LAB TITLE AND CODE : 19CLE284 SOFTWARE DEFINED RADIO LAB
EXPERIMENT NUMBER : 4
DATE : 09/05/2022, MONDAY

POLYPHASE STRUCTURES

* AIM:
For the filters designed to be used in interpolators and decimators, implement the simplified polyphase architecture. Analyze the complexity and latency of the implemented filters.

* SOFTWARE REQUIRED :
Spyder IDE - Python 3.9

* CODE: ORDER OF FILTER = 2)

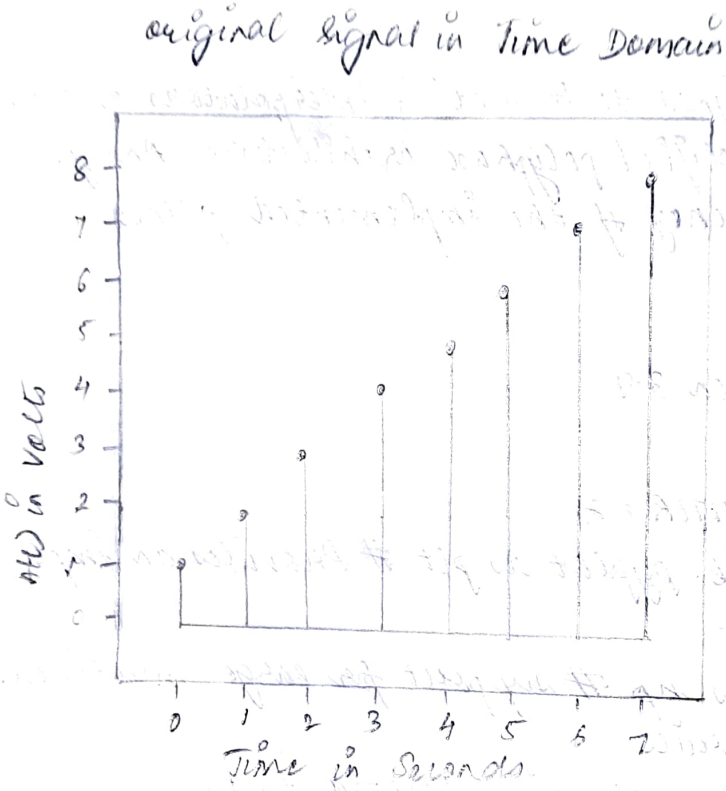
```
import matplotlib.pyplot as plt # Provides an implicit way of plotting
import numpy as np # support for large, multi-dimensional arrays and matrices
import time # Handle time-related tasks

# Plot time-domain spectrum of given signal :-
def time_domain(signal, plot_name):

    plt.xlabel("Time in Seconds")
    plt.ylabel("Alt) in Volts")
    plt.title("" + str(plot_name) + " in Time Domain")
    plt.stem(np.arange(0, len(signal)), signal)
    plt.grid()
    plt.show()
```

