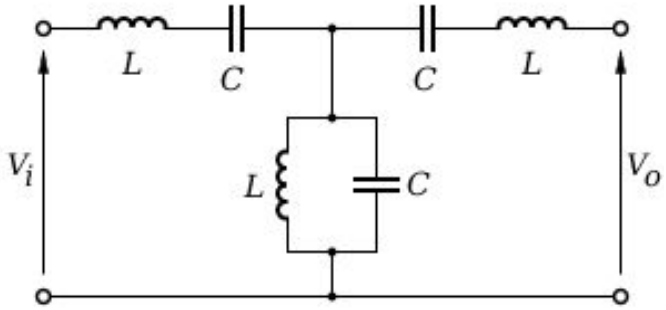
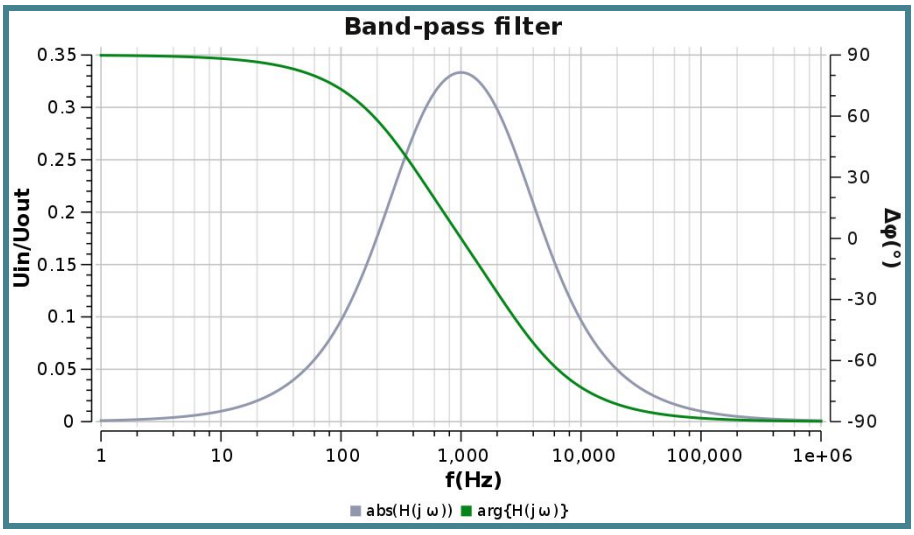
BANDPASS FILTER

Definition:



Bandpass filter allows only a certain point of frequencies and it rejects the frequencies which are outside of this band. There are many ways in which we can design a band pass filter and one of the ways is to design this band pass filter using the series RLC circuit.
The bandpass filter passes signals with frequencies which falls between the passband of both the highpass and lowpass filters. The frequency analyzer displays the response of the bandpass filter by adding the previous low pass and high pass frequencies. The low pass removes the higher frequency and high pass removes the lower frequency, only the mid range frequency is passed through it.
The frequency difference between the higher band cutoff and lower band cutoff frequency is known as the bandwidth and the geometric mean of these two cutoff frequencies is known as a centre frequency.



Bandpass filter applications:

1. Used extensively in wireless communication (transmitter and receiver):
2. To limit the bandwidth in accordance with the allotment.
3. In devices where signal to noise ratio is important and to increase or decrease the sensitivity of a receiver.
4. Used in Sonar, Seismology, LASER, LIDAR and in medical applications .
5. In audio signal processing where only a particular range of frequency is required.

Open Ended problem: Explore the design of digital filters on Xilinx software. Design a bandpass filter of centre frequency 300 KHz and a pass bandwidth of 1 KHz. Due date for this assignment is 1st September 2020. If you can simulate the proposed model then it will give additional weight.

