Write a simple function in MATLAB, with the name *cflip_bee()* that implements circular flipping of an input array ***x***.

Check your function for different lengths of input and values of *N*.

SOLUTION TASK 1:

Required Code:

```
function y = cflip_bee(x,N)
x=[1,2,3,4];
N=6;
%where x = input array

% N = the number of points for circular flipping (DFT points)

%y = output that should be Modulo N circularly flipped version of x
if(N>length(x))

    x1 = [x, zeros(1, N-length(x))];

end



y = [x1(1), x1(length(x1):-1:2)];

return y;
```

REQUIRED OUTPUTS:

Example 1

```
>> z=cflip_bee([1,2,3,4,6,7,4],2)

z =

     1     0     0     4     3     2
```

Example 2

```
>> z=cflip_bee([1,2,3,4],6)

z =

     1     0     0     4     3     2
```

Implementation of Circular Shifting

Write a function in MATLAB, with the name *cshift_bee()* that implements circular shifting of an input array *x*. The format of the function should be

REQUIRED CODE:

```
function y = cshift_bee(x,r,N)
x1=x
if(N>length(x))

    x1 = [x, zeros(1, N-length(x))];

end

if (r<=0)

 y = [x1((length(x1)+r+1):length(x1)), x1(1:(length(x1)+r))];

else

 y = [x1(r+1:length(x1)), x1(1:r)];



end
```

Verify your result for different values of *x*, *r*, and *N*.

REQUIRED OUTPUTS

```
>> z=cshift_bee([1,2,3,4],-2,4)

z =

     3     4     1     2
```

```
>> z=cshift_bee([1,2,3],-2,6)

z =

     0     0     1     2     3     0
```

Implementation of Circular Convolution

Write a function in MATLAB, with the name *cconv_bee()* that implements circular convolution of an input array *x* with an array *h*.
The format of the function should be

Required Code:

```
function y = cconv_bee(x,h,N)

%where x, h = input arrays

% N = the number of DFT points

%y = N point output of circular convolution

if(N>length(x))

 x = [x, zeros(1, N-length(x))]

end

if(N>length(h))

 h = [h, zeros(1, N-length(h))]
 %h will remain intact and make matrix of impulse h
end
flippled_h=cflip_bee(h,N)
flipped_matrix=zeros(N,N)
 for i=0:N-1

     temp_vector=cshift_bee(flippled_h,-i,N)

     flipped_matrix(i+1,:)=temp_vector
 end

 %now time for matrix multiplication

 my_result=transpose(flipped_matrix*transpose(x))


 % cconv_bee([1,1,1,1,1,1],[1,1,1,1,1,1],11]
end
```

OUTPUTS:

Test your result on some values of *x*, *h* and *N* and compare your results with the MATLAB built-in function cconv().

```
>> cconv_bee([1,1,1],[1,1,1],5)

my_result =

     1     2     3     2     1

>> conv([1,1,1],[1,1,1])

ans =

     1     2     3     2     1

>>
```

2nd Example

```
>> cconv_bee([1,2,1],[1,0,1],5)

my_result =

     1     2     2     2     1

>> conv([1,2,1],[1,0,1])

ans =

     1     2     2     2     1

>>
```

a. Write a code for implementing Overlap and Add Method.

REQUIRED CODE

```
function y = add_method_conv(x,h,L)

%x input signal
%h filter signal (impulse response)
%L chunks size of x divisions into smaller inpu signals


%step one --> finding individual outputs of chunks of x[n]

remainder=mod(x,L)
if remainder==0
    total_chunks=length(x)/L
    last_fraction=0
else
    total_chunks=floor(length(x)/L)+1
    last_fraction=1
end
total_chunks
my_chunks_array=zeros(total_chunks,L)
for  i=0:total_chunks-1

    if(i==total_chunks-1 && last_fraction==1)
        %means last chunk needs padding
        disp('camehere')
        temp_single_chunk=[x(i*L+1:length(x)),zeros(1,L-length(x(i*L+1:length(x))))];
```

```
        my_chunks_array(i+1,:)=temp_single_chunk;
    else

        temp_single_chunk=x(i*L+1:i*L+L);
        my_chunks_array(i+1,:)=temp_single_chunk;
         disp('camehere')
    end

end

%now next step , we will convolving each chunk with the filter's response
chunks_convolved_matrix=zeros(total_chunks,L+length(h)-1)

for i=1:total_chunks

    single_chunk_convolved=cconv_bee(my_chunks_array(i,:),h,L+length(h)-1);

    chunks_convolved_matrix(i,:)=single_chunk_convolved;

end

chunks_convolved_matrix
%now last step, adding all the outputs together
final_ans=[]
for i=1:total_chunks
    if i==1
        final_ans=chunks_convolved_matrix(i,:);

    else

        temp=chunks_convolved_matrix(i,:)

        temp_common_points_addition=final_ans(length(final_ans)-
length(h)+2:length(final_ans))+temp(1:length(h)-1)

        final_ans=[final_ans(1:length(final_ans)-
length(h)+1),temp_common_points_addition,temp(length(h):length(temp))]

    end

    final_ans
end
```

REQUIRED OUTPUTS:

```
>>  add_method_conv([1,2,3,4,5,6,2,3],[5,4,2,1],2)

final_ans =

     5    14    25    37    49    61    48    40    22     8     3     0     0

>> conv([1,2,3,4,5,6,2,3],[5,4,2,1])

ans =

     5    14    25    37    49    61    48    40    22     8     3

>>
```

b. Write a code for implementing Overlap Save Method.

