EXP 11 RAM

```
module ram(
  input clk,
  input write_enable,
  input [9:0] address,
  input [7:0] data_in,
  output reg [7:0] data_out
);
  reg [7:0] ram_block [0:1023];
  always @(posedge clk) begin
    if (write_enable)
      ram_block[address] <= data_in;
    else
      data_out <= ram_block[address];
  end
endmodule
```

TEST BENCH:

```
`timescale 1ns / 1ps
module ram_tb;
reg clk;
reg write_enable;
reg [9:0] address;
reg [7:0] data_in;
wire [7:0] data_out;
ram ram_instance (
  .clk(clk),
  .write_enable(write_enable),
  .address(address),
  .data_in(data_in),
  .data_out(data_out)
);
always #5 clk = ~clk;
initial begin
  clk = 0; write_enable = 0; address = 0;
  data_in = 8'b10101010;
  write_enable = 1;
  address = 10; data_in = 8'b11001100;
  #10 write_enable = 0
address = 10;
  #10; $finish;
end
initial begin
  $monitor("Time=%t, Write Enable=%b, Address=%d, Data In=%h, Data Out=%h",
       $time, write_enable, address, data_in, data_out);
end
endmodule
```

EXP NO :10 ALU

```
module ALU_7(
  input [7:0] A, B,
  input [3:0] ALU_Sel,
  output [7:0] ALU_Out,
  output CarryOut
);
  reg [7:0] ALU_Result;
  wire [8:0] tmp;
  assign ALU_Out = ALU_Result;
  assign tmp = {1'b0, A} + {1'b0, B};
  assign CarryOut = tmp[8];
  always @(*) begin
    case (ALU_Sel)
      4'b0000: ALU_Result = A + B;
      4'b0001: ALU_Result = A - B;
      4'b0010: ALU_Result = A & B;
      4'b0011: ALU_Result = A | B;
      4'b0100: ALU_Result = A ^ B;
      4'b0101: ALU_Result = ~(A & B);
      4'b0110: ALU_Result = ~(A ^ B);
      default: ALU_Result = A + B;
    endcase
  end
endmodule
```

EXP 9: TRAFFIC LIGHT CONTROLLER

```
`define TRUE 1'b1
`define FALSE 1'b0
`define RED 2'd0
`define YELLOW 2'd1
`define GREEN 2'd2
`define s0 3'd0
`define s1 3'd1
`define s2 3'd2
`define s3 3'd3
`define s4 3'd4
`define Y2RDELAY 3
`define R2GDELAY 2

module trafficlightcontroller(hwy, cntry, X, clock, clear);
  output reg [1:0] hwy, cntry;
  input X, clock, clear;
  reg [2:0] state, next_state;
  initial begin
    state = `s0; next_state = `s0;
    hwy = `GREEN; cntry = `RED;
  end
  always @(posedge clock or posedge clear) begin
    if (clear)
      state <= `s0;
    else
      state <= next_state;
  end
  always @(state) begin
    case (state)
      `s0: begin
        hwy = `GREEN; cntry = `RED;
      end
      `s1: begin
        hwy = `YELLOW; cntry = `RED;
      end
      `s2: begin
        hwy = `RED; cntry = `RED;
      end
      `s3: begin
        hwy = `RED;
        cntry = `GREEN;
      end
      `s4: begin
        hwy = `RED;
        cntry = `YELLOW;
      end
```

```
    endcase
  end
  always @(state or X) begin
    case (state)
      `s0: if (X) next_state = `s1; else next_state = `s0;
      `s1: begin
        repeat(`Y2RDELAY) @(posedge clock);
        next_state = `s2;
      end
      `s2: begin
        repeat(`R2GDELAY) @(posedge clock);
        next_state = `s3;
      end
      `s3: if (X) next_state = `s3; else next_state = `s4;
      `s4: begin
        repeat(`Y2RDELAY) @(posedge clock);
        next_state = `s0;
      end
      default: next_state = `s0;
    endcase
  end
endmodule
```