Source code	***Test Bench***

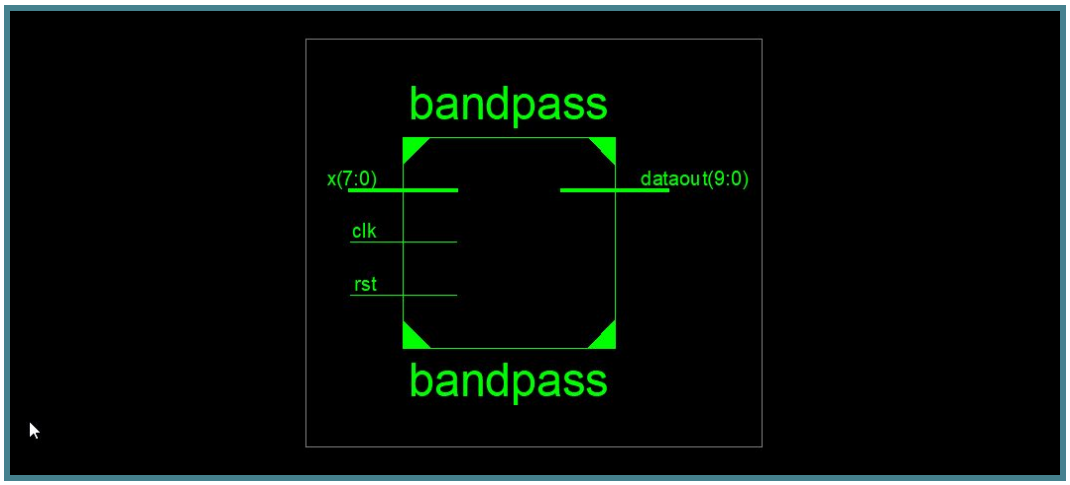
`timescale 1 ns / 1 ns
 	
module filter (clk, clk_en, reset, xin, xout);
 	
input clk;
input clk_en;
input reset;
input signed [15:0] xin;
output signed [15:0] xout;
 	
parameter signed [15:0] constant_scale = 45;
parameter signed [15:0] co_b1sec = 16'b0000000000000011;
parameter signed [15:0] co_b2sec = 16'b0000000000000000;
parameter signed [15:0] co_b3sec = 16'b0000000000000001;
parameter signed [15:0] co_a2sec = 16'b0000000000000010;
parameter signed [15:0] co_a3sec = 16'b0000000000000111;
 	
wire signed [15:0] in_data_convert;
wire signed [15:0] scale1;
wire signed [31:0] multemp;
 	
wire signed [33:0] a1sum1;
wire signed [33:0] a2sum1;
wire signed [33:0] b1sum1;
wire signed [33:0] b2sum1;
wire signed [15:0] num_data_conv;
wire signed [15:0] den_data_conv;
reg signed [15:0] num_delay_sec [0:1] ;
reg signed [15:0] den_delay_sec [0:1] ;
wire signed [31:0] a2mul1;
wire signed [31:0] a3mul1;
wire signed [31:0] b1mul1;
wire signed [31:0] b3mul1;
wire signed [16:0] single_neg_temp;
	module testbench;
	reg clk;
	reg clk_en;
	reg reset;
	reg [15:0] xin;
	wire [15:0] xout;
	
	filter uut (
	.clk(clk),
	.clk_en(clk_en),
	.reset(reset),
	.xin(xin),
	.xout(xout)
	);
	
	initial begin
	clk = 0;
	#0 clk_en = 1;
	reset = 1;
	#10 reset = 0;
	xin = 0;
	forever #5 clk = ~clk;
	xin = 271391; #10;
	xin = 524288; #10;
	xin = 741455; #10;
	xin = 908093; #10;
	xin = 1012846; #10;
	xin = 1048576; #10;
	xin = 1012846; #10;
	xin = 908093; #10;
	xin = 741455; #10;
	xin = 524288; #10;
	xin = 271391; #10;
	xin = 0; #10;
	xin = -271391; #10;
	xin = -524288; #10;
	xin = -741455; #10;
	xin = -908093; #10;
	xin = -1012846; #10;

