Verilog

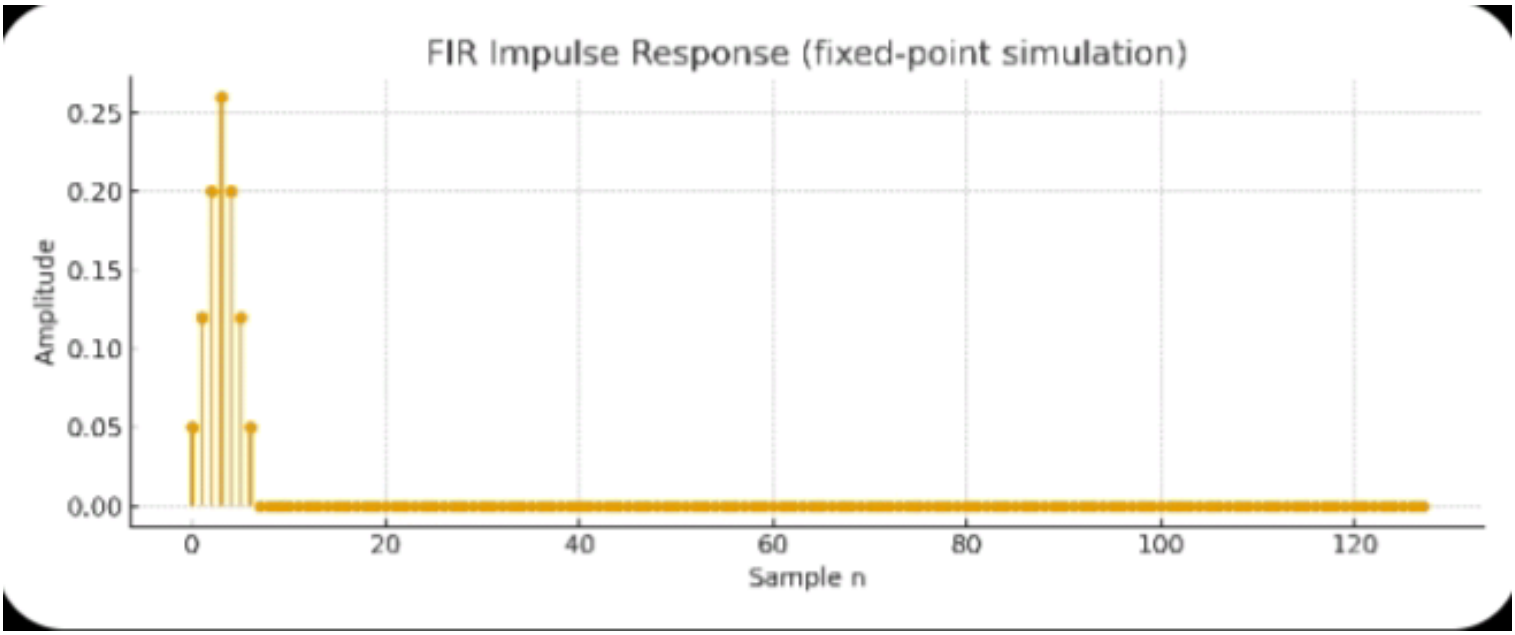
```
// fir8.v
// Simple parameterizable FIR filter (8 taps) - signed
fixed-point (Q1.15 assumed)
// Synthesizable Verilog (uses generate/for loops).
module fir8 #(
    parameter integer WIDTH = 16,        // input/output
data width (signed)
    parameter integer COEFF_WIDTH = 16,  // coefficient
width (signed)
    parameter integer TAPS = 8
)(
    input  wire clk,
    input  wire rst,
    input  wire signed [WIDTH-1:0] din,  // Q1.15 input
    output reg  signed [WIDTH-1:0] dout  // Q1.15 output
);

    // Coefficients (Q1.15). Edit these coefficients as
needed.
    // Example symmetric low-pass coefficients normalized
(floating -> quantized externally).
    // Here we store as signed integers representing Q1.15
values.
    localparam signed [COEFF_WIDTH-1:0] h0 = 16'sd1638; //
~0.05
    localparam signed [COEFF_WIDTH-1:0] h1 = 16'sd3932; //
~0.12
    localparam signed [COEFF_WIDTH-1:0] h2 = 16'sd6554; //
~0.20
    localparam signed [COEFF_WIDTH-1:0] h3 = 16'sd8519; //
~0.26
    localparam signed [COEFF_WIDTH-1:0] h4 = 16'sd6554; //
~0.20
    localparam signed [COEFF_WIDTH-1:0] h5 = 16'sd3932; //
~0.12
    localparam signed [COEFF_WIDTH-1:0] h6 = 16'sd1638; //
~0.05
    localparam signed [COEFF_WIDTH-1:0] h7 = 16'sd0;    //
~0.00

    // coefficient array for easier loop use
(SystemVerilog style allowed in many tools)
    // If your tool doesn't support packed arrays in
Verilog, unroll the MAC manually.
    wire signed [COEFF_WIDTH-1:0] h [0:TAPS-1];
    assign h[0] = h0;
    assign h[1] = h1;
    assign h[2] = h2;
    assign h[3] = h3;
    assign h[4] = h4;
    assign h[5] = h5;
    assign h[6] = h6;
    assign h[7] = h7;
```

