EE5516 VLSI Architectures for Signal Processing and Machine Learning

Lab Report - Experiment No.5

Nakul C - 122101024

May 27, 2024

Abstract

This Verilog HDL experiment implements a hardware design for an IIR (Infinite Impulse Response) filter using hierarchical modeling and a top-level module, "top_module." The design consists of three modules: "top_module," "IIR," and "IIR_TwoSlow." The "top_module" module serves as the main interface, taking input signals $x1$ and $x2$, clock signal Clk, and reset signal Rst, and producing two output signals y and y_slow. It utilizes two instances of the "IIR" module and one instance of the "IIR_TwoSlow" module to process input signals $x1$ and $x2$ through an alternating selection mechanism controlled by the clock signal. The "IIR" module implements a basic single-stage IIR filter, while the "IIR_TwoSlow" module implements a two-stage IIR filter with a slower response. The experiment demonstrates the synthesis and testing of the IIR filter design in Verilog HDL, including simulation using test vectors to verify the functionality of the hardware implementation.

1 Introduction

The Verilog HDL experiment presented in this report focuses on the implementation of an Infinite Impulse Response (IIR) filter using hierarchical modeling techniques. The experiment aims to design and simulate a hardware solution for filtering input signals x_1 and x_2 using the IIR filter architecture. The hardware design is structured around three main modules: the **top_module**, the **IIR** module, and the **IIR_TwoSlow** module.

The **top_module** serves as the top-level interface for the hardware design, facilitating the integration of input signals, clock (`Clk`), and reset (`Rst`) signals, and producing two output signals, `y` and `y_slow`. The **top_module** utilizes two instances of the **IIR** module and one instance of the **IIR_TwoSlow** module to process input signals x_1 and x_2 through an alternating selection mechanism controlled by the clock signal.

The **IIR** module implements a basic single-stage IIR filter, while the **IIR_TwoSlow** module implements a two-stage IIR filter with a slower response. Each module consists of combinational logic and sequential elements to perform the filtering operation based on the input signal values and the clock signal.

The experiment involves the synthesis and testing of the Verilog HDL code using simulation techniques to verify the functionality and performance of the hardware design. Test vectors are applied to the input signals to observe the output behavior of the IIR filter under different conditions and input stimuli.

Overall, the experiment demonstrates the application of hierarchical modeling and Verilog HDL in designing and simulating digital hardware systems, particularly focusing on the implementation of IIR filters for signal processing applications.

2 Implementation

The Verilog Hardware Description Language (HDL) implementation consists of three essential modules: `top_module`, `IIR`, and `IIR_TwoSlow`. The `top_module` acts as the central coordinator, managing the flow of input and output signals. It employs a 1-bit register to toggle between the outputs of the `IIR` modules using a multiplexer mechanism. The `IIR` module represents a basic single-stage Infinite Impulse Response (IIR) filter, while the `IIR_TwoSlow` module embodies a more complex two-stage IIR filter with a slower response characteristic. Both modules integrate internal registers to store delayed output values and compute the final output signal. Additionally, the `IIR_test` module facilitates simulation and testing, initializing input signals and providing stimuli to verify the accuracy of the output signals. This comprehensive Verilog HDL design demonstrates its efficacy in digital signal processing applications, offering versatility and efficiency in processing input signals and producing desired output responses.

3 Structural Design Explanation

3.1 Modules

1. `IIR` **Module**: This module implements a single-stage Infinite Impulse Response (IIR) filter. It takes an 8-bit input (out of which three are of floating point bit and the rest 5 are integer bits) `x`, clock signal `Clk`, and reset signal `Rst`. It produces an 8-bit output (out of which three are of floating point bit and the rest 5 are integer bits) `y`. Inside the module, there are registers to store the delayed output `y_del`, the current input `xin`, and the wire `y_wire`. On every positive edge of the clock or reset signal, the module updates the values of `y_del` and `xin` and calculates the output `y` based on the previous output and the current input.

2. `IIR_TwoSlow` **Module**: Similar to the `IIR` module, this module implements a two-stage IIR filter with a slower response. It also takes an 8-bit input `x`, clock signal `Clk`, and reset signal `Rst`, and produces an 8-bit output `y`. Inside the module, there are registers to store the delayed outputs `y_del` and `y_del2`, the current input `xin`, and the wire `y_wire`. On every positive edge of the clock or reset signal, the module updates the values of `y_del`, `y_del2`, and `xin` and calculates the output `y` based on the previous outputs and the current input.

