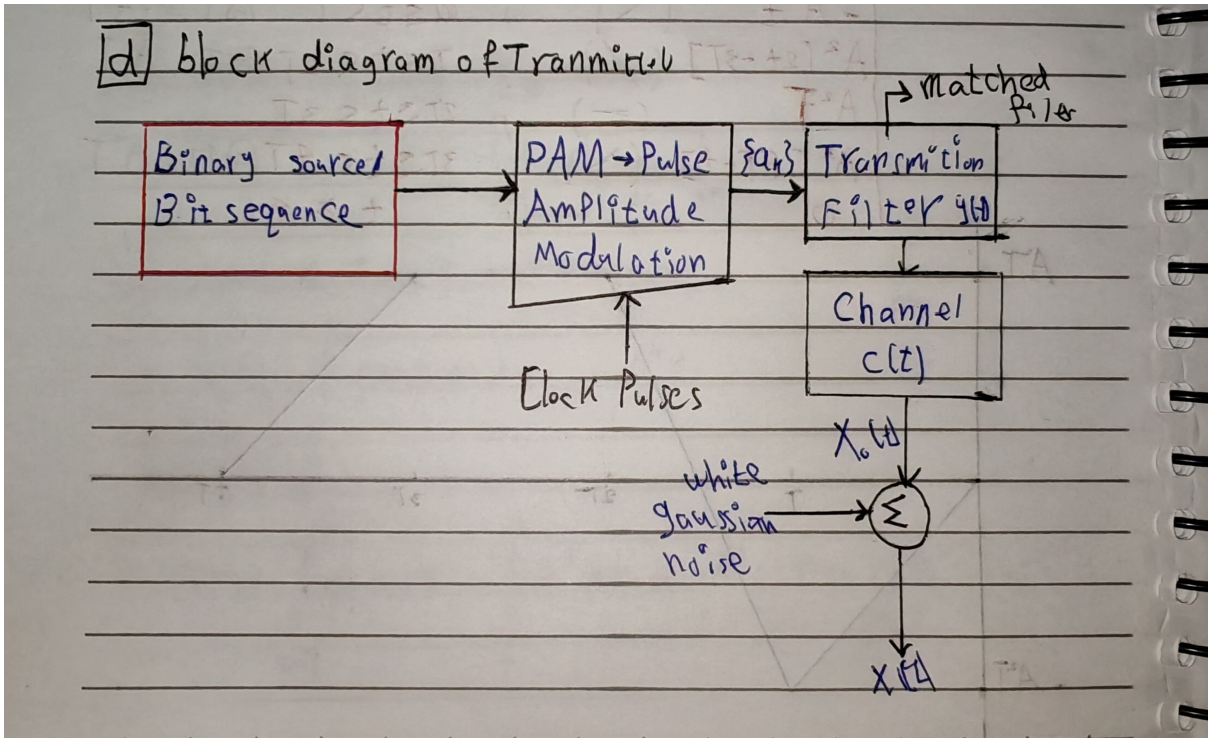


Figure 5: Transmitter Block Diagram

1.6 Part 1-e: Receiver Block Diagram

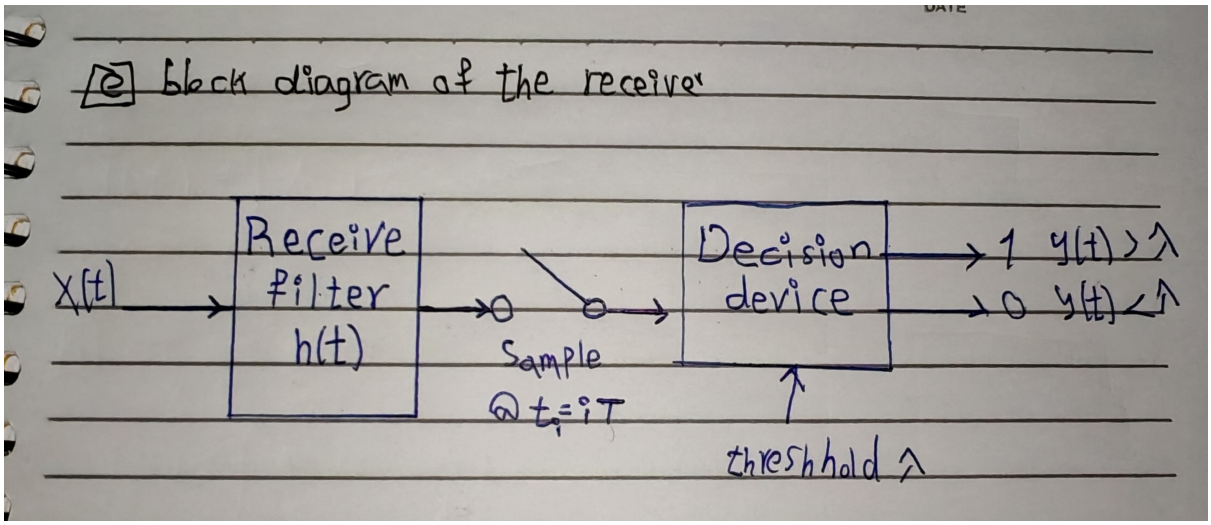


Figure 6: Receiver Block Diagram

1.7 Part 1-d,e: General form as the before was for binary.

