Problem 1: Building the MIPS ALU

(a) module alu(
    input [2:0] funct,
    input [31:0] op1,
    input [31:0] op2,
    output reg [31:0] alu_out
);

always @(*) begin
    case(funct)
        3'b000: alu_out = ($signed(op1)) + ($signed(op2));
        3'b001: alu_out = op1 + op2;
        3'b010: alu_out = ($signed(op1)) - ($signed(op2));
        3'b011: alu_out = op1 - op2;
        3'b100: alu_out = op1 & op2;
        3'b101: alu_out = op1 | op2;
        3'b110: alu_out = $signed(op1) >>> op2[4:0];
        3'b111: alu_out = op1 >> op2[4:0];
        else: alu_out = 32'd0;
    endcase

end
endmodule

(b) /*
   In this testbench only one example is given for each case, but you should
test more comprehensively while designing circuits
*/
module alu_test();

    reg [2:0] funct;
    reg [31:0] op1;
    reg [31:0] op2;

endmodule
wire [31:0] alu_out;

alu a0 (
    .funct(funct),
    .op1(op1),
    .op2(op2),
    .alu_out(alu_out)
);

initial begin
    funct = 3'b000;
    op1 = 32'd0;
    op2 = 32'd0;

    // Test for signed addition
    #(5)
    op1 = -4;
    op2 = 2;

    #(1)
    if ($signed(alu_out) != -2)
        $display("Failed signed add,\n        output should be -2, but was %d", $signed(alu_out));
    else
        $display("Passed signed add");

    // Test for unsigned addition
    #(5)
    funct = 3'b001;
    op1 = 32'd4;
    op2 = 32'd2;

    #(1)
    if (alu_out != 32'd6)
        $display("Failed unsigned add,\n        output should be 6, but was %d", alu_out);
    else
        $display("Passed unsigned add");

    // Test for signed subtraction
    #(5)
    funct = 3'b010;
    op1 = -4;
    op2 = 2;

    #(1)
if ($signed(alu_out) != -6)
    $display("Failed signed subtract,\n          output should be -6, but was %d", $signed(alu_out));
else
    $display("Passed signed subtract");

//Test for unsigned subtraction
#(5)
funct = 3'b011;
op1 = 32'd4;
op2 = 32'd2;

#(1)
if (alu_out != 2)
    $display("Failed unsigned subtract,\n             output should be 2, but was %d", alu_out);
else
    $display("Passed unsigned subtract");

//Test for and
#(5)
funct = 3'b100;
op1 = 32'hFFFFFFFF;
op2 = 32'h00000001;

#(1)
if (alu_out != 32'h00000001)
    $display("Failed and, output should be 32'h00000001, but was %h", alu_out);
else
    $display("Passed and");

//Test for or
#(5)
funct = 3'b101;
op1 = 32'hFFFFFFFE;
op2 = 32'h00000001;

#(1)
if (alu_out != 32'hFFFFFFFF)
    $display("Failed or, output should be 32'hFFFFFFFF, but was %h", alu_out);
else
    $display("Passed or");

//Test for SRA
funct = 3'b110;
op1 = 32'hFFFFFFFF;
op2 = 32'd2;

if (alu_out != 32'hFFFFFFFF)
   $display("Failed arithmetic right shift, \
          output should be 32'hFFFFFFFF, but was %h", alu_out);
else
   $display("Passed arithmetic right shift");

//Test for SRL
funct = 3'b111;
op1 = 32'hFFFFFFFF;
op2 = 32'd2;

if (alu_out != 32'h3FFFFFFF)
   $display("Failed logical right shift, \
          output should be 32'h3FFFFFFF, but was %h", alu_out);
else
   $display("Passed logical right shift");

$finish();
end
endmodule
Problem 2: Hamming Code

(a) Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>No Error</td>
</tr>
<tr>
<td>01</td>
<td>Parity Error</td>
</tr>
<tr>
<td>10</td>
<td>Double Error</td>
</tr>
<tr>
<td>11</td>
<td>Single Error</td>
</tr>
</tbody>
</table>

