Tushar prabhu

PES1UG22EC321

AHP Component – 1

```
import numpy as np
import heapq

# Define the symbols and their probabilities
S = ["s"+str(i) for i in range(5)]
pk = [0.4,0.2,0.2,0.1,0.1]

# Create a priority queue with the probabilities and symbols
queue = [[weight, [symbol, ""]] for weight, symbol in zip(pk, S)]
heapq.heapify(queue)

# While there are more than one items in the queue
while len(queue) > 1:
    # Get the two symbols with the smallest probabilities
    lo = heapq.heappop(queue)
    hi = heapq.heappop(queue)

    # Append '0' to the Huffman code of the first symbol and '1' to the
second
    for pair in lo[1:]:
        pair[1] = '0' + pair[1]
    for pair in hi[1:]:
        pair[1] = '1' + pair[1]

    # Add a new item to the queue with the combined probability and both
symbols
    heapq.heappush(queue, [lo[0] + hi[0]] + lo[1:] + hi[1:])

# Print the Huffman codes
huff_codes = sorted(heapq.heappop(queue)[1:], key=lambda p: (len(p[-1]),
p))
for symbol, huff_code in huff_codes:
    print(f"{symbol} : {huff_code}")
```

AHP Component – 2

A) Product Modulator

On changing the above parameters we observe the following the changes:

1) fc:

Effect on the Spectrum: As the carrier frequency the bandwidth of the modulated signal increases in the spectrum.

Effect on the Graphs: In the time domain , higher the carrier frequency faster is the rate of frequency variations according to the modulated signal.

2) fm:

Effect on Spectrum: As the fm increases the bandwidth of the modulated signal increases.

Effect on Graph: The waveform exhibits more oscillations in a given interval of time.

3) Ac:

Effect on Spectrum: As the Ac increases, this results in a more powerful or strongly

modulated signal.

Effect on Graph: A larger Ac generally results in a higher amplitude of modulated signal.

4) Am:

B)Envelope Detector:

Effect on Spectrum: As the Ac increases, this results in a more powerful or strongly

modulated signal.

Effect on Graph: A larger Ac generally results in a higher amplitude of modulated signal. When the Am is too high , it can lead to overmodulation and causes distortions in the graphs.

The output obtained from the envelope detector must be similar to the input signal since it is demodulating the initially modulated signal. However there certain parameters that affect the speed of the response.

1) Rs (Series Resistor):

A larger resistance value provides a smoother output similar to the initial signal, however the time required to produce the response increases. Hence a larger the resistance guarantees a more efficient output but increases the response time.

2)C (Capacitor):

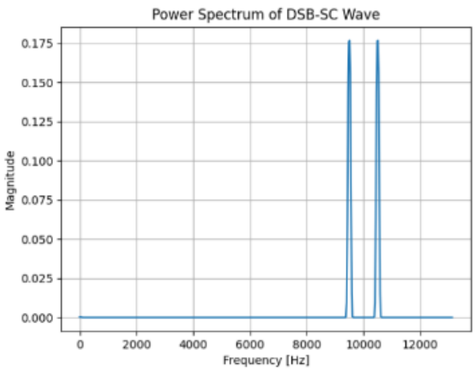
A larger capacitor value provides a more smooth output similar to the initial signal, however the time required to produce the response increases. Hence a larger the capacitance guarantees a more efficient output but increases the response time.

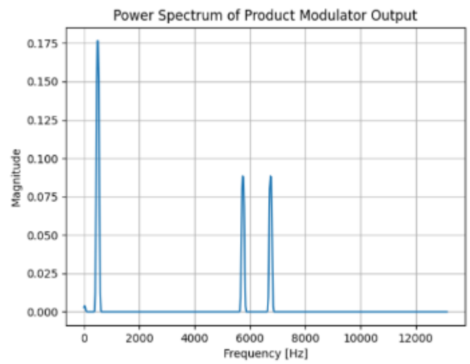
Conclusion: Hence there is a Trade-Off between smoothness of the signal and the response time of the demodulated signal.

