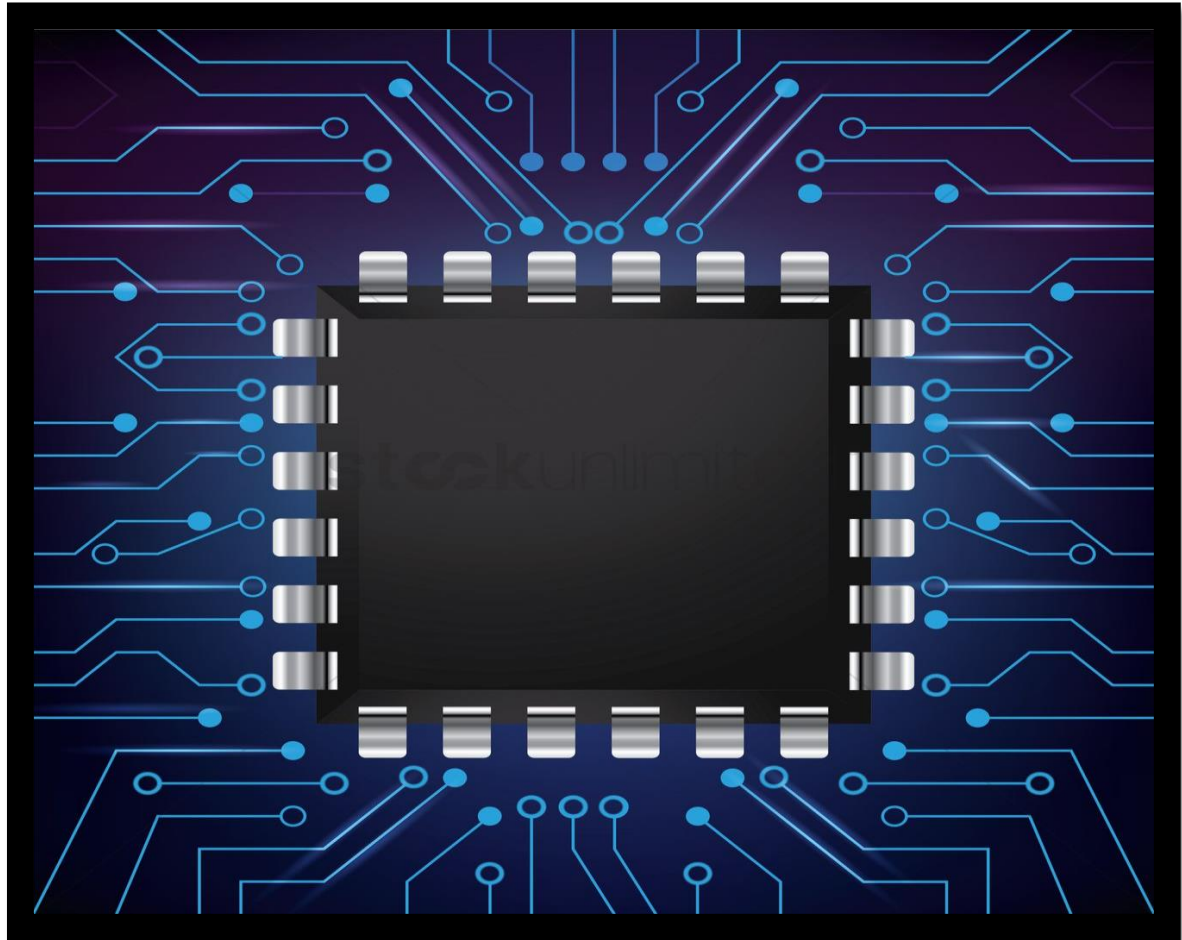


# VLSI | OEP

Submitted to: **Dr. Saad Qasim Khan**



**Name: Muhammad Abdullah**

**Roll no.: CS-080**

**Couse Code: CS-319**

**CIS DEPARTMENT**

# TABLE OF CONTENTS

<b>INTRODUCTION .....</b>	<b>3</b>
OPEN ENDED PROBLEM .....	1
DIGITAL FILTERS .....	2
BANDPASS FILTER.....	3
<b>ANALOG BAND PASS FILTER .....</b>	<b>4</b>
DIAGRAM .....	1
EXPLANATION .....	2
OUTPUT.....	3
<b>CALCULATIONS .....</b>	<b>5</b>
<b>FILTER DESIGN .....</b>	<b>7</b>
<b>RTL SCHEMATIC .....</b>	<b>11</b>
<b>SIMULATION .....</b>	<b>12</b>
<b>VERILOG CODE.....</b>	<b>13</b>

# INTRODUCTION

## OPEN ENDED PROBLEM

Explore the design of digital filters on Xilinx software. Design a band pass filter of center frequency 300 KHz and a pass bandwidth of 1 KHz.