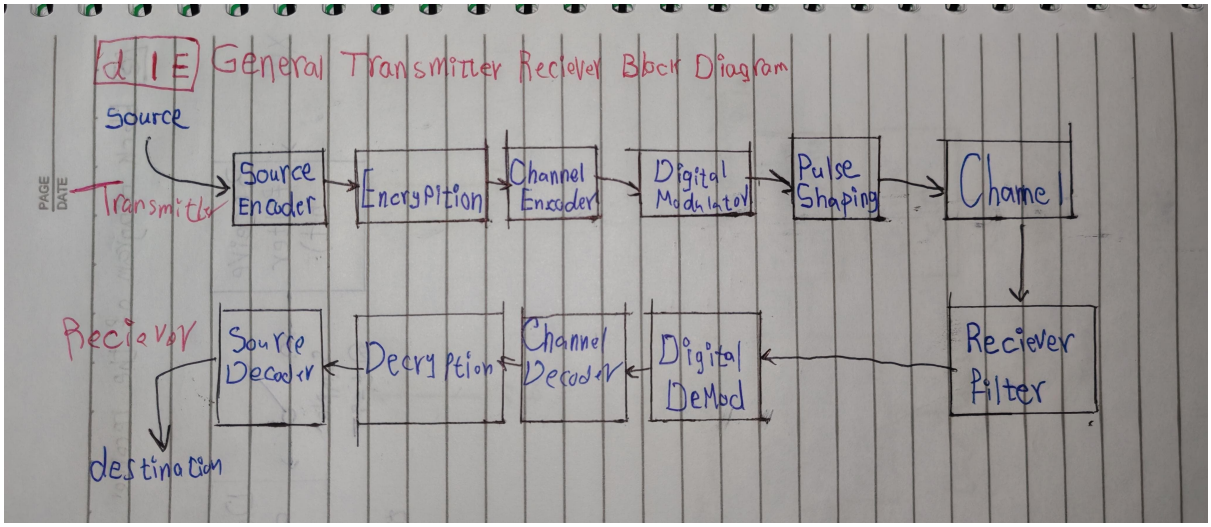


Figure 7: Transmitter-Receiver Block Diagram

2 Part 2

2.1 Hand-Analysis - Matched Filter

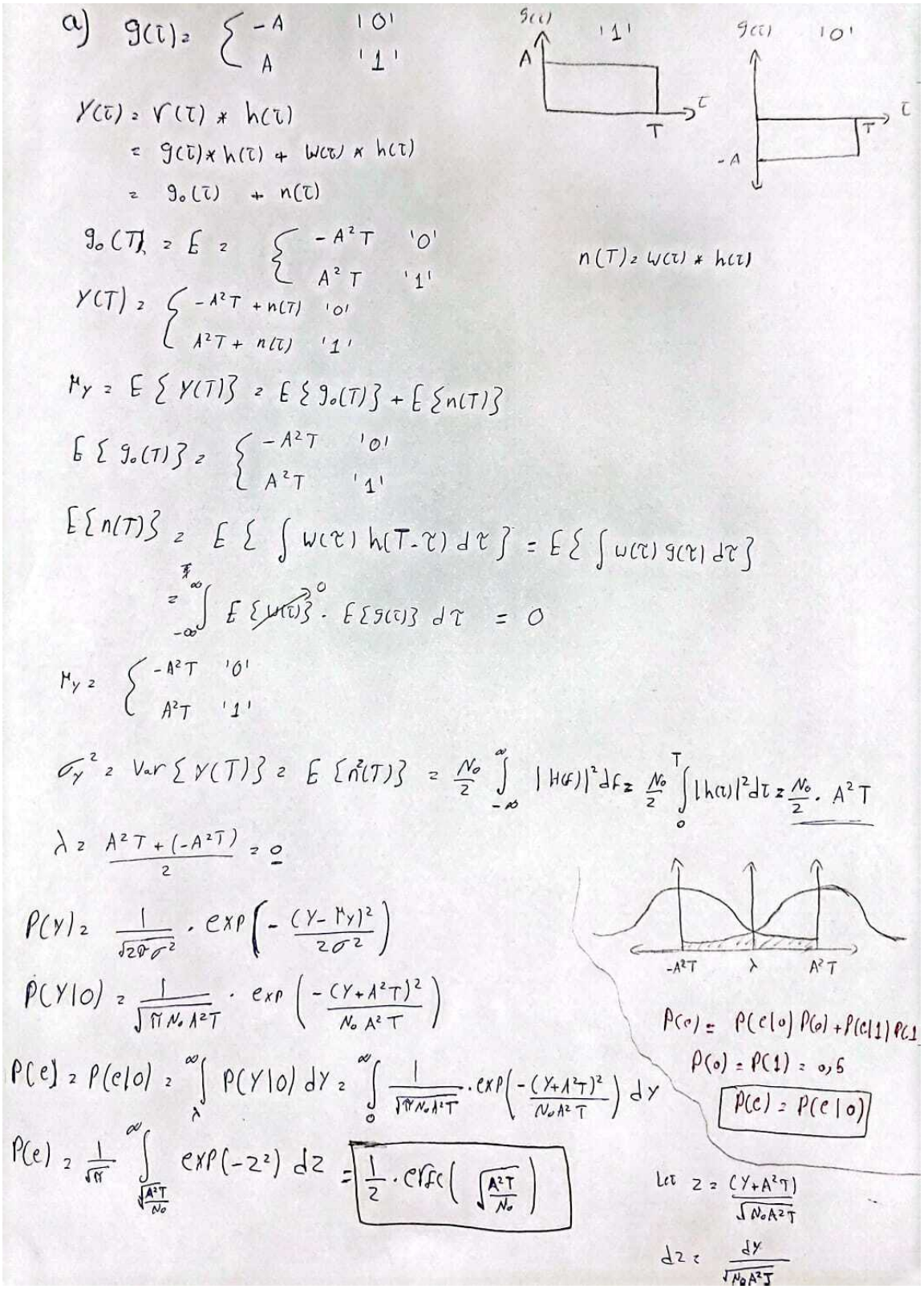


Figure 8: Receiver Block Diagram

2.2 Hand Analysis - No Filter ($\delta(t)$)

b) $g(t) = \begin{cases} -A & '0' \\ A & '1' \end{cases}$

$g_0(t) = g(t) * h(t) = g(t) * \delta(t) = g(t)$

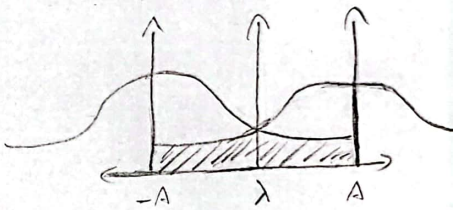
$g(T) = \begin{cases} -A & '0' \\ A & '1' \end{cases}$

