Assignment 2

# Majority Gate

A majority gate will only output a 1 if more than 50% of the inputs are 1.

Majority Table

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | Y |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Diagram

Diagram

Description automatically generated with medium confidence

Design.sv

module majority\_gate (input a, b, c, output y);

  assign y = (a & b) | (a & c) | (b & c);

endmodule

Testbench.sv

module test();

  reg a, b, c;

  wire y;

  majority\_gate TEST (.a(a), .b(b), .c(c), .y(y));

  initial begin

    $dumpfile("dump.vcd");

    $dumpvars(1);

    a = 0; b = 0; c = 0; #10;

    a = 0; b = 1; c = 0; #10;

    a = 0; b = 0; c = 1; #10;

    a = 1; b = 1; c = 0; #10;

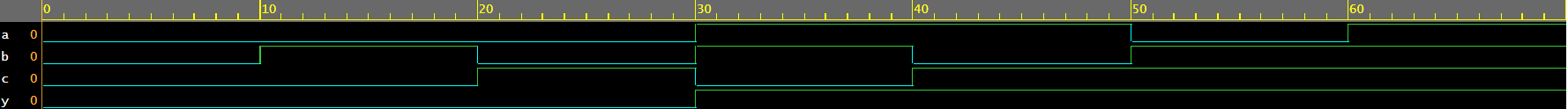
    a = 1; b = 0; c = 1; #10;

    a = 0; b = 1; c = 1; #10;

    a = 1; b = 1; c = 1; #10;

  end

endmodule

Waveform

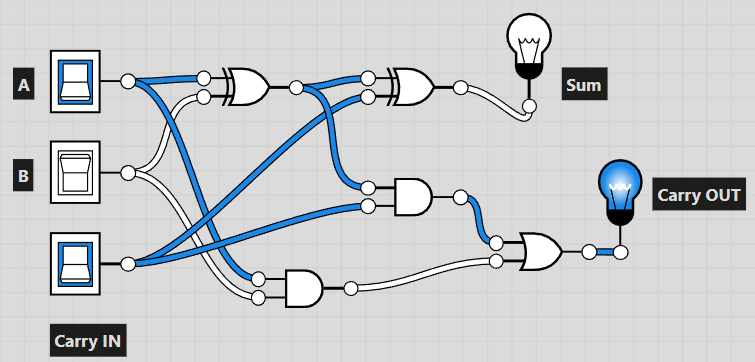
# Full Adder

A full adder adds two 1-bit inputs, plus a carry bit. It outputs the sum. The second output bit is a carry bit that can be chained to another full adder.

Full Adder Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Carry In | Sum | Carry Out |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Diagram



Design.sv

module full\_adder(input a, b, c\_in, output sum, c\_out);

  wire h1\_sum, h1\_carry, h2\_carry;

  half\_adder had1 (.a(a), .b(b), .sum(h1\_sum), .carry(h1\_carry));

  half\_adder had2 (.a(h1\_sum), .b(c\_in), .sum(sum), .carry(h2\_carry));

  assign c\_out = h1\_carry | h2\_carry;

endmodule

module half\_adder (input a, b, output sum, carry);

  assign sum = a ^ b;

  assign carry = a & b;

endmodule

Testbench.sv

module test();

  reg a, b, c\_in;

  wire sum, c\_out;

  full\_adder TEST (.a(a), .b(b), .c\_in(c\_in), .sum(sum), .c\_out(c\_out));

  initial begin

    $dumpfile("dump.vcd");

    $dumpvars(1);

    a = 0; b = 0; c\_in = 0; #10;

    a = 0; b = 1; c\_in = 0; #10;

    a = 0; b = 0; c\_in = 1; #10;

    a = 1; b = 1; c\_in = 0; #10;

