Md Maniruzzaman

ID: 155009

**EE461-HW#03**

***Q1: Answer***

|  |  |
| --- | --- |
| **testbench** | **design** |
| `timescale 1ns/1ps  module testbench;  reg a\_r;  reg b\_r;  reg c\_r;  wire f\_w;  designModule u(  .a\_i(a\_r),  .ab\_i(b\_r),  .c\_i(c\_r),  .f\_o(f\_w)  );  initial begin  $display("Time(ns) | A | B | C | F");  $display("---------|---|---|---|---");  a\_r = 0; b\_r = 0; c\_r = 0; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 0; b\_r = 0; c\_r = 1; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 0; b\_r = 1; c\_r = 0; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 1; b\_r = 0; c\_r = 0; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 1; b\_r = 1; c\_r = 0; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 1; b\_r = 1; c\_r = 1; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 1'bx; b\_r = 1'b0; c\_r = 1'b1; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  a\_r = 1'bz; b\_r = 1'b1; c\_r = 1'b0; #5; $display("%6t | %b | %b | %b | %b", $time, a\_r, b\_r, c\_r, f\_w);  #5 $finish;  end  endmodule | `timescale 1ns/1ps  module designModule (  input wire a\_i,  input wire ab\_i,  input wire c\_i,  output wire f\_o  );  supply1 Vdd;  supply0 Gnd;  wire n\_ab;  wire p\_ab;  // PMOS Pull-Up  pmos (p\_ab, Vdd, a\_i);  pmos (p\_ab, Vdd, ab\_i);  pmos (f\_o, p\_ab, c\_i);  // NMOS Pull-Down  nmos (n\_ab, Gnd, a\_i);  nmos (n\_ab, Gnd, ab\_i);  nmos (f\_o, n\_ab, c\_i);  pulldown(f\_o);  endmodule |

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

***Q2: Answer***

|  |  |  |
| --- | --- | --- |
| **testbench** | **design** | **udp** |
| `timescale 1ns/1ps  module testbench;  reg sel;  reg d0, d1;  wire y;  mux2to1\_top dut (  .sel(sel),  .d0(d0),  .d1(d1),  .y(y)  );  initial begin  $display("Time(ns) | sel | d0 | d1 | y");  $display("---------|-----|-----|-----|-----");  d0 = 0; d1 = 1;  sel = 0; #5; $display("%8t | %b | %b | %b | %b", $time, sel, d0, d1, y);  sel = 1; #5; $display("%8t | %b | %b | %b | %b", $time, sel, d0, d1, y);  sel = 1'bx; #5; $display("%8t | %b | %b | %b | %b", $time, sel, d0, d1, y);  sel = 1'bz; #5; $display("%8t | %b | %b | %b | %b", $time, sel, d0, d1, y);  #5 $finish;  end  endmodule | `include "udp.sv"  module mux2to1\_top (  input wire sel,  input wire d0,  input wire d1,  output wire y  );  mux2to1\_udp U1 (y, sel, d0, d1);  endmodule | primitive mux2to1\_udp (out, sel, d0, d1);  output out;  input sel, d0, d1;  table  // sel d0 d1 : out  0 0 ? : 0;  0 1 ? : 1;  1 ? 0 : 0;  1 ? 1 : 1;  x ? ? : x;  z ? ? : x;  endtable  endprimitive |

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

***Q3: Answer***

**Longest data-path calculation (general case)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Path label** | **Data path (from input to out)** | **Gate delays along path** | **Total delay** |
| P1 (longest) | A/B => OR => MUX1 => OR => MUX2 | 5 + 8 + 5 + 8 | 26 |
| P2 | C => NOT =>MUX1 => OR => MUX2 | 3 + 8 + 5 + 8 | 24 |
| P3 | C => NOT => MUX1 => NOT => MUX2 | 3 + 8 + 3 + 8 | 22 |
| P4 | (C direct into OR2 only) => OR => MUX2 | 5 + 8 | 13 |
| P5 | D => NOT => MUX2 | 3 + 8 | 11 |

The longest input => output delay = 26 time units (via P1).

# **If D is held constant**

|  |  |  |  |
| --- | --- | --- | --- |
| **D fixed** | **Active longest data path** | **Gate delays** | **Longest delay** |
| D = 0 | A/B => OR => MUX1 => OR => MUX2 | 5 + 8 + 5 + 8 | 26 |
| D = 1 | D => NOT => MUX2  (A/B/C paths are masked at MUX2) | 3 + 8 | 11 |