$Y(T) = \begin{cases} -A + n(t) & '0' \\ A + n(t) & '1' \end{cases}$

$\mu_Y = E\{g_0(T)\} = \begin{cases} -A & '0' \\ A & '1' \end{cases}$

$\sigma_Y^2 = \frac{N_0}{2} \int_{-\infty}^{\infty} |h(t)|^2 dt = \frac{N_0}{2}$

$\lambda = \frac{A + (-A)}{2} = 0$



$P(Y|0) = \frac{1}{\sqrt{\pi N_0}} \cdot \exp\left(-\frac{(Y+A)^2}{N_0}\right)$

$P(e) = P(e|0) = \int_{\lambda}^{\infty} P(Y|0)\, dY = \int_{0}^{\infty} \frac{1}{\sqrt{\pi N_0}} \exp\left(-\frac{(Y+A)^2}{N_0}\right) dY$

Let $z = \frac{Y+A}{\sqrt{N_0}}$ $\quad dz = \frac{dY}{\sqrt{N_0}}$

$P(e) = \frac{1}{\sqrt{\pi}} \int_{\frac{A}{\sqrt{N_0}}}^{\infty} \exp(-z^2)\, dz = \frac{1}{2} \operatorname{erfc}\left(\frac{A}{\sqrt{N_0}}\right)$.

