PBL#5 Report: AND-OR-Reg Sequential Circuit

Name: Goutam Krapa
ID: 2215855

1. Introduction

In the following design project PBL5, our goal is to implement a and or sequential circuit with a register. We must follow the block diagram in the Figure 1 we have. The circuit has four 4-bit inputs, a, b, c and d. The Circuit performs and operation on a and b making it (a&b) and also and operation is performed on c and d making it (c&d) . After this level of circuit, the outputs of each and operation are taken as input for the or gate where “e” is the output for or gate. Thus, making it (a&b) | (c&d). The signal e is given the Register and based on the reset signal “rst” and clock signal “clk” output f is given.

A register is very important in the circuit. It works with two control signals: CLK and RST. CLK is a clock signal that makes the register work on time period. RST is a reset signal is the control signal which controls the output f. If reset signal RST=0, it makes "f" become the value 1'h0. But if RST =1, then "f" shows the output signal.Based on the

This report provides a comprehensive explanation of the design process, design code, test bench code, and key comments.

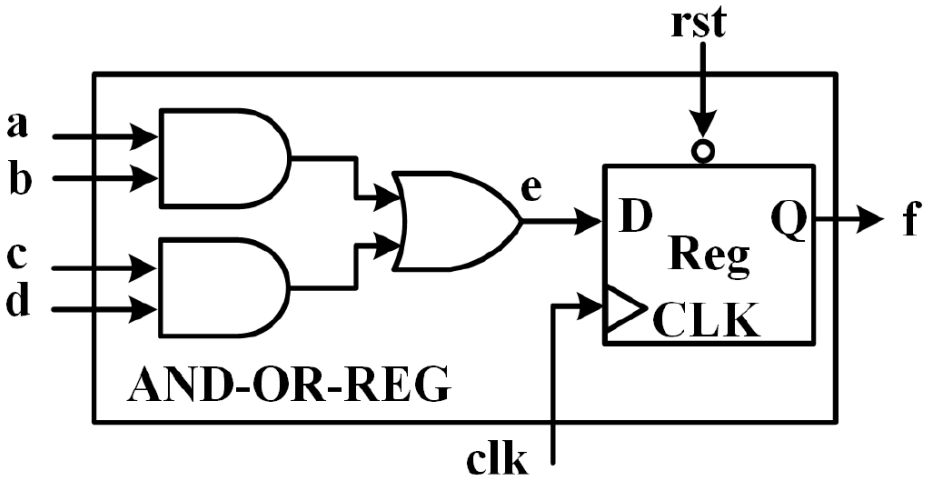

Figure 1: Design Structure of AND-OR-REG Circuit

2. Verilog HDL Design Code

2.1. And_or_reg.v

```
//module defining the and or  with register circuit
module and_or_reg (
input clk,
input rst,
```

```
input a ,
input b,
input c,
input d,
output reg f
//if wire or reg not mentioned its default as wire
);

// Internal signal to hold the combinational logic result
// Combinational logic: AND and OR gates
wire e = (a & b) | (c & d);
//OR
//wire e;
//wire e = (a & b) | (c & d);
//assign e = (a & b) | (c & d);

  // Sequential logic: Register with asynchronous reset
  //always @(posedge clk or negedge rst) begin
  always @(posedge clk, negedge rst) begin
    if (~rst) begin                                          // or !rst
      f <= 1'b0;
                    end       // Asynchronous reset (active-low)
    else begin
      f <= e;
                    end       // On clock edge, store the value of e
  end

endmodule
```

2.2. tb_and_or_reg.v

```
// `timescale 1ns/1ps // Uncomment for a specific timescale
// Defining the test bench module
//Test bench module
module tb_and_or_reg();

// Inputs to the circuit
  reg a, b, c, d;
// Clock and Reset signals
  reg clk, rst;
```

```
// Output f
  wire f;

  // Instantiate the design under test (DUT)
  and_or_reg uut (       .clk(clk),
                                          .rst(rst),
                                          .a(a),
                                          .b(b),
                                          .c(c),
                                          .d(d),
                                          .f(f)
  );

  // Clock generation: toggle every 5 time units
  always #5 clk = ~clk;

  // Testbench logic
  initial begin
    // Initialize signals
    clk = 1'b0; rst = 1'b0;
    a = 1'b0; b = 1'b0; c = 1'b0; d = 1'b0;
    // Apply reset
    #10 rst = 1'b1;
    // Apply test cases
    #10; a = 1'b1; b = 1'b1; c = 1'b0; d = 1'b0;  // Test case 1
    #10; a = 1'b0; b = 1'b1; c = 1'b0; d = 1'b1;  // Test case 2
    #10; a = 1'b0; b = 1'b1; c = 1'b1; d = 1'b1;  // Test case 3
    #10; a = 1'b0; b = 1'b1; c = 1'b0; d = 1'b1;  // Test case 4
  end

endmodule
```