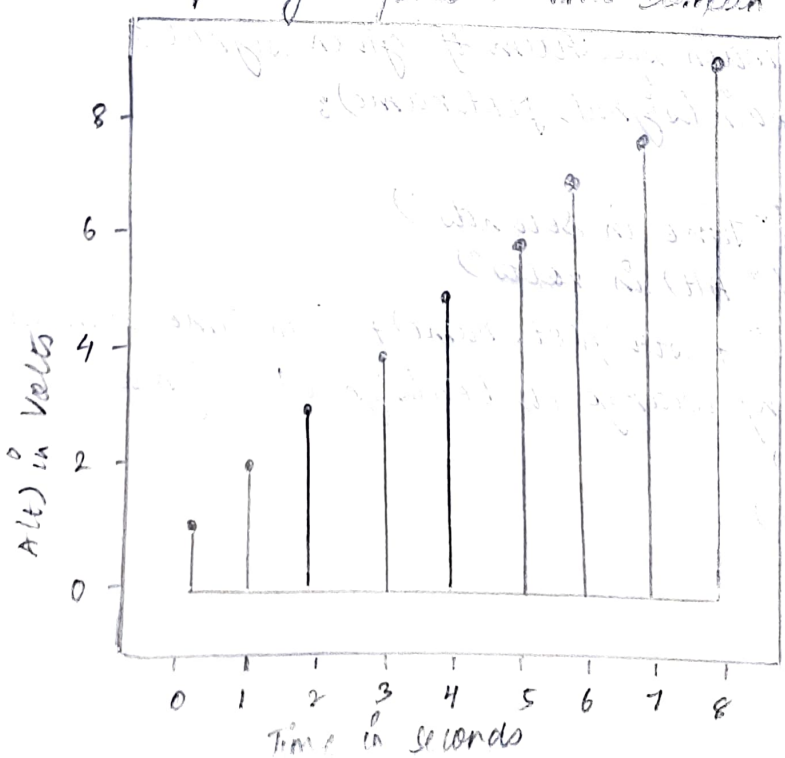
PTO

```
# Down-sample the given signal by a factor of "m":-
def down-sample(signal, m):
    down-sampler = []
    # Copy the element value from original signal to down
sampler array with the increment value set to "m":
    for i in range (0, len(signal), m):
        down-sampler.append(signal[i])
    return down-sampler

# Up-sample the given signal by a factor of "l":-
def up-sample (signal, l)
    up-sampler = []
    for i in range (len(signal)):
        up-sampler.append(signal[i]) # Copy the element value
from original signal to up sampler array
        if i != len(signal)-1:
            for k in range
                up-sampler.append(0) # Insert "l-1" zeros between the
elements of the array
        else:
            break
    return up-sampler

# shift the samples of given signal by "phase-shift" units:-
def shift-samples(signal, phase-shift)
    for i in range (phase-shift):
        signal.append(0) # Adds required zeros at the end of list
    for i in range (len(signal)-1, 0, -1):
        signal[i] = signal[i-1] # shifts the zeros to the beginning of
the list
    return signal
```

→ OUTPUT (CONSOLE & PLOTS) : (ORDER OF FILTER = 2)

The original signal is : [1, 2, 3, 4, 5, 6, 7, 8]
The frequency response is : [1, 2, 3, 4, 5, 6, 7, 8, 9]
The order of the filter is 2.





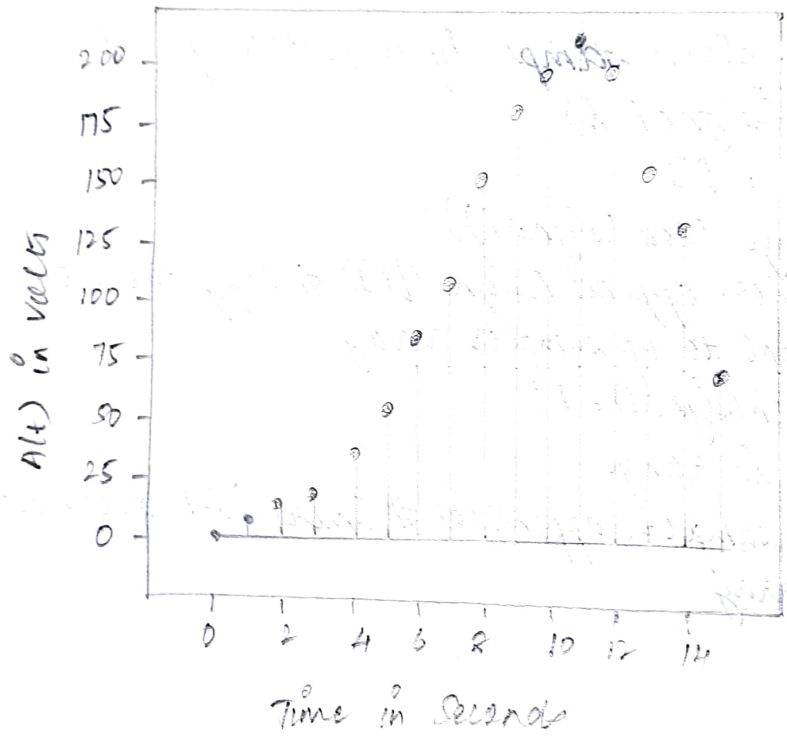
```
# Perform convolution, x[n]* h[n] :-
def convolution (x, h):

    # string padding refers to adding, usually, non-informative
characters to a string to one or both ends of it. This is most often
done for output formatting and alignment purposes, but it can
have useful practical applications. numpy.pad() function is used to
pad the numpy arrays.

    size = (len(x) + len(h)) - 1 # Compute the size of system
response
    x = np.pad(x, (0, size - len(x)), 'constant')
    h = np.pad(h, (0, size - len(h)), 'constant')
    y = np.zeros(size, dtype = int) # Returns a new array of given
shape and type, with zeros.

    for i in range(size):
        for j in range(size):
            if i >= j:
                y[i] = int(y[i] + (int(x[i-j]) * int(h[j])))

    return y # System Response y[n]

# Driver Code : main(); Execution starts here.

# Generate the original signal :-
title = "original signal"
original_signal = [1, 2, 3, 4, 5, 6, 7, 8]
print("The original response is: ", original_signal)
time_domain(original_signal, title)
```