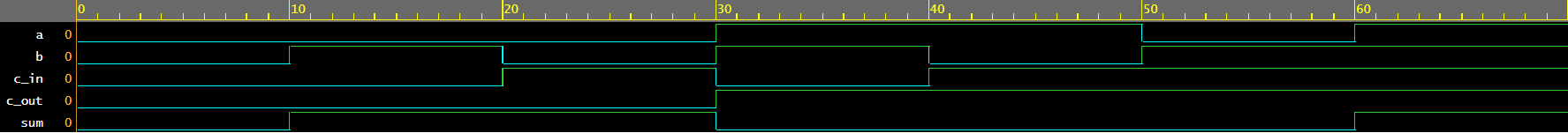
    a = 1; b = 0; c\_in = 1; #10;

    a = 0; b = 1; c\_in = 1; #10;

    a = 1; b = 1; c\_in = 1; #10;

  end

endmodule

Waveform

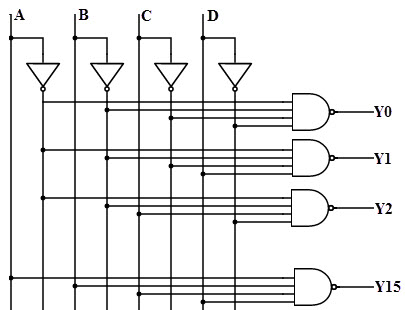
# 4:16 Encoder

Takes in a 4bit input and outputs 16bits. The 16bit output has 1 high and the rest low. The position of the 1 is dependent on the 4bit number. 0000 gives 0000000000000001, 0001 shifts left by one position 0000000000000010 and 1111 gives 1000000000000000.

4:16 Encoder Table

|  |  |
| --- | --- |
| ABCD | Y0-Y15 |
| 0000 | 0000000000000001 |
| 0001 | 0000000000000010 |
| 0010 | 0000000000000100 |
| 0011 | 0000000000001000 |
| 0100 | 0000000000010000 |
| 0101 | 0000000000100000 |
| 0110 | 0000000001000000 |
| 0111 | 0000000010000000 |
| 1000 | 0000000100000000 |
| 1001 | 0000001000000000 |
| 1010 | 0000010000000000 |
| 1011 | 0000100000000000 |
| 1100 | 0001000000000000 |
| 1101 | 0010000000000000 |
| 1110 | 0100000000000000 |
| 1111 | 1000000000000000 |

Diagram



Design.sv

module encoder416(input [3:0] in, output reg [15:0] out);

  always @ (in) begin

    case (in)

            4'b0000: out = 16'b0000000000000001;

            4'b0001: out = 16'b0000000000000010;

            4'b0010: out = 16'b0000000000000100;

            4'b0011: out = 16'b0000000000001000;

            4'b0100: out = 16'b0000000000010000;

            4'b0101: out = 16'b0000000000100000;

            4'b0110: out = 16'b0000000001000000;

            4'b0111: out = 16'b0000000010000000;

            4'b1000: out = 16'b0000000100000000;

            4'b1001: out = 16'b0000001000000000;

            4'b1010: out = 16'b0000010000000000;

            4'b1011: out = 16'b0000100000000000;

            4'b1100: out = 16'b0001000000000000;

            4'b1101: out = 16'b0010000000000000;

            4'b1110: out = 16'b0100000000000000;

            4'b1111: out = 16'b1000000000000000;

            default: out = 16'b0000000000000001;

        endcase

    end

endmodule

Testbench.sv

module test();

  reg [3:0] in;

  wire [15:0] out;

  encoder416 TEST (.in(in), .out(out));

  initial begin

    $dumpfile("dump.vcd");

    $dumpvars(1);

    in = 4'b0000; #5;

    in = 4'b0001; #5;

    in = 4'b0010; #5;

    in = 4'b0011; #5;

    in = 4'b0100; #5;

    in = 4'b0101; #5;

    in = 4'b0110; #5;

    in = 4'b0111; #5;

    in = 4'b1000; #5;

    in = 4'b1001; #5;

    in = 4'b1010; #5;

    in = 4'b1011; #5;

    in = 4'b1100; #5;