TEST BENCH

```
module testbench_trafficlightcontroller;
reg X, clock, clear;
wire [1:0] hwy, cntry;
trafficlightcontroller u1 (.hwy(hwy), .cntry(cntry), .X(X), .clock(clock), .clear(clear));
always #5 clock = ~clock;
initial begin
  X = 0; clock = 0; clear = 0;
  $display("Time\tX\tclock\tclear\thwy\tcntry");
  $monitor("%d\t%b\t%b\t%b\t%b\t%b", $time, X, clock, clear, hwy, cntry);
  clear = 1;
  X = 1; #10;
  X = 0; #10;
  X = 0; #20;
  X = 0; #10;
  X = 1; #10;
  X = 0; #10;
  X = 0; #10;
  X = 0; #20;
  clear = 0;
  X = 0; #10;
  X = 1; #10;
  $finish;
end
endmodule
```

EXP NO 8: SEQUENCE DETECTOR

```
module sequence_detector(sequence_in, clock, reset, detector_out);
  input clock;
  input reset;
  input sequence_in;
  output reg detector_out;
  parameter zero = 3'b000,
        one = 3'b001,
        onezero = 3'b011,
        onezeroone = 3'b010,
        onezerooneone = 3'b110;
  reg [2:0] current_state, next_state;
  always @(posedge clock or posedge reset) begin
    if (reset == 1)
      current_state <= zero;
    else
      current_state <= next_state;
  end
  always @(current_state, sequence_in) begin
    case (current_state)
      zero: begin
        if (sequence_in == 1)
          next_state = one;
        else
          next_state = zero;
      end
      one: begin
        if (sequence_in == 0)
          next_state = onezero;
        else
          next_state = one;
      end
      onezero: begin
        if (sequence_in == 0)
          next_state = zero;
        else
          next_state = onezeroone;
      end
      onezeroone: begin
        if (sequence_in == 0)
          next_state = onezero;
        else
          next_state = onezerooneone;
      end
      onezerooneone: begin
        if (sequence_in == 0)
          next_state = onezero;
```

```
        else
          next_state = one;
      end
      default: next_state = zero;
    endcase
  end

  always @(current_state) begin
    case (current_state)
      zero: detector_out = 0;
      one: detector_out = 0;
      onezero: detector_out = 0;
      onezeroone: detector_out = 0;
      onezerooneone: detector_out = 1;
      default: detector_out = 0;
    endcase
  end
endmodule
```

EXP NO 7 : COUNTERS

UP COUNTER

```
module up_counter(
  input wire clk,
  input wire reset,
  output reg [3:0] count
);
  always @(posedge clk or posedge reset) begin
    if (reset)
      count <= 4'b0000;
    else if (count == 4'b1111)
      count <= 4'b0000;
    else
      count <= count + 1;
  end
endmodule
```

TEST BENCH

```
module up_counter_tb;
  reg clk;
  reg reset;
  wire [3:0] count;
  up_counter counter (
    .clk(clk),
    .reset(reset),
    .count(count)
  );
  always #5 clk = ~clk;
  initial begin
    clk = 0;
    reset = 0;
    reset = 1;
    #10 reset = 0;
    #20;
    #20;
    $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    $finish;
  end
endmodule
```

DOWN COUNTER

```
module down_counter(
  input wire clk,
  input wire reset,
  output reg [3:0] count
);
  always @(posedge clk or posedge reset) begin
    if (reset)
      count <= 4'b1111;
    else if (count == 4'b0000)
      count <= 4'b1111;
    else
      count <= count - 1;
  end
endmodule
```

TEST BENCH:

```
module down_counter_tb;
  reg clk;
  reg reset;
  wire [3:0] count;
  down_counter counter (
    .clk(clk),
    .reset(reset),
    .count(count)
  );
  always #5 clk = ~clk;
  initial begin
    clk = 0;
    reset = 0;
    reset = 1;
    #10 reset = 0;
    #20;
    #20;
    $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);
    #10 $display("Time=%0t, Counter=%b", $time, count);

    $finish;
  end
endmodule
```

EXP NO 6: SHIFT REGISTER

SISO

```
module siso(
  input clk,
  input reset,
  input si,
  output so
);
  reg [3:0] shift_register;
  always @(posedge clk or posedge reset) begin
    if (reset)
      shift_register <= 4'b0000;
    else
      shift_register <= {shift_register[2:0], si};
  end
  assign so = shift_register[3];
endmodule
```

