```
1  /*-------------------------------------------------------------------------
2  Name:    Lamin Jammeh
3  CLass:   CE6325 Fall_2024
4  Project: 1
5  Project Description: this is a Finite Impulse Response (FIR) filter design using Verilog HDL
6  The design follows the concepts below
7  **The filter order is selected and parameterized so the design can be scaled in the fututre
8  **The filter cooefficents are pre-determined
9  **Data_in samples will be provided in the testbench to determine the filter behavior
10 **The Data_in sample is Multiplied and accumalated through the diffrent filter stages/taps
11 **The Data_out word_size = word_in + coeff_size + [log2[N]]  where N= # of taps in the filte
12
13 -------------------------------------------------------------------------*/
14 module FIR_Filter_Project1 #(parameter order = 5,                          //filter
   order [pre-determined]
15                              parameter word_size_in = 4,                   //size of
   data_in [pre-determined]
16                              parameter word_size_out = 16)  //word_size = word_in +
   coeff_size + [log2[N]] N=4, & coeff=word_in
17
18                            //declare inputs and outputs
19                            (
20                            output   [word_size_out - 1:0]   Data_out,
21                            input    [word_size_in - 1:0]    Data_in,
22                            input                            clock, reset
23                            );
24
25 reg      [word_size_in -1:0]     Samples[order-1:0];     //temp storage for input samples
   (x(n))
26 reg      [word_size_out -1:0] acc;              //temp storage for output data
27 integer  k;
28
29 //Filter Coefficients
30 parameter b0 = 4'd7;
31 parameter b1 = 4'd8;
32 parameter b2 = 4'd9;
33 parameter b3 = 4'd12;
34 parameter b4 = 4'd4;
35
36 //define the formula for the output
37 assign   Data_out = acc;
38
39 always @(posedge clock)
40  begin
41    if (reset == 1)
42    //when reset is high and clock is rising samples[k] = 0 regardless of k value
43       begin
44          for (k = 0; k < order;  k = k+1)
45          begin
46             Samples[k] <= 0;
47             acc        <= 0;
48          end
49       end
50       else
51       //when reset is low and clock is rising compute Samples with data_in to get Data_out
52          begin
53             Samples[0] <= Data_in;                  //@ k=0 Samples[0] <= Data_in
54             for (k = 1; k < order;  k = k + 1)        // from k=1 to k=order Samples[k]
   <=Samples[k-1]
55             begin
56                Samples[k] <= Samples[k - 1];          //Data_in will go throught all the
   filter coefficeints
57                acc <=   b0*Data_in
58                     + b1*Samples[0]
59                     + b2*Samples[1]
60                     + b3*Samples[2]
61                     + b4*Samples[3];
62             end
63          end
```

```
64      end
65    endmodule
```