PTO

The system response after convolution for 'Traditional Filter - Direct Representation' is : [1 4 10 20 35 56 84 120 156 182 197 200 190 166 127 72]

The time of execution of 'Traditional Filter - Direct Representation' is (in seconds) : 0.14633822441101074

Traditional Filter - Direct Representation in Time Domain



e0[n] : [1, 0, 3, 0, 5, 0, 7, 0, 9]

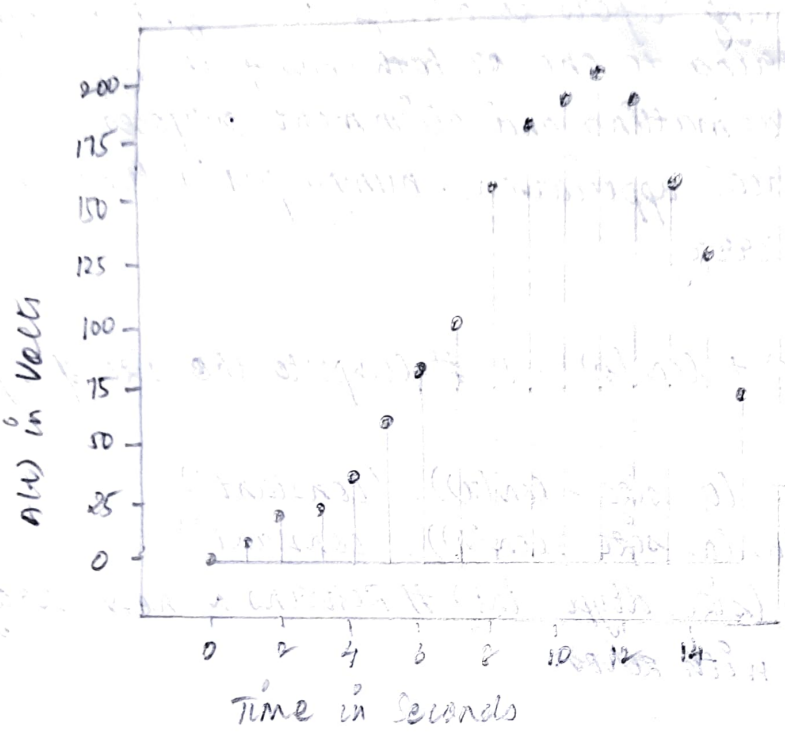
e1[n] : [0, 2, 0, 4, 0, 6, 0, 8, 0]