TEST BENCH:

```
module siso_tb;
  reg clk;
  reg reset;
  reg si;
  wire so;
  siso uut (
    .clk(clk),
    .reset(reset),
    .si(si),
    .so(so)
  );
  always #5 clk = ~clk;
  initial begin
    clk = 0;
    reset = 0;
    si = 0;
    reset = 1;
    #10 reset = 0;
    si = 1; #10;
    si = 0; #10;
    si = 1; #10;
    si = 0; #10;
    #100;
    $finish;
  end
  always @(posedge clk) begin
    $display("Time = %0t: clk = %b, reset = %b, si = %b, so = %b", $time, clk, reset, si, so);
  end
endmodule
```

PIPO

```
module pipo(
  input clk,
  input reset,
  input [3:0] data_in,
  output reg [3:0] data_out
);
  always @(posedge clk or posedge reset) begin
    if (reset)
      data_out <= 4'b0000;
    else
      data_out <= data_in;
  end
endmodule
```

TEST BENCH:

```
module pipo_tb;
  reg clk;
  reg reset;
  reg [3:0] data_in;
  wire [3:0] data_out;
  pipo uut (
    .clk(clk),
    .reset(reset),
    .data_in(data_in),
    .data_out(data_out)
  );
  always #5 clk = ~clk;
  initial begin
    clk = 0;
    reset = 1;
    data_in = 4'b0000;
    #10 reset = 0;

    #10 data_in = 4'b1110;
    #10 data_in = 4'b0001;
    #10 data_in = 4'b0010;
    #10 data_in = 4'b0100;
    #10 data_in = 4'b1000;

    #100;
    $finish;
  end

  always @(posedge clk) begin
    $display("Time=%0t, data_in=%b, data_out=%b", $time, data_in, data_out);
  end
endmodule
```

SIPO

```
module sipomod (
  input clk,
  input clear,
  input si,
  output reg [3:0] po
);
  always @(posedge clk) begin
    if (clear)
      po <= 4'b0000;
    else
      po <= {po[2:0], si};
  end
endmodule
```

TEST BENCH:

```
module tb_SIPO;
reg clk;
reg reset;
reg si;
wire [3:0] po;
sipo uut (
  .clk(clk),
  .reset(reset),
  .si(si),
  .po(po)
);
always #5 clk = ~clk;
initial begin
  clk = 0;
  reset = 1;
  si = 0;
  #10 reset = 0;
  si = 1; #10;
  si = 0; #10;
  si = 1; #10;
  si = 1; #10;
  si = 0; #10;
  $display("Time=%0t | si=%b | po=%b", $time, si, po);
  #20 $finish;
end

always @(posedge clk) begin
  $display("Time=%0t | clk=%b | si=%b | po=%b", $time, clk, si, po);
end
endmodule
```

PISO

```
module PISO_ShiftRegister (
  input wire clk,
  input wire reset,
  input wire [7:0] parallel_in,
  output wire serial_out
);
  reg [7:0] shift_register;
  always @(posedge clk or posedge reset) begin
    if (reset)
      shift_register <= 8'b0;
    else
      shift_register <= {1'b0, shift_register[7:1]};
  end
  assign serial_out = shift_register[0];
endmodule
```

TEST BENCH:

```
module tb_PISO_ShiftRegister;
reg clk;
reg reset;
reg [7:0] parallel_in;
wire serial_out;

PISO_ShiftRegister PISO_inst (
  .clk(clk),
  .reset(reset),
  .parallel_in(parallel_in),
  .serial_out(serial_out)
);

always #5 clk = ~clk;

initial begin
  clk = 0;
  reset = 1;
  parallel_in = 8'b00000000;
  #10 reset = 0;
  parallel_in = 8'b10101010;
  #10;
  repeat(10) #10 $display("Time=%0t serial_out=%b", $time, serial_out);
  $finish;
end
endmodule
```

EXP NO 5: FLIP FLOPS
T FLIP FLOP

```
module t_flip_flop(
input T,
input clk,
input rst,
output reg Q
);

always @(posedge clk or posedge rst)
begin
if (rst)
Q <= 1'b0;
else if (T)
Q <= ~Q;
end
endmodule
```