Figure 9: Receiver Block Diagram

2.3 Hand Analysis - Triangle Filter

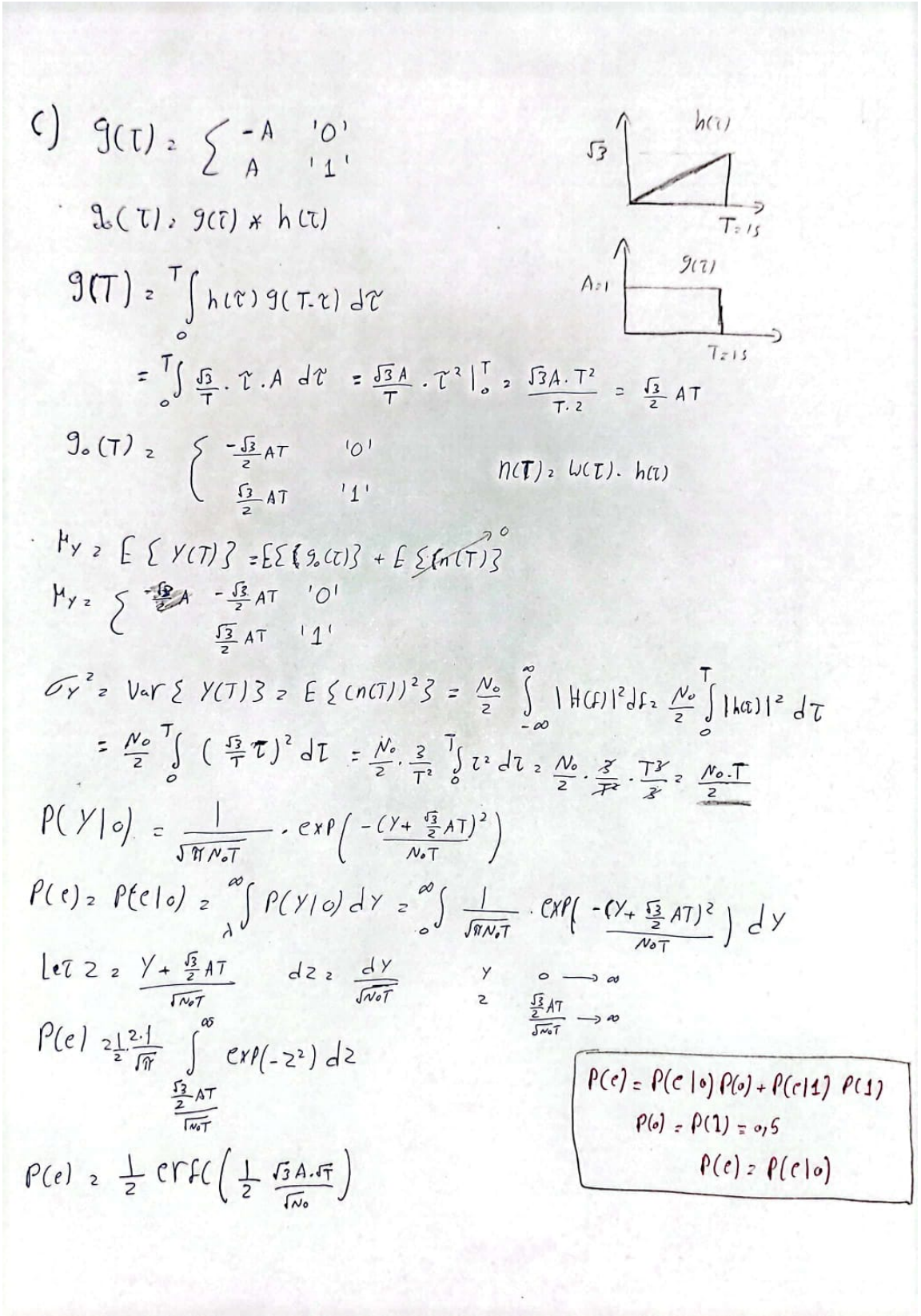


Figure 10: Receiver Block Diagram

2.4 Receive Filters Output

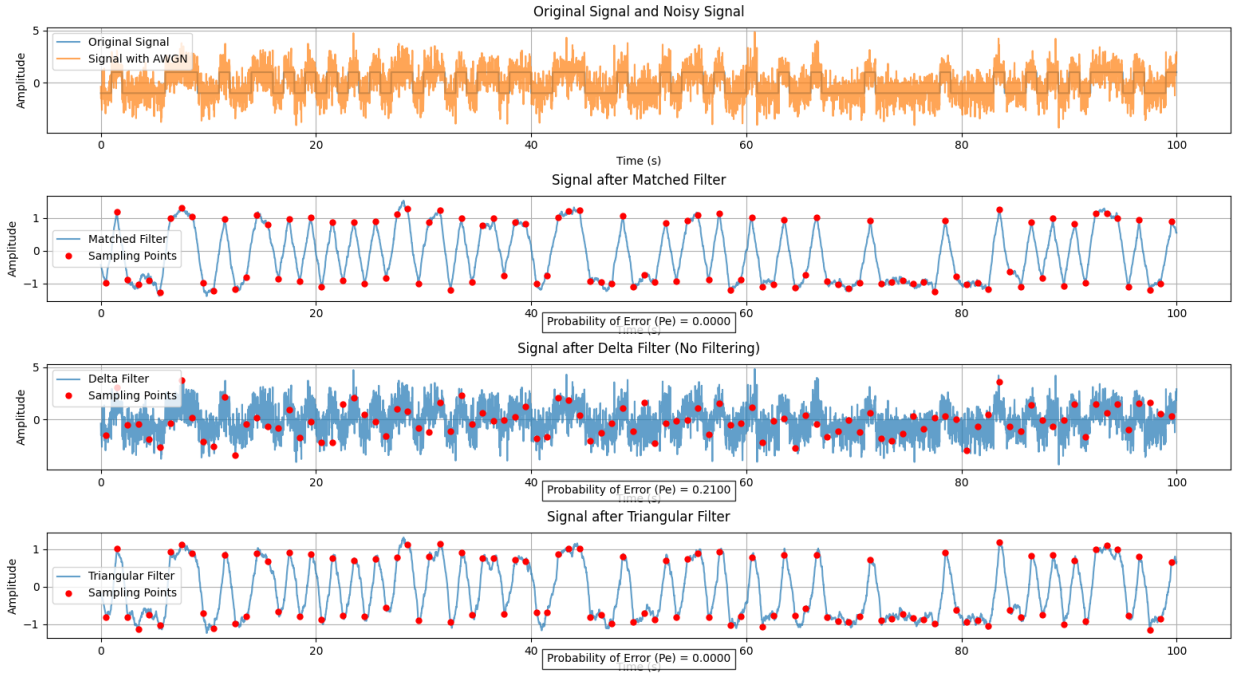


Figure 11: Filters Output

2.5 BER vs E/No

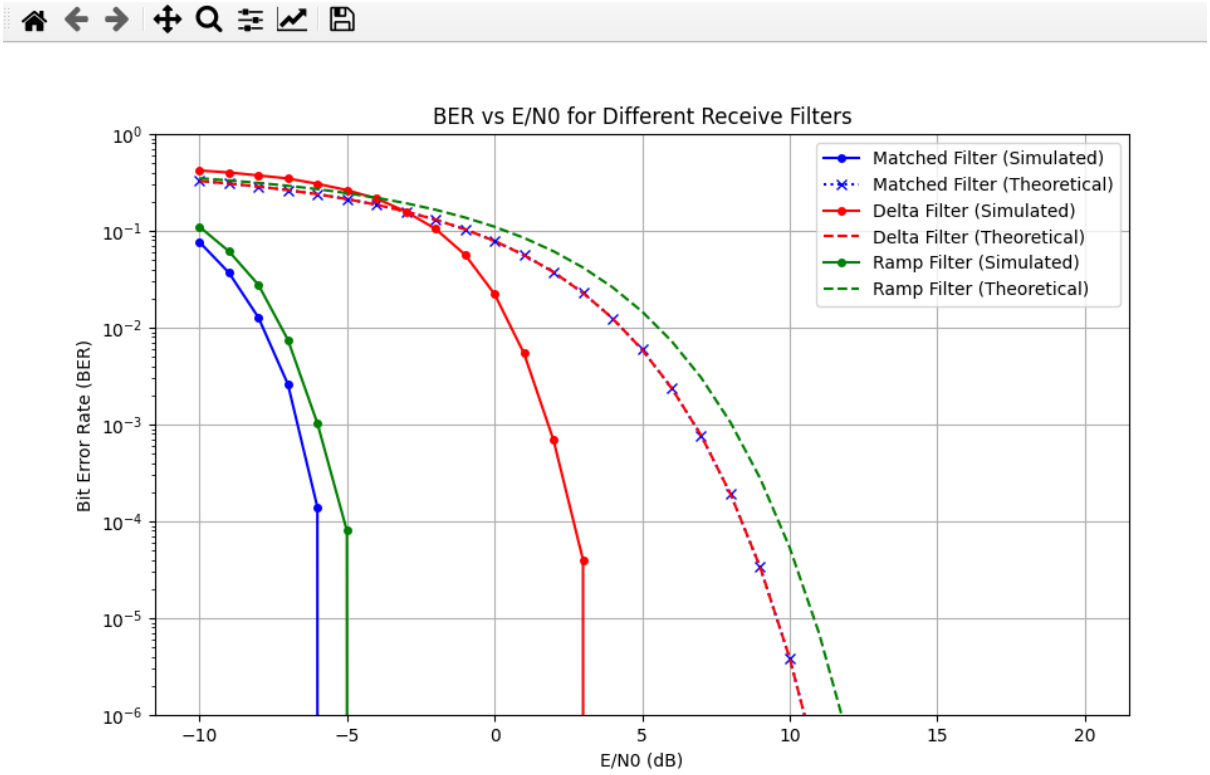


Figure 12: BER vs E/No

2.6 Is the Bit Error Rate (BER) an Increasing or Decreasing Function of E/N_0?

Answer:
The Bit Error Rate (BER) is a decreasing function of E/N_0. As E/N_0 increases, the signal energy becomes larger relative to the noise energy, which makes the signal less prone to errors. Hence, the likelihood of incorrect bit detection decreases.

2.7 Which Case Has the Lowest BER?

Answer:
The matched filter yields the lowest bit error rate (BER) because it convolves the received signal with a time-reversed version of the known signal, thereby maximizing the signal-to-noise ratio (SNR) at the sampling instant. This enhanced SNR at the point of decision reduces the influence of noise and improves detection accuracy.

3 Appendix

3.1 Signal Generation and Processing Functions

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfc

def generate_random_bits(N):
    return np.random.randint(0, 2, N)

def bits_to_bipolar_nrz(bits, A=1, T=1, Fs=100):
    t = np.linspace(0, T, int(Fs * T), endpoint=False)
    pulse = A * np.ones_like(t)
    signal = np.array([pulse if b == 1 else -pulse for b in bits])
    total_time = np.linspace(0, len(bits) * T, len(bits) * len(t), endpoint=False)
    return signal.flatten(), total_time

def add_awgn(signal, N0):
    noise = np.random.normal(0, (N0 / 2), size=len(signal))
    return signal + noise
```

Listing 1: Signal Generation and Processing Functions

3.2 Filter Implementation Functions