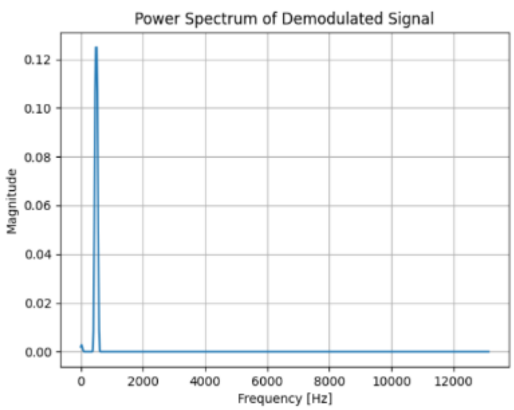
AHP COMPONENT – 2

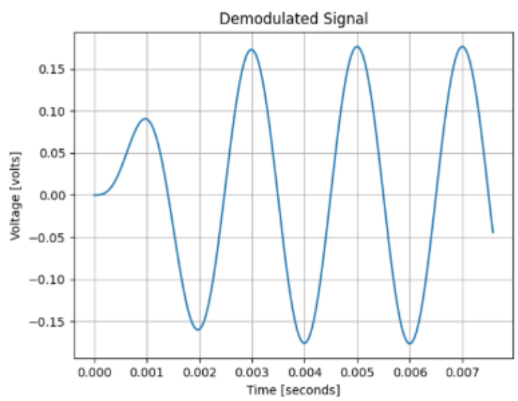
Generation and demodulation of DSB-SC Signal:

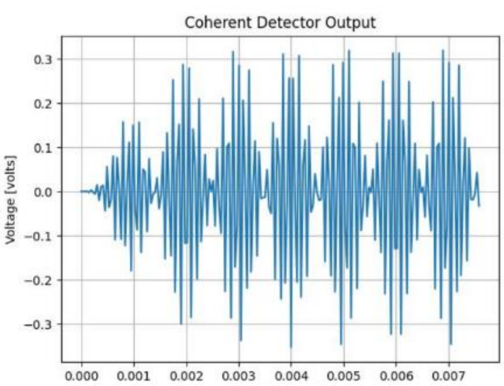
Graphs:











3/22/2024

AHP Component – 3

Generation and demodulation of DSB-SC signal

```
from scipy import signal
from scipy.signal import butter, lfilter
import numpy as np
import matplotlib.pyplot as plt

N = 1e5 # Number of samples for visualization as a waveform
Ac = 1 # Carrier peak amplitude
Am = 0.5 # Message signal peak amplitude
freq_c = 10e3 # carrier frequency
freq_m = 500 # message frequency
fs = 2.5*(freq_c+freq_m) # sampling frequency is at least twice the
highest frequency

def butter_lowpass(cutoff, fs, order=4):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq # the critical frequencies must be
normalized
    b, a = butter(order, normal_cutoff, btype='lowpass', analog=False)
    return b, a

def butter_lowpass_filter(data, cutoff, fs, order=4):
    b, a = butter_lowpass(cutoff, fs, order=order) # a and b coefficients
of the LPF modeled as FIR/IIR filter
    y = lfilter(b, a, data) # filter output
    return y

order=4
cutoff=freq_m

time = np.arange(N) / fs # time instants of the samples of duration (1/fs)
seconds
c = Ac*np.cos(2*np.pi*freq_c*time) # Carrier signal
m = Am*np.cos(2*np.pi*freq_m*time) # Message signal
s= m*c # DSB-SC signal
y=s*c # product modulator output at receiver

x = butter_lowpass_filter(y, cutoff, fs, order) # demodulated signal

#
#### Either use signal.welch or signal.spectrogram
# Set first argument to s, y or x to see the spectrum of DSBSC wave,
product modulator output at the receiver or demodulated signal
f, Pxx_spec = signal.welch(s, fs, 'flattop', 1024,
scaling='spectrum',return_onesided=True) # for two sided spectrum set
False
# f, t, Pxx_spec = signal.spectrogram(s, fs)
```

```
plt.figure()
plt.plot(f,np.sqrt(Pxx_spec)) # magnitude spectrum
# plt.plot(f,Pxx_spec) # power spectrum
#plt.semilogy(f, np.sqrt(Pxx_spec)) # magnitude spectrum in decibels
(20*log_10(value))
plt.xlabel('frequency [Hz]')
plt.ylabel('Magnitude')
plt.title('Power spectrum')
plt.show()

plt.plot(time[0:200],x[0:200]) # demodulated signal waveform
plt.xlabel('Time (seconds)')
plt.ylabel('Voltage [volts]')
plt.title('Demodulated signal')
plt.grid()
```

SRN : PES1UG22EC321

AHP – 2

Observation:

- Increasing fs will improve frequency resolution and may provide a more accurate representation of the power spectrum.

- Decreasing fs below the Nyquist frequency could lead to aliasing, distorting the power spectrum.

- By varying the above parameters in the power spectrum graph of the modulated and demodulated graph we observe that :

1. Number of Samples (N):

- Increasing N will improve frequency resolution in the power spectrum graphs, allowing for a clearer representation of signal components.

- Decreasing N may lead to spectral leakage and less accurate frequency information.

2. Carrier Peak Amplitude (Ac):

- Increasing Ac will increase the amplitude of the carrier signal, affecting the amplitude of the sidebands in the power spectrum of the modulated signal.