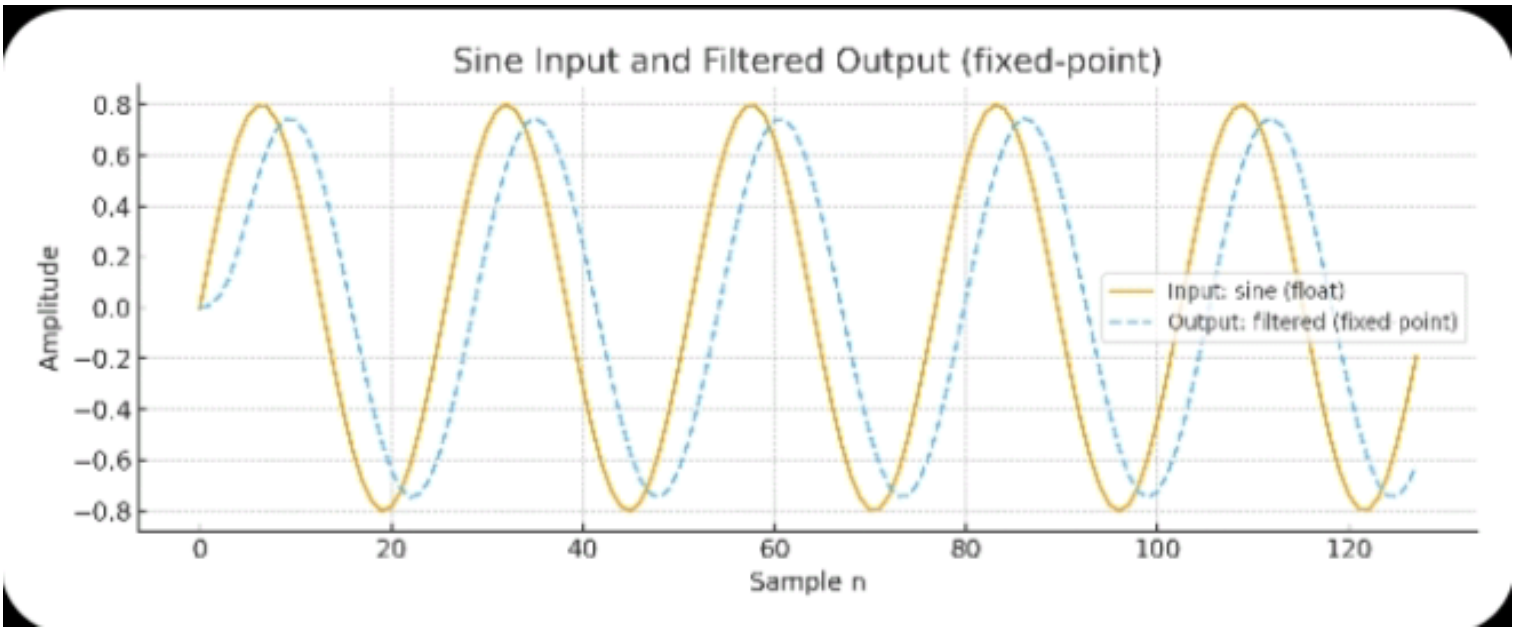
```
    // shift register for samples
    reg signed [WIDTH-1:0] shift_reg [0:TAPS-1];

    integer i;
    always @(posedge clk) begin
        if (rst) begin
            for (i = 0; i < TAPS; i = i + 1)
                shift_reg[i] <= 0;
            dout <= 0;
        end else begin
            // shift
            for (i = TAPS-1; i > 0; i = i - 1)
                shift_reg[i] <= shift_reg[i-1];
            shift_reg[0] <= din;

            // Multiply-accumulate
            // Multiply: WIDTH * COEFF_WIDTH -> product
width ~ (WIDTH+COEFF_WIDTH)
            // We'll accumulate into a wider signed reg to
avoid overflow.
            // After accumulation (Q1.15 * Q1.15 ->
Q2.30), shift right by 15 to get Q1.15.
            // We'll implement as blocking combinational
ops inside this clocked block for clarity.
            reg signed [WIDTH+COEFF_WIDTH+7:0] acc; //
wide accumulator
            acc = 0;
            for (i = 0; i < TAPS; i = i + 1) begin
                acc = acc + ( $signed(shift_reg[i]) *
$signed(h[i]) );
            end
            // acc is Q30 (sum of Q15*Q15 products).
Convert to Q15 by arithmetic shift right 15.
            // Add rounding (optional): add 1<<14 before
shift for rounding positive values.
            acc = acc + ( (acc >= 0) ? (1 <<< 14) : - (1
<<< 14) );
            // assign to output with saturation to signed
WIDTH
            // shift right by 15
            reg signed [WIDTH+COEFF_WIDTH+7:0] acc_q15;
            acc_q15 = acc >>> 15;

            // saturation to WIDTH-bit signed
            if (acc_q15 > $signed({1'b0,
{WIDTH-1{1'b1}}})) begin
                dout <= {1'b0, {WIDTH-1{1'b1}}};
            end else if (acc_q15 < $signed({1'b1,
{WIDTH-1{1'b0}}})) begin
                dout <= {1'b1, {WIDTH-1{1'b0}}};
            end else begin
                dout <= acc_q15[WIDTH-1:0];
            end
        end
    end

endmodule
```