```python
def matched_filter(signal, T=1, Fs=100):
    t = np.linspace(0, T, int(Fs * T), endpoint=False)
    h = np.ones_like(t)
    h = np.flip(h)
    h = h / np.sqrt(np.sum(h**2) / Fs)
    return np.convolve(signal, h, mode='same') / Fs

def delta_filter(signal, Fs=100):
    return signal

def triangular_filter(signal, T=1, Fs=100):
    t = np.linspace(0, T, int(Fs * T), endpoint=False)
    h = (np.sqrt(3) / T) * t
    return np.convolve(signal, h, mode='same') / Fs
```

Listing 2: Filter Implementation Functions

3.3 Detection and Analysis Functions

```python
def sample_and_detect(filtered_signal, N, T=1, Fs=100):
    samples_per_bit = int(Fs * T)
    sample_indices = np.arange(samples_per_bit//2, len(filtered_signal), samples_per_bit
    )
    sampled_values = filtered_signal[sample_indices]
    return (sampled_values > 0).astype(int)

def calculate_error_probability(original_bits, detected_bits):
    errors = np.sum(original_bits != detected_bits)
    return errors / len(original_bits)
```

Listing 3: Detection and Analysis Functions

3.4 Theoretical Error Probability Functions

```python
def theoretical_matched_filter_pe(A, T, N0):
    E = A**2 * T
    return 0.5 * erfc(np.sqrt(E/N0))

def theoretical_no_filter_pe(A, N0):
    E = A**2
    return 0.5 * erfc(np.sqrt(E/N0))
```

```

def theoretical_ramp_filter_pe(A, T, N0):
    return 0.5 * erfc((np.sqrt(3) * A * np.sqrt(T)) / (2 * np.sqrt(N0)))
```

Listing 4: Theoretical Error Probability Functions

3.5 Signal Generation Helper Function

```
def generate_noisy_signal(N, N0, A=1, T=1, Fs=1000):
    bits = generate_random_bits(N)
    signal, time = bits_to_bipolar_nrz(bits, A, T, Fs)
    noisy_signal = add_awgn(signal, N0)
    return signal, noisy_signal, time, bits
```

Listing 5: Signal Generation Helper Function

3.6 Simulation Function for Filter Comparison

```
def filters_sim():
    N = 100
    T = 1
    Fs = 50
    A = 1
    N0 = 2

    signal, noisy_signal, time, bits = generate_noisy_signal(N, N0, A, T, Fs)

    matched_filtered = matched_filter(noisy_signal, T, Fs)
    delta_filtered = delta_filter(noisy_signal, Fs)
    triangular_filtered = triangular_filter(noisy_signal, T, Fs)

    samples_per_bit = int(Fs * T)
    sample_indices = np.arange(samples_per_bit//2, len(time), samples_per_bit)
    sample_times = time[sample_indices]

    matched_bits = sample_and_detect(matched_filtered, N, T, Fs)
    delta_bits = sample_and_detect(delta_filtered, N, T, Fs)
    triangular_bits = sample_and_detect(triangular_filtered, N, T, Fs)

    pe_matched = calculate_error_probability(bits, matched_bits)
    pe_delta = calculate_error_probability(bits, delta_bits)
    pe_triangular = calculate_error_probability(bits, triangular_bits)

    plt.figure(figsize=(15, 20))

    plt.subplot(4, 1, 1)
    plt.plot(time, signal, label='Original Signal', alpha=0.7)
    plt.plot(time, noisy_signal, label='Signal with AWGN', alpha=0.7)
    plt.title('Original Signal and Noisy Signal', pad=10)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)

    plt.subplot(4, 1, 2)
    plt.plot(time, matched_filtered, label='Matched Filter', alpha=0.7)
    plt.plot(sample_times, matched_filtered[sample_indices], 'ro', label='Sampling
    Points', markersize=5)
    plt.title('Signal after Matched Filter', pad=10)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)
    plt.text(0.5, -0.2, f'Probability of Error (Pe) = {pe_matched:.4f}',
             horizontalalignment='center', verticalalignment='center',
             transform=plt.gca().transAxes, fontsize=10, bbox=dict(facecolor='white',
    alpha=0.8))

    plt.subplot(4, 1, 3)
    plt.plot(time, delta_filtered, label='Delta Filter', alpha=0.7)
```