3. Message Signal Peak Amplitude (Am):

- Increasing Am will increase the modulation depth and potentially result in a broader frequency spectrum with more pronounced sidebands.

4. Carrier Frequency (freq_c):

- Increasing freq_c will shift the entire power spectrum to higher frequencies.

5. Message Frequency (freq_m):

- Increasing freq_m will shift the spectrum to higher frequencies and may increase the

spacing between sidebands in the modulated signal's power spectrum.

6. Sampling Frequency (fs): Like taking faster pictures, it helps capture the signals more accurately, especially for fast changes.Increasing fs will improve frequency resolution in the power spectrum graphs.

AHP Component – 4

PART-A

Python Code:

a)

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

def continuous_time_signal(t):

# Example continuous-time sinc

return np.sinc(0.25 * (t - 5))**2

def ideal_sampling(signal, fs, Ts):

# Ideal sampling of continuous-time signal

sampled_indices = np.arange(0, len(signal), int(fs * Ts))

sampled_signal = signal[sampled_indices]

sampled_time = np.arange(0, len(signal), int(fs * Ts)) / fs

return sampled_time, sampled_signal


def ideal_reconstruction(sampled_time, sampled_signal, fs):

# Ideal reconstruction using a low-pass filter

reconstructed_signal = signal.lfilter([1], [1, 0], sampled_signal)

reconstructed_time = np.linspace(0, sampled_time[-1], len(reconstructed_signal))
```

```
return reconstructed_time, reconstructed_signal

# Parameters

fs_original = 1000 # Original continuous-time signal frequency

Ts_original = 1/fs_original # Original continuous-time signal sampling interval

# Generate continuous-time sinc signal

t_continuous = np.linspace(0, 10, 1000)

signal_continuous = continuous_time_signal(t_continuous)

# Ideal sampling

fs_sampled = 100 * fs_original # Choose sampling frequency (e.g., 100 times higher)

Ts_sampled = 1/fs_sampled

sampled_time, sampled_signal = ideal_sampling(signal_continuous, fs_sampled, Ts_sampled)

# Ideal reconstruction

reconstructed_time, reconstructed_signal = ideal_reconstruction(sampled_time, sampled_signal, fs_sampled)

# Plotting

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

plt.subplot(3, 1, 1)

plt.plot(t_continuous, signal_continuous, label='Continuous-Time Signal')

plt.title('Continuous-Time Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.legend()

plt.subplot(3, 1, 2)

plt.stem(sampled_time, sampled_signal, markerfmt='ro', basefmt='r', label='Sampled Signal')

plt.title('Sampled Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.legend()

plt.subplot(3, 1, 3)

plt.plot(reconstructed_time, reconstructed_signal, label='Reconstructed Signal')

plt.title('Reconstructed Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.legend()

plt.tight_layout()

plt.show()
```

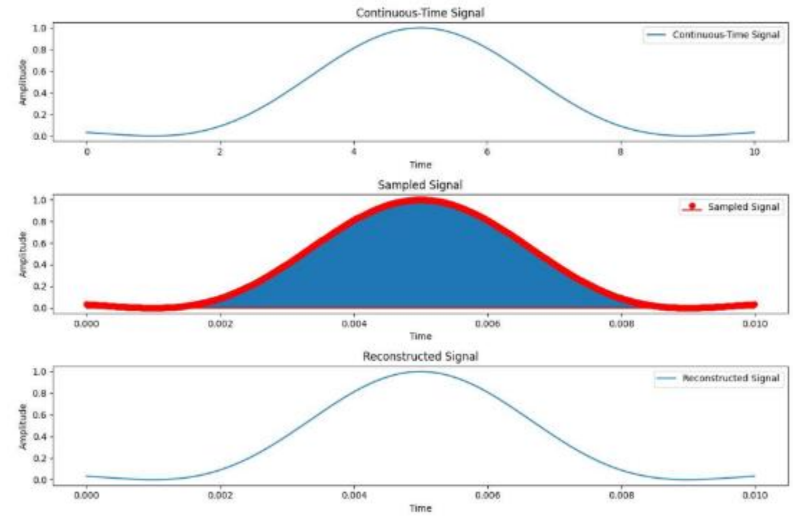
b)

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

def continuous_time_signal(t):


return 2 * np.cos(6 * np.pi * t + 0.1) + 2 * np.cos(10 * np.pi * t + 0.2)

def ideal_sampling(signal, fs, Ts):

# Ideal sampling of continuous-time signal

sampled_indices = np.arange(0, len(signal), int(fs * Ts))

sampled_signal = signal[sampled_indices]

sampled_time = np.arange(0, len(signal), int(fs * Ts)) / fs

return sampled_time, sampled_signal

def ideal_reconstruction(sampled_time, sampled_signal, fs):

# Ideal reconstruction using a low-pass filter

reconstructed_signal = signal.lfilter([1], [1, 0], sampled_signal)

reconstructed_time = np.linspace(0, sampled_time[-1], len(reconstructed_signal))

return reconstructed_time, reconstructed_signal

# Parameters

fs_original = 1000 # Original continuous-time signal frequency

Ts_original = 1/fs_original # Original continuous-time signal sampling interval

# Generate continuous-time sinc signal

t_continuous = np.linspace(0, 1, 1000)

signal_continuous = continuous_time_signal(t_continuous)

# Ideal sampling

fs_sampled = 100 * fs_original # Choose sampling frequency (e.g., 100 times higher)

Ts_sampled = 1/fs_sampled

sampled_time, sampled_signal = ideal_sampling(signal_continuous, fs_sampled, Ts_sampled)

# Ideal reconstruction

reconstructed_time, reconstructed_signal = ideal_reconstruction(sampled_time, sampled_signal, fs_sampled)

# Plotting

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

plt.subplot(3, 1, 1)

plt.plot(t_continuous, signal_continuous, label='Continuous-Time Signal')
```

