EE417 Advanced Digital Logic with Verilog HDL Name: Ron Kalin / Lamin Jammeh Date: 07/10/24

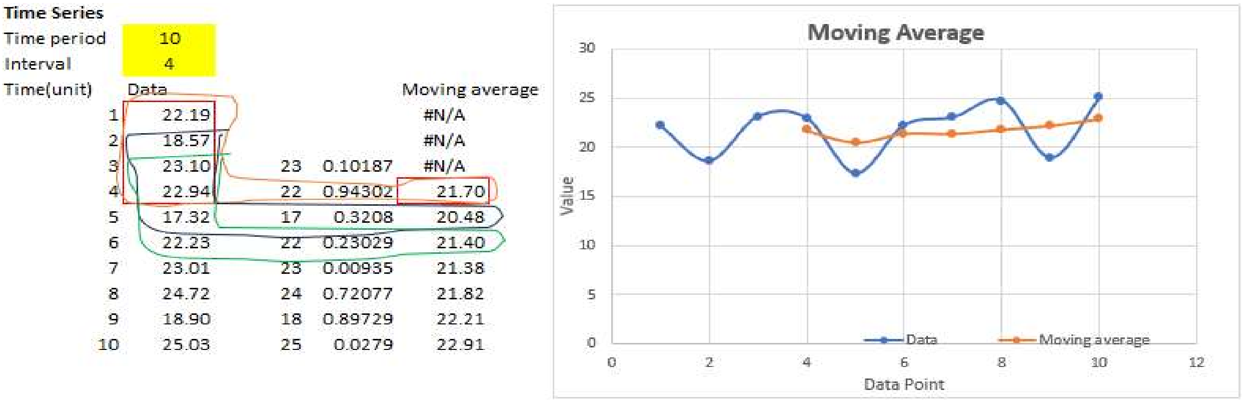
Lesson 9 Digital Signal Processing

Digital Signal Processing is essential in different applications. Filtering audio signals for clear audio output hearing, or for speaker identification, or speech recognition are one of few applications of DSP. Filtering image pixels to remove salt and pepper noise, or embedding security information in image steganography is also part of some applications of DSP. Integrators, Differentiators, Decimators and Interpolators are also commonly used DSP applications. Implementing DSP in

One of the special FIR filters is the moving average filter of overlapping windows of 4 input samples. The module should have reset and enable input signals. The code should be parametrizable: The order of the filter should be one parameter (starting with just 4 samples). We will restrict the FIR to always have an order that is a power of 2 (2, 4, 8, 16 and so on). This allows us to use the right shifting to implement division (Reference: lesson 3). The input sample word-Size should also be a parameter (starting with 4 bits). Calculate the size of the output and the internal registers correspondingly for the moving average filter.
Create a test-bench that tests the functionality of the FIR. The test should include an impulse response (one sample equal to 4 embedded between 4 samples of zeros, the reset case, the disabled and enabled cases, and maximum possible sample values, as well as other test vectors. Investigate the effect of the rounding on the moving average filter. Combine your code, and simulation results in one pdf file.

FIR Moving Average Filter of Overlapping windows of 4 input samples.
A moving average, also called a moving mean or a rolling mean, is a calculation that relies on a series of averages from data subsets within an entire data set. It's a term statisticians, technical analysts and financial analysts use to describe changes to averages as new data becomes available. It explains how a data series changes over a set period. The moving average also updates to include recent data along with data points from pre-determined intervals.Moving average can be a helpful metric for tracking price trends because it won't reflect short-term fluctuations or infrequent outliers as drastically. In stock trading, experts often rely on 200-day moving averages, but short-, medium- and long-term averages can all be useful metrics to track.

Example - The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 10 days, 20 minutes, 30 weeks, or any time period the trader chooses.



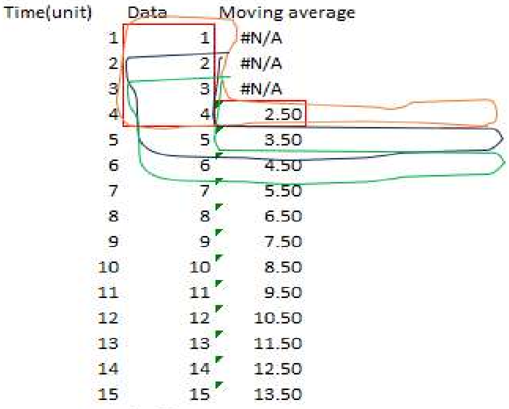
Time Series

Time period	10
Interval	4

Time(unit)	Data			Moving average
1	22.19			#N/A
2	18.57			#N/A
3	23.10	23	0.10187	#N/A
4	22.94	22	0.94302	21.70
5	17.32	17	0.3208	20.48
6	22.23	22	0.23029	21.40
7	23.01	23	0.00935	21.38
8	24.72	24	0.72077	21.82
9	18.90	18	0.89729	22.21
10	25.03	25	0.0279	22.91

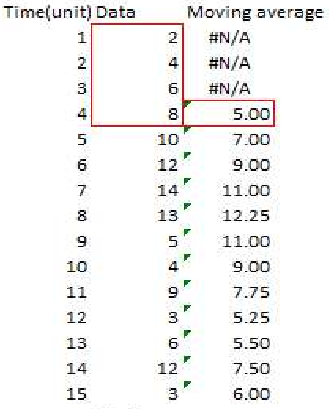
Fig. 1

As can be seen above in Fig. 1, the moving average is an average of the last interval of data points. In this case the interval = 4.
The first three datapoints of the moving average produce an error (#N/A) because there is not enough data (at least four) to take an average.
Each moving average datapoint overlaps the last data point by one datapoint less than the interval.
The below data Fig. 2 and Fig. 3 will be used to input into the FIR moving average filter.