2.3. TCL Script: run

```
#!/bin/csh -f
##############################################################################
##
# simulation top script by Goutam Krapa 2215855
# change modelsim in this file dir and run "do this_file.do"
# project dir ---+---> hdl source dir
#                +---> sim script dir
#
```

```
##############################################################################
##

# check if current dir has modelsim config file
set has_config  [file exists modelsim.do]
# config modelsim
if {$has_config==1} { do modelsim.do  ; }

echo "+====================================="
echo "| Creat Lib Work soc                  "
echo "+====================================="
vlib work
vmap work work

echo "+====================================="
echo "| Complile RTL Code of soc            "
echo "+====================================="

vlog -f ../filelist/filelist.f
##or##vlog -v ../dut/and_or.v ../tb/tb_and_or_reg.v
##vlog -v ../dut/and_or_reg.v ../tb/tb_and_or_reg.v

echo "+====================================="
echo "| Compiler Pass                       "
echo "| Being to Run Simulation             "
echo "+====================================="
vsim work.tb_and_or_reg -t 1ns

################## ################### mus
####################################
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/clk
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/a
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/b
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/c
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/d
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/rst
add wave -noupdate -format logic -radix hexadecimal tb_and_or_reg/f

add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/clk
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/a
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/b
```

```
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/c
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/d
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/rst
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/e
add wave -noupdate -format logic -radix hexadecimal -color blue tb_and_or_reg/uut/f
run 200ns
```

2.4. Project code

Please find the zip file of the code in the attachment.

PBL-Report-GoutamKrapa-2215855-1.zip

3. Simulation Waveform

The below following simulation waveforms are the resultant simulations generated from model sim software a tool used for simulation. We can validate that the output f is generated as 0 when the reset signal rst=0 and generates the result f based on inputs a ,b , c and d when rst= 1.

After the rst is 1 and the next rising edge of the clock “clk” we see the output f. The implementation of register is shown at 20ns as we obtain the result “e”=1’b1 at 20ns but the result f is displayed as 1’b1 after 5ns of “e” being displayed

