Weighted Moving Average Filter Documentation

Submited by: Amr Hossam
GitHub Repo: https://github.com/amrhossam9/ADI_Assignments

Assignment Overview

implementing a **Weighted Moving Average Filter** using Verilog.

Key Specifications

`H(Z) = [1, 0.5, 0.25, 0.125]`

- **Frequency**: 100 MHz
- **Implementation**: Verilog
- **Deliverables**:
    1. Filter RTL
    2. Filter Testbench

Filter RTL Design

Module: `Filter`

The **Filter** module takes an 8-bit input signal and applies a weighted moving average over the last three samples. The weightings used are powers of two to simplify the design, achieved using bit shifts.

Inputs:

- **x**: 8-bit input signal
- **CLK**: Clock signal
- **RST**: Reset signal (active low)

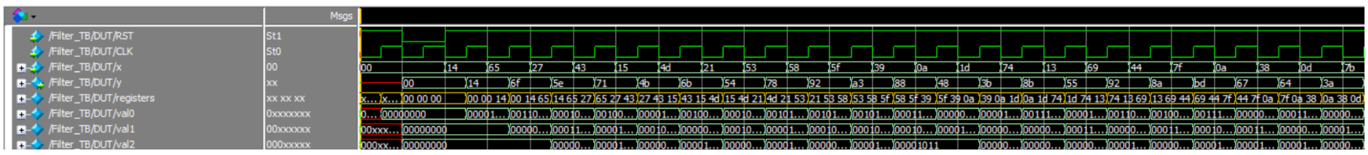
Outputs:

- **y**: 8-bit filtered output

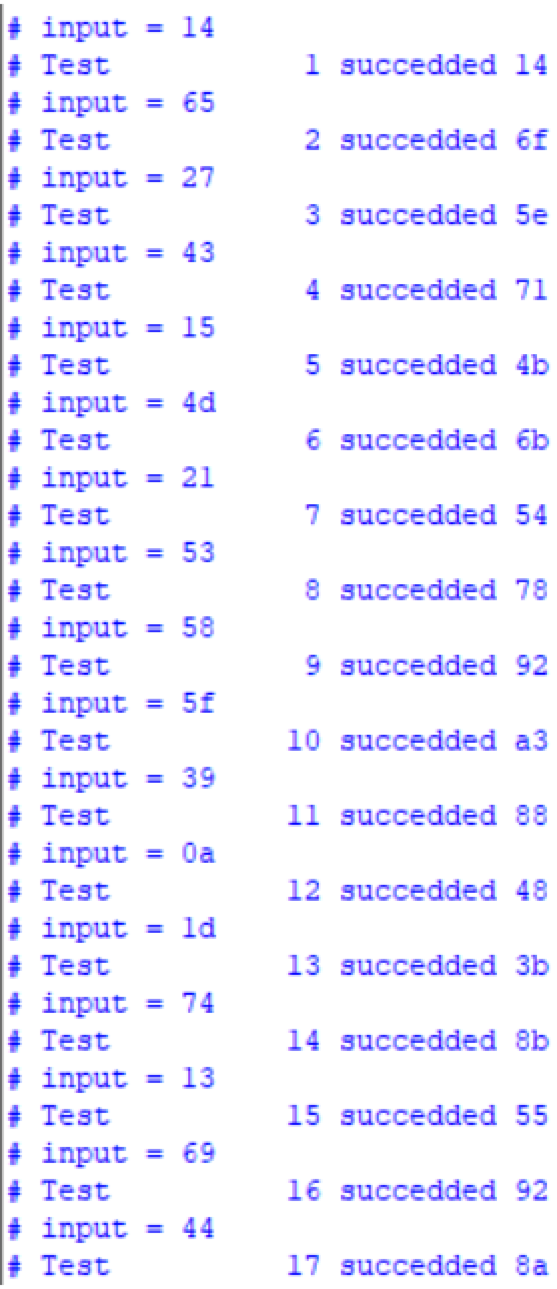
Internal Registers:

- **registers[0:2]**: Stores the last three input samples.