Time(unit)	Data	Moving average
1	1	#N/A
2	2	#N/A
3	3	#N/A
4	4	2.50
5	5	3.50
6	6	4.50
7	7	5.50
8	8	6.50
9	9	7.50
10	10	8.50
11	11	9.50
12	12	10.50
13	13	11.50
14	14	12.50
15	15	13.50

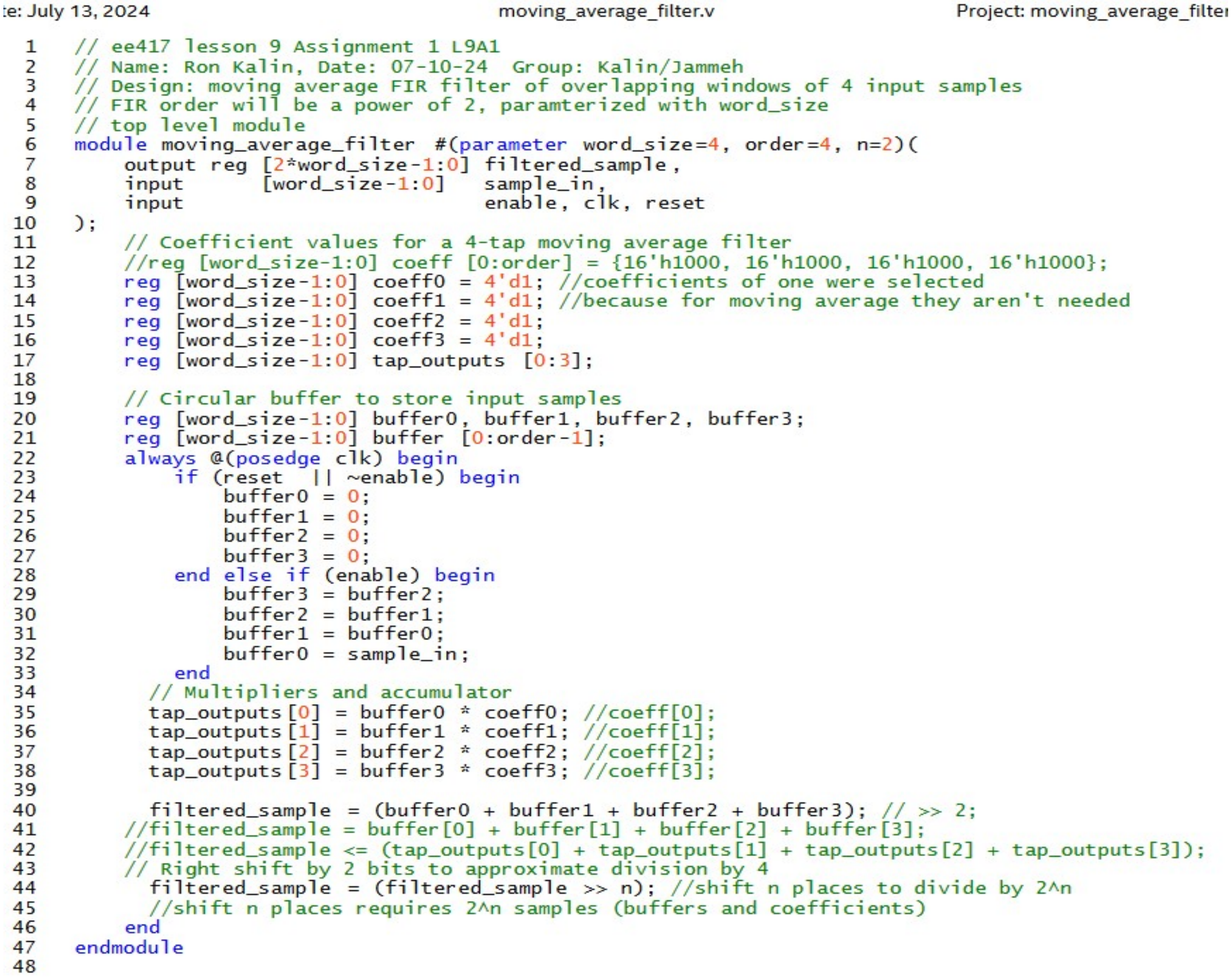
Fig. 2



Time(unit)	Data	Moving average
1	2	#N/A
2	4	#N/A
3	6	#N/A
4	8	5.00
5	10	7.00
6	12	9.00
7	14	11.00
8	13	12.25
9	5	11.00
10	4	9.00
11	9	7.75
12	3	5.25
13	6	5.50
14	12	7.50
15	3	6.00

