```
1   // ee417 lesson 9 Assignment 1 L9A1
2   // Name: Ron Kalin, Date: 07-10-24  Group: Kalin/Jammeh
3   // Design: moving average FIR filter of overlapping windows of 4 input samples
4   // FIR order will be a power of 2, paramterized with word_size
5   // top level module
6   module moving_average_filter #(parameter word_size=4, order=4, n=2)(
7       output reg [2*word_size-1:0] filtered_sample,
8       input      [word_size-1:0]   sample_in,
9       input                        enable, clk, reset
10  );
11      // Coefficient values for a 4-tap moving average filter
12      //reg [word_size-1:0] coeff [0:order] = {16'h1000, 16'h1000, 16'h1000, 16'h1000};
13      reg [word_size-1:0] coeff0 = 4'd1; //coefficients of one were selected
14      reg [word_size-1:0] coeff1 = 4'd1; //because for moving average they aren't needed
15      reg [word_size-1:0] coeff2 = 4'd1;
16      reg [word_size-1:0] coeff3 = 4'd1;
17      reg [word_size-1:0] tap_outputs [0:3];
18
19      // Circular buffer to store input samples
20      reg [word_size-1:0] buffer0, buffer1, buffer2, buffer3;
21      reg [word_size-1:0] buffer [0:order-1];
22      always @(posedge clk) begin
23          if (reset  || ~enable) begin
24              buffer0 = 0;
25              buffer1 = 0;
26              buffer2 = 0;
27              buffer3 = 0;
28          end else if (enable) begin
29              buffer3 = buffer2;
30              buffer2 = buffer1;
31              buffer1 = buffer0;
32              buffer0 = sample_in;
33          end
34        // Multipliers and accumulator
35        tap_outputs[0] = buffer0 * coeff0; //coeff[0];
36        tap_outputs[1] = buffer1 * coeff1; //coeff[1];
37        tap_outputs[2] = buffer2 * coeff2; //coeff[2];
38        tap_outputs[3] = buffer3 * coeff3; //coeff[3];
39
40        filtered_sample = (buffer0 + buffer1 + buffer2 + buffer3); // >> 2;
41      //filtered_sample = buffer[0] + buffer[1] + buffer[2] + buffer[3];
42      //filtered_sample <= (tap_outputs[0] + tap_outputs[1] + tap_outputs[2] + tap_outputs[3]);
43      // Right shift by 2 bits to approximate division by 4
44        filtered_sample = (filtered_sample >> n); //shift n places to divide by 2^n
45        //shift n places requires 2^n samples (buffers and coefficients)
46      end
47  endmodule
48
49
```