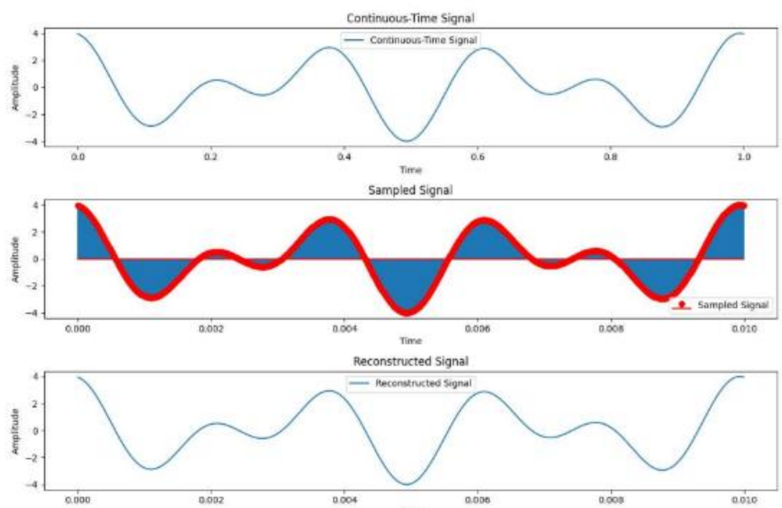
```
plt.title('Continuous-Time Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.subplot(3, 1, 2)
plt.stem(sampled_time, sampled_signal, markerfmt='ro', basefmt='r', label='Sampled Signal')
plt.title('Sampled Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.subplot(3, 1, 3)
plt.plot(reconstructed_time, reconstructed_signal, label='Reconstructed Signal')
plt.title('Reconstructed Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.tight_layout()
plt.show()
```

Graphs For Ideal Sampling and Reconstruction of the Input Continuous Time Signal

A. sinc(0.25*(t - 5))**2



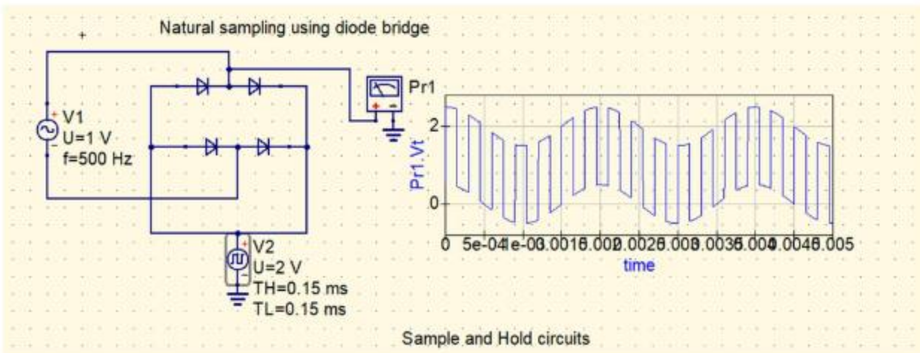
B. cos(6 * pi * t + 0.1) + 2 * cos(10 * pi * t + 0.2)