```
    plt.plot(sample_times, delta_filtered[sample_indices], 'ro', label='Sampling Points', markersize=5)
    plt.title('Signal after Delta Filter (No Filtering)', pad=10)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)
    plt.text(0.5, -0.2, f'Probability of Error (Pe) = {pe_delta:.4f}',
             horizontalalignment='center', verticalalignment='center',
             transform=plt.gca().transAxes, fontsize=10, bbox=dict(facecolor='white', alpha=0.8))

    plt.subplot(4, 1, 4)
    plt.plot(time, triangular_filtered, label='Triangular Filter', alpha=0.7)
    plt.plot(sample_times, triangular_filtered[sample_indices], 'ro', label='Sampling Points', markersize=5)
    plt.title('Signal after Triangular Filter', pad=10)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)
    plt.text(0.5, -0.2, f'Probability of Error (Pe) = {pe_triangular:.4f}',
             horizontalalignment='center', verticalalignment='center',
             transform=plt.gca().transAxes, fontsize=10, bbox=dict(facecolor='white', alpha=0.8))

    plt.subplots_adjust(hspace=0.6, bottom=0.1)
    plt.show()
```

Listing 6: Simulation Function for Filter Comparison

3.7 BER vs. E/N0 Simulation

```
def ber_vs_eno_sim():
    N = 100000
    T = 1
    Fs = 50
    A = 1

    E_div_N0_dB = np.arange(-10, 21)

    BER_simulated_match = []
    BER_theoretical_match = []
    BER_simulated_delta = []
    BER_theoretical_delta = []
    BER_simulated_ramp = []
    BER_theoretical_ramp = []

    for E_div_N0_db in E_div_N0_dB:
        E_div_N0 = 10**(E_div_N0_db/10)
        E = A**2 * T
        N0 = E / E_div_N0

        bits = generate_random_bits(N)
        signal, time = bits_to_bipolar_nrz(bits, A, T, Fs)
        noisy_signal = add_awgn(signal, N0)

        matched_filtered = matched_filter(noisy_signal, T, Fs)
        delta_filtered = delta_filter(noisy_signal, Fs)
        triangular_filtered = triangular_filter(noisy_signal, T, Fs)

        matched_bits = sample_and_detect(matched_filtered, N, T, Fs)
        delta_bits = sample_and_detect(delta_filtered, N, T, Fs)
        triangular_bits = sample_and_detect(triangular_filtered, N, T, Fs)

        BER_simulated_match.append(calculate_error_probability(bits, matched_bits))
        BER_simulated_delta.append(calculate_error_probability(bits, delta_bits))
        BER_simulated_ramp.append(calculate_error_probability(bits, triangular_bits))

        BER_theoretical_match.append(theoretical_matched_filter_pe(A, T, N0))
        BER_theoretical_delta.append(theoretical_no_filter_pe(A, N0))
        BER_theoretical_ramp.append(theoretical_ramp_filter_pe(A, T, N0))
```

```
40
41      plt.figure(figsize=(10, 6))
42      plt.semilogy(E_div_N0_dB, BER_simulated_match, 'bo-', label='Matched Filter (
        Simulated)', markersize=4)
43      plt.semilogy(E_div_N0_dB, BER_theoretical_match, 'bx:', label='Matched Filter (
        Theoretical)', markersize=6)
44      plt.semilogy(E_div_N0_dB, BER_simulated_delta, 'ro-', label='Delta Filter (Simulated
        )', markersize=4)
45      plt.semilogy(E_div_N0_dB, BER_theoretical_delta, 'r--', label='Delta Filter (
        Theoretical)')
46      plt.semilogy(E_div_N0_dB, BER_simulated_ramp, 'go-', label='Ramp Filter (Simulated)'
        , markersize=4)
47      plt.semilogy(E_div_N0_dB, BER_theoretical_ramp, 'g--', label='Ramp Filter (
        Theoretical)')
48
49      plt.grid(True)
50      plt.xlabel('E/N0 (dB)')
51      plt.ylabel('Bit Error Rate (BER)')
52      plt.title('BER vs E/N0 for Different Receive Filters')
53      plt.legend()
54      plt.ylim(1e-6, 1)
55      plt.show()
```

Listing 7: BER vs. E/N0 Simulation