REQURIED CODE

```
function y = add_method_overlap_convolve(x,h,L)
%remember this method works when L>length(h)

%step one , getting required chunks , but this time we will use while loop
%and break by force

breaking_loop=1;
my_chunks_array=[];
tracking_index=1;

%padd zeros to input signal of size p-1

x=[zeros(1,length(h)-1),x];
while(breaking_loop==1)

    if(length(x(tracking_index:length(x)))<L)
        disp('came in if ')

        x=[x,zeros(1,L-length(x(tracking_index:length(x))))];
        my_chunks_array=[my_chunks_array;x(tracking_index:tracking_index+L-
1)];
        tracking_index= (tracking_index+L-1)  - length(h)+1;

        breaking_loop=0
        disp('came in if ')

    else
        disp('came in else')

        my_chunks_array=[my_chunks_array;x(tracking_index:tracking_index+L-
1)];
        tracking_index=tracking_index+1;
        tracking_index= (tracking_index+L-1)  - length(h)+1;
        tracking_index;

    end
    if(tracking_index+L-1==length(x))

my_chunks_array=[my_chunks_array;x(tracking_index:tracking_index+L-1)];
            breaking_loop=0;

    end

end
%now to find convolution for each chunk of input signal but L point
z=size(my_chunks_array);
z=z(1);

convolved_chunk_matrix=[];
for i=1:z

    single_chunk_convolved=cconv_bee(my_chunks_array(i,:),h,L);
```

```
    convolved_chunk_matrix=[convolved_chunk_matrix;single_chunk_convolved];

end

%now discarding the first p-1 points of each convolved_input_chunk vector

discarded_p_minus_1_matrix=[];
for i=1:z

discarded_p_minus_1_matrix=[discarded_p_minus_1_matrix;convolved_chunk_matri
x(i,length(h):L)];
end
%now concatinating into single vector
final_ans=[];
for i=1:z
   final_ans=[final_ans,discarded_p_minus_1_matrix(i,:)]

end
```

c. Compare the results of both against each other and against the direct convolution of whole length ***x***.

COMPARISION BETWEEN ALL THREE IMPLEMENTATIONS

DIRECT CONV BUILTIN FUNCTION :

```
>> conv([1,1,1,1,1,1,1,1],[1,1])

ans =

     1     2     2     2     2     2     2     2     1
```

NOW USING OVERLAP ADD METHOD:

```
>> add_method_conv([1,1,1,1,1,1,1,1],[1,1],4)

final_ans =

     1     2     2     2     2     2     2     2     1     0     0     0     0
```

USING OVERLAP SAVE METHOD

```
>> add_method_overlap_convolve([1,1,1,1,1,1,1,1],[1,1],4)

final_ans =

     1     2     2     2     2     2     2     2     1

fx >> 
```

Implement the two block convolution methods but this time using the MATLAB functions *fft()* and *ifft()*.

Now instead of using the previous built cconv_bee(x,h,N) function , we will get the convolution results using the new implemented function ,

linear_conv_using_fft(x,h,N) which uses the FFT to compute the convolution between two signals.
The code for this function is given as:

```
function y = linear_conv_using_fft(x,h,N)

if(N>length(x))

 x = [x, zeros(1, N-length(x))];

end

if(N>length(h))

 h = [h, zeros(1, N-length(h))];

end

x_FFT=fft(x);
h_FFT=fft(h);


dot_product=x_FFT.*h_FFT;

y=ifft(dot_product);

end
```

We have now, just replaced the function of cconv_bee(x,h,N) with this above function in both overlap add , add and overlap save method.
And we have verified the results, and they are exactly same as answers obtained from previous implementations.

Verification of results:

(Each convolution block function uses the FFT implemented function for calculating the convolutions)

Overlap save method

```
>> add_method_overlap_convolve([1,1,1,1,1,1,1,1],[1,1],4)

final_ans =

     1     2     2     2     2     2     2     2     1

>>
>> |
```

Overlap add method

```
>
> add_method_conv([1,1,1,1,1,1,1,1],[1,1],4)

inal_ans =

     1     2     2     2     2     2     2     2     1     0     0     0     0

> |
```

Builtin Conv function

```
ans =

     1     2     2     2     2     2     2     2     1
```