## DIGITAL FILTERS

Digital filters are commonly used in discrete signal processing to eliminate or preserve certain portions of the signal. Digital filters are divided into two categories which are Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). The most common types of digital filters are following

- Low pass filter
- High pass filter
- Band pass filter

## BAND PASS FILTER

Band pass filter is a device that allows frequencies within a specified range and ignores frequencies beyond that range.

### CENTER FREQUENCY

The frequency at the center of the high cut off frequency and low cut off frequency is called center frequency.

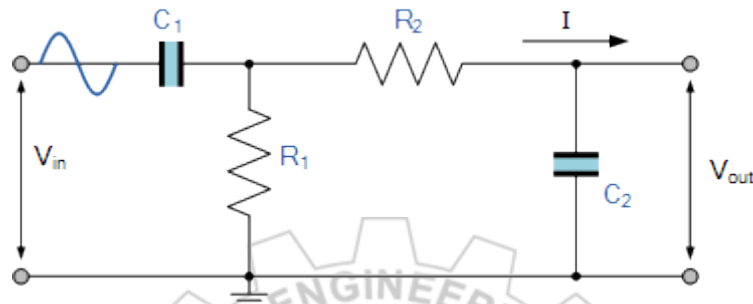
### PASS BANDWIDTH

The difference between the high cut off frequency and low cut off frequency is called pass bandwidth.

# ANALOG BAND PASS FILTER

Before we jump in to design a digital FIR band pass filter we should understand how an analog band pass filter works.

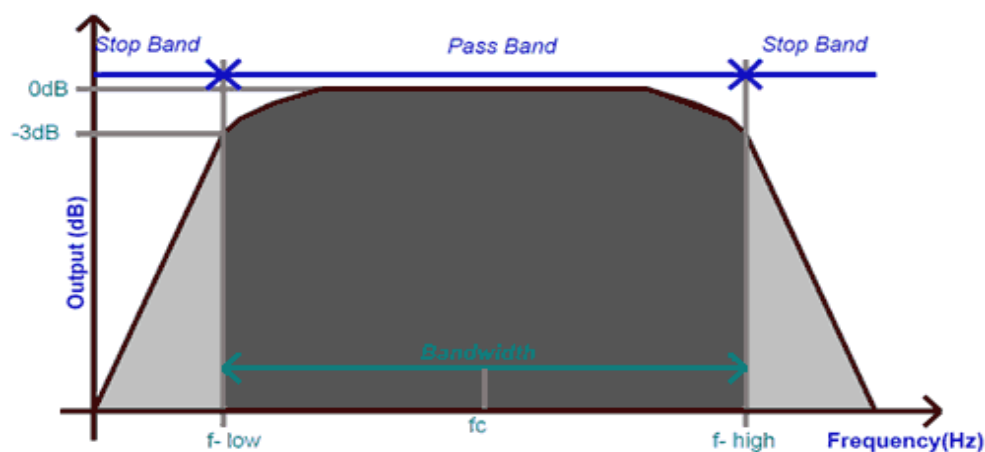
## DIAGRAM



## EXPLANATION

The analog band pass filter circuit is created by cascading a high pass filter and a low pass filter by choosing appropriate values of resistors and capacitors shown in diagram. The high cut-off frequency is calculated by the formula  $f_{c1} = 1 / 2\pi \cdot R_1 \cdot C_1$ . While the low cut-off frequency is calculated by the formula  $f_{c2} = 1 / 2\pi \cdot R_2 \cdot C_2$ . By using the values of  $f_{c1}$  and  $f_{c2}$  we can calculate pass bandwidth and center frequency of an analog band pass filter.

## OUTPUT



# CALCULATIONS

Here we will calculate fpass1, fpass2 and sampling frequency according to the given OBE requirements.

## Center Frequency Formulas

For  $f_2/f_1 \geq 1.1$

$$\text{Center Frequency} = \sqrt{f_1 f_2}$$

For  $f_2/f_1 < 1.1$

$$\text{Center Frequency} = \frac{f_1 + f_2}{2}$$

Enter the Lower Cutoff Frequency

Enter the Upper Cutoff Frequency

Center Frequency: 30000 Hz

Here we found out that Fpass1 = 29500 Hz and Fpass2 = 30500 KHz, and if we use these values we gets a pass bandwidth of 1 KHz.