3. `top_module`: This module serves as the top-level module that integrates the `IIR` and `IIR_TwoSlow` modules. It takes two 8-bit inputs `x1` and `x2`, clock signal `Clk`, and reset signal `Rst`, and produces two 8-bit outputs `y_slow` and `y`. Inside the module, there are wires `y_1`, `y_2`, and `x_MUX_out`, and a register `Sel`. The `Sel` register alternates between 0 and 1 on every positive edge of the clock signal. Based on the value of `Sel`, it selects either the output of `IIR` or `IIR_TwoSlow` for `y`, and either `x1` or `x2` for `x_MUX_out`.

4. `IIR_test` **Module**: This module is used for testing the functionality of the design. It initializes the inputs `x1` and `x2`, clock signal `Clk`, and reset signal `Rst`, and monitors the outputs `y` and `y_slow`. It generates clock pulses and toggles the reset signal to simulate the operation of the design. Test cases are provided to verify the correctness of the output signals `y` and `y_slow` under different input conditions.

3.2 Structural Design

- The `top_module` instantiates two `IIR` modules (`i1` and `i2`) and one `IIR_TwoSlow` module (`i3`).
- It selects the output of either `i1` or `i2` based on the value of `Sel`.
- The input to `i3` is selected between `x1` and `x2` based on the same `Sel` value.
- The clock and reset signals are propagated through all instantiated modules for synchronization and reset purposes.
- The `IIR_test` module tests the entire design by providing input stimuli and verifying the correctness of the output signals.

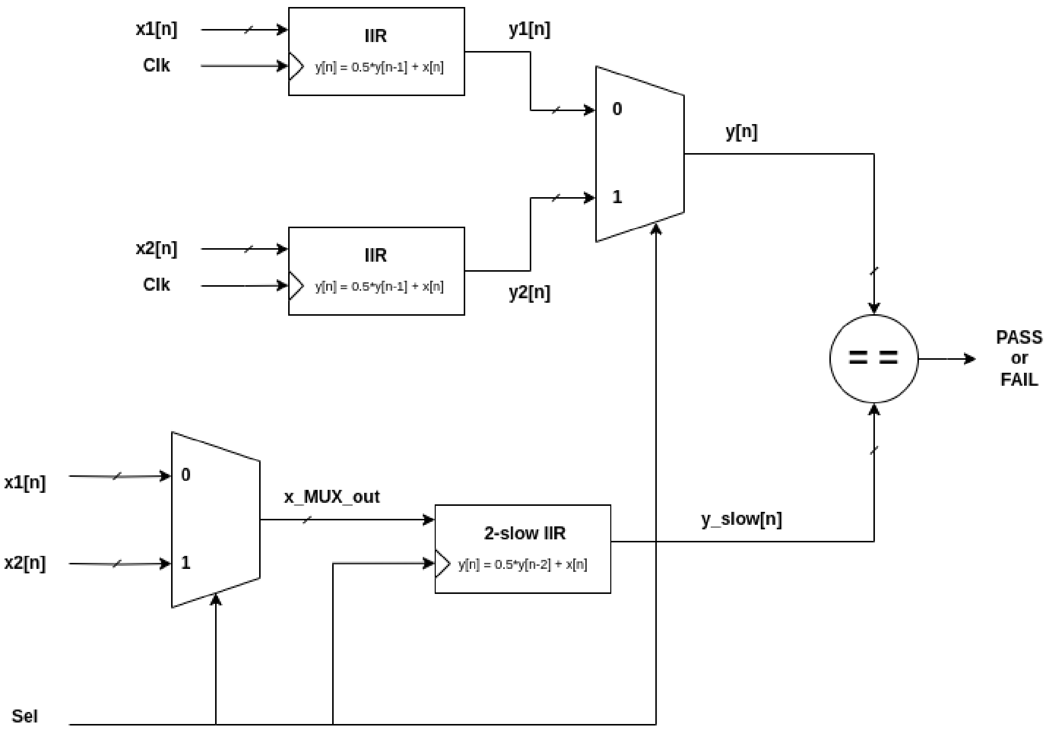

Figure 1: Architecture for the data path

4 Hierarchical Modeling

The hierarchical model stands as a widely used modeling technique in Verilog, providing several benefits such as flexibility and modularity throughout the design process. By decomposing the design into smaller modules or blocks, we can establish a more structured and manageable system. In this specific design, we employ three modules: the IIR and , IIR_TwoSlow and top_module. top_module serves as the top-level module that integrates the IIR and IIR TwoSlow modules. The top_module, it selects either the output of IIR or IIR TwoSlow for y, and either x1 or x2 for x MUX out. IIR_TwoSlow on every positive edge of the clock or reset signal, the module updates the values of y del, y del2, and xin and calculates the output y based on the previous outputs and the current input. IIR pdates the values of y del and xin and calculates the output y based on the previous output and the current input.