TEST BENCH:

```
module testbench_t_flip_flop;
reg T, clk, rst;
wire Q;

t_flip_flop u1 (
  .T(T),
  .clk(clk),
  .rst(rst),
  .Q(Q)
);

always #5 clk = ~clk;

initial begin
  T = 0; clk = 0; rst = 0;
  $display("Time\tT\tclk\trst\tQ");
  $monitor("%0t\t%b\t%b\t%b\t%b", $time, T, clk, rst, Q);

  #10;
  T = 0; #20;       // No toggle
  T = 1; #20;       // Toggle Q
  T = 0; #20;       // Hold Q
  rst = 1; #10;     // Reset Q
  rst = 0; #10;     // Normal operation
  T = 1; #20;       // Toggle again
  $finish;
end
endmodule
```

D FLIP FLOP

```
module D_FlipFlop (
  input wire D,
  input wire CLK,
  input wire CLR,
  output wire Q
);
reg Q_reg;

always @(posedge CLK or posedge CLR) begin
  if (CLR)
    Q_reg <= 1'b0;
  else
    Q_reg <= D;
end

assign Q = Q_reg;
endmodule
```

TEST BENCH:

```
module D_FlipFlop_TB;
reg D, CLK, CLR;
wire Q;
D_FlipFlop UUT (
  .D(D),  .CLK(CLK), .CLR(CLR), .Q(Q)
);
always begin
  #5 CLK = ~CLK;
end
initial begin
  CLK = 0; CLR = 0; D = 0;
  #10; $display("Time=%0t | D=%b | Q=%b", $time, D, Q);
  D = 1;
  #10; $display("Time=%0t | D=%b | Q=%b", $time, D, Q);
  D = 0;
  #10; $display("Time=%0t | D=%b | Q=%b", $time, D, Q);
  D = 1;
  #10;  $display("Time=%0t | D=%b | Q=%b", $time, D, Q);
  CLR = 1;
  #10;
  $display("Time=%0t | CLR=%b | Q=%b", $time, CLR, Q);
  CLR = 0;
  #10;
  $display("Time=%0t | CLR=%b | Q=%b", $time, CLR, Q);
  $finish;
end
endmodule
```

RS FLIP FLOP

```
module rs_flipflop (
input R,
input S,
input clk,
output reg Q
);
always @(posedge clk) begin
if (R)
Q <= 1'b0;
else if (S)
Q <= 1'b1;
end
endmodule
```

TEST BENCH:

```
module testbench_rs_flipflop;
reg R, S, clk;
wire Q;

rs_flipflop u1 (.R(R), .S(S), .clk(clk), .Q(Q));

always #5 clk = ~clk;

initial begin
  R = 0; S = 0; clk = 0;
  $display("Time R S clk Q");
  $monitor("Time=%0t R=%b S=%b clk=%b Q=%b", $time, R, S, clk, Q);

  S = 1; #10;
  clk = 1; #10;
  S = 0; #10;

  R = 1; #10;
  clk = 1; #10;
  R = 0; #10;

  $finish;
end
endmodule
```

JK FLIP FLOPS

```
module jk (
input wire clk,
input wire reset,
input wire j, input wire k,
output reg q, output reg qbar
);
always @(posedge clk or posedge reset) begin
   if (reset) begin
      q <= 1'b0; qbar <= 1'b1;
   end else if (j & ~k) begin
      q <= 1'b1; qbar <= 1'b0;
   end else if (~j & k) begin
      q <= 1'b0;  qbar <= 1'b1;
   end else if (j & k) begin
      q <= ~q; qbar <= ~qbar;
   end
end
endmodule
```

TEST BENCH :

```
module stimulus;
reg clk, reset, j, k;
wire q, qbar;
jk uut (
   .j(j), .k(k), .clk(clk), .reset(reset), .q(q), .qbar(qbar)
);
always #5 clk = ~clk;
initial begin
   clk = 0; reset = 1;  j = 0;  k = 0;
   #10 reset = 0;
   j = 0; k = 0; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   j = 0; k = 1; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   j = 1; k = 0; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   j = 1; k = 1; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   j = 0; k = 1; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   j = 1; k = 1; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   j = 1; k = 1; #10;
   $display("j=%b k=%b q=%b qbar=%b", j, k, q, qbar);
   #100;
   $finish;
End endmodule
```