Fig. 3

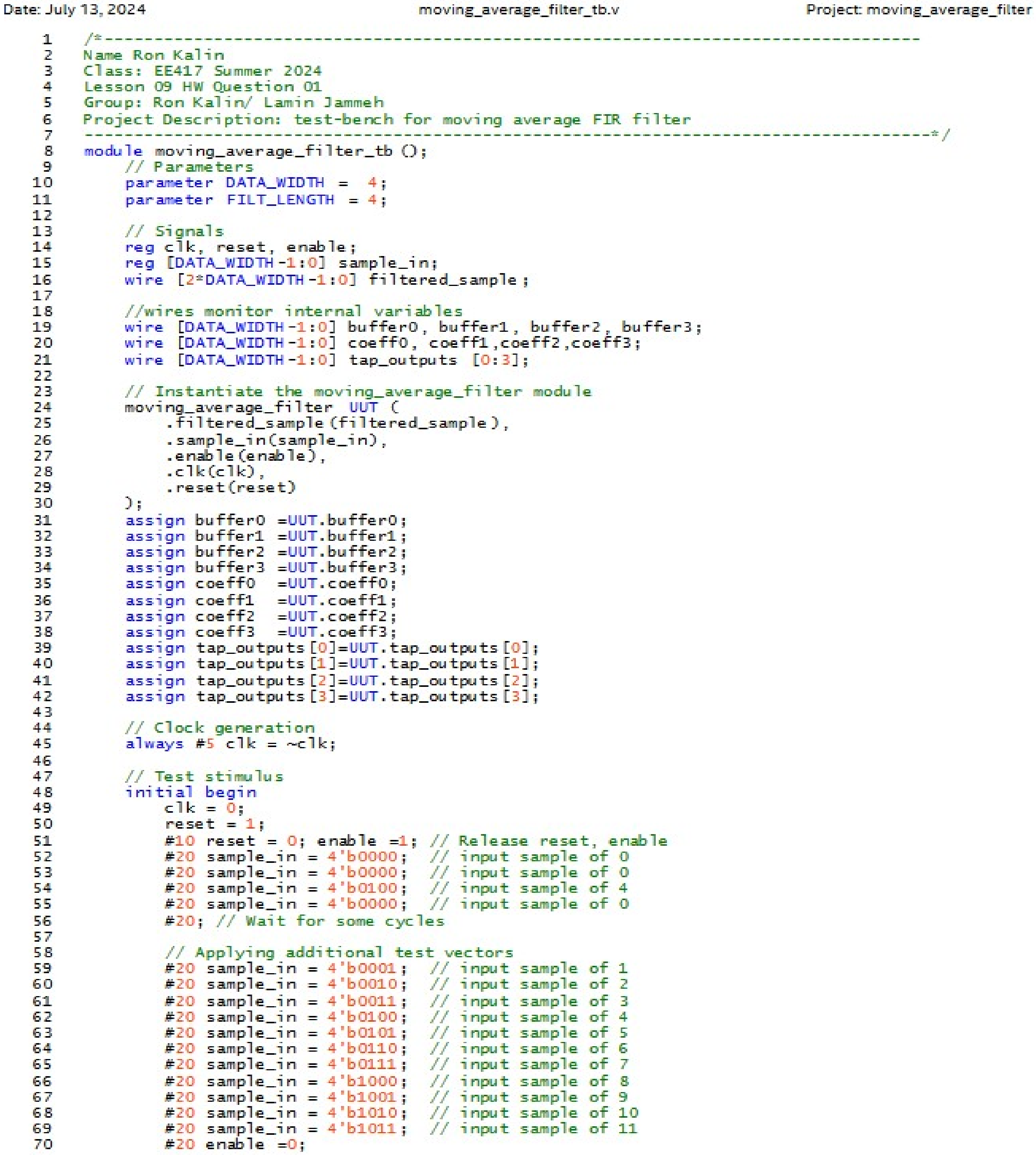
Design Verilog HDL Code - 4bit words



te: July 13, 2024 moving_average_filter.v Project: moving_average_filte

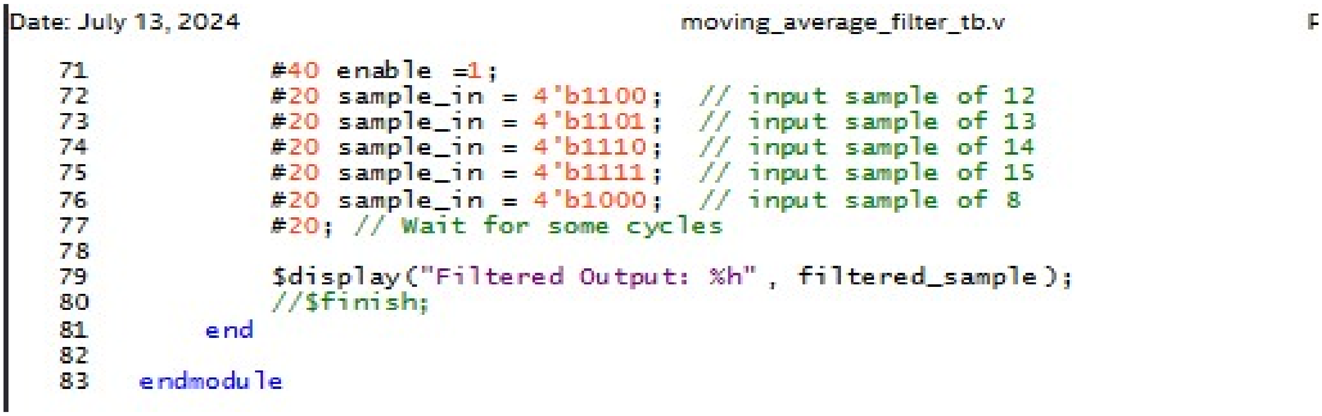
```verilog
// ee417 lesson 9 Assignment 1 L9A1
// Name: Ron Kalin, Date: 07-10-24  Group: Kalin/Jammeh
// Design: moving average FIR filter of overlapping windows of 4 input samples
// FIR order will be a power of 2, paramterized with word_size
// top level module
module moving_average_filter #(parameter word_size=4, order=4, n=2)(
    output reg [2*word_size-1:0] filtered_sample ,
    input      [word_size-1:0]   sample_in,
    input                        enable, clk, reset
);
    // Coefficient values for a 4-tap moving average filter
    //reg [word_size-1:0] coeff [0:order] = {16'h1000, 16'h1000, 16'h1000, 16'h1000};
    reg [word_size-1:0] coeff0 = 4'd1; //coefficients of one were selected
    reg [word_size-1:0] coeff1 = 4'd1; //because for moving average they aren't needed
    reg [word_size-1:0] coeff2 = 4'd1;
    reg [word_size-1:0] coeff3 = 4'd1;
    reg [word_size-1:0] tap_outputs [0:3];

    // Circular buffer to store input samples
    reg [word_size-1:0] buffer0, buffer1, buffer2, buffer3;
    reg [word_size-1:0] buffer [0:order-1];
    always @(posedge clk) begin
        if (reset  || ~enable) begin
            buffer0 = 0;
            buffer1 = 0;
            buffer2 = 0;
            buffer3 = 0;
        end else if (enable) begin
            buffer3 = buffer2;
            buffer2 = buffer1;
            buffer1 = buffer0;
            buffer0 = sample_in;
        end
      // Multipliers and accumulator
      tap_outputs[0] = buffer0 * coeff0; //coeff[0];
      tap_outputs[1] = buffer1 * coeff1; //coeff[1];
      tap_outputs[2] = buffer2 * coeff2; //coeff[2];
      tap_outputs[3] = buffer3 * coeff3; //coeff[3];

      filtered_sample = (buffer0 + buffer1 + buffer2 + buffer3); // >> 2;
    //filtered_sample = buffer[0] + buffer[1] + buffer[2] + buffer[3];
    //filtered_sample <= (tap_outputs[0] + tap_outputs[1] + tap_outputs[2] + tap_outputs[3]);
    // Right shift by 2 bits to approximate division by 4
      filtered_sample = (filtered_sample >> n); //shift n places to divide by 2^n
      //shift n places requires 2^n samples (buffers and coefficients)
    end
endmodule
```