- ⇒ Fpass2 – Fpass1 = Bandwidth
- ⇒ 30500 – 29500 = Bandwidth
- ⇒ 1000 Hz = Bandwidth

To calculate sampling frequency we will use Nyquist formula:

**"it is necessary to use a sampling rate "fs" at least twice the highest waveform frequency."**

$$\Rightarrow f_{\text{pass}} = \frac{1}{2}(f_s)$$

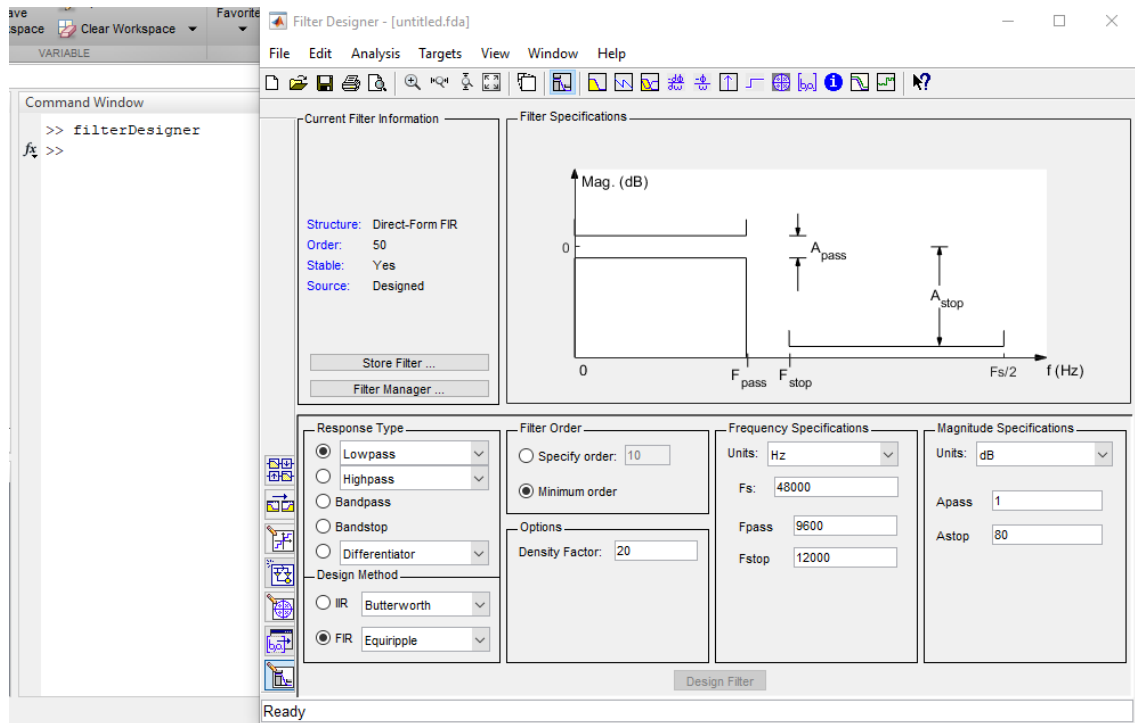
$$\Rightarrow f_s = 62000 \text{ Hz}$$

We can apply sampling frequency greater than or equals to 62000 Hz in our band pass filter.