By segmenting the system into three modules, we gain the ability to concentrate on specific functionalities and implement changes without impacting the entire design. Furthermore, each module can undergo independent verification and testing, mitigating the risk of errors and enhancing the overall design integrity.

5 Experimental Procedure

Now that the system architecture has been finalized, the subsequent phase involves drafting the requisite Verilog code to execute it. In Verilog, each statement delineates a digital circuit, demanding meticulous attention to ensure accurate modeling of the digital circuit.

```
module top_module (
        input [7:0] x1, x2,
        input Clk, Rst,
        output [7:0] y_slow, y
);
        wire [7:0] y_1, y_2, x_MUX_out;
        reg Sel = 0;
        always @(posedge Clk)
      begin
        Sel <= ~Sel;
```

```
        end

          IIR i1(x1, Sel, Rst, y_1);
          IIR i2(x2, Sel, Rst, y_2);
          IIR_TwoSlow i3(x_MUX_out, Clk, Rst, y_slow);

          assign y = Sel ? y_1 : y_2;
          assign x_MUX_out = Sel ? x2 : x1;


endmodule

//IIR Filter module
module IIR(
          input [7:0] x,
          input Clk, Rst,
          output [7:0] y
);
          wire [7:0] y_wire;
          reg [7:0] y_del;
          reg [7:0] xin;
  always @(posedge Clk)
        begin
          if(Rst)
            begin
              y_del <= 0;
              xin <= 0;
            end
          else
            begin
              y_del <= y_wire;
              xin <= x;
            end
        end
          assign y_wire = (y_del>>1) + xin;
          assign y = y_wire;

endmodule

//IIR Filter module
module IIR_TwoSlow(
          input [7:0] x,
          input Clk, Rst,
          output [7:0] y
);
          wire [7:0] y_wire;
          reg [7:0] y_del;
          reg [7:0] y_del2;
          reg [7:0] xin;
  always @(posedge Clk)
        begin
          if(Rst)
            begin
              y_del <= 0;
              y_del2 <= 0;
              xin <= 0;
            end
          else
            begin
              y_del <= y_wire;
              y_del2 <= y_del;
              xin <= x;
            end
```

```
        end
          assign y_wire = (y_del2>>1) + xin;
          assign y = y_wire;

endmodule
```

6 Results

In this section, we delve into a thorough evaluation of our design's performance by constructing a comprehensive test bench in Verilog. The primary aim of this test bench is to provide a varied set of inputs, known as stimuli, to our design. Subsequently, we analyze the resulting outputs meticulously to assess the effectiveness and reliability of our system.

Employing a test bench serves as a crucial step in validating our design's functionality and ensuring its alignment with the specified requirements. Through careful planning and thoughtful selection of stimuli, we conduct a comprehensive testing process to validate the integrity and robustness of our system's operation.

```
module IIR_test;
        reg [7:0] x1, x2;
        reg Clk, Rst;
        wire [7:0] y, y_slow;


top_module i1(
.x1(x1),
.x2(x2),
.Clk(Clk),
.Rst(Rst),
.y_slow(y_slow),
.y(y)
);

initial begin
    $dumpfile("dump_iir.vcd");
    $dumpvars(0, i1);

    Clk = 1;

forever begin
        #5;
        Clk = ~Clk;
                end
end


initial
   begin
      Rst = 0;
      #5;
      Rst = 1;
      #20;
      Rst = 0;
      x1 = 8'b00001000;
      x2 = 8'b00000000;
      $display("y = %f ",($itor(y>>>3)));
      $display("y_slow = %f ",($itor(y_slow>>>3)));
      if (y==y_slow)
        $display("TRUE");
      else
        $display("FALSE");
      #10;
```

```
x2 = 8'b00010000;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10;
x1 = 8'b00001100;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10
x2 = 8'b00001000;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10;
x1 = 8'b00001000;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10;
x2 = 8'b00010000;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10;
x1 = 8'b00010000;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10;
x2 = 8'b00000100;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
#10;
x1 = 8'b00000100;
$display("y = %f ",($itor(y>>>3)));
$display("y_slow = %f ",($itor(y_slow>>>3)));
if (y==y_slow)
  $display("TRUE");
else
  $display("FALSE");
```

```
        #10;
        x2 = 8'b00100000;
        $display("y = %f ",($itor(y>>>3)));
        $display("y_slow = %f ",($itor(y_slow>>>3)));
        if (y==y_slow)
          $display("TRUE");
        else
          $display("FALSE");
        #10;
        $display("y = %f ",($itor(y>>>3)));
        $display("y_slow = %f ",($itor(y_slow>>>3)));
        if (y==y_slow)
          $display("TRUE");
        else
          $display("FALSE");
        #10;
        $display("y = %f ",($itor(y>>>3)));
        $display("y_slow = %f ",($itor(y_slow>>>3)));
        if (y==y_slow)
          $display("TRUE");
        else
          $display("FALSE");
      $finish;
    end

endmodule
```