Testbench Verilog HDL Code -4bit



Date: July 13, 2024 moving_average_filter_tb.v Project: moving_average_filter

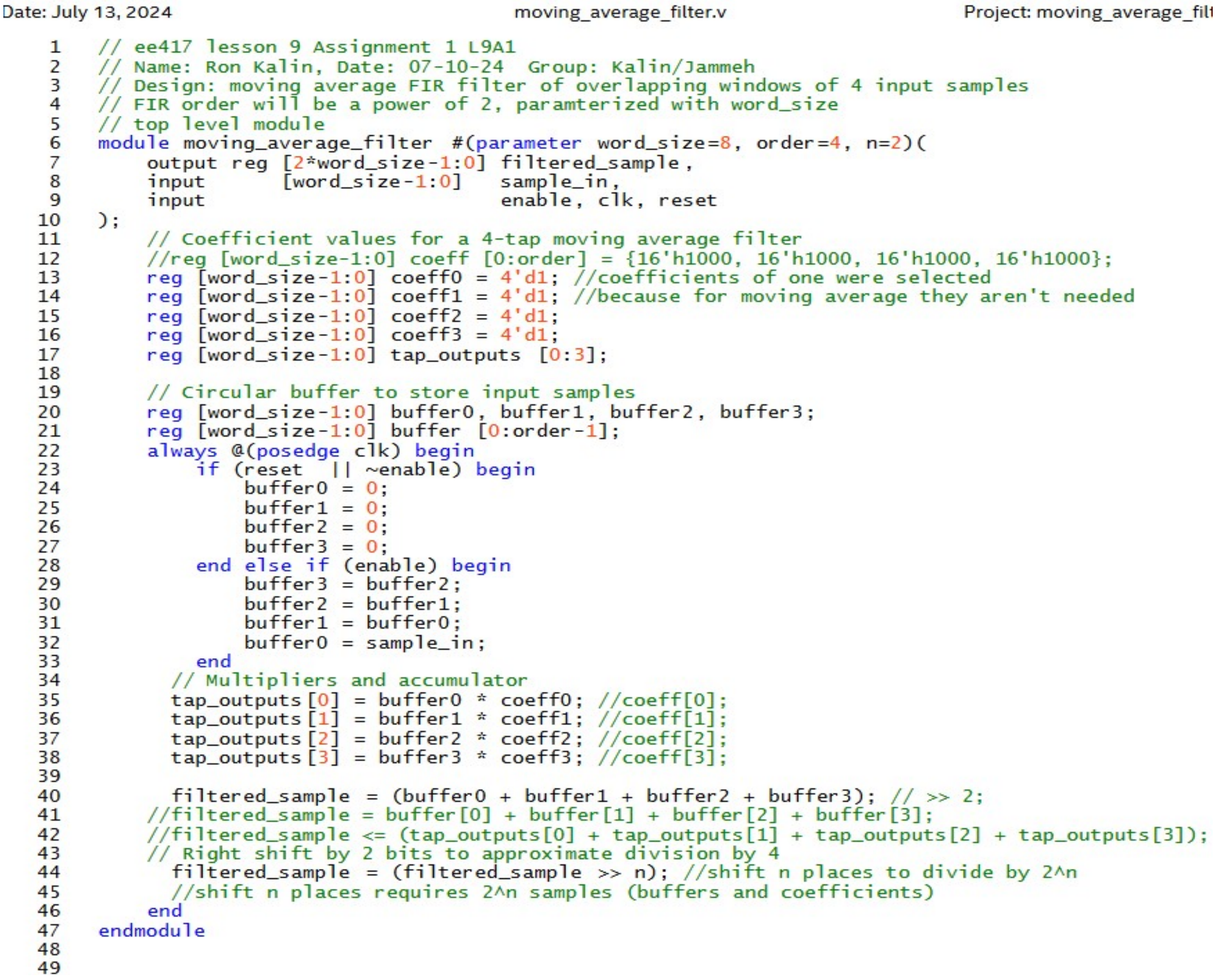
```verilog
/*-----------------------------------------------------------------------------------
Name Ron Kalin
Class: EE417 Summer 2024
Lesson 09 HW Question 01
Group: Ron Kalin/ Lamin Jammeh
Project Description: test-bench for moving average FIR filter
-------------------------------------------------------------------------------------*/
module moving_average_filter_tb ();
    // Parameters
    parameter DATA_WIDTH =  4;
    parameter FILT_LENGTH  = 4;

    // Signals
    reg clk, reset, enable;
    reg [DATA_WIDTH-1:0] sample_in;
    wire  [2*DATA_WIDTH-1:0] filtered_sample ;

    //wires monitor internal variables
    wire  [DATA_WIDTH-1:0] buffer0, buffer1, buffer2, buffer3;
    wire  [DATA_WIDTH-1:0] coeff0, coeff1,coeff2,coeff3;
    wire  [DATA_WIDTH-1:0] tap_outputs  [0:3];

    // Instantiate the moving_average_filter module
    moving_average_filter  UUT (
        .filtered_sample(filtered_sample),
        .sample_in(sample_in),
        .enable(enable),
        .clk(clk),
        .reset(reset)
    );
    assign buffer0 =UUT.buffer0;
    assign buffer1 =UUT.buffer1;
    assign buffer2 =UUT.buffer2;
    assign buffer3 =UUT.buffer3;
    assign coeff0  =UUT.coeff0;
    assign coeff1  =UUT.coeff1;
    assign coeff2  =UUT.coeff2;
    assign coeff3  =UUT.coeff3;
    assign tap_outputs[0]=UUT.tap_outputs[0];
    assign tap_outputs[1]=UUT.tap_outputs[1];
    assign tap_outputs[2]=UUT.tap_outputs[2];
    assign tap_outputs[3]=UUT.tap_outputs[3];

    // Clock generation
    always #5 clk = ~clk;

    // Test stimulus
    initial begin
        clk = 0;
        reset = 1;
        #10 reset = 0; enable =1; // Release reset, enable
        #20 sample_in = 4'b0000;  // input sample of 0
        #20 sample_in = 4'b0000;  // input sample of 0
        #20 sample_in = 4'b0100;  // input sample of 4
        #20 sample_in = 4'b0000;  // input sample of 0
        #20; // Wait for some cycles

        // Applying additional test vectors
        #20 sample_in = 4'b0001;  // input sample of 1
        #20 sample_in = 4'b0010;  // input sample of 2
        #20 sample_in = 4'b0011;  // input sample of 3
        #20 sample_in = 4'b0100;  // input sample of 4
        #20 sample_in = 4'b0101;  // input sample of 5
        #20 sample_in = 4'b0110;  // input sample of 6
        #20 sample_in = 4'b0111;  // input sample of 7
        #20 sample_in = 4'b1000;  // input sample of 8
        #20 sample_in = 4'b1001;  // input sample of 9
        #20 sample_in = 4'b1010;  // input sample of 10
        #20 sample_in = 4'b1011;  // input sample of 11
        #20 enable =0;
```