The system response after summing up the individual convolutions for 'Polyphase Filter Representation' is : [1 4 10 20 35 56 84 120 156 182 197 200 190 166 127 72]

The time of execution of 'Polyphase Filter Representation' is (in seconds) : 0.16259789466857910

```
# Generate the frequency response:-
title = "Frequency Response"
frequency_response = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print("The frequency response is: ", frequency_response)
time_domain(frequency_response, title)

m = int(input("Enter the order of filter: "))

# Case - I:-
start = time.time()
title_I = "Traditional Filter- Direct Representation"
output_I = convolution(original_signal, frequency_response)
print("\n The system response after convolution for '" +
str(title_I) + "' is: ", output_I)
time_domain(output_I, title_I)
end = time.time()
print("The time of execution of '" + str(title_I) + "' is (in
seconds): ", end-start)

print("\n")

Case - II:-
start = time.time()

title_II = "Polyphase Filter Representation"
down_sampler_0 = down_sample(frequency_response, m)
up_sampler_0 = up_sample(down_sampler_0, m)
print("E0[n]: ", up_sampler_0)

phase_shift = m-1
delay_signal = shift_samples(frequency_response, phase_shift)
down_sampler_1 = down_sample(delay_signal, m)
up_sampler_1 = up_sample(down_sampler_1, m)
```

Polyphase Filter Representation in Time Domain



```
for i in range (m-1):
    up_sampler_1.remove (up_sampler_1 [i])
    up_sampler_1.append (0)
print ("H[n]: ", up_sampler_1)

convolve_I = convolution (original_signal, up_sampler_0)
convolve_II = convolution (original_signal, up_sampler_1)
output_II = convolve_I + convolve_II
print ("\n The system response after summing up the individual
convolutions for "" + str (title_II) + "" is: ", output_II)
time_domain (output_II, title_II)
end = time.time ()

print ("The time of execution of '" + str (title_II) + "' is (in
seconds): ", end-start)
```

* CODE: (ORDER OF FILTER = 3)

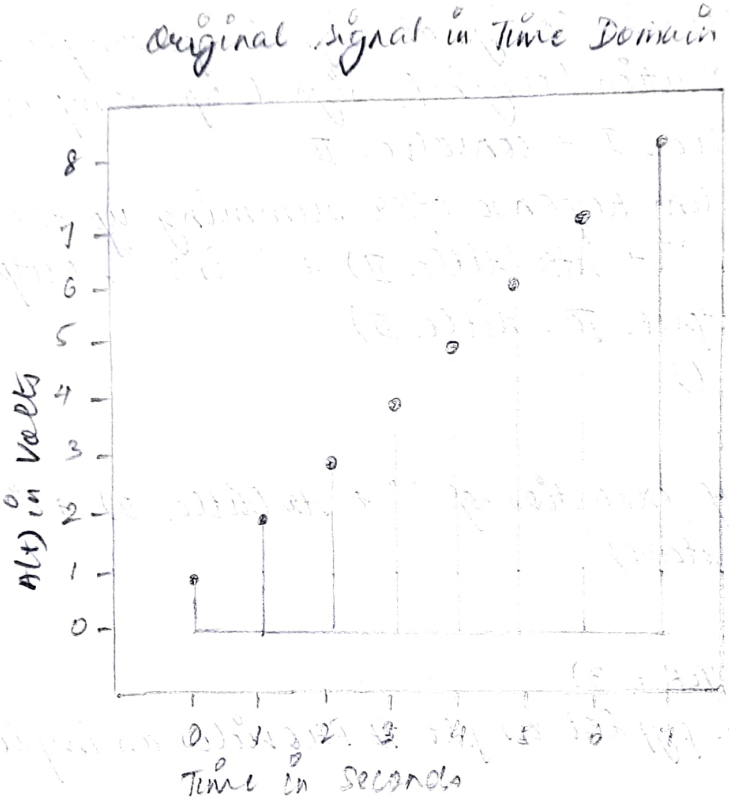
```
import matplotlib.pyplot as plt # Provides an implicit way of
plotting
import numpy as np # Support for large, multi-dimensional
arrays and matrices
import time # Handle time-related tasks

# Plot time_domain spectrum of given signal:-
def time_domain (signal, plot_name):

    plt.xlabel ("Time in Seconds")
    plt.ylabel ("A(t) in Volts")
    plt.title ("Spectral Analysis of '" + str (plot_name) + "' in
Time Domain")
    plt.stem (np.arange (0, len(signal)), signal)
    plt.grid ()
    plt.show ()
```