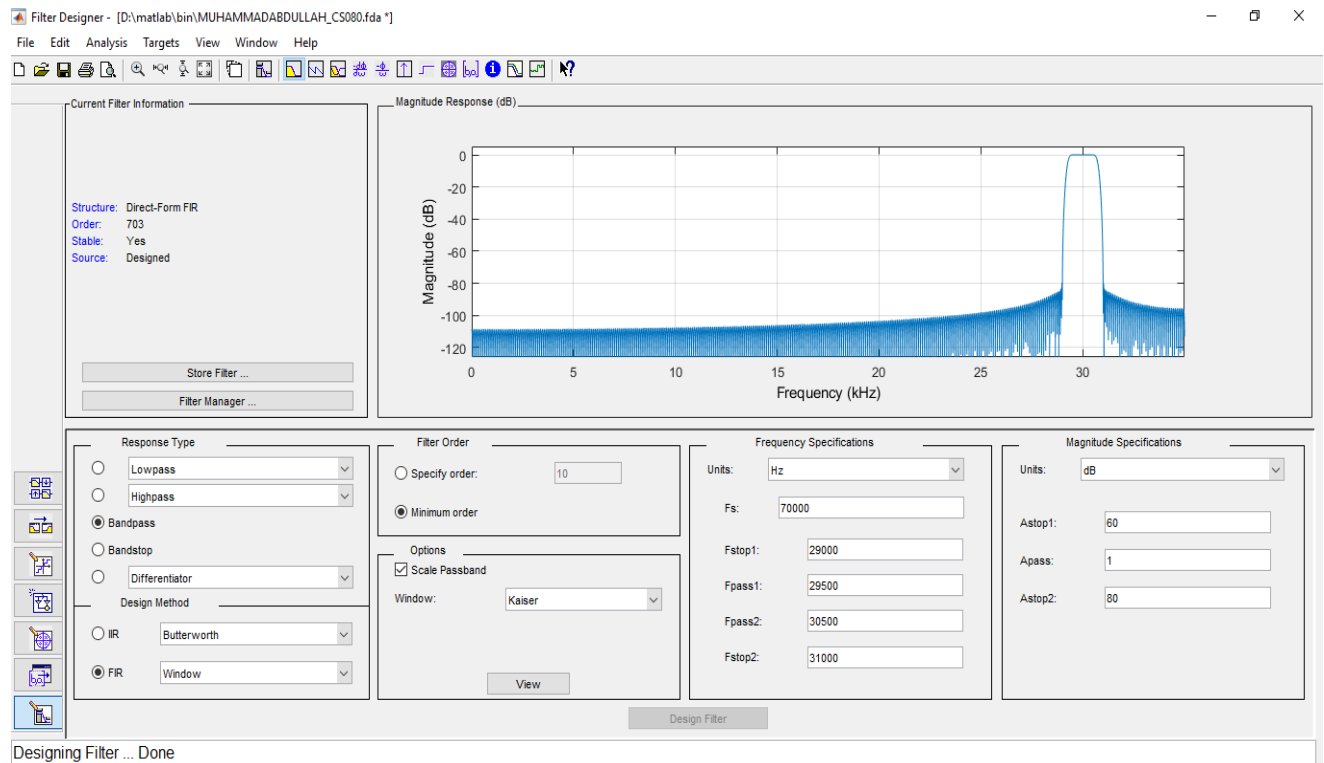


# FILTER DESIGN

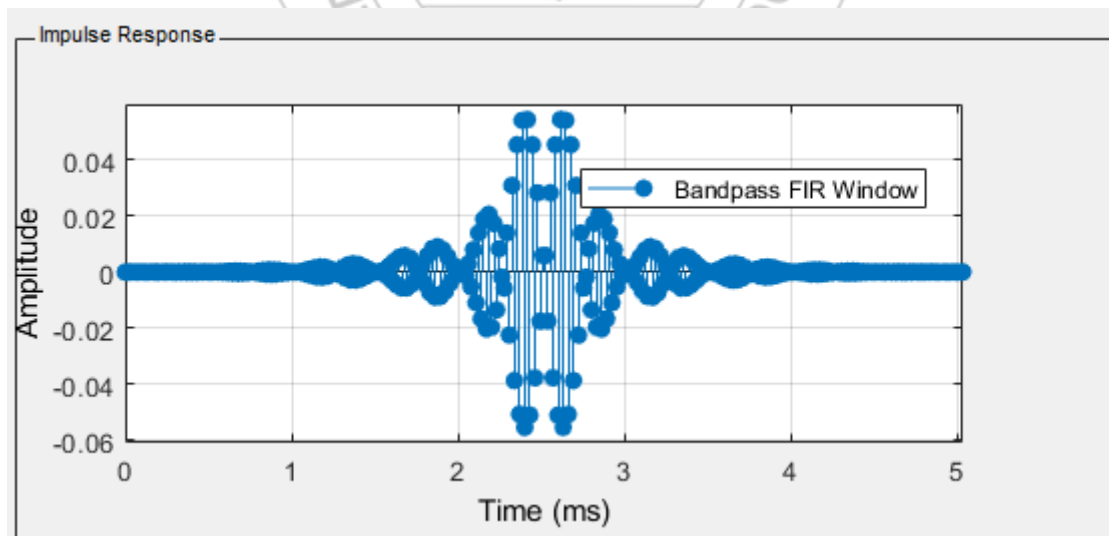
First we will open Matlab and type command “filterDesigner” in command window. Filter Designer window appears as shown below.



Now we will select options i.e. Response Type, Design Method, Frequency Specifications according to the calculations we made earlier. By putting all the values we get a band pass filter of center frequency 30000 Hz as shown below.

