In modules.v we design several modules following various requirements:

> Write and demonstrate a Verilog module that takes a 4 bit binary input and displays the equivalent hexadecimal value on the seven-segment display.
>
> Write and demonstrate a Verilog module that performs BITWISE logical XOR of two 4-bit values.
>
> Write and demonstrate a Verilog module that receives two, 4-bit unsigned binary numbers and after pressing a button, it adds the two numbers and shows the least significant digit of the result in hex on one digit of the seven segment display.
>
> Design a counter that that counts from F to 0 and goes back to F. Use the 7-segment display to display the counter's output in hexadecimal from F to 0.

MODULES.v

```
`timescale 1ns / 1ps
module hex_display (
    input [3:0] value, // value will decide hex on 7 seg
    output reg [6:0] seven_segment, // output of seven seg
    output [3:0] anodes // far right seven seg will only be used
);
/*
    From MSB to LSB
    GFEDCBA is the order of bits corresponding to seven seg display
    the following comment is from diligent

    //////////////gfedcba/////////////////////////////////////////////              a
    0:a_to_g = 7'b1000000;////0000                                                  __
    1:a_to_g = 7'b1111001;////0001                                               f/    /b
    2:a_to_g = 7'b0100100;////0010                                                 g
    //                                                                           __
    3:a_to_g = 7'b0110000;////0011                                             e /   /c
    4:a_to_g = 7'b0011001;////0100                                                 __
    5:a_to_g = 7'b0010010;////0101                                                 d
*/
    assign anodes = 4'b1110; //FAR RIGHT SEV SEG DISPLAY
    always @(*) begin
        case (value)
            4'b0000: seven_segment = 7'b100_0000; // "0"
            4'b0001: seven_segment = 7'b111_1001; // "1"
            4'b0010: seven_segment = 7'b010_0100; // "2"
            4'b0011: seven_segment = 7'b011_0000; // "3"
            4'b0100: seven_segment = 7'b001_1001; // "4"
            4'b0101: seven_segment = 7'b001_0010; // "5"
            4'b0110: seven_segment = 7'b000_0010; // "6"
            4'b0111: seven_segment = 7'b111_1000; // "7"
            4'b1000: seven_segment = 7'b000_0000; // "8"
            4'b1001: seven_segment = 7'b001_0000; // "9"
            4'b1010: seven_segment = 7'b000_1000; // "A"
            4'b1011: seven_segment = 7'b000_0011; // "b"
            4'b1100: seven_segment = 7'b100_0110; // "c"
            4'b1101: seven_segment = 7'b010_0001; // "d"
            4'b1110: seven_segment = 7'b000_0110; // "e"
            4'b1111: seven_segment = 7'b000_1110; // "f"
```

```
            default: seven_segment = 7'b000_0000; // "0"
        endcase
    end
endmodule

module switches_to_hex (
    input [9:0] switches,
    output reg [3:0] value
);
    always @(*) begin
        if (switches[9:8] == 2'b00) begin
            value = switches[3:0]; // far right switches outputted
        end
    end
endmodule

module switches_xor (
    input [9:0] switches,
    output reg [3:0] led,
    output reg [3:0] clear // only purpose is to send zero to display_hex
);
    reg [3:0] xor_value;
    wire [3:0] a = switches [3:0];
    wire [7:4] b = switches [7:4];
    always @(*) begin
        if (switches[9:8] == 2'b01) begin
            xor_value = a ^ b;
            led = xor_value;
            clear = 4'b0000;
        end
        else begin
            led = 4'b0000;
        end
    end
endmodule


module switches_addition (
    input [9:0] switches,
    input btn_center,
    output reg [3:0] value
);
    reg [5:0] sum;
    wire [3:0] a = switches [3:0];
    wire [7:4] b = switches [7:4];
    always @(*) begin
        if (switches[9:8] == 2'b10) begin
            if (btn_center == 1'b1) begin
                sum = a + b;
                value = sum[3:0];
            end
        end
    end
```

```
endmodule

module switches_auto_counter (
    input clk,
    input [9:0] switches,
    output reg [3:0] value
);
    reg [3:0]counter = 4'b1111; //counter to decrement each posedge
    always @(posedge clk) begin
        if(counter > 0) begin
            value <= counter;
            counter <= counter - 4'b0001;
        end
        else begin
            counter <= 4'b1111;
            value <= counter;
        end
    end
endmodule

module clock_divider(
        input clk,
        output reg clk_new
    );
    reg [31:0] counter = 0;
    always @(posedge clk) begin
        if(counter == 50_000_000) begin
            counter <= 0;
            clk_new = ~clk_new;
        end
        else begin
            counter <= counter + 1;
        end
    end
endmodule
```

TB_MODULES.v

```
`timescale 1ns / 1ps
module stimmy (
     input [9:0] sw, // switches
     input btnC, // center button
     input clk, // clk from basys3
     output [3:0] led, // leds
     output [6:0] seg, // seven segment display
     output [3:0] an //anodes
);
     reg [3:0] display_hex; // register to store value to be displayed in seven seg
     wire [3:0]temp0; // output from using switches as hexvalue on seven seg (async)
     wire [3:0]temp1; // output from xor is zero because asingnment reequested the use of LEDS
     wire [3:0]temp2; // output from addition of two 4-bit unsigned numbers after button press
     wire [3:0]temp3; // output hexadecmial from F to 0 using clock divider
     wire tmp_clk; // result of clock divider
```

```
/*
using always switch on two switches on basys because assignment requested different functions
to be used in correspondence to two switches on board

Visually we are only seeing one output on the seven seg because of the case statements.
But every other function is in action in the background
*/
always @(sw[9:8]) begin
     case (sw[9:8])
          2'b00: begin
               display_hex <= temp0; // switches to hex
          end
          2'b01: begin
               display_hex <= temp1; // zero, leds are the spotlight
          end
          2'b10: begin
               display_hex <= temp2; // LSB of addition two two binary numbers in hex
          end
          2'b11: begin
               display_hex <= temp3; // synchronous hex
          end
          default: begin
               display_hex <= temp0;
          end
     endcase
end

hex_display inst1 (
     .value(display_hex),
     .seven_segment(seg),
     .anodes(an)
);
switches_to_hex inst2 (
     .switches(sw),
     .value(temp0)
);
switches_xor inst3 (
     .switches(sw),
     .led(led),
     .clear(temp1)
);
switches_addition inst4(
     .switches(sw),
     .btn_center(btnC),
     .value(temp2)
);
switches_auto_counter inst5(
     .switches(sw),
     .clk(tmp_clk),
     .value(temp3)
);
clock_divider inst6 (
```

```
        .clk(clk),
        .clk_new(tmp_clk)
    );
endmodule
```

Basys-3-Master.xdc

```
# # Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
# # Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
# LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


# #Buttons
set_property PACKAGE_PIN U18 [get_ports btnC]
set_property IOSTANDARD LVCMOS33 [get_ports btnC]
## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```