EXP NO 4 : MULTIPLEXER & DECODER

DECODER

```
module decoder(in, out, en);
input [2:0] in;
input en;
output reg [7:0] out;
always @(in or en) begin
  if (en) begin
    out = 8'd0;
    case (in)
      3'b000: out[0] = 1'b1;
      3'b001: out[1] = 1'b1;
      3'b010: out[2] = 1'b1;
      3'b011: out[3] = 1'b1;
      3'b100: out[4] = 1'b1;
      3'b101: out[5] = 1'b1;
      3'b110: out[6] = 1'b1;
      3'b111: out[7] = 1'b1;
      default: out = 8'd0;
    endcase
  end else
    out = 8'd0;
end
endmodule
```

TEST BENCH:

```
module decoder_tb;
wire [7:0] out;
reg en;
reg [2:0] in;
integer i;

decoder dut(in, out, en);

initial begin
  $monitor("en=%b, in=%b, out=%b", en, in, out);
  for (i = 0; i < 16; i = i + 1) begin
    {en, in} = i;
    #5;
  end
  $finish;
end
endmodule
```

MUX

```
module mux_4to1(
  input wire [3:0] a,
  input wire [1:0] sel,
  output wire y
);
assign y = (sel == 2'b00) ? a[0] :
       (sel == 2'b01) ? a[1] :
       (sel == 2'b10) ? a[2] :
       (sel == 2'b11) ? a[3] :
       1'b0;
Endmodule
```

TEST BENCH:

```
module testbench_mux_4to1;
reg [3:0] a;
reg [1:0] sel;
wire y;
mux_4to1 u1 (.a(a), .sel(sel), .y(y));
initial begin
$display("Time a sel y");
$monitor("%d %b %b %b", $time, a, sel, y);
a = 4'b0000; sel = 2'b00; #5;
a = 4'b0011; sel = 2'b01; #5;
a = 4'b1111; sel = 2'b10; #5;
a = 4'b0111; sel = 2'b11; #5;
a = 4'b1010; sel = 2'b00; #5;
a = 4'b1010; sel = 2'b01; #5;
a = 4'b1010; sel = 2'b10; #5;
a = 4'b1010; sel = 2'b11; #5;
$finish;
end
endmodule
```

EXP NO 3:FULL ADDER AND FULL SUBTRACTOR

FULL ADDER

```
module full_adder(a, b, cin, sum1, cout);
input a, b, cin;
output sum1, cout;
reg sum1, cout;
always @(a or b or cin)
begin
{cout, sum1} = a + b + cin;
end
endmodule
```

TEST BENCH:

```
module testbench_full_adder;
reg a, b, cin;
wire sum1, cout;
full_adder u1 (.a(a), .b(b), .cin(cin), .sum1(sum1), .cout(cout));
initial begin
$display("Time a b cin sum1 cout");
$monitor("%d %b %b %b %b %b", $time, a, b, cin, sum1, cout);
a = 0; b = 0; cin = 0; #5;
a = 0; b = 1; cin = 0; #5;
a = 1; b = 0; cin = 0; #5;
a = 1; b = 1; cin = 0; #5;
a = 0; b = 0; cin = 1; #5;
a = 0; b = 1; cin = 1; #5;
a = 1; b = 0; cin = 1; #5;
a = 1; b = 1; cin = 1; #5;
$finish;
end
endmodule
```

FULL SUBTRACTOR

```
module full_subtractor(
input A,
input B,
input Bin,
output Difference,
output Bout
);
assign Difference = A ^ B ^ Bin;
assign Bout = (~A & B) | (B & Bin) | (Bin & ~A);
endmodule
```