Wave form


Msgs
/Filter_TB/DUT/RST St1
/Filter_TB/DUT/CLK St0
/Filter_TB/DUT/x 00
/Filter_TB/DUT/y xx
/Filter_TB/DUT/registers xx xx xx
/Filter_TB/DUT/val0 0xxxxxxx
/Filter_TB/DUT/val1 00xxxxxx
/Filter_TB/DUT/val2 000xxxxx

Transcript



```
# input = 14
# Test            1 succedded 14
# input = 65
# Test            2 succedded 6f
# input = 27
# Test            3 succedded 5e
# input = 43
# Test            4 succedded 71
# input = 15
# Test            5 succedded 4b
# input = 4d
# Test            6 succedded 6b
# input = 21
# Test            7 succedded 54
# input = 53
# Test            8 succedded 78
# input = 58
# Test            9 succedded 92
# input = 5f
# Test           10 succedded a3
# input = 39
# Test           11 succedded 88
# input = 0a
# Test           12 succedded 48
# input = 1d
# Test           13 succedded 3b
# input = 74
# Test           14 succedded 8b
# input = 13
# Test           15 succedded 55
# input = 69
# Test           16 succedded 92
# input = 44
# Test           17 succedded 8a
```

Verilog Code:

```verilog
module Filter (
    input  wire [7:0] x,
    input  wire CLK, RST,
    output reg  [7:0] y
);

    reg [7:0] registers [2:0];
    wire [7:0] val0, val1, val2;

    assign val0 = registers[0] >> 1;
    assign val1 = registers[1] >> 2;
    assign val2 = registers[2] >> 3;

    always @(posedge CLK or negedge RST)
    begin
        if(!RST)
        begin
            registers[0] <= 0;
            registers[1] <= 0;
            registers[2] <= 0;
            y <= 0;
        end
        else
        begin
            y <= (x + val0 + val1 + val2);
            registers[2] <= registers[1];
            registers[1] <= registers[0];
            registers[0] <= x;
        end
    end
endmodule
```

Testbench for Filter

Testbench Code:

```
module Filter_TB ();

    reg [7:0] x_TB;
    reg CLK_TB, RST_TB;
    wire [7:0] y_TB;

    parameter CLK_PERIOD = 10;
    parameter N = 100;
    parameter DATA_LENGTH = 8;
    integer operations;

    reg [DATA_LENGTH - 1:0] Test_Inputs [N - 1:0];
    reg [DATA_LENGTH - 1:0] Expec_Outs [N - 1:0];

    initial begin
        $dumpfile("Filter_DUMP.vcd") ;
        $dumpvars;

        $readmemh("input.txt", Test_Inputs);
        $readmemh("output.txt", Expec_Outs);

        initialize();

        for (operations = 0; operations < N; operations = operations + 1)
        begin
            do_operation(Test_Inputs[operations], operations);
            checkout(Expec_Outs[operations], operations);
        end

        #100;
        $finish;
    end

    task initialize;
    begin
        CLK_TB = 0;
        x_TB = 0;
        rest();
    end
    endtask

    task rest;
    begin
        RST_TB = 'b1;
        #CLK_PERIOD;
        RST_TB = 'b0;
        #CLK_PERIOD;
        RST_TB = 'b1;
    end
    endtask

    task do_operation;
    input reg [DATA_LENGTH - 1:0] INPUT_TEST;
    input integer operation_number;
    begin
        x_TB = INPUT_TEST;
        #CLK_PERIOD;

        $display("input = %h", INPUT_TEST);
    end
    endtask

    task checkout;
    input reg [DATA_LENGTH - 1:0] expected_output;
    input integer operation_number;
    reg [DATA_LENGTH - 1:0] general_output;
    begin
        general_output = y_TB;

        if (expected_output == general_output)
        begin
            $display("Test %d succedded %h", operation_number + 1,general_output);
        end
        else
        begin
            $display("Test %d failed %h", operation_number + 1,general_output);
        end
    end
    endtask

    always #(CLK_PERIOD/2) CLK_TB = ~CLK_TB;

    Filter DUT(
        .x(x_TB),
        .CLK(CLK_TB),
        .RST(RST_TB),
        .y(y_TB)
    );

endmodule
```