Date: July 13, 2024 moving_average_filter_tb.v P

```
        #40 enable  =1;
        #20 sample_in = 4'b1100;  // input sample of 12
        #20 sample_in = 4'b1101;  // input sample of 13
        #20 sample_in = 4'b1110;  // input sample of 14
        #20 sample_in = 4'b1111;  // input sample of 15
        #20 sample_in = 4'b1000;  // input sample of 8
        #20; // Wait for some cycles

        $display("Filtered Output: %h" , filtered_sample);
        //$finish;
    end

endmodule
```

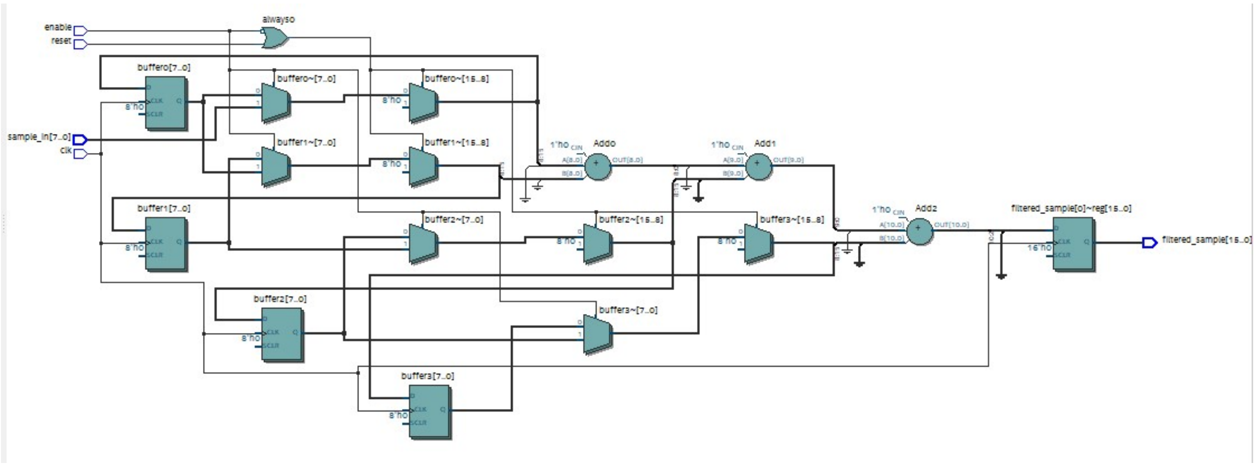
Design Verilog HDL Code - 8bit words



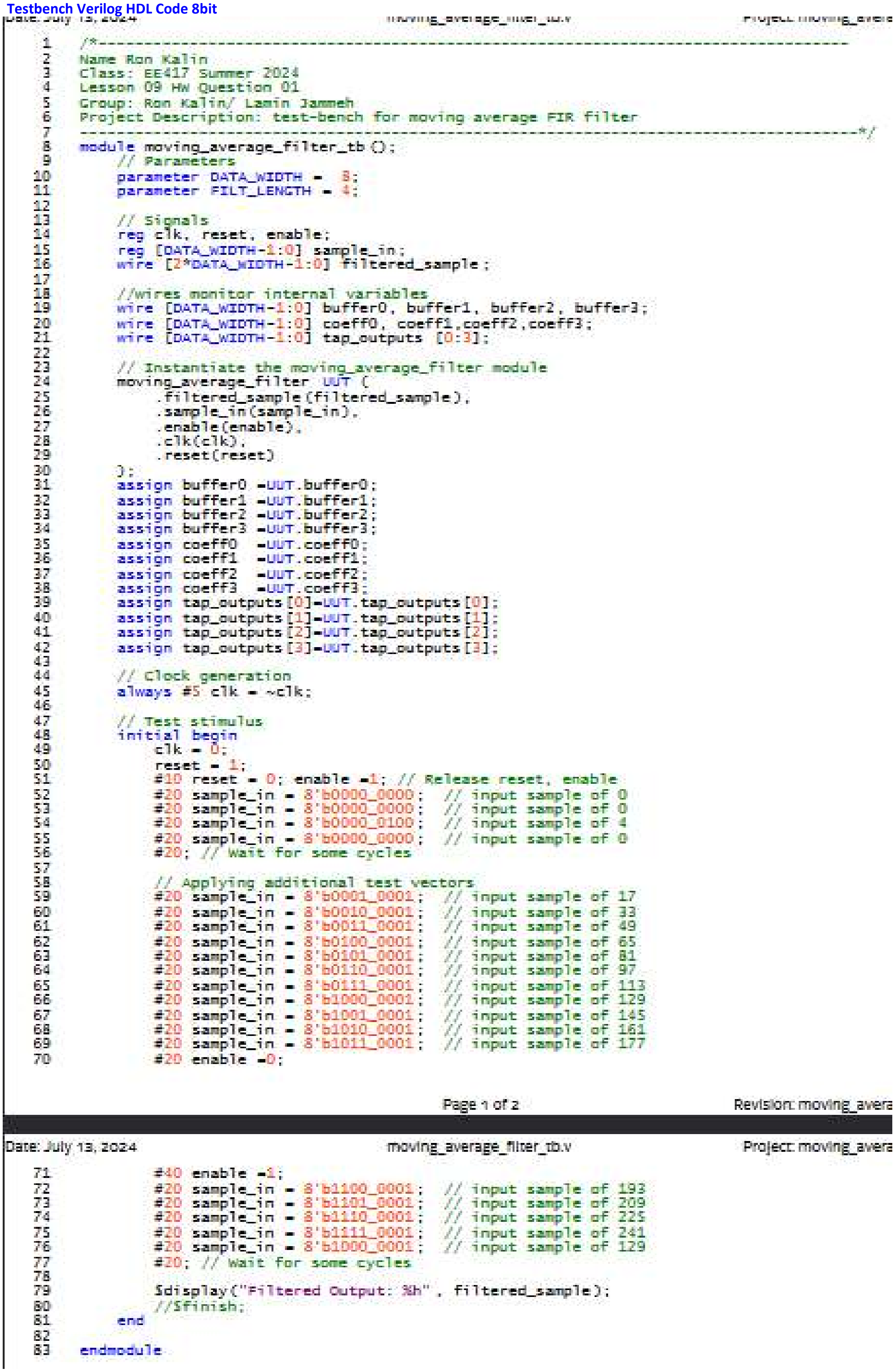
Date: July 13, 2024 moving_average_filter.v Project: moving_average_filt

```
// ee417 lesson 9 Assignment 1 L9A1
// Name: Ron Kalin, Date: 07-10-24  Group: Kalin/Jammeh
// Design: moving average FIR filter of overlapping windows of 4 input samples
// FIR order will be a power of 2, paramterized with word_size
// top level module
module moving_average_filter #(parameter word_size=8, order=4, n=2)(
    output reg [2*word_size-1:0] filtered_sample ,
    input      [word_size-1:0]   sample_in,
    input                        enable, clk, reset
);
    // Coefficient values for a 4-tap moving average filter
    //reg [word_size-1:0] coeff [0:order] = {16'h1000, 16'h1000, 16'h1000, 16'h1000};
    reg [word_size-1:0] coeff0 = 4'd1; //coefficients of one were selected
    reg [word_size-1:0] coeff1 = 4'd1; //because for moving average they aren't needed
    reg [word_size-1:0] coeff2 = 4'd1;
    reg [word_size-1:0] coeff3 = 4'd1;
    reg [word_size-1:0] tap_outputs [0:3];

    // Circular buffer to store input samples
    reg [word_size-1:0] buffer0, buffer1, buffer2, buffer3;
    reg [word_size-1:0] buffer [0:order-1];
    always @(posedge clk) begin
        if (reset  || ~enable) begin
            buffer0 = 0;
            buffer1 = 0;
            buffer2 = 0;
            buffer3 = 0;
        end else if (enable) begin
            buffer3 = buffer2;
            buffer2 = buffer1;
            buffer1 = buffer0;
            buffer0 = sample_in;
        end
      // Multipliers and accumulator
      tap_outputs[0] = buffer0 * coeff0; //coeff[0];
      tap_outputs[1] = buffer1 * coeff1; //coeff[1];
      tap_outputs[2] = buffer2 * coeff2; //coeff[2];
      tap_outputs[3] = buffer3 * coeff3; //coeff[3];

      filtered_sample = (buffer0 + buffer1 + buffer2 + buffer3); // >> 2;
    //filtered_sample = buffer[0] + buffer[1] + buffer[2] + buffer[3];
    //filtered_sample <= (tap_outputs[0] + tap_outputs[1] + tap_outputs[2] + tap_outputs[3]);
    // Right shift by 2 bits to approximate division by 4
      filtered_sample = (filtered_sample >> n); //shift n places to divide by 2^n
      //shift n places requires 2^n samples (buffers and coefficients)
    end
endmodule


