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;

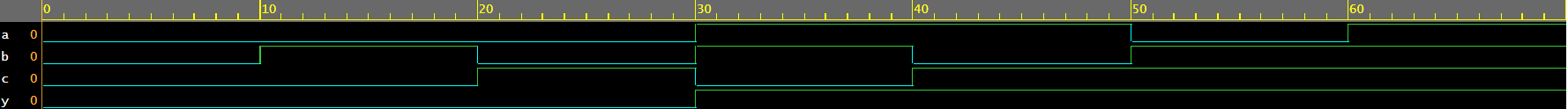
    a = 1; b = 0; c = 1; #10;

    a = 0; b = 1; c = 1; #10;

    a = 1; b = 1; c = 1; #10;

  end

endmodule

Waveform

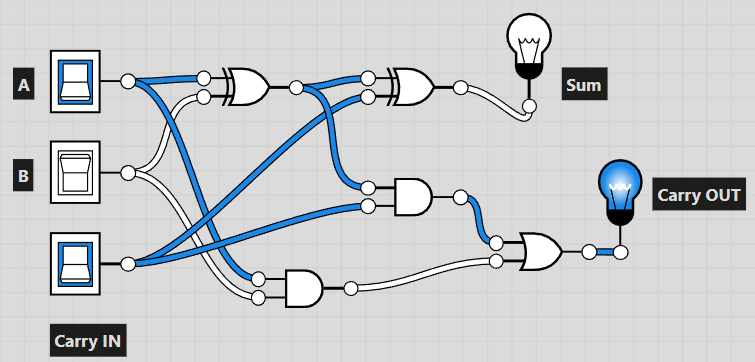
# Full Adder

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.

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;

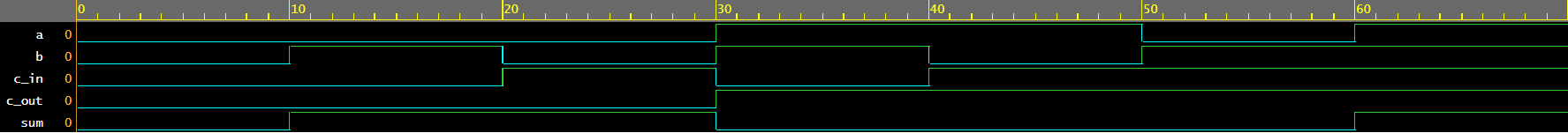
    a = 1; b = 0; c\_in = 1; #10;

    a = 0; b = 1; c\_in = 1; #10;

    a = 1; b = 1; c\_in = 1; #10;

  end

endmodule

Waveform

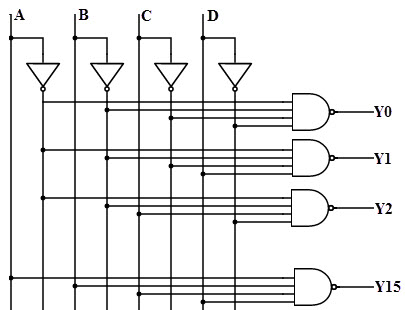
# 4:16 Encoder

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.

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;

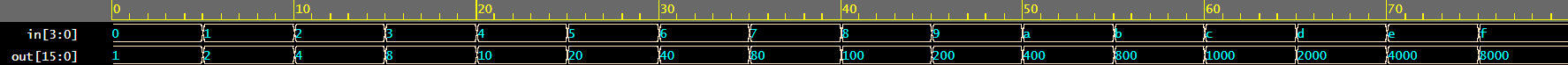
    in = 4'b1101; #5;

    in = 4'b1110; #5;

    in = 4'b1111; #5;

  end

endmodule

Waveform

# Multiplexer and Demultiplexer

Table

Diagram

Diagram

Description automatically generated

Design.sv

Testbench.sv

Waveform

# Down Counter

Table

Diagram

Design.sv

Testbench.sv

Waveform