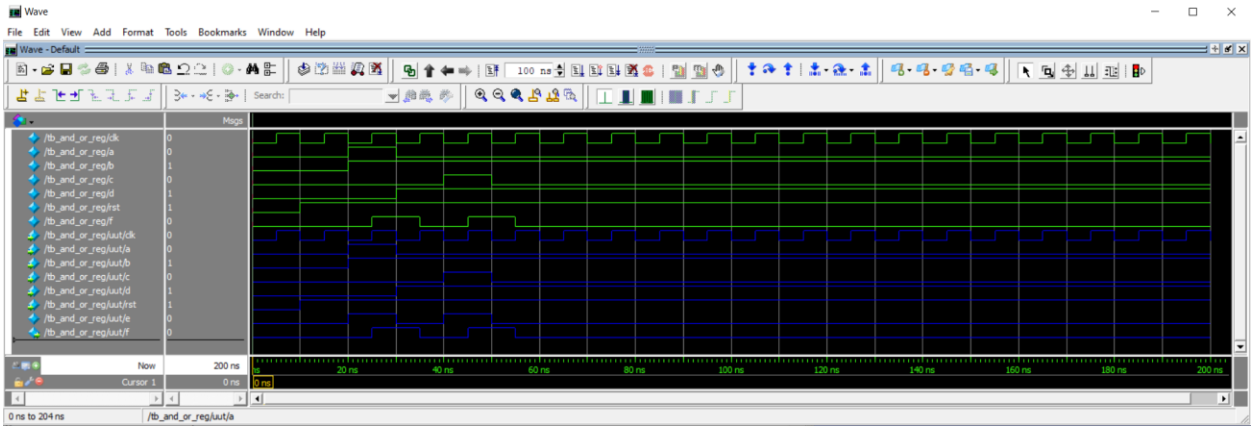

Figure 2: Simulation waveform for the and_or_reg.v with tb_and_or_reg.v

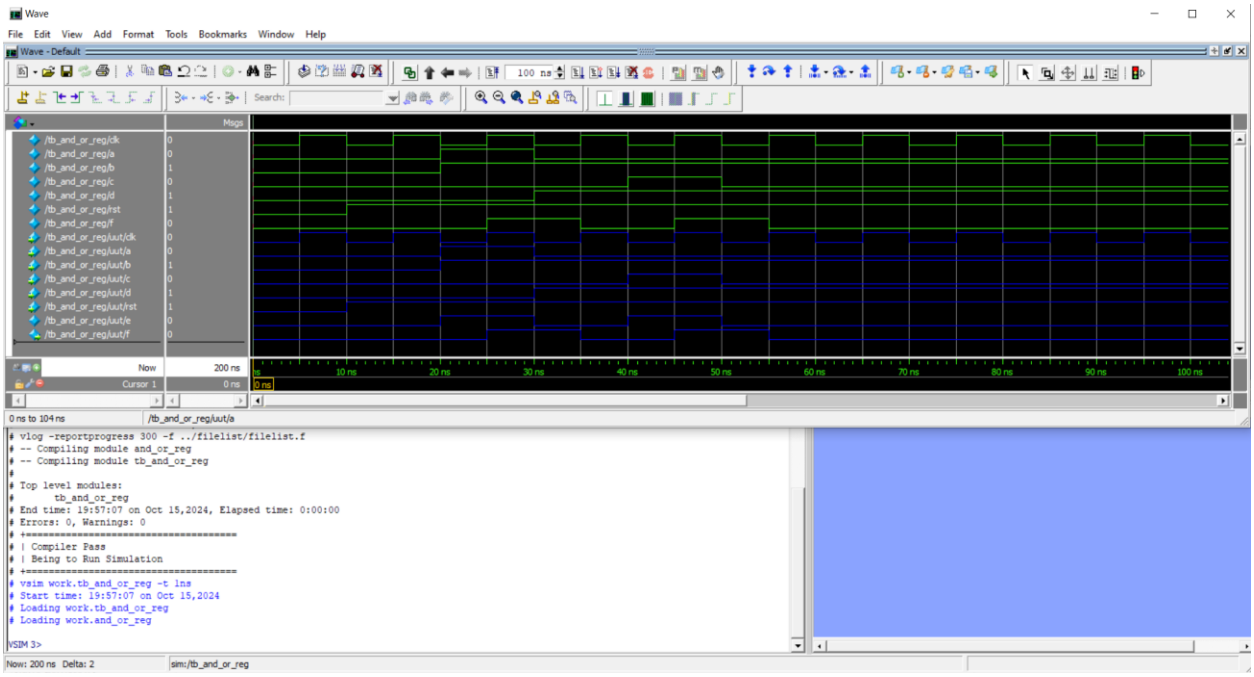

Figure 3: Simulation waveform for the and_or_reg.v with tb_and_or_reg.v

4. Comments and Conclusion

4.1. Design project

Our project involves two main components And-Or Sequential circuit and a Register .Our Design code Test Becnh code and TCL script is used to implement a and-or sequential circuit with a register as given in the Figure 1. The input signals a, b, c, and d are 1-bit values which are given to the and-or sequential circuit with a register. The output from the sequential circuit is given to the register which is a 1-bit signal. Based on the block diagram we use rst, negedge, clk and posedge.

The Register incorporated in the given circuit additionally has two necessary control signals named CLK (clock signal) and RST (reset signal). The reset signal ensures the output is set to 1'b0 i.e. when the reset is true (RST=0) the output value c is 1'b0 and if reset is True the output value f is the 1-bit (a&b)|(c&d) value.

The design of the circuit based on above points is in the and_or_reg.v file and the test bench used to define the test cases is in tb_ and_or_reg.v file where as the TCL script used to generate simulated waveforms is in the run file.

4.2. Design Challenges

- Timing Control and Edge Sensitivity: The circuit is designed with and-or sequential circuit with register logic, responsible for result "f" signal. One of the main challenges faced in this PBL is to manage the edge triggering behavior of the register. It needs to respond correctly to both the rising edge of the clock signal and falling edge of the reset signal.

- Synchronization of Signals: During simulation we need to ensuring that all inputs a, b, c, and d are changed at the correct time while maintaining synchronization with the clock and reset.
- Reset Logic: The reset logic must be implemented such that a consistent reset operation is performed and it resets the register output to 1'h0 when activated (RST=0).

4.3. Constraint

- **Simple Logic Structure:** The given circuit design is constrained by the combinational blocks of 2 AND gates, 1 OR gate and 1 Register. This limits the functionality of the design as it may not be scalable.
- **1-Bit Signals:** The inputs a, b, c, and d and the output f are all 1-bit signals, which are a constrain for other designs that may need multi-bit inputs or outputs.

4.4. Conclusion

On cross validating the generated simulation waveform we can conclude that with the help of Verilog HDL code we have translated the design specification into logical functional code and through simulation we displayed the simulated waveform. Essential control signals such as A, B, C, D, E, F, CLK and RST have been monitored. Through simulation of different test cases the design code and the block diagram design has been verified ensuring correctness of the scenario.

Thus, we have usefully implemented the design for AND-OR sequential circuit with a register while the register stored the output at the correct clock edge, allowing for the sequential behavior to function as expected and allowing control over output by utilizing the reset signal and clock signal. The register's responds to both the falling-edge reset (posedge) and rising-edge clock (negedge) works correctly in the simulation, resetting the output when the reset was active (i.e 1’b1) and updating it with the combinational logic when the reset was inactive (i.e 1’b0). The follow approach of a modular and controlled approach to perform logical operations of input.