TEST BENCH:

```
module testbench_full_subtractor;
reg A, B, Bin;
wire Difference, Bout;
full_subtractor u1 (.A(A), .B(B), .Bin(Bin), .Difference(Difference), .Bout(Bout));
initial begin
$display("Time A B Bin Difference Bout");
$monitor("%d %b %b %b %b %b", $time, A, B, Bin, Difference, Bout);
A = 0; B = 0; Bin = 0; #5;
A = 0; B = 1; Bin = 0; #5;
A = 1; B = 0; Bin = 0; #5;
A = 1; B = 1; Bin = 0; #5;
A = 0; B = 0; Bin = 1; #5;
A = 0; B = 1; Bin = 1; #5;
A = 1; B = 0; Bin = 1; #5;
A = 1; B = 1; Bin = 1; #5;
$finish;
end
endmodule
```

EXP NO 2: BOOLEAN FUNCTIONS
NAND GATE USING AND OPERATOR

```
module nand_example(
input wire A,
input wire B,
output wire NAND_output
);
assign NAND_output = ~(A & B);
endmodule
```

TEST BENCH:

```
module testbench;
reg A, B;
wire NAND_output;
nand_example uut (
.A(A),
.B(B),
.NAND_output(NAND_output)
);
initial begin
A = 0; B = 0; #10;
A = 0; B = 1; #10;
A = 1; B = 0; #10;
A = 1; B = 1; #10;
$finish;
end
always @(A, B, NAND_output) begin
$display("A = %b, B = %b, NAND_output = %b", A, B, NAND_output);
end
endmodule
```

NOR USING OR OPERATOR

```
module nor_gate(
input wire A,
input wire B,
output wire Y
);
assign Y = ~(A|B);
endmodule
```

TEST BENCH:

```
module testbench_nor;
reg A, B;
wire Y_NOR;
nor_gate u1 (.A(A), .B(B), .Y(Y_NOR));
initial begin
$display("Time A B Y_NOR");
$monitor("%d %b %b %b", $time, A, B, Y_NOR);
A = 0; B = 0; #5;
```

```
A = 0; B = 1; #5;
A = 1; B = 0; #5;
A = 1; B = 1; #5;
$finish;
end
endmodule

module or_gate(
input wire A,
input wire B,
output wire Y
);
assign Y = A | B;
endmodule

module and_gate(
input wire A,
input wire B,
output wire Y
);
assign Y = A & B;
endmodule
```

XOR GATE Using AND OPERATOR and OR OPERATOR

```
module xor_gate(
input wire A,
input wire B,
output wire Y
);
wire A_OR_B, A_AND_B;
or_gate or1 (.A(A), .B(B), .Y(A_OR_B));
and_gate and1 (.A(A), .B(B), .Y(A_AND_B));
assign Y = A_OR_B & ~A_AND_B;
endmodule
```

TEST BENCH:

```
module testbench_xor;
reg A, B;
wire Y_XOR;
xor_gate u1 (.A(A), .B(B), .Y(Y_XOR));
initial begin
$display("Time A B Y_XOR");
$monitor("%d %b %b %b", $time, A, B, Y_XOR);
A = 0; B = 0; #5;
A = 0; B = 1; #5;
A = 1; B = 0; #5;
A = 1; B = 1; #5;
$finish;
end
endmodule
```

EXP NO 1: BASIC GATES

```
module and_gate(
input wire A,
input wire B,
output wire Y
);
assign Y = A & B;
endmodule

module or_gate(
input wire A,
input wire B,
output wire Y
);
assign Y = A | B;
endmodule

module not_gate(
input wire A,
output wire Y
);
assign Y = ~A;
endmodule
```

TEST BENCH:

```
module testbench;
reg A, B;
wire Y_AND, Y_OR, Y_NOT;
and_gate u1 (.A(A), .B(B), .Y(Y_AND));
or_gate  u2 (.A(A), .B(B), .Y(Y_OR));
not_gate u3 (.A(A),      .Y(Y_NOT));
initial begin
  $display("Time A B Y_AND Y_OR Y_NOT");
  $monitor("%d %b %b %b %b %b", $time, A, B, Y_AND, Y_OR, Y_NOT);
  A = 0; B = 0; #5;
  A = 0; B = 1; #5;
  A = 1; B = 0; #5;
  A = 1; B = 1; #5;
  $finish;
end
endmodule
```