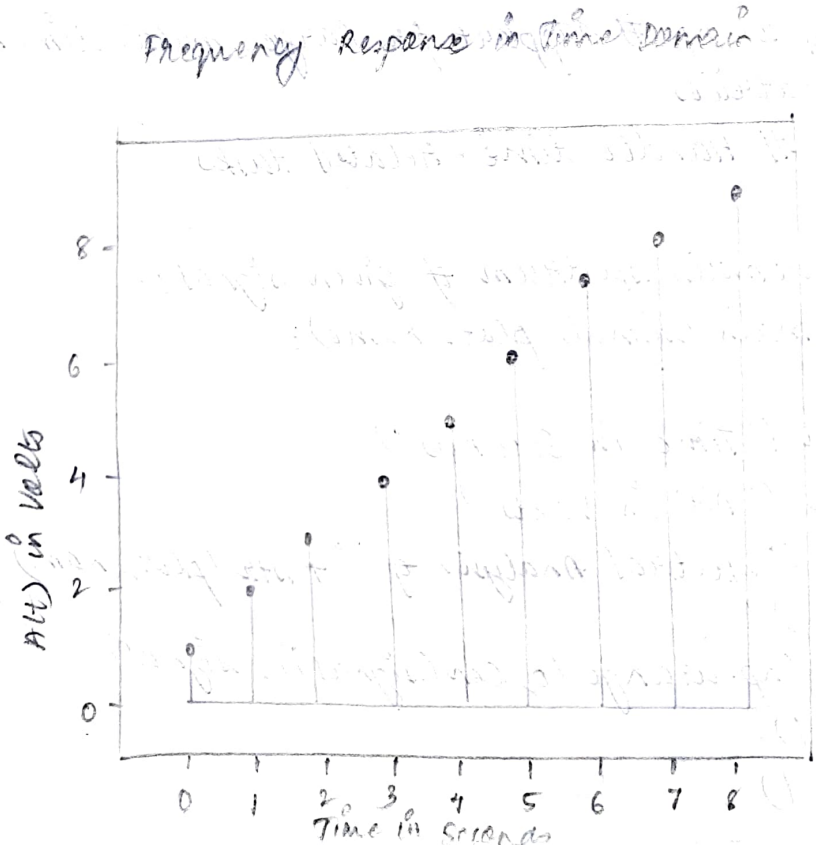
```
# Down-sample the given signal by a factor of "m":-
def down_sample(signal, m):
    down_sampler = []
    # Copy the element value from original signal to down
sampler array with the increment value set to "m":
    for i in range(0, len(signal), m):
        down_sampler.append(signal[i])
    return down_sampler

# Up-sample the given signal by a factor of "l":-
def up_sample(signal, l):
    up_sampler = []
    for i in range(len(signal)):
        up_sampler.append(signal[i]) # Copy the element value
from original signal to up sampler array
        if i != len(signal)-1:
            for k in range(l-1):
                up_sampler.append(0) # Insert "l-1" zeros
between the elements of the array
        else:
            break

    return up_sampler

# Perform convolution, x[n] * h[n]:-
def convolution(x, a):
```

string padding refers to adding, usually, non-informative characters to a string to one or both ends of it. This is most often done for output formatting and alignment purposes, but it can have useful practical applications. numpy.pad() function is used to pad the numpy arrays.