(b) Figure 1: Hierarchy Diagram of SECDED

(c) //Compute the received parity

module read_parity
(input [21:0] data,
output [4:0] new_parity);

assign new_parity[0] = (data[2]*data[4]*data[6]*data[8]*
data[10]*data[12]*data[14]*data[16]*data[18]*data[20]);
assign new_parity[1] = (data[2]*data[5]*data[6]*data[9]*
data[10]*data[13]*data[14]*data[17]*data[18]);
data[12]*data[13]*data[14]*data[19]*data[20]);
data[12]*data[13]*data[14]);
endmodule

//Compute the syndrome
module syndrome (  
    input [4:0] old_parity, new_parity,  
    output [4:0] syndrome  
);  
    
    assign syndrome = old_parity ^ new_parity;  
endmodule

//Compute overall error code
module secded(
    input [21:0] data,  
    output [1:0] error  
);

wire [4:0] new_parity, old_parity, syndrome;  
wire overall_parity;

read_parity rp(.data(data), .new_parity(new_parity));  
syndrome syn(.old_parity(old_parity), .new_parity(new_parity), .syndrome(syndrome));

assign old_parity = {data[15], data[7], data[3], data[1], data[0]};  
assign overall_parity = ~data;

assign error = {~(syndrome == 0), overall_parity};
endmodule

/*
In this testbench only one example is given for each case, but you should test more comprehensively while designing circuits */
module secded_test();

    reg [21:0] data;  
    wire [1:0] error;

    secded s0(.data(data), .error(error));

    initial begin  
        data = 22'd0;
        error = 1'b0;
    end  
endmodule
//Test for correct data
#(5)
data = 22'b11_1101_0111_1011_0101_0010;

#(1)
if (error != 2'b00)
    // here's an example of how you check internal signals
    // with testbench
    $display("Error is wrong, should be 00 but was %b, \ 
          old syndrome is %b, new syndrome is %b", 
          error, s0.old_parity, s0.new_parity);
else
    $display("No error detected correctly");

//Test for single bit error
#(5)
data = 22'b11_1101_0111_1101_1101_0010;

#(1)
if (error != 2'b11)
    $display("Error is wrong, should be 11 but was %b", error);
else
    $display("Single error detected correctly");

//Test for double bit error
#(5)
data = 22'b11_1101_0111_1010_1101_0010;

#(1)
if (error != 2'b10)
    $display("Error is wrong, should be 10 but was %b", error);
else
    $display("Double error detected correctly");

//Test for parity bit error
#(5)
data = 22'b01_1101_0111_1011_0101_0010;

#(1)
if (error != 2'b01)
    $display("Error is wrong, should be 01 but was %b", error);

7
else
    $display("Parity error detected correctly");
endmodule

end
Problem 3: Counters

(a) module counter(
    input clk,
    input reset,
    input en,
    input up,
    output reg [15:0] count
);

always @(posedge clk) begin
    if (reset == 1'b1) count <= 16'd0;
    else begin
        if (en == 1'b1)
            if (up == 1'b1) count <= count + 1;
            else count <= count - 1;
        else count <= count;
    end
end
endmodule

(b) module counter_test();

    reg clk, reset, en, up;
    wire [15:0] count;

    counter cn(.clk(clk), .reset(reset), .en(en), .up(up), .count(count));

    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    initial begin
        clk = 1'b0;
        reset = 1'b1;
        en = 1'b0;
        up = 1'b0;

        repeat (2) @(posedge clk);
        reset = 1'b0;
    //makes sure count stays unchanged when en = 0
    repeat (5) @(posedge clk);
        if (count == 16'b0)
$display("So far so good");
else
    $display("Failed when enable is 0");

//Start counting
@(posedge clk);
en = 1'b1;
up = 1'b1;

repeat (100) @(posedge clk);
if (count == 16'd100)
    $display("So far so good");
else
    $display("Count incorrect");
up = 1'b0;
repeat (50) @(posedge clk);
if (count == 16'd50)
    $display("Passed!");
else
    $display("Up/down incorrect");
$finish();
end
endmodule