PART-B

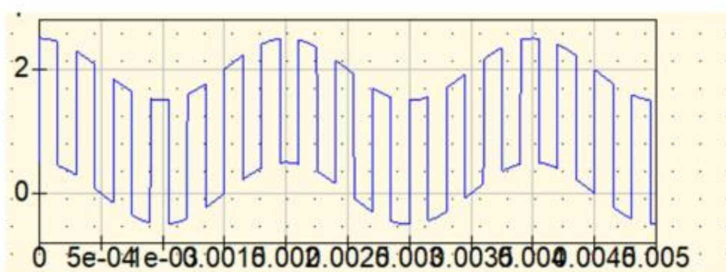
1)

(a)

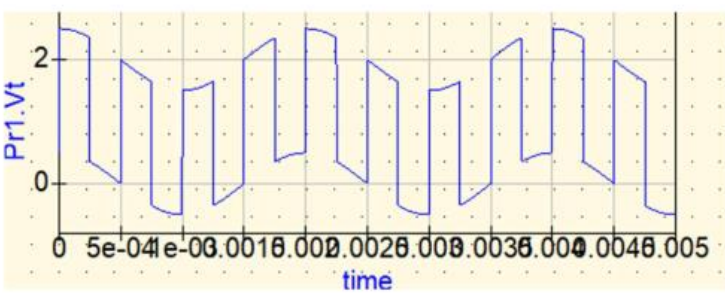


On changing the values of duration of the high pulse and low pulse we obtain the following graphs:

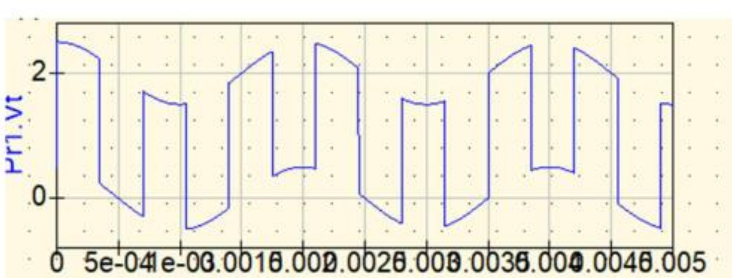
TH = TL = 0.15ms



TH = TL = 0.25ms

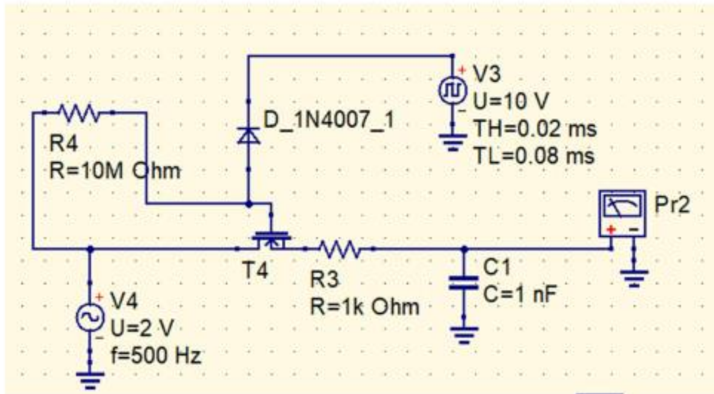


TH = TL = 0.35ms

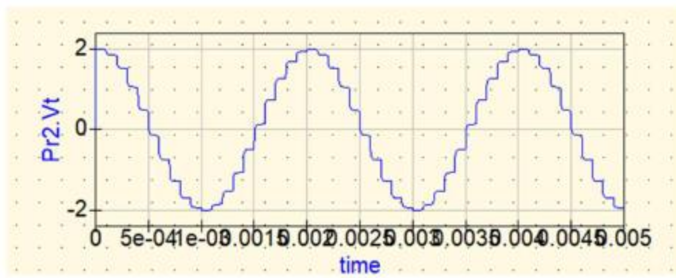


Hence it is observed that if the TH and TL values are increased the Frequency of the Output Voltages gets significantly decreased .

Circuit Diagram:

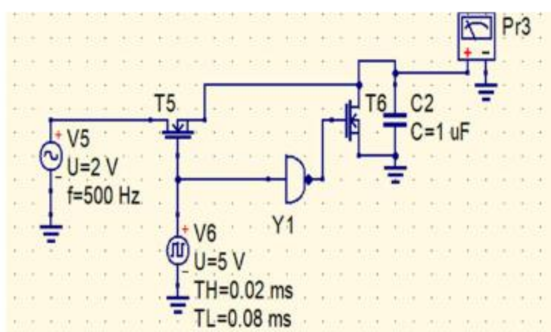


Graphs:

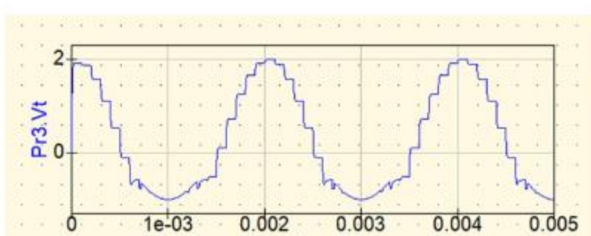


(c)