```

RTL Viewer


enable
reset
always0
buffer0[7..0]
D
CLK
SCLR
Q
8'h0
sample_in[7..0]
clk
buffer0~[7..0]
buffer0~[15..8]
buffer1~[7..0]
buffer1~[15..8]
buffer1[7..0]
buffer2~[7..0]
buffer2~[15..8]
buffer2[7..0]
buffer3~[7..0]
buffer3~[15..8]
buffer3[7..0]
0
1
1'h0 CIN
Add0
A[8..0]
B[8..0]
OUT[8..0]
Add1
A[9..0]
B[9..0]
OUT[9..0]
Add2
A[10..0]
B[10..0]
OUT[10..0]
filtered_sample[0]~reg[15..0]
16'h0
filtered_sample[15..0]

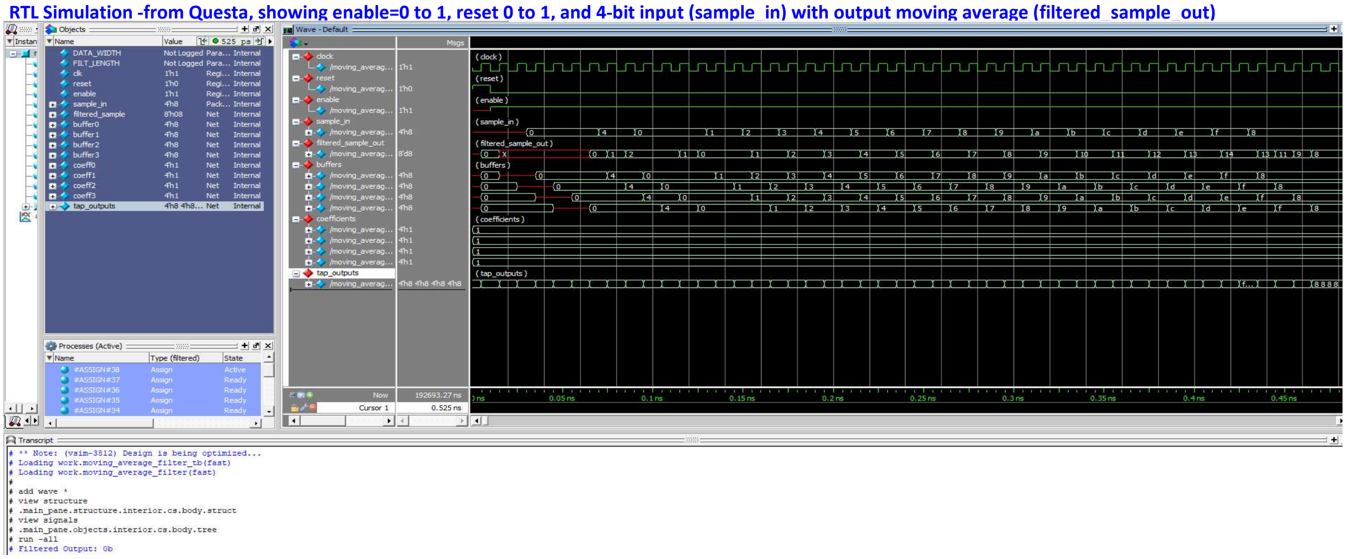
Testbench Verilog HDL Code 8bit



```verilog
/*------------------------------------------------------------------------------
Name Ron Kalin
Class: EE417 Summer 2024
Lesson 09 HW Question 01
Group: Ron Kalin/ Lamin Jammeh
Project Description: test-bench for moving average FIR filter
---------------------------------------------------------------------------------*/
module moving_average_filter_tb ();
    // Parameters
    parameter DATA_WIDTH =  8;
    parameter FILT_LENGTH = 4;

    // Signals
    reg clk, reset, enable;
    reg [DATA_WIDTH-1:0] sample_in;
    wire [2*DATA_WIDTH-1:0] filtered_sample;

    //wires monitor internal variables
    wire [DATA_WIDTH-1:0] buffer0, buffer1, buffer2, buffer3;
    wire [DATA_WIDTH-1:0] coeff0, coeff1,coeff2,coeff3;
    wire [DATA_WIDTH-1:0] tap_outputs [0:3];

    // Instantiate the moving_average_filter module
    moving_average_filter  UUT (
        .filtered_sample(filtered_sample),
        .sample_in(sample_in),
        .enable(enable),
        .clk(clk),
        .reset(reset)
    );
    assign buffer0 =UUT.buffer0;
    assign buffer1 =UUT.buffer1;
    assign buffer2 =UUT.buffer2;
    assign buffer3 =UUT.buffer3;
    assign coeff0  =UUT.coeff0;
    assign coeff1  =UUT.coeff1;
    assign coeff2  =UUT.coeff2;
    assign coeff3  =UUT.coeff3;
    assign tap_outputs[0]=UUT.tap_outputs[0];
    assign tap_outputs[1]=UUT.tap_outputs[1];
    assign tap_outputs[2]=UUT.tap_outputs[2];
    assign tap_outputs[3]=UUT.tap_outputs[3];

    // Clock generation
    always #5 clk = ~clk;

    // Test stimulus
    initial begin
        clk = 0;
        reset = 1;
        #10 reset = 0; enable =1; // Release reset, enable
        #20 sample_in = 8'b0000_0000;  // input sample of 0
        #20 sample_in = 8'b0000_0000;  // input sample of 0
        #20 sample_in = 8'b0000_0100;  // input sample of 4
        #20 sample_in = 8'b0000_0000;  // input sample of 0
        #20; // Wait for some cycles

        // Applying additional test vectors
        #20 sample_in = 8'b0001_0001;  // input sample of 17
        #20 sample_in = 8'b0010_0001;  // input sample of 33