* OUTPUT (CONSOLE & PLOTS) : (ORDER OF FILTER = 3)

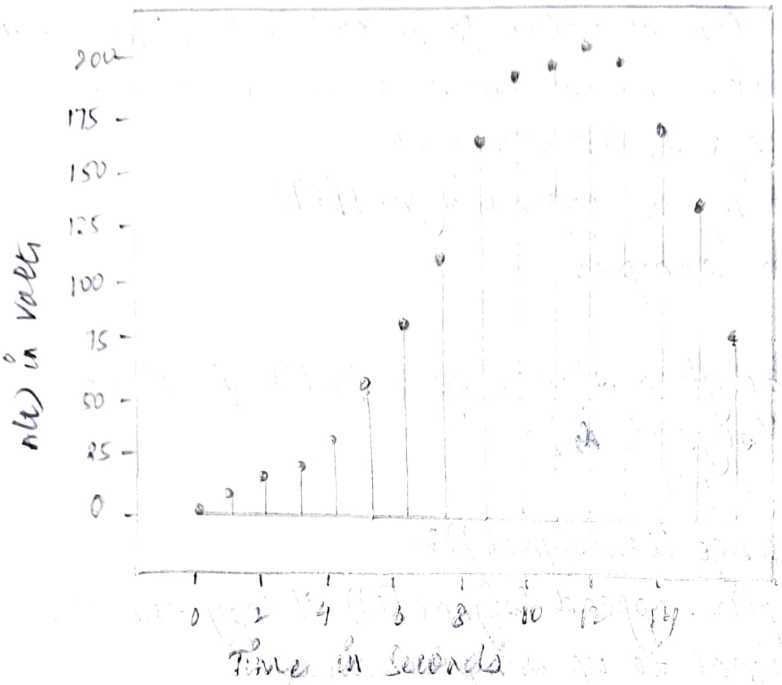
The original signal is : [1, 2, 3, 4, 5, 6, 7, 8]
The frequency response is : [1, 2, 3, 4, 5, 6, 7, 8, 9]
The order of filter is 3.





```
    size = (len(x) + len(h)) -1 # Compute the size of system response
    x = np.pad(x,(0, size - len(x)), 'constant')
    h = np.pad(h,(0, size - len(h)), 'constant')
    y = np.zeros(size, dtype = int) # Returns a new array of given shape and type, with zeros.

    for i in range(size):
        for j in range(size):
            if i >= j:
                y[i] = int(y[i] + (int(x[i-j]) * int(h[j])))
    return y # system Response y[n]

# Driver Code: main(); Execution starts here.

# Generate the original signal:-
title = "Original signal"
original_signal = [1, 2, 3, 4, 5, 6, 7, 8]
print("The original signal is: ", original_signal)
time_domain(original_signal, title)

# Generate the frequency response:-
title = "Frequency Response"
frequency_response = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print("The frequency response is: ", frequency_response)
time_domain(frequency_response, title)

m = int(input("enter the order of filter: "))

# Case - I:-
start = time.time()
title_I = "Traditional Filter - Direct Representation"
```

Traditional filter Direct Representation in Time Domain



The system response after convolution for 'Traditional Filter - Direct Representation' is: [1 4 10 20 35 56 84 120 156 182 197 200 190 166 127 72]

The time of execution of 'Traditional Filter - Direct Representation' is (in seconds): 0.0791355705261230 47