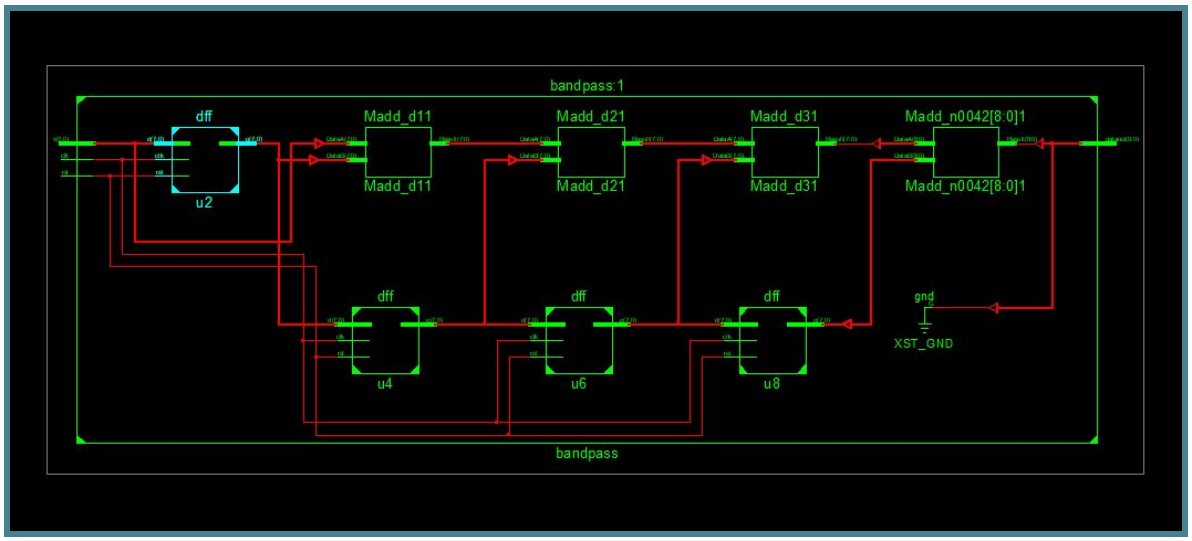
```
wire signed [33:0] b1_mul_conv1;
wire signed [33:0] addcast;
wire signed [33:0] addcast1;
wire signed [34:0] addtemp;
wire signed [33:0] subcast;
wire signed [33:0] subcast1;
wire signed [34:0] subtemp;
wire signed [33:0] subcast2;
wire signed [33:0] subcast3;
wire signed [34:0] subtemp1;
wire signed [15:0] out_data_conv;

assign in_data_convert = xin;
assign multemp = in_data_convert *
constant_scale;
assign scale1 = (multemp[29:13] + 1)>>>1;
assign num_data_conv = scale1;
assign den_data_conv = (a1sum1[29:13] + 1)>>>1;
always @( posedge clk or posedge reset)
begin: num_delay_process_sec
if (reset == 1'b1) begin
num_delay_sec[0] <= 0;
num_delay_sec[1] <= 0;
end
else begin
if (clk_en == 1'b1) begin
num_delay_sec[0] <= num_data_conv;
num_delay_sec[1] <= num_delay_sec[0];
end
end
end

always @( posedge clk or posedge reset)
begin: den_delay_process_sec
if (reset == 1'b1) begin
den_delay_sec[0] <= 0;
den_delay_sec[1] <= 0;
end
else begin
if (clk_en == 1'b1) begin
den_delay_sec[0] <= den_data_conv;
den_delay_sec[1] <= den_delay_sec[0];
end
end
end

assign a2mul1 = den_delay_sec[0] * co_a2sec;
```