|  |  |  |
| --- | --- | --- |
| **testbench** | **design** | **udp** |
| module tb\_top\_module;  reg A, B, C, D;  wire out;    top\_module uut (out, A, B, C, D);    initial begin  $dumpfile("mux\_circuit.vcd");  $dumpvars(0, tb\_top\_module);    $display("Time\t| A | B | C | D | OUT");  $display("--------|---|---|---|---|----");  $monitor("%0t\t\t| %b | %b | %b | %b | %b", $time, A, B, C, D, out);    A=0; B=0; C=0; D=0;  #50;    A=1; B=0; C=1; D=0;  #50;    A=0; B=1; C=1; D=1;  #50;    A=1; B=1; C=0; D=1;  #50;    A=0; B=0; C=1; D=0;  #50;    $finish;  end  endmodule | `include "udp.sv"  module top\_module (output out, input A, B, C, D);  wire signal\_AB, signal\_notC, signal\_MUX1\_out;  wire signal\_OR2, signal\_NOT2, signal\_notD;    or #5 or\_gate1(signal\_AB, A, B);  not #3 not\_gate1(signal\_notC, C);  mux2to1 #8 mux1(signal\_MUX1\_out, signal\_AB, signal\_notC, D);  or #5 or\_gate2(signal\_OR2, signal\_MUX1\_out, C);  not #3 not\_gate2(signal\_NOT2, signal\_MUX1\_out);  not #3 not\_gate3(signal\_notD, D);  mux2to1 #8 mux2(out, signal\_OR2, signal\_NOT2, signal\_notD);    endmodule | primitive mux2to1 (out, in0, in1, sel);  output out;  input in0, in1, sel;    table  // in0 in1 sel : out  0 0 ? : 0;  1 1 ? : 1;  0 ? 0 : 0;  1 ? 0 : 1;  ? 0 1 : 0;  ? 1 1 : 1;  x x 0 : x;  x x 1 : x;  endtable  endprimitive |

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

***Q4: Answer***

`include "basicPrimitive.v"

primitive example(a0, a1, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r0, r1);

output a0, a1, r0, r1;

input b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q;

reg a0, a1, r0, r1;

table

// a0 a1 c r0 r1 b d e f g h i j k l m n o p q

0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0;

1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ?;

0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0;

// more rows as needed

endtable

endprimitive

**Explanation of Fixes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Issue** | **Problem in Original** | **Fix Applied** | **Why** |
| Include directive | Used #include | Changed to `include | Verilog uses backtick, not hash, for include directives. |
| Multi-bit outputs | Used [1:0] a, r inside a UDP | Split into single bit a0, a1, r0, r1 | UDPs support scalar outputs only, not vectors. |
| In-out port | c declared as in-out | Changed to input c | UDPs cannot have in-out ports - only input and output. |
| Table format | Had two colons (:) per row | Rewritten with one colon separator between input and output fields (semicolon to end each row) | UDP syntax allows only one colon to separate the input pattern from the output. |
| Nested instantiation | Tried to instantiate another primitive (basicPrimitive u0) inside a UDP | Removed — not allowed inside UDPs | UDPs are atomic; cannot instantiate other modules or primitives. |

***Q5: Answer***

`timescale 1ns/1ps

// UDP: Positive-Edge-Triggered D Flip-Flop with Asynchronous Reset

primitive DFF (q, d, clk, rst);

output q;

reg q;

input d, clk, rst;

table

// clk d rst : q : q+

? ? 1 : ? : 0; // Async reset (active-high)

(01) 1 0 : ? : 1; // Rising-edge clock, capture 1

(01) 0 0 : ? : 0; // Rising-edge clock, capture 0

(0?) ? 0 : ? : -; // Ignore falling edge

(1?) ? 0 : ? : -; // Ignore glitches

? ? 0 : ? : -; // Hold otherwise

endtable

endprimitive

// 3-Bit Ripple Counter: 3 DFF + 2 XOR + 1 NAND + 1 NOT

module rippleCounter(input clk, input rst, output [2:0] out);

wire rst\_bar;

wire n1, n2, n3;

not (rst\_bar, rst);

nand (n1, out[0], out[1]);

xor (n2, n1, ~out[2]);

xor (n3, out[0], out[1]);

DFF dff0(out[0], n3, clk, rst\_bar); // LSB toggles every clk

DFF dff1(out[1], ~out[0], out[0], rst\_bar); // middle bit

DFF dff2(out[2], n2, out[1], rst\_bar); // MSB

endmodule

// TESTBENCH

module tb\_rippleCounter;

reg clk, rst;

wire [2:0] out;

rippleCounter uut (.clk(clk), .rst(rst), .out(out));

// Generate 10 ns clock

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

$dumpfile("ripple\_counter.vcd");

$dumpvars(0, tb\_rippleCounter);

$display("Time(ns)\tclk\trst\tout[2:0]");

$display("-----------------------------------");

$monitor("%0t\t%b\t%b\t%b", $time, clk, rst, out);

rst = 1; // assert reset

#10 rst = 0; // de-assert reset

#200 $finish;

end

endmodule