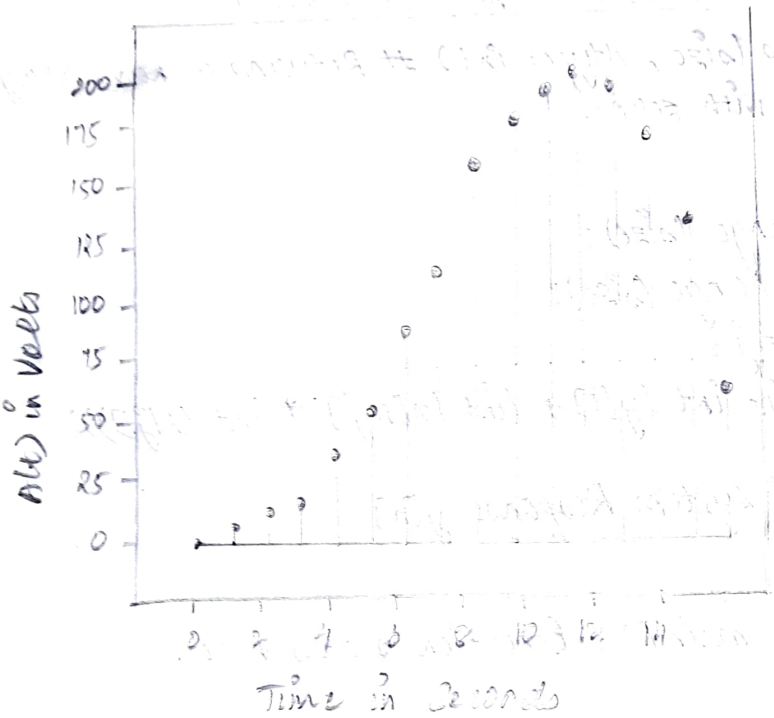
E0[n] : [1, 0, 0, 4, 0, 0, 7, 0, 0]
E1[n] : [0, 2, 0, 0, 5, 0, 0, 8, 0]
E2[n] : [0, 0, 3, 0, 0, 6, 0, 0, 9]

The system response after summing up the individual convolutions for 'Polyphase Filter Representation' is: [1 4 10 20 35 56 84 120 156 182 197 200 190 166 127 72]

The time of execution of 'Polyphase Filter Representation' is (in seconds): 0.16875016239345312

```
output_I = convolution (original_signal, frequency_response)
print ("\n The system response after convolution for '" +
str(title_I) + "' is : ", output_I)
time_domain (output_I, title_I)
end = time.time()
print ("The time of execution of '" + str(title_I) + "' is (in
seconds) : ", end-start)

print ("\n")

# Case- II :-
start = time.time()

title_II = "Polyphase Filter Representation"
down_sampler_0 = down_sample (original_signal, m)
up_sampler_0 = up_sample (down_sampler_0, m)
for i in range (m-1):
    up_sampler_0.append(0)
print ("E0[n] : ", up_sampler_0)

pad_frequency_response = np.pad (frequency_response, (m-1, 0),
'constant', constant_values = 0)
down_sampler_1 = down_sample (pad_frequency_response, m)
up_sampler_1 = up_sample (down_sampler_1, m)
for i in range (m-1):
    up_sampler_1.remove (up_sampler_1[i])
size_up_sampler_1 = len (up_sampler_0) - len (up_sampler_1)
for i in range (size_up_sampler_1):
    up_sampler_1.append(0)
print ("E1[n] : ", up_sampler_1)
```

Polyphase Filter Representation in Time Domain



```
pad_frequency_response = np.pad(frequency_response, (m-2, 0),
'constant', constant_values = 0)
down_sampler_2 = down_sample(pad_frequency_response, m)
up_sampler_2 = up_sample(down_sampler_2, m)
for i in range(m-2):
    up_sampler_2.remove(up_sampler_1[i])
print("H2[n]: ", up_sampler_2)

convolve_I = convolution(original_signal, up_sampler_0)
convolve_II = convolution(original_signal, up_sampler_1)
convolve_III = convolution(original_signal, up_sampler_2)
output_II = convolve_I + convolve_II + convolve_III
print("\n The system response after summing up the individual
convolutions for "" + str(title_II) + "" is: ", output_II)
time_domain(output_II, title_II)
end = time.time()
print("The time of execution of "" + str(title_II) + "" is
(in seconds): ", end-start)
```

RESULT:

For the filters designed to be used in interpolators and decimators implemented the simplified polyphase architecture. Also, analyzed the complexity and latency of the implemented filters. All the simulation results were verified successfully.