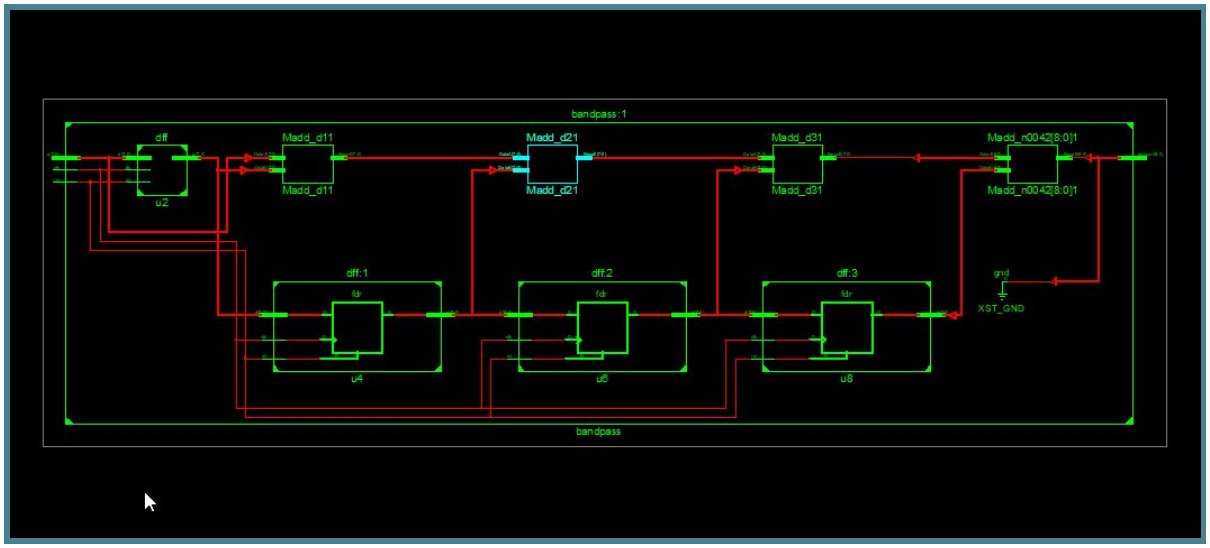
```
    xin = -1048576; #10;
    xin = -1012846; #10;
    xin = -908093; #10;
    xin = -741455; #10;
    xin = -524288; #10;
    xin = -271391; #10;
    xin = 0; #10;
    xin = 271391; #10;
    xin = 524288; #10;
    xin = 741455; #10;
    xin = 908093; #10;
    xin = 1012846; #10;
    xin = 1048576; #10;
    xin = 1012846; #10;
    xin = 908093; #10;
    xin = 741455; #10;
    end
endmodule
```

```
assign a3mul1 = den_delay_sec[1] * co_a3sec;
assign b1mul1 = $signed({num_data_conv[15:0],
14'b00000000000000});
assign single_neg_temp =
(num_delay_sec[1]==16'b0000000000000001) ?
$signed({1'b0, num_delay_sec[1]}) :
-num_delay_sec[1];
assign b3mul1 = $signed({single_neg_temp[16:0],
14'b0});
assign b1_mul_conv1 = $signed({{2{b1mul1[31]}},
b1mul1});
assign b1sum1 = b1_mul_conv1;
assign addcast = b1sum1;
assign addcast1 = $signed({{2{b3mul1[31]}},
b3mul1});
assign addtemp = addcast + addcast1;
assign b2sum1 = addtemp[33:0];
assign subcast = b2sum1;
assign subcast1 = $signed({{2{a2mul1[31]}},
a2mul1});
assign subtemp = subcast - subcast1;
assign a2sum1 = subtemp[33:0];
assign subcast2 = a2sum1;
assign subcast3 = $signed({{2{a3mul1[31]}},
a3mul1});
assign subtemp1 = subcast2 - subcast3;
assign a1sum1 = subtemp1[33:0];
assign out_data_conv = ({{4{den_data_conv[15]}},
den_data_conv[15:3]} + 1)>>>1;

assign xout = out_data_conv;

endmodule
```

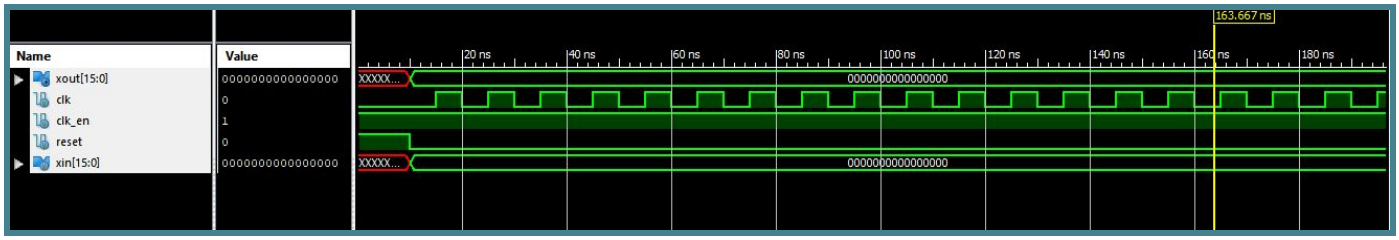
RTL DIAGRAM:


bandpass
x(7:0)
clk
rst
dataout(9:0)
bandpass


bandpass:1
dff
u2
Madd_d11
Madd_d11
Madd_d21
Madd_d21
Madd_d31
Madd_d31
Madd_n0042[8:0]1
Madd_n0042[8:0]1
dff
u4
dff
u6
dff
u8
gnd
XST_GND
bandpass


bandpass:1
dff
u2
Madd_d11
Madd_d11
Madd_d21
Madd_d21
Madd_d31
Madd_d31
Madd_n0042[8:0]1
Madd_n0042[8:0]1
dff:1
fdr
u4
dff:2
fdr
u6
dff:3
fdr
u8
gnd
XST_GND
bandpass

SIMULATION:



Note: I wasn't able to generate the timing diagram with the manual testbench code. The testbench code generated from the Matlab dfa tool was over 2000 lines and I couldn't understand the working of the code, so didn't include the testbench code and timing diagram generated from it.