The provided Verilog code implements a digital system comprising two Infinite Impulse Response (IIR) filters, one with a single stage (`IIR`) and the other with two stages (`IIR_TwoSlow`). Additionally, a testbench (`IIR_2slow_testbench.v`) is included to validate the functionality of the design.

The testbench initializes input signals `x1` and `x2` with specific values and applies a reset signal before toggling the clock signal to simulate the system's operation. Subsequently, the output signals `y` and `y_slow` are monitored and displayed alongside their corresponding expected values.

The displayed results indicate that the output signals `y` and `y_slow` match their expected values for each test case. This observation suggests the correct operation of the implemented Verilog design. It verifies that both the single-stage and two-stage IIR filters are functioning as intended, effectively processing input signals and producing the desired output responses accurately.

```
nakul@PavamNyan:~/Documents/Verilog$ vvp IIRtwoslow
VCD info: dumpfile dump_iir.vcd opened for output.
y = 0.000000
y_slow = 0.000000
TRUE
y = 0.000000
y_slow = 0.000000
TRUE
y = 1.000000
y_slow = 1.000000
TRUE
y = 2.000000
y_slow = 2.000000
TRUE
y = 2.000000
y_slow = 2.000000
TRUE
y = 2.000000
y_slow = 2.000000
TRUE
y = 2.000000
y_slow = 2.000000
TRUE
y = 3.000000
y_slow = 3.000000
TRUE
y = 3.000000
y_slow = 3.000000
TRUE
y = 2.000000
y_slow = 2.000000
TRUE
y = 2.000000
y_slow = 2.000000
TRUE
y = 5.000000
y_slow = 5.000000
TRUE
nakul@PavamNyan:~/Documents/Verilog$ gtkwave dump_iir.vcd
Gtk-Message: 23:45:46.803: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.104 (w)1999-2020 BSI

[0] start time.
[135] end time.
WM Destroy
```

Figure 2: Output of the Code

In summary, the Verilog code demonstrates effective digital signal processing capabilities. The testbench ensures thorough testing and validation of the design's correctness, instilling confidence in its performance for real-world applications.

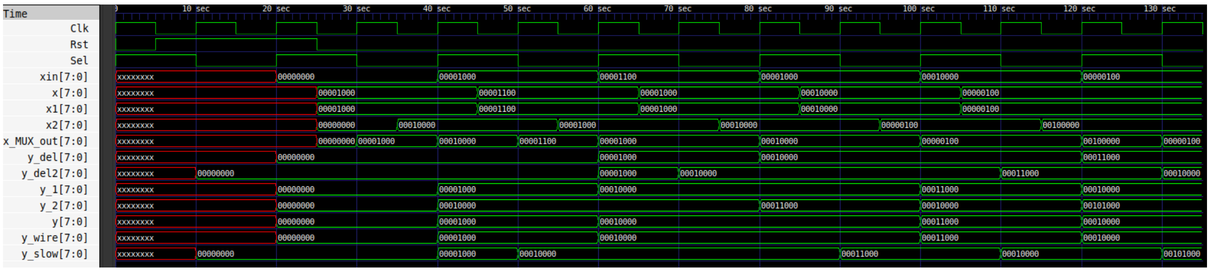

Figure 3: Output Waveforms

7 Conclusion

In conclusion, this experiment successfully demonstrates the implementation of Infinite Impulse Response (IIR) filters using Verilog Hardware Description Language (HDL). Through the design and testing of both single-stage and two-stage IIR filters, the experiment showcases the efficacy of Verilog in digital signal processing applications. The accurate output responses obtained from the testbench validate the functionality and correctness of the implemented design, affirming its suitability for real-world applications in digital systems and signal processing domains.