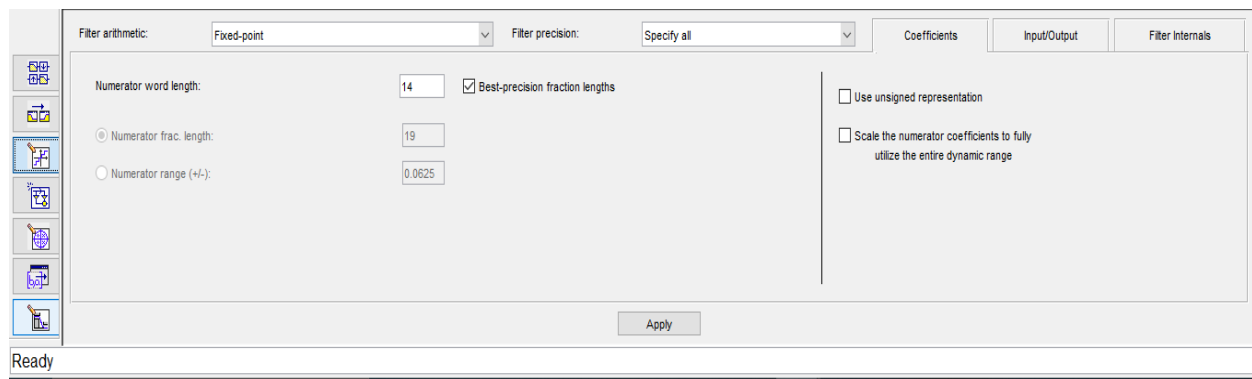


The impulse response of band pass filter

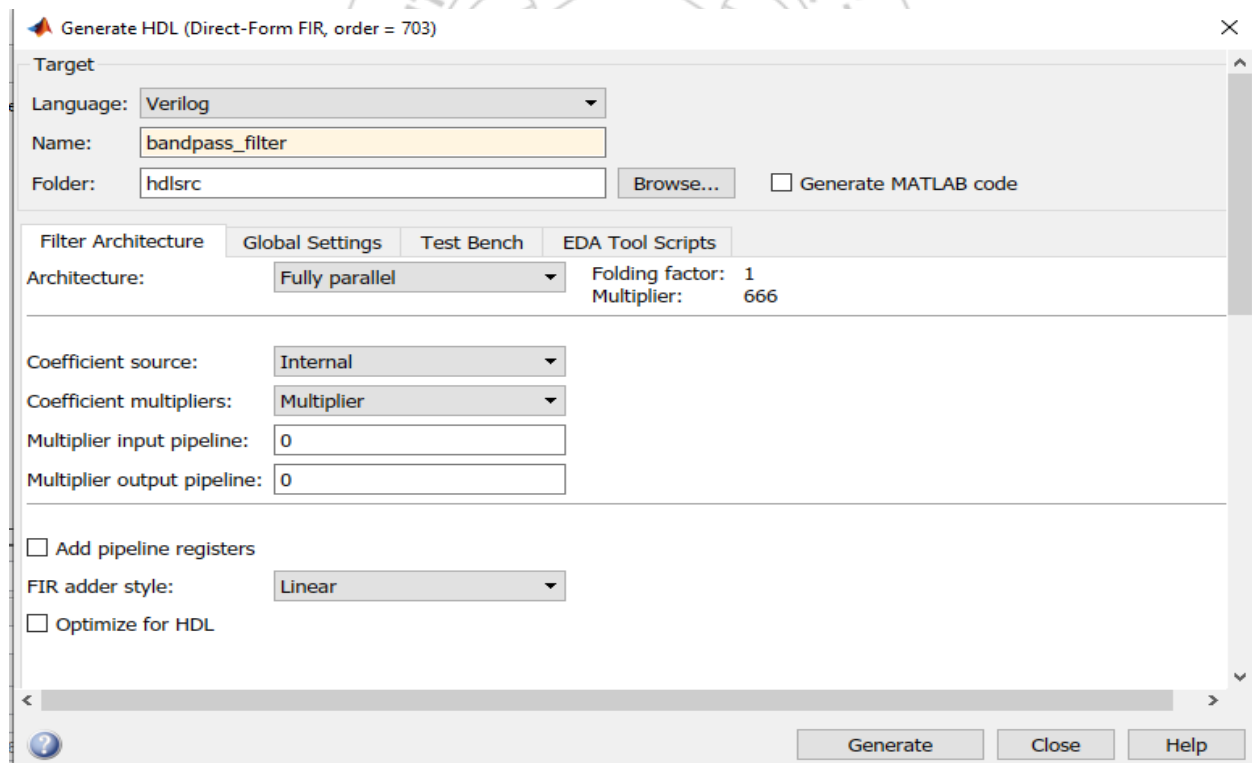




Now we will quantize the filter before generating its Verilog code.



Now select Target => Generate HDL following dialog box appears we will select appropriate options and then click "Generate" button.








Verilog code generation of our band pass filter starts as shown in command window below

```
### Starting Verilog code generation process for filter: bandpass_filter
### Generating: D:\matlab\bin\hdlsrc\bandpass_filter.v
### Starting generation of bandpass_filter Verilog module
### Starting generation of bandpass_filter Verilog module body
### Successful completion of Verilog code generation process for filter: bandpass_filter
### HDL latency is 1 samples
### Starting generation of VERILOG Test Bench.
Warning: Structure fir has symmetric coefficients, consider converting to structure symmetricfir for reduced area.
> In hdlfilter.abstracdfir/setimplementation
   In hdlfilter.AbstractHDLFilter/generatetbcode>localgentb
   In hdlfilter.AbstractHDLFilter/generatetbcode
   In hdlfilter.AbstractHDLFilter/generatehdlcode
   In fdhdlcodernui.fdhdltooldlg/dialogCallback (line 87)
### Generating input stimulus
### Done generating input stimulus; length 8000 samples.
### Generating Test bench: D:\matlab\bin\hdlsrc\bandpass_filter_tb.v
### Creating stimulus vectors ...
### Done generating VERILOG Test Bench.
fx >>
```

