```
% Index number = 180631J
% Therefore the required parameters
A = 6;
B = 3;
C = 1;
```

### Prescribed Filter specifications

```
tilde_A_p = 0.03+0.01*A;    % Maximum passband ripple
tilde_A_a = 45 + B;         % Minimum stopband attenuation
omega_p1 = C*100 + 300;     % Lower passband edge
omega_p2 = C*100 + 700;     % Upper passband edge
omega_a1 = C*100 + 150;     % Lower stopband edge
omega_a2 = C*100 + 800;     % Upper stopband edge
omega_s  = 2*(C*100 +1200); % Sampling frequency
```

### Derivation of filter Parameters

```
B_t1 = omega_p1 - omega_a1; % Lower transition width
B_t2 = omega_a2 - omega_p2; % Upper transition width
B_t  = min(B_t1,B_t2);      % Critical transition width
omega_c1 = omega_p1-B_t/2;  % Lower cutoff frequency
omega_c2 = omega_p2+B_t/2;  % Upper cutoff frequency
T = 2*pi /omega_s;          % Sampling period
```

### Derivation of the Kaiser Window Parameters

```
tilde_delta_p  =  (10^(0.05*tilde_A_p) -1)/(10^(0.05*tilde_A_p)  +1);
tilde_delta_a  = 10^(-0.05*tilde_A_a);
delta = min(tilde_delta_p, tilde_delta_a);

A_a = -20*log10(delta);% Actual stopband attenuation

% Choose parameter alpha as,
if A_a <=21
    alpha = 0;
elseif 21 < A_a && A_a <=50
    alpha = 0.5842*(A_a - 21)^0.4 + 0.07886*(A_a - 21);
else
    alpha = 0.1102*(A_a - 8.7);
end

% Choose parameter D as,
if A_a <= 21
    D = 0.9222;
else
    D = (A_a - 7.95)/14.36;
```

```
end

% Select the lowest odd value of N that satisfies the inequality
N = ceil(omega_s*D/B_t + 1);
if mod(N,2) ==0
    N = N+1; % If calculated N is evn, make it odd by adding 1
end
```

### Creating the Kaise Window

```
n = -(N-1)/2:1:(N-1)/2;                              % Range of the Kaizer window
beta = alpha*sqrt(1-(2*n/(N-1)).^2);                 % betas for I(beta)
terms = 100;                                         % Number of terms to be considered for bessel
w_k_nT = ZerothOrderModifiedBessel(beta,terms)...     % Custom function is defined at the end.
            /ZerothOrderModifiedBessel(alpha,terms); % Calculating window coefficients
stem(n,w_k_nT,'filled');
title('Kaiser Window - Time Domain');
xlabel('Samples(n)');
ylabel('Amplitude');
grid on
```

### Derivation of The Ideal Impulse Response

```
h_d_nT = (sin(omega_c2*n*T) - sin(omega_c1*n*T))./(pi*n); % For each n != 0
h_d_nT((N+1)/2) = (omega_c2 - omega_c1)*(2/omega_s);      % For n = 0
stem(n,h_d_nT,'filled');
title('The Ideal Impulse Response');
xlabel('Samples(n)');
ylabel('Amplitude');
grid on
```

### Truncating the Ideal Impulse Response to obtain Finite Impulse Response(Anti-Causal)

```
h_nT = h_d_nT.*w_k_nT; % Windowing using Kaiser window
stem(n,h_nT,'filled');
title('Finite Impulse Response- Anti-Causal')
xlabel('Samples(n)');
ylabel('Amplitude');
grid on
```

### Finite Impulse Response(Causal)

```
n_causal = 0:1:N-1; % Making the range of n positive
stem(n_causal,h_nT,'filled');
xlim([0 N-1]);
title('Finite Impulse Response-Causal')
xlabel('Samples(n)');
```

```
ylabel('Amplitude');
grid on;
```

### Plotting the magnitude response of filter in the range (0,omega_s/2)

```
%fvtool(h_nT,'magnitude')
[H_ejomegaT, omega] = freqz(h_nT); %compute the frequency, magnitude, and phase response
omega = (omega/pi)*(omega_s/2);          % rad/s = (normalized freq)*(sampling freq/2)
magnitude = 20*log10(abs(H_ejomegaT));  % converting the magnitude into dB
plot(omega, magnitude);
xlim([0 omega_s/2]);
title('Magnitude Response of Filter');
xlabel('Angular Frequency (rad/s)');
ylabel('Magnitude (dB)');
grid on;
```