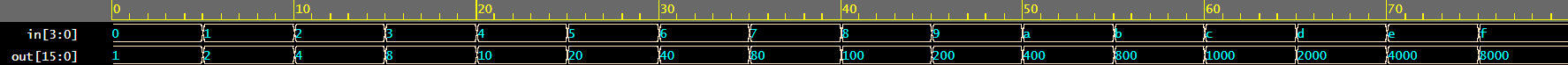
    in = 4'b1101; #5;

    in = 4'b1110; #5;

    in = 4'b1111; #5;

  end

endmodule

Waveform

# Multiplexer and Demultiplexer

A multiplexer takes inputs A, B, C, D, S0 and S1. It’s a data selector. The 2bit value of S0 and S1 from 00 to 11 determines which of A, B, C and D is the output.

A demultiplexer takes inputs Data, S0 and S1. The 2bit value of S0 and S1 from 00 to 11 determines which if the outputs Y0, Y1, Y2 and Y3 is high.

Multiplexer and Demultiplexer Table

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | | | | | Outputs | | | | |
| A | B | | C | D | | Data |
| Choice | | S0 | | | S1 | | Out | Y0 | Y1 | Y2 | Y3 |
| 1 | | 0 | | | 0 | | A | X | X | X | X |
| 1 | | 0 | | | 1 | | B | X | X | X | X |
| 1 | | 1 | | | 0 | | C | X | X | X | X |
| 1 | | 1 | | | 1 | | D | X | X | X | X |
| 0 | | 0 | | | 0 | | X | Data | 0 | 0 | 0 |
| 0 | | 0 | | | 1 | | X | 0 | Data | 0 | 0 |
| 0 | | 1 | | | 0 | | X | 0 | 0 | Data | 0 |
| 0 | | 1 | | | 1 | | X | 0 | 0 | 0 | Data |

Diagram

Diagram

Description automatically generated

Design.sv

module mux\_demux (input choice, a, b, c, d, data, s0, s1,

                  output out, y0, y1, y2, y3);

  reg out, y0, y1, y2, y3;

  always @(\*) begin

    if (choice == 1) begin // multiplexer

      case ({s0,s1})

        {1'b0,1'b0}: out = a;

        {1'b0,1'b1}: out = b;

        {1'b1,1'b0}: out = c;

        {1'b1,1'b1}: out = d;

      endcase

    end

    else begin // demultiplexer

      case ({s0,s1})

        {1'b0,1'b0}: y0 = data;

        {1'b0,1'b1}: y1 = data;

        {1'b1,1'b0}: y2 = data;

        {1'b1,1'b1}: y3 = data;

      endcase

    end

  end

endmodule

Testbench.sv

module test();

  reg choice, a, b, c, d, data, s0, s1;

  wire out, y0, y1, y2, y3;

  mux\_demux TEST (.choice(choice), .a(a), .b(b), .c(c), .d(d), .data(data),

                  .s0(s0), .s1(s1),

                  .out(out), .y0(y0), .y1(y1), .y2(y2), .y3(y3));

  initial begin

    $dumpfile("dump.vcd");

    $dumpvars(1);

    choice = 1; a = 0; s0 = 1'b0; s1 = 1'b0; #5;

    choice = 1; a = 1; s0 = 1'b0; s1 = 1'b0; #5;

    choice = 1; b = 0; s0 = 1'b0; s1 = 1'b1; #5;

    choice = 1; b = 1; s0 = 1'b0; s1 = 1'b1; #5;

    choice = 1; c = 0; s0 = 1'b1; s1 = 1'b0; #5;

    choice = 1; c = 1; s0 = 1'b1; s1 = 1'b0; #5;

    choice = 1; d = 0; s0 = 1'b1; s1 = 1'b1; #5;

    choice = 1; d = 1; s0 = 1'b1; s1 = 1'b1; #5;

    choice = 0; data = 0; s0 = 1'b0; s1 = 1'b0; #5;

    choice = 0; data = 1; s0 = 1'b0; s1 = 1'b0; #5;

    choice = 0; data = 0; s0 = 1'b0; s1 = 1'b1; #5;

    choice = 0; data = 1; s0 = 1'b0; s1 = 1'b1; #5;

    choice = 0; data = 0; s0 = 1'b1; s1 = 1'b0; #5;

