```
/*--------------------------------------------------------------------------
Name:    Lamin Jammeh
CLass:   CE6325 Fall_2024
Project: 1 Scaled up
Project Description: this is a Finite Impulse Response (FIR) filter design using Verilog HDL
The design follows the concepts below
**The filter order is selected and parameterized so the design can be scaled in the fututre
**The filter cooefficents are pre-determined
**Data_in samples will be provided in the testbench to determine the filter behavior
**The Data_in sample is Multiplied and accumalated through the diffrent filter stages/taps
**The Data_out word_size = word_in + coeff_size + [log2[N]]  where N= # of taps in the filte

--------------------------------------------------------------------------*/
module FIR_Filter_Project1
    #(parameter order = 15,   // Filter order [N=15] coeff size =8 [Max_coeff_size = 2^8 -1 = 255]
      parameter word_size_in = 8,   // Size of data_in [Max_Data_in = 2^8 - 1 = 255]
      parameter word_size_out = 20) // Output word size = log_2 (N * Max_Data_in * Max_coeff) = roughly 20

    // Declare inputs and outputs
    (
    output reg [word_size_out - 1:0]   Data_out,
    input      [word_size_in - 1:0]    Data_in,
    input                              clock, reset
    );

    reg [word_size_in  -1:0] Samples[order-1:0];   // Temporary storage for input samples (x(n))
    reg [word_size_out -1:0] acc;                  // Temporary storage for output data
    reg [word_size_out -1:0] PR_mul[order-1:0];    // Storage for multiplication results
    reg [word_size_out -1:0] PR_add[order-1:0];    // Pipeline registers for add operations
    integer k;

    // Filter Coefficients
    parameter b0 = 8'd7;
    parameter b1 = 8'd8;
    parameter b2 = 8'd9;
    parameter b3 = 8'd12;
    parameter b4 = 8'd4;

    parameter b5 = 8'd7;
    parameter b6 = 8'd8;
    parameter b7 = 8'd9;
    parameter b8 = 8'd12;
    parameter b9 = 8'd4;

    parameter b10 = 8'd7;
    parameter b11 = 8'd8;
    parameter b12 = 8'd9;
    parameter b13 = 8'd12;
    parameter b14 = 8'd4;

    always @(posedge clock) begin
        if (reset == 1) begin
            // Reset all samples, accumulation, and pipeline registers
            for (k = 0; k < order; k = k + 1) begin
                Samples[k] <= 0;
                PR_mul[k] <= 0;
                PR_add[k] <= 0;
            end
            acc <= 0;
            Data_out <= 0;
        end else begin
            // Shift samples
            Samples[0] <= Data_in;
            for (k = 1; k < order; k = k + 1) begin
                Samples[k] <= Samples[k - 1];
            end
```

```
67
68              // Compute multiplication results (pipeline stage 1)
69              PR_mul[0] <= b0 * Data_in;
70              PR_mul[1] <= b1 * Samples[0];
71              PR_mul[2] <= b2 * Samples[1];
72              PR_mul[3] <= b3 * Samples[2];
73              PR_mul[4] <= b4 * Samples[3];
74              PR_mul[5] <= b5 * Samples[4];
75              PR_mul[6] <= b6 * Samples[5];
76              PR_mul[7] <= b7 * Samples[6];
77              PR_mul[8] <= b8 * Samples[7];
78              PR_mul[9] <= b9 * Samples[8];
79
80              PR_mul[10] <= b10 * Samples[9];
81              PR_mul[11] <= b11 * Samples[10];
82              PR_mul[12] <= b12 * Samples[11];
83              PR_mul[13] <= b13 * Samples[12];
84              PR_mul[14] <= b14 * Samples[13];
85
86              // Pipeline stage for addition
87              PR_add[0] <= PR_mul[0] + PR_mul[1];
88              PR_add[1] <= PR_add[0] + PR_mul[2];
89              PR_add[2] <= PR_add[1] + PR_mul[3];
90              PR_add[3] <= PR_add[2] + PR_mul[4];
91              PR_add[4] <= PR_add[3] + PR_mul[5];
92              PR_add[5] <= PR_add[4] + PR_mul[6];
93              PR_add[6] <= PR_add[5] + PR_mul[7];
94              PR_add[7] <= PR_add[6] + PR_mul[8];
95              PR_add[8] <= PR_add[7] + PR_mul[9];
96
97              PR_add[9]  <= PR_add[8] + PR_mul[10];
98              PR_add[10] <= PR_add[9] + PR_mul[11];
99              PR_add[11] <= PR_add[10] + PR_mul[12];
100             PR_add[12] <= PR_add[11] + PR_mul[13];
101             PR_add[13] <= PR_add[12] + PR_mul[14];
102             // Final result after the last addition
103             acc <= PR_add[13];
104             Data_out <= acc;
105         end
106     end
107 endmodule
108
```