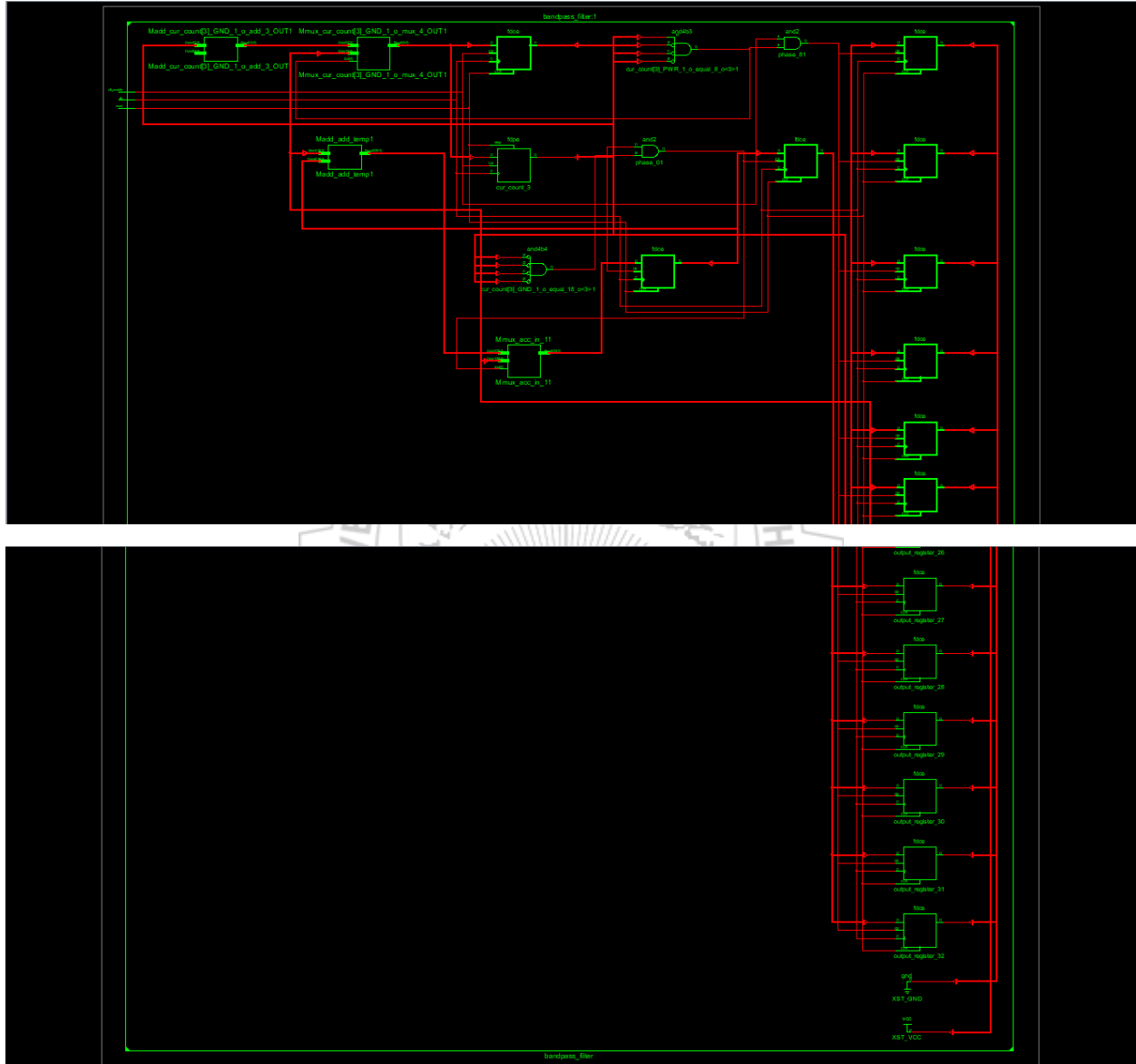
The verilog code after being generated gets stored as shown below.

This PC > New Volume (D:) > matlab > bin > hdlsrc

Name	Date modified	Type	Size
 bandpass_filter.v	9/6/2020 2:29 AM	V File	656 KB
 bandpass_filter_compile.do	9/6/2020 2:29 AM	DO File	1 KB
 bandpass_filter_tb.v	9/6/2020 2:29 AM	V File	643 KB
 bandpass_filter_tb_compile.do	9/6/2020 2:29 AM	DO File	1 KB
 bandpass_filter_tb_sim.do	9/6/2020 2:29 AM	DO File	1 KB

# RTL SCHEMATIC

The following RTL schematic is obtained after synthesizing the generated Verilog code.