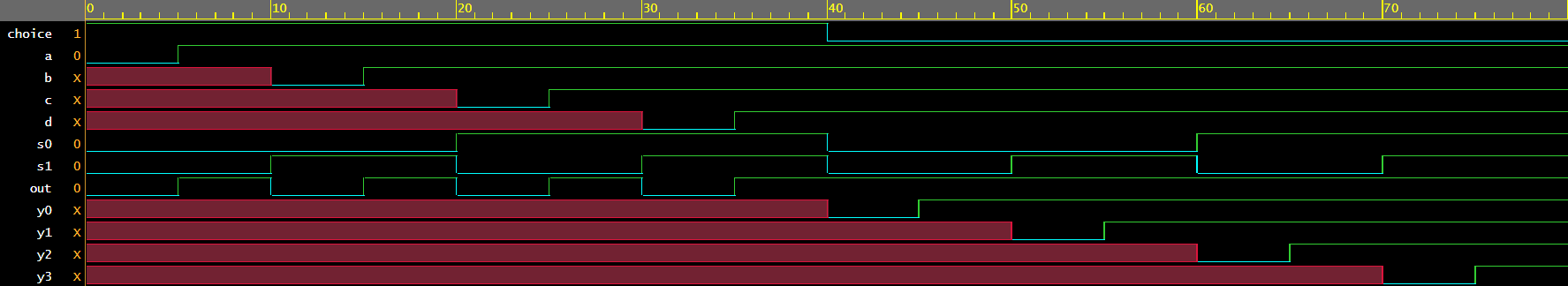
    choice = 0; data = 1; s0 = 1'b1; s1 = 1'b0; #5;

    choice = 0; data = 0; s0 = 1'b1; s1 = 1'b1; #5;

    choice = 0; data = 1; s0 = 1'b1; s1 = 1'b1; #5;

  end

endmodule

Waveform

# Downcounter

Downcounter counts to 0 from any value. When reset is high, the counter starts from it’s max value and each clock cycle, the counter value is decremented by 1 and the output is updated to reflect the new count value.

Design.sv

module down\_counter(input clk, reset, output[15:0] counter);

  reg [15:0] counter\_down;

    always @(posedge clk or posedge reset)

    begin

        if(reset)

            counter\_down <= 16'b1111111111111111;

        else

            counter\_down <= counter\_down - 16'b0000000000000001;

    end

    assign counter = counter\_down;

endmodule

Testbench.sv

module test();

reg clk, reset;

  wire [15:0] counter;

down\_counter Test(clk, reset, counter);

initial begin

    $dumpfile("dump.vcd");

    $dumpvars(1);

clk=0;

//forever #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

    #5 clk=~clk;

  end

initial begin

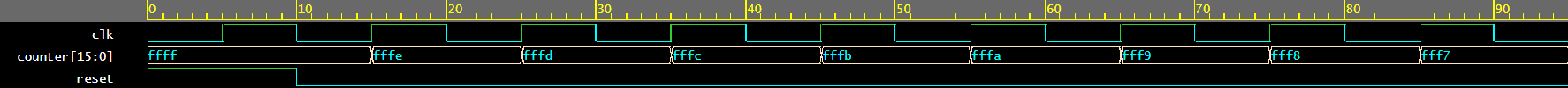
reset=1;

#10;

reset=0;

end

endmodule

Waveform