        #20 sample_in = 8'b0011_0001;  // input sample of 49
        #20 sample_in = 8'b0100_0001;  // input sample of 65
        #20 sample_in = 8'b0101_0001;  // input sample of 81
        #20 sample_in = 8'b0110_0001;  // input sample of 97
        #20 sample_in = 8'b0111_0001;  // input sample of 113
        #20 sample_in = 8'b1000_0001;  // input sample of 129
        #20 sample_in = 8'b1001_0001;  // input sample of 145
        #20 sample_in = 8'b1010_0001;  // input sample of 161
        #20 sample_in = 8'b1011_0001;  // input sample of 177
        #20 enable =0;
```

```verilog
        #40 enable =1;
        #20 sample_in = 8'b1100_0001;  // input sample of 193
        #20 sample_in = 8'b1101_0001;  // input sample of 209
        #20 sample_in = 8'b1110_0001;  // input sample of 225
        #20 sample_in = 8'b1111_0001;  // input sample of 241
        #20 sample_in = 8'b1000_0001;  // input sample of 129
        #20; // Wait for some cycles

        $display("Filtered Output: %h", filtered_sample);
        //$finish;
    end

endmodule
```

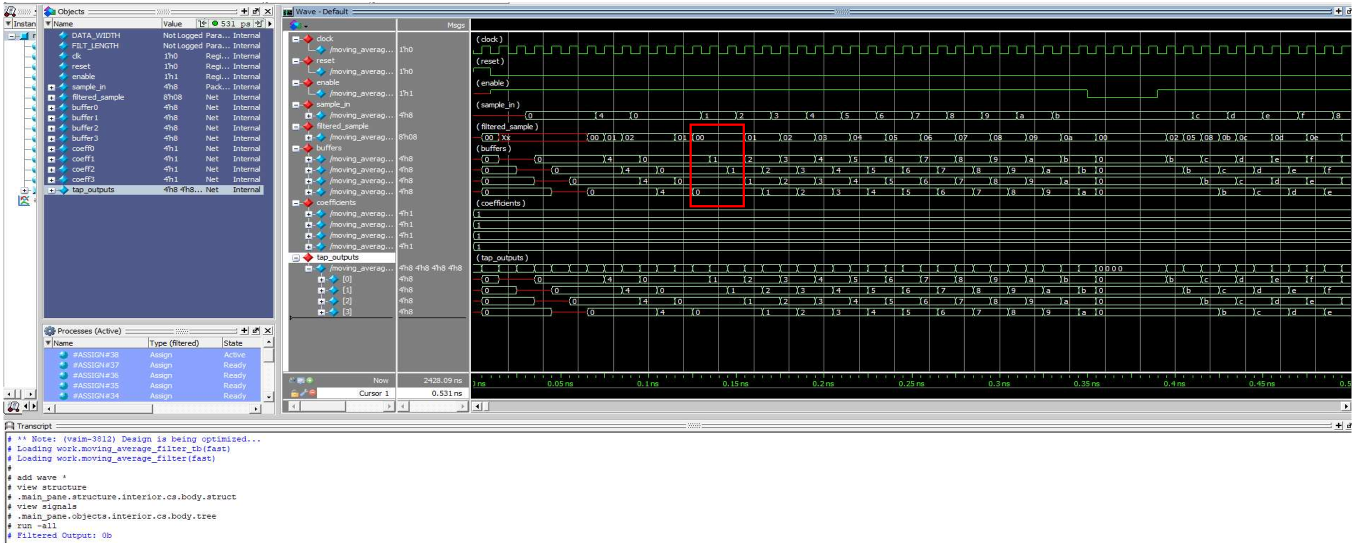
RTL Simulation -from Questa, showing enable=0 to 1, reset 0 to 1, and 4-bit input (sample_in) with output moving average (filtered_sample_out)



```
Objects
Name              Value        525 ps
DATA_WIDTH        Not Logged   Para...  Internal
FILT_LENGTH       Not Logged   Para...  Internal
clk               1'h1         Regi...  Internal
reset             1'h0         Regi...  Internal
enable            1'h1         Regi...  Internal
sample_in         4'h8         Pack...  Internal
filtered_sample   8'h08        Net      Internal
buffer0           4'h8         Net      Internal
buffer1           4'h8         Net      Internal
buffer2           4'h8         Net      Internal
buffer3           4'h8         Net      Internal
coeff0            4'h1         Net      Internal
coeff1            4'h1         Net      Internal
coeff2            4'h1         Net      Internal
coeff3            4'h1         Net      Internal
tap_outputs       4'h8 4'h8... Net      Internal

Processes (Active)
Name          Type (filtered)  State
#ASSIGN#38    Assign           Active
#ASSIGN#37    Assign           Ready
#ASSIGN#36    Assign           Ready
#ASSIGN#35    Assign           Ready
#ASSIGN#34    Assign           Ready

Now      192693.27 ns
Cursor 1 0.525 ns
```

```
Transcript
# ** Note: (vsim-3812) Design is being optimized...
# Loading work.moving_average_filter_tb(fast)
# Loading work.moving_average_filter(fast)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Filtered Output: 0b
```