# VERILOG CODE

```
// -----  
//  
// Module: bandpass_filter  
// Generated by MATLAB(R) 8.6 and the Filter Design HDL Coder 2.10.  
// Generated on: 2020-05-09 13:21:29  
// -----  
  
// -----  
// HDL Code Generation Options:  
//  
// TargetDirectory: D:\matlab\bin\hdlsrc\matlab_filter  
// Name: bandpass_filter  
// SerialPartition: 9  
// TargetLanguage: Verilog  
// TestBenchStimulus: impulse step ramp chirp noise  
  
// -----  
// HDL Implementation : Fully Serial  
// Multipliers : 1  
// Folding Factor : 9  
// -----  
  
// Filter Settings:  
//  
// Discrete-Time FIR Filter (real)  
// -----  
// Filter Structure : Direct-Form FIR  
// Filter Length : 11  
// Stable : Yes  
// Linear Phase : Yes (Type 1)  
// Arithmetic : fixed  
// Numerator : s6,5 -> [-1 1)
```

```

// Input      : s18,17 -> [-1 1)
// Filter Internals : Specify Precision
// Output     : s33,30 -> [-4 4)
// Product    : s26,24 -> [-2 2)
// Accumulator : s26,24 -> [-2 2)
// Round Mode  : convergent
// Overflow Mode : wrap
// -----

```

```

`timescale 1 ns / 1 ns

```

```

module bandpass_filter

```

```

(
    clk,
    clk_enable,
    reset,
    data_in,
    data_out
);

```

```

input  clk;
input  clk_enable;
input  reset;
input  signed [17:0] data_in; //sfix18_En17
output signed [32:0] data_out; //sfix33_En30

```

```

////////////////////////////////////////////////////////////////

```

```

//Module Architecture: bandpass_filter

```

```

////////////////////////////////////////////////////////////////

```

```

// Local Functions

```

```

// Type Definitions

```

```

// Constants

```

```

parameter signed [5:0] coeff1 = 6'b111101; //sfix6_En5

```

```

parameter signed [5:0] coeff2 = 6'b111000; //sfix6_En5

```

```

parameter signed [5:0] coeff3 = 6'b111000; //sfix6_En5

```



```

parameter signed [5:0] coeff4 = 6'b000000; //sfix6_En5
parameter signed [5:0] coeff5 = 6'b001011; //sfix6_En5
parameter signed [5:0] coeff6 = 6'b010000; //sfix6_En5
parameter signed [5:0] coeff7 = 6'b001011; //sfix6_En5
parameter signed [5:0] coeff8 = 6'b000000; //sfix6_En5
parameter signed [5:0] coeff9 = 6'b111000; //sfix6_En5
parameter signed [5:0] coeff10 = 6'b111000; //sfix6_En5
parameter signed [5:0] coeff11 = 6'b111101; //sfix6_En5

```

```
// Signals
```

```

reg [3:0] cur_count; // ufix4
wire phase_8; // boolean
wire phase_0; // boolean
reg signed [17:0] delay_pipeline [0:10] ; // sfix18_En17
wire signed [17:0] inputmux_1; // sfix18_En17
reg signed [25:0] acc_final; // sfix26_En24
reg signed [25:0] acc_out_1; // sfix26_En24
wire signed [25:0] product_1; // sfix26_En24
wire signed [5:0] product_1_mux; // sfix6_En5
wire signed [23:0] mul_temp; // sfix24_En22
wire signed [25:0] prod_typeconvert_1; // sfix26_En24
wire signed [25:0] acc_sum_1; // sfix26_En24
wire signed [25:0] acc_in_1; // sfix26_En24
wire signed [25:0] add_signext; // sfix26_En24
wire signed [25:0] add_signext_1; // sfix26_En24
wire signed [26:0] add_temp; // sfix27_En24
wire signed [32:0] output_typeconvert; // sfix33_En30
reg signed [32:0] output_register; // sfix33_En30

```

```
// Block Statements
```

```

always @ (posedge clk or posedge reset)
begin: Counter_process
    if (reset == 1'b1) begin
        cur_count <= 4'b1000;
    end
end

```

```

else begin
    if (clk_enable == 1'b1) begin
        if (cur_count == 4'b1000) begin
            cur_count <= 4'b0000;
        end
    end
    else begin
        cur_count <= cur_count + 1;
    end
end
end
end // Counter_process

assign phase_8 = (cur_count == 4'b1000 && clk_enable == 1'b1)? 1 : 0;

assign phase_0 = (cur_count == 4'b0000 && clk_enable == 1'b1)? 1 : 0;

always @(posedge clk or posedge reset)
begin: Delay_Pipeline_process
    if (reset == 1'b1) begin
        delay_pipeline[0] <= 0;
        delay_pipeline[1] <= 0;
        delay_pipeline[2] <= 0;
        delay_pipeline[3] <= 0;
        delay_pipeline[4] <= 0;
        delay_pipeline[5] <= 0;
        delay_pipeline[6] <= 0;
        delay_pipeline[7] <= 0;
        delay_pipeline[8] <= 0;
        delay_pipeline[9] <= 0;
        delay_pipeline[10] <= 0;
    end
    else begin
        if (phase_8 == 1'b1) begin
            delay_pipeline[0] <= data_in;
            delay_pipeline[1] <= delay_pipeline[0];
        end
    end
end