### Magnitude response of the digital filter for the frequencies in the passband

```
plot(omega, magnitude);
xlim([omega_p1 omega_p2]);  % Passband = [Lower passband edge, Upper passband edge]
title('Magnitude Response of Filter  in the passband');
xlabel('Angular Frequency (rad/s)');
ylabel('Magnitude (dB)');
grid on;
```

### Input Signal Generation

```
omega_1 =  (omega_a1 + 0)/2;          % Middle frequency of the lower stopband
omega_2 =  (omega_p1 + omega_p2)/2;  % Middle frequency of the passband
omega_3 =  (omega_a2 + omega_s/2)/2; % Middle frequency of the upper stopband

samples = 400;      % To achieve a steady-state response we need much samples
n1 = 0:1:samples;   % Discrete Values for sampling
n2 = 0:0.01:samples;% Near Continuous values for envelope drawing

x_nT = sin(omega_1*n1*T) +  sin(omega_2*n1*T) +sin(omega_3*n1*T); % Sampled I/P signal
x_t  = sin(omega_1*n2*T) +  sin(omega_2*n2*T) +sin(omega_3*n2*T); % I/P signal envelope

%================ Visulaization of Input Signal in time domain ================
subplot(2,1,1)
stem(n1, x_nT,'filled', 'b');
title('Input Signal in time domain');
xlabel('Samples(n)');
ylabel('Amplitude');
hold on;
plot(n2, x_t,'--','Color','r');xlim([100 150]);

%============= Visulaization of Input Signal in frequency domain ==============
X_f = fft(x_nT);                       % Discrete Fourier Transform of the input signal
```

```
X_ff = ((n1/length(n1))*omega_s)-omega_s/2; % map the range into (-omega_s to omega_s)

subplot(2,1,2)
plot(X_ff, abs(X_f));
title('Input Signal in frequency domain');
xlabel('Angular Frequency (rad/s)');
ylabel('Magnitude');

subplot(2,1,1)
grid on
subplot(2,1,2)
grid on
```

### Filtering through frequency domain multiplication

```
% Convolution in time domain is computationally expensive.
% Therefore frequency domain multiplication is used in this implementation.

% x_nT and h_nT have different lenghts,
% Therefore we need to get their fft outputs to a common lenghts.
% In order to multiply them element wise matrix dimensions must agree.
n_points = length(x_nT) + length(h_nT) -1;

X_f = fft(x_nT, n_points); % Discrete Fourier Transform of the input signal
H_f = fft(h_nT, n_points); % Discrete Fourier Transform of the FIR bandpass filter
Y_f = X_f.*H_f;            % Filtering through freq domain multiplication
y_nT = ifft(Y_f,n_points); % Inverse Fourier Transform to get the output signal.

% Truncating to the required length to account added time shift
y_nT = y_nT(floor(N/2)+1:length(y_nT)-floor(N/2));
% Output of an Ideal Filter
y_t = sin(omega_2*n2*T);
```

### Analyzing Output Signal

```
% ============Visulaization of Output Signal in time domain===============
figure;
subplot(2,1,1)
stem(n1, y_nT,'filled', 'b');
title('Output Signal in time domain');
xlabel('Samples(n)');
ylabel('Amplitude');
hold on;
plot(n2, y_t,'--','Color','r');xlim([100 150]);

% ===========Visulaization of Output Signal in frequency domain===========
Y_f = fft(y_nT);                          % Discrete Fourier Transform of the output signal
Y_ff = ((n1/length(n1))*omega_s)-omega_s/2; % map the range into (-omega_s to omega_s)

subplot(2,1,2)
```

```
plot(Y_ff, abs(Y_f));
title('Otput Signal in frequency domain');
xlabel('Angular Frequency (rad/s)');
ylabel('Magnitude');

subplot(2,1,1)
grid on
subplot(2,1,2)
grid on
```

### Input Signal and Desired Output Signal

```
figure;
plot(n2, x_t,'Color','r')
hold on
plot(n2, y_t,'Color','b')
legend('Input Signal', 'Desired Output Signal')
xlim([100 150]);
title('Input Signal and Desired Output Signal')
xlabel('Samples(n)')
ylabel('Amplitude')
grid on
```

### Local Function definitions

```
% function to calculate zeroth-order modified Bessel function of the first kind
% Upto a given terms.
function value = ZerothOrderModifiedBessel(x,terms)
value = 1;
for k = 1:terms
    value = value + ((1/factorial(k))*(x/2).^k).^2;
end
end
```