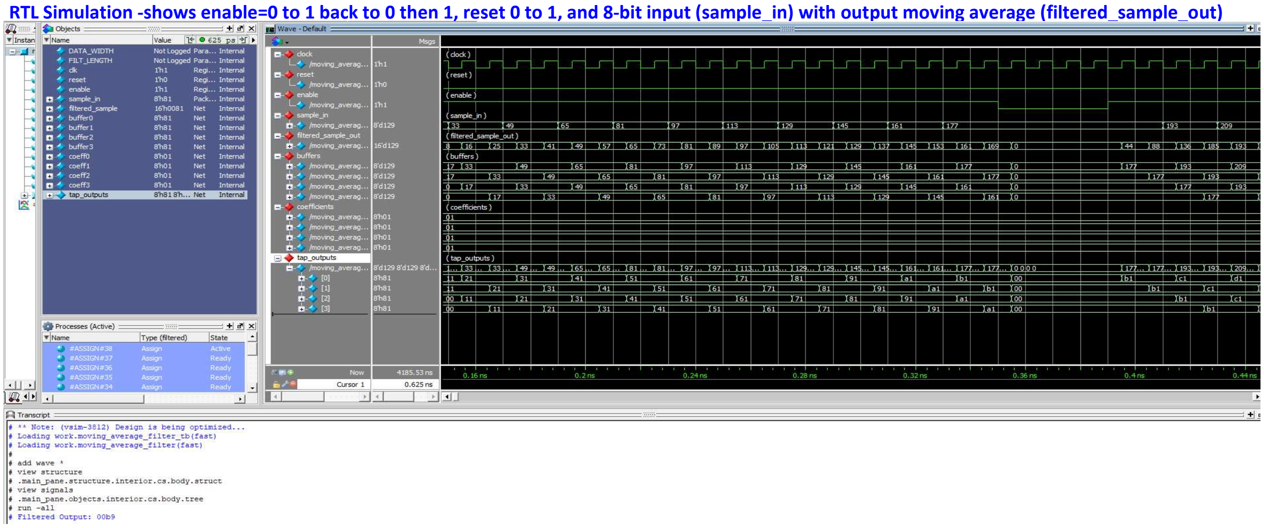
RTL Simulation -shows enable=0 to 1 back to 0 then 1, reset 0 to 1, and 4-bit input (sample_in) with output moving average (filtered_sample_out)



```
Objects
Name              Value        531 ps
DATA_WIDTH        Not Logged   Para...  Internal
FILT_LENGTH       Not Logged   Para...  Internal
clk               1'h0         Regi...  Internal
reset             1'h0         Regi...  Internal
enable            1'h1         Regi...  Internal
sample_in         4'h8         Pack...  Internal
filtered_sample   8'h08        Net      Internal
buffer0           4'h8         Net      Internal
buffer1           4'h8         Net      Internal
buffer2           4'h8         Net      Internal
buffer3           4'h8         Net      Internal
coeff0            4'h1         Net      Internal
coeff1            4'h1         Net      Internal
coeff2            4'h1         Net      Internal
coeff3            4'h1         Net      Internal
tap_outputs       4'h8 4'h8... Net      Internal

Processes (Active)
Name          Type (filtered)  State
#ASSIGN#38    Assign           Active
#ASSIGN#37    Assign           Ready
#ASSIGN#36    Assign           Ready
#ASSIGN#35    Assign           Ready
#ASSIGN#34    Assign           Ready

Now      2428.09 ns
Cursor 1 0.531 ns
```

```
Transcript
# ** Note: (vsim-3812) Design is being optimized...
# Loading work.moving_average_filter_tb(fast)
# Loading work.moving_average_filter(fast)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Filtered Output: 0b
```

RTL Simulation -shows enable=0 to 1 back to 0 then 1, reset 0 to 1, and 8-bit input (sample_in) with output moving average (filtered_sample_out)



```
Objects
Name              Value        625 ps
DATA_WIDTH        Not Logged   Para...  Internal
FILT_LENGTH       Not Logged   Para...  Internal
clk               1'h1         Regi...  Internal
reset             1'h0         Regi...  Internal
enable            1'h1         Regi...  Internal
sample_in         8'h81        Pack...  Internal
filtered_sample   16'h0081     Net      Internal
buffer0           8'h81        Net      Internal
buffer1           8'h81        Net      Internal
buffer2           8'h81        Net      Internal
buffer3           8'h81        Net      Internal
coeff0            8'h01        Net      Internal
coeff1            8'h01        Net      Internal
coeff2            8'h01        Net      Internal
coeff3            8'h01        Net      Internal
tap_outputs       8'h81 8'h... Net      Internal

Processes (Active)
Name          Type (filtered)  State
#ASSIGN#38    Assign           Active
#ASSIGN#37    Assign           Ready
#ASSIGN#36    Assign           Ready
#ASSIGN#35    Assign           Ready
#ASSIGN#34    Assign           Ready

Now      4185.53 ns
Cursor 1 0.625 ns
```

```
Transcript
# ** Note: (vsim-3812) Design is being optimized...
# Loading work.moving_average_filter_tb(fast)
# Loading work.moving_average_filter(fast)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Filtered Output: 00b9
```

Conclusion

Reset low means system active. Reset high means reset to zero. Enable =0 is the same as reset. Enable =1 means system active.

There is a longer disabled period in the second figure above.

filtered_sample_out = moving average of sample_in staggerred in overlapping windows using four buffers (0 thru 3).

Since the output can only be a whole number, results are rounded by truncation. This can be seen in the red box 2nd figure.

In the first two figures 4-bit words were used. In the last Figure 8-bit words were used.