Circuit Diagram:

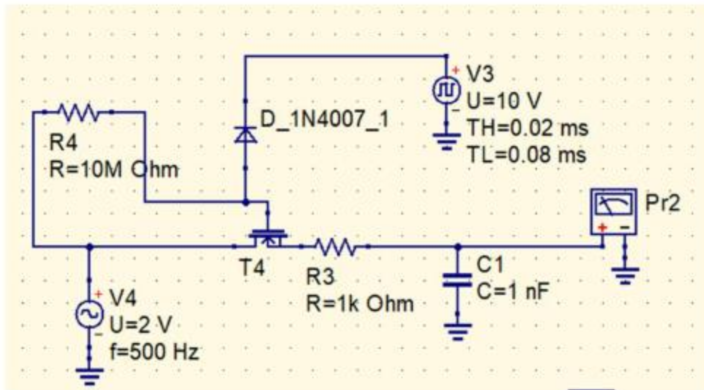


Graph:

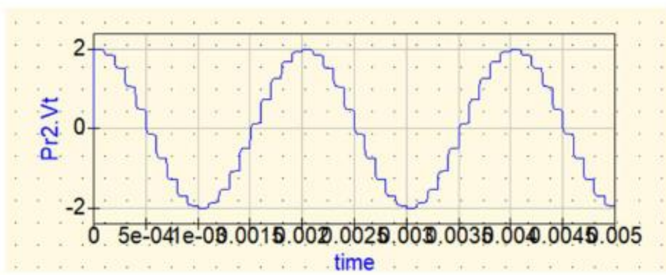


(b)

Circuit Diagram:

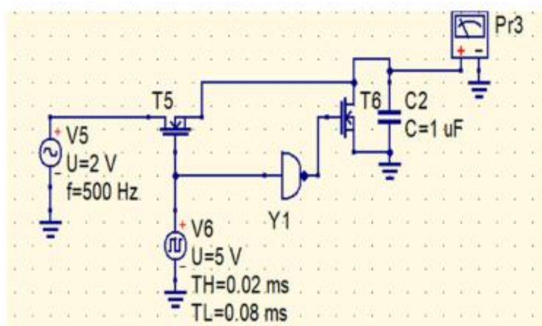


Graphs:

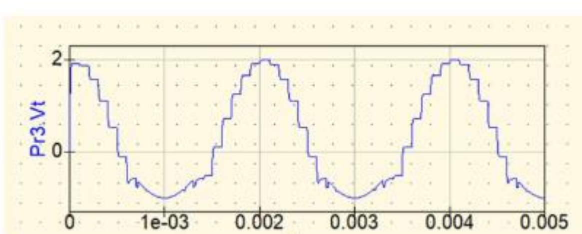


(c)

Circuit Diagram:

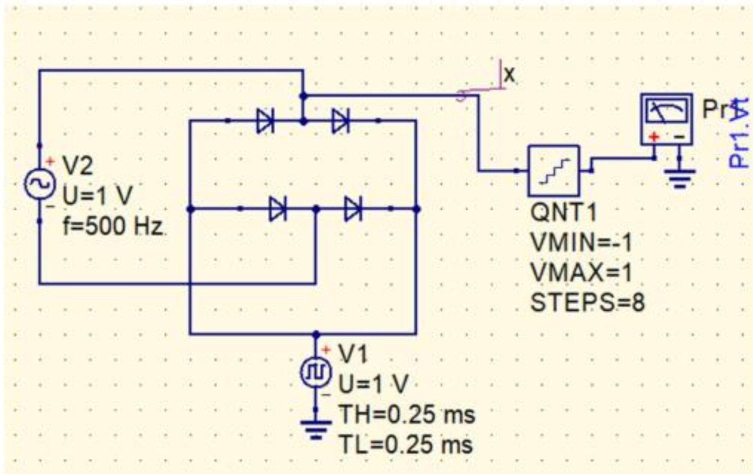


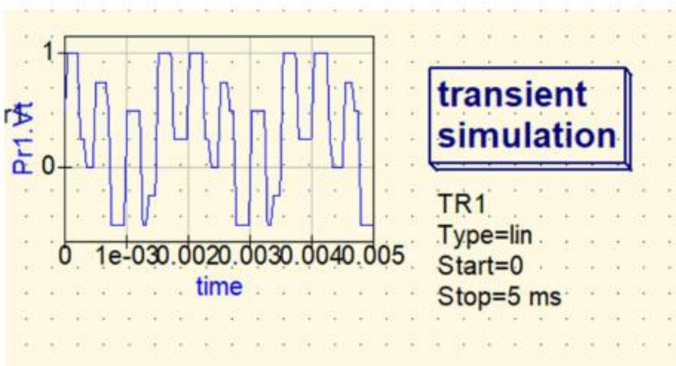
Graph:



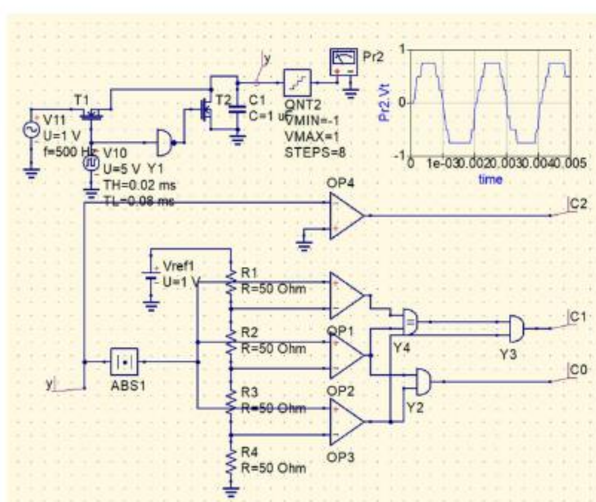
AHP Component – 5

1)



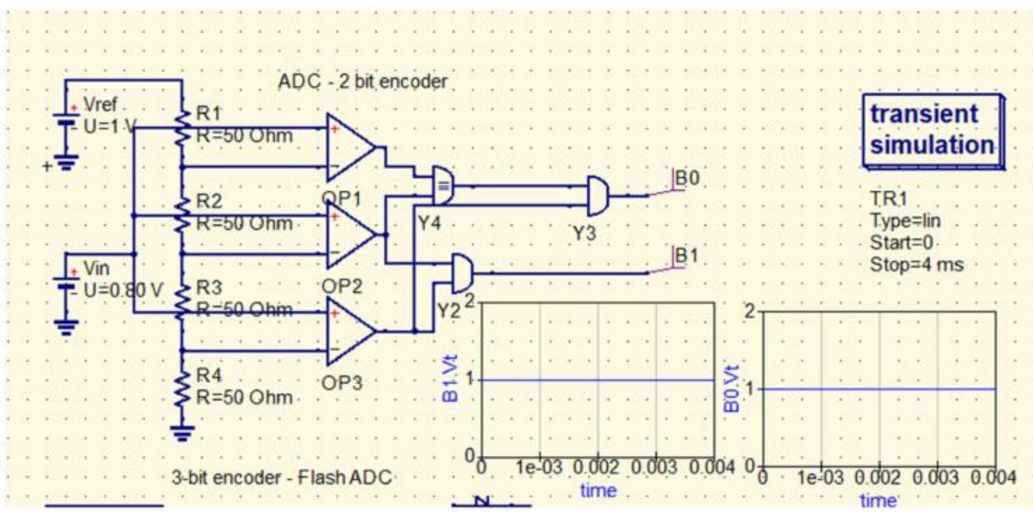


2)