Verilog

```
// tb_fir8.v
`timescale 1ns/1ps
module tb_fir8;
    reg clk = 0;
    reg rst = 1;
    reg signed [15:0] din;
    wire signed [15:0] dout;

    fir8 uut (
        .clk(clk),
        .rst(rst),
        .din(din),
        .dout(dout)
    );

    // clock
    always #5 clk = ~clk; // 100 MHz hypothetical

    integer i;
    initial begin
        // reset
        rst = 1;
        din = 0;
        #20;
        rst = 0;
        #10;

        // 1) Impulse
        din = 16'sh7fff; // max Q1.15 ~ +0.99997 to
represent 1.0 (or use exact quantized 1.0)
        #10;
        din = 16'sh0000;
        for (i = 0; i < 50; i = i + 1) begin
            #10;
        end
```

```
        // 2) Step
        for (i = 0; i < 40; i = i + 1) begin
            din = 16'sh4000; // 0.5 (approx)
            #10;
        end
        din = 0;
        #50;

        // 3) Sine-like test: precomputed samples or a
small ramp approximating sine
        for (i = 0; i < 128; i = i + 1) begin
            // simple pseudo-sine using sin from python
exported values - replace with real samples if needed
            // for demonstration use a small pattern:
            din = $signed(16'sd0 + $signed(i % 32 *
100)); // replace by real test vector for better test
            #10;
        end

        #200;
        $stop;
    end
endmodule
```


FIR Impulse Response (fixed-point simulation)
Amplitude
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
Sample n


Sine Input and Filtered Output (fixed-point)
Amplitude
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
20
40
60
80
100
120
Sample n
Input: sine (float)
Output: filtered (fixed point)