```



```

delay_pipeline[2] <= delay_pipeline[1];
delay_pipeline[3] <= delay_pipeline[2];
delay_pipeline[4] <= delay_pipeline[3];
delay_pipeline[5] <= delay_pipeline[4];
delay_pipeline[6] <= delay_pipeline[5];
delay_pipeline[7] <= delay_pipeline[6];
delay_pipeline[8] <= delay_pipeline[7];
delay_pipeline[9] <= delay_pipeline[8];
delay_pipeline[10] <= delay_pipeline[9];
end
end
end // Delay_Pipeline_process

```

```

assign inputmux_1 = (cur_count == 4'b0000) ? delay_pipeline[0] :
    (cur_count == 4'b0001) ? delay_pipeline[1] :
    (cur_count == 4'b0010) ? delay_pipeline[2] :
    (cur_count == 4'b0011) ? delay_pipeline[4] :
    (cur_count == 4'b0100) ? delay_pipeline[5] :
    (cur_count == 4'b0101) ? delay_pipeline[6] :
    (cur_count == 4'b0110) ? delay_pipeline[8] :
    (cur_count == 4'b0111) ? delay_pipeline[9] :
    delay_pipeline[10];

```

```

// ----- Serial partition # 1 -----

```

```

assign product_1_mux = (cur_count == 4'b0000) ? coeff1 :
    (cur_count == 4'b0001) ? coeff2 :
    (cur_count == 4'b0010) ? coeff3 :
    (cur_count == 4'b0011) ? coeff5 :
    (cur_count == 4'b0100) ? coeff6 :
    (cur_count == 4'b0101) ? coeff7 :
    (cur_count == 4'b0110) ? coeff9 :
    (cur_count == 4'b0111) ? coeff10 :
    coeff11;

```

```

assign mul_temp = inputmux_1 * product_1_mux;
assign product_1 = $signed({mul_temp[23:0], 2'b00});

assign prod_typeconvert_1 = product_1;

assign add_signext = prod_typeconvert_1;
assign add_signext_1 = acc_out_1;
assign add_temp = add_signext + add_signext_1;
assign acc_sum_1 = add_temp[25:0];

assign acc_in_1 = (phase_0 == 1'b1) ? prod_typeconvert_1 :
    acc_sum_1;

```

```

always @ (posedge clk or posedge reset)

```

```

begin: Acc_reg_1_process

```

```

    if (reset == 1'b1) begin

```

```

        acc_out_1 <= 0;

```

```

    end

```

```

    else begin

```

```

        if (clk_enable == 1'b1) begin

```

```

            acc_out_1 <= acc_in_1;

```

```

        end

```

```

    end

```

```

end // Acc_reg_1_process

```

```

always @ (posedge clk or posedge reset)

```

```

begin: Finalsum_reg_process

```

```

    if (reset == 1'b1) begin

```

```

        acc_final <= 0;

```

```

    end

```

```

    else begin

```

```

        if (phase_0 == 1'b1) begin

```

```

            acc_final <= acc_out_1;

```

```

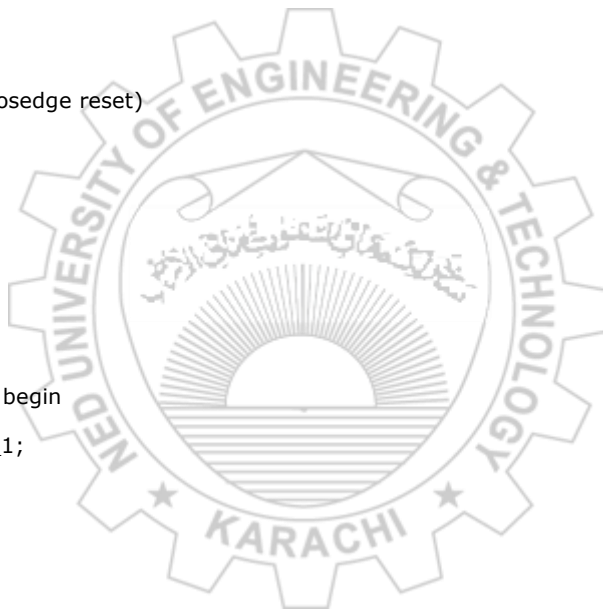
        end

```

```

    end

```



```

end // Finalsum_reg_process

assign output_typeconvert = $signed({acc_final[25:0], 6'b000000});

always @ (posedge clk or posedge reset)
begin: Output_Register_process
    if (reset == 1'b1) begin
        output_register <= 0;
    end
    else begin
        if (phase_8 == 1'b1) begin
            output_register <= output_typeconvert;
        end
    end
end // Output_Register_process

// Assignment Statements
assign data_out = output_register;
endmodule // bandpass_filter

```