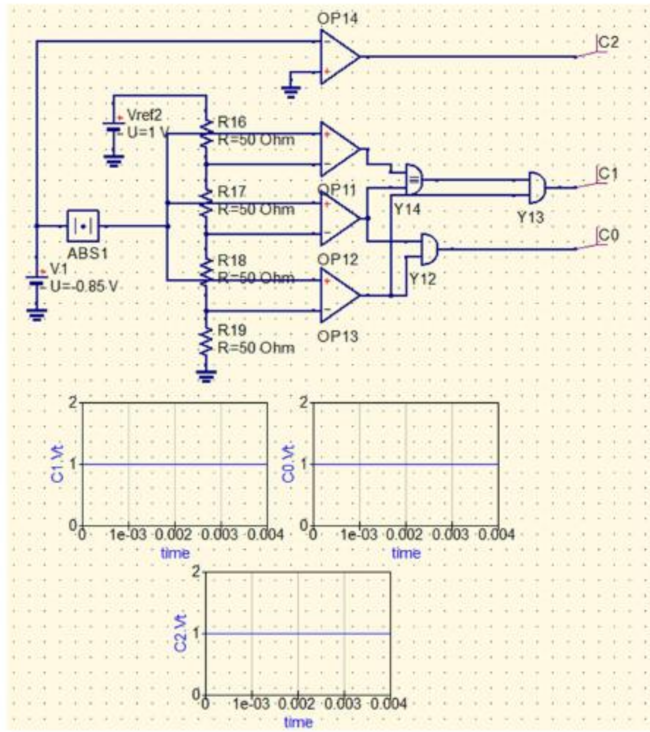


time	C2.Vt	C1.Vt	C0.Vt	x.Vt
5.05e-05	-1	1e-12	5e-13	1.49
0.000101	-1	1e-12	5e-13	1.48
0.000152	-1	1	1e-12	1.44
0.000202	-1	9.08e-05	1	1.4
0.000253	-1	9.08e-05	1	0.351
0.000303	-1	9.08e-05	1	0.29
0.000354	-1	1	1	0.222
0.000404	-1	1	1	0.148
0.000455	-1	1	1	0.0712
0.000505	-1	1	1	0.992
0.000556	-1	1	1	0.913
0.000606	-1	1	1	0.836
0.000657	-1	1	1	0.764
0.000707	-1	1	1	0.697
0.000758	-1	1	1	-0.362
0.000808	-1	9.08e-05	1	-0.412
0.000859	-1	9.08e-05	1	-0.451
0.000909	-1	1	1e-12	-0.48
0.00096	-1	1	1e-12	-0.496
0.00101	-1	1e-12	5e-13	0.5
0.00106	1	1e-12	5e-13	0.509
0.00111	1	1	1e-12	0.53
0.00116	1	1	1e-12	0.563

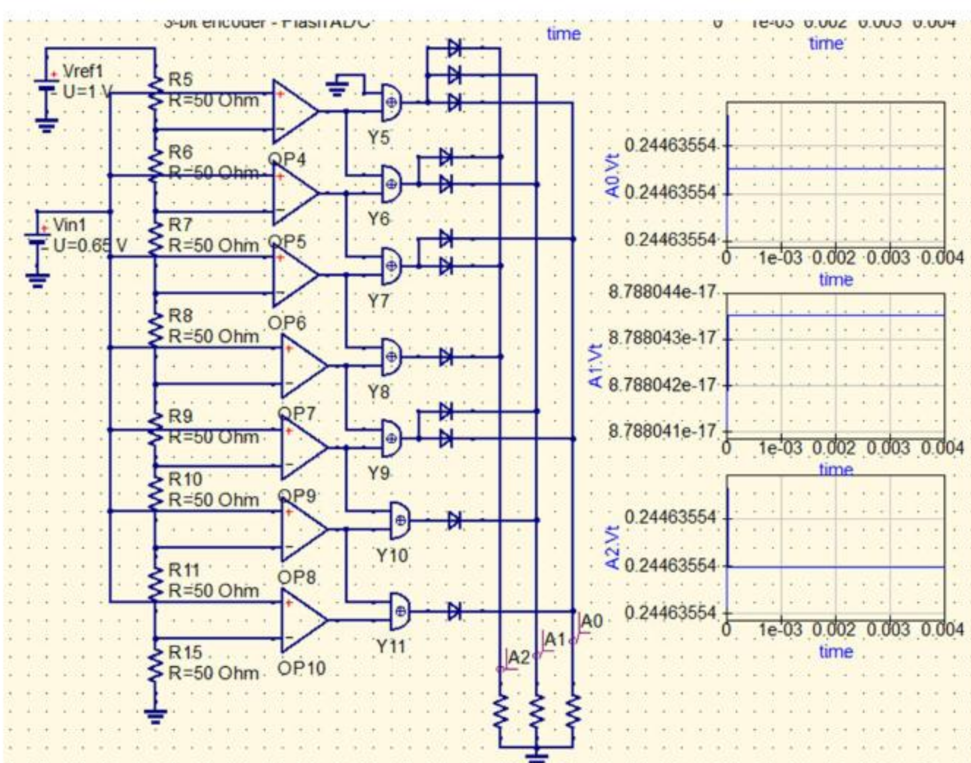
3)



4)


OP14
C2
Vref2
U=1 V
R16
R=50 Ohm
OP11
C1
Y14
Y13
R17
R=50 Ohm
ABS1
V1
U=-0.85 V
R18
R=50 Ohm
OP12
Y12
C0
OP13
R19
R=50 Ohm
C1.Vt
C0.Vt
C2.Vt
time

5)


Vref1
U=1 V
R5
R=50 Ohm
Y5
R6
R=50 Ohm
OP4
Y6
Vin1
U=0.65 V
R7
R=50 Ohm
OP5
Y7
R8
R=50 Ohm
OP6
Y8
R9
R=50 Ohm
OP7
Y9
R10
R=50 Ohm
OP9
Y10
R11
R=50 Ohm
OP8
Y11
R15
R=50 Ohm
OP10
A0
A1
A2
A0.Vt
0.24463554
A1.Vt
8.788044e-17
8.788043e-17
8.788042e-17
8.788041e-17
A2.Vt
0.24463554
time