Embedded Systems

CS429 Lab3

Name: Dipean Dasgupta **ID:202151188**

Task1 Solution:

Module code:

```
module eight_bit_adder (
    input [7:0] A,
    input [7:0] B,
    output [7:0] sum,
    output [4:0] flags
);

wire [8:0] adder_output;

assign adder_output = {1'b0, A} + {1'b0, B};

assign sum = adder_output[7:0];

assign flags[0] = (sum == 8'b00000000) ? 1'b1 : 1'b0; // Zero flag
assign flags[1] = (sum ^ (sum >> 1) ^ (sum >> 2) ^ (sum >> 3) ^ (sum >> 4) ^ (sum
>> 5) ^ (sum >> 6) ^ (sum >> 7)) ? 1'b1 : 1'b0; // Parity flag
assign flags[2] = (A[3] & B[3]) ? 1'b1 : 1'b0; // Auxiliary carry flag
assign flags[3] = (sum[0] == 1'b0) ? 1'b1 : 1'b0; // Even_or_odd flag
assign flags[4] = (adder_output[8] == 1'b1) ? 1'b1 : 1'b0; // Carry flag

endmodule
```

Testbench Code:

```
`include "lab3_1.v"
module lab3_1tb;
  reg [7:0] A;
  reg [7:0] B;
  wire [7:0] sum;
  wire [4:0] flags;

  eight_bit_adder uut (
    .A(A),
    .B(B),
    .sum(sum),
```

```
    .flags(flags)
  );

  reg clk = 0;
  always #5 clk = ~clk;


  initial begin
    $dumpfile("test_lab3_1.vcd");
    $dumpvars(0, lab3_1tb);

    // (5+3=8)
    A = 8'b00000101;
    B = 8'b00000011;
    #10;
    // (127+1=128)
    A = 8'b01111111;
    B = 8'b00000001;
    #10;
    //(255+1=0 carry)
    A = 8'b11111111;
    B = 8'b00000001;
    #10;
    $finish;
  end

  always @(posedge clk) begin
    $display("A = %b, B = %b, Sum = %b, Flags = %b", A, B, sum, flags);
  end
endmodule
```

OUTPUT:

```
[Running] lab3_1tb.v
VCD info: dumpfile test_lab3_1.vcd opened for output.
A = 00000101, B = 00000011, Sum = 00001000, Flags = 01010
A = 01111111, B = 00000001, Sum = 10000000, Flags = 01010
A = 11111111, B = 00000001, Sum = 00000000, Flags = 11001
lab3_1tb.v:40: $finish called at 30 (1s)
[Done] exit with code=0 in 0.124 seconds
```

Task 2(a) Solution:

Module code:

```
module parity_encoder (
    input [7:0] message_signal,
    output reg [8:0] coded_message
);

integer ones_count;

always @(message_signal) begin

    ones_count = 0;
    for (ones_count = 0; ones_count < 8; ones_count = ones_count + 1) begin
        if (message_signal[ones_count] == 1'b1) begin
            ones_count = ones_count + 1;
        end
    end

    // Set the parity bit based on the evenness of ones_count
    if (ones_count % 2 == 0) begin
        coded_message = {1'b0, message_signal};
    end
    else begin
        coded_message = {1'b1, message_signal};
    end
end

endmodule
```

Testbench code:

```
`include "lab3_2a.v"
module lab3_2atb;

reg [7:0] message_signal;
wire [8:0] coded_message;


parity_encoder pe (
    .message_signal(message_signal),
    .coded_message(coded_message)
);
```

```
initial begin
    //even ones
    message_signal = 8'b11001100;
    #10;
    $display("Message: %b, Coded Message: %b", message_signal, coded_message);

    //odd ones
    message_signal = 8'b11001101;
    #10;
    $display("Message: %b, Coded Message: %b", message_signal, coded_message);

    $finish;
end

endmodule
```

OUTPUT:

```
[Running] lab3_2atb.v
Message: 11001100, Coded Message: 011001100
Message: 11001101, Coded Message: 011001101
lab3_2atb.v:25: $finish called at 20 (1s)
[Done] exit with code=0 in 0.171 seconds
```

Task 2(b) Solution:

Module code:

```
module parity_encoder (
    input [7:0] message_signal,
    output reg [8:0] coded_message
);

reg parity_bit;

integer ones_count;

always @(message_signal) begin
    ones_count = 0;
    for (ones_count = 0; ones_count < 8; ones_count= ones_count + 1) begin
        if (message_signal[ones_count] == 1'b1) begin
```

```
            ones_count = ones_count + 1;
        end
    end

    if (ones_count % 2 == 0) begin
        parity_bit = 1'b0;
    end else begin
        parity_bit = 1'b1;
    end

    coded_message = {parity_bit, message_signal};
end
endmodule

module error_detector (
    input [8:0] coded_message,
    output reg error_detected
);
integer ones_counts;
always @(coded_message) begin
    ones_counts = 0;
    for (ones_counts= 0;ones_counts< 9;ones_counts=ones_counts+ 1) begin
        if (coded_message[ones_counts] == 1'b1) begin
            ones_counts = ones_counts + 1;
        end
    end

    // Check for an error (odd number of ones)
    if (ones_counts % 2 == 0) begin
        error_detected = 1'b0;
    end
    else begin
        error_detected = 1'b1;
    end
end

endmodule
```

TestBench Code:

```
`include "lab3_2b.v"
module lab3_2btb;
```

```
reg [7:0] message_signal;
wire [8:0] coded_message;
wire error_detected;

parity_encoder pe (
    .message_signal(message_signal),
    .coded_message(coded_message)
);

error_detector ed (
    .coded_message(coded_message),
    .error_detected(error_detected)
);

initial begin
    message_signal = 8'b11011100;
    #10;
    $display("Message: %b, Coded Message: %b, Error Detected: %b",
message_signal, coded_message, error_detected);


    message_signal = 8'b11001101;
    #10;
    $display("Message: %b, Coded Message: %b, Error Detected: %b",
message_signal, coded_message, error_detected);

    $finish;
end
endmodule
```

OUTPUT:

```
[Running] lab3_2btb.v
Message: 11011100, Coded Message: 011011100, Error Detected: 1
Message: 11001101, Coded Message: 011001101, Error Detected: 1
lab3_2btb.v:32: $finish called at 20 (1s)
[Done] exit with code=0 in 0.133 seconds
```

Task3 Solution:

Module and Testbench code:

```
module eight_bit_comparator (
    input [7:0] num1,
    input [7:0] num2,
    output equal,
    output lesser,
    output greater
);
    assign equal = (num1 == num2);
    assign lesser = (num1 < num2);
    assign greater = (num1 > num2);
endmodule

module tb_eight_bit_comparator;
    reg [7:0] num1;
    reg [7:0] num2;
    wire equal;
    wire lesser;
    wire greater;

    eight_bit_comparator uut (
        .num1(num1),
        .num2(num2),
        .equal(equal),
        .lesser(lesser),
        .greater(greater)
    );
    initial begin
        //num1 = num2
        num1 = 8'b01100100;
        num2 = 8'b01100100;
        #10;
        $display("Case 1: num1=%d, num2=%d, equal=%b, lesser=%b,
greater=%b",num1,num2, equal, lesser, greater);

        // num1 < num2
        num1 = 8'b00111000;
        num2 = 8'b10101010;
        #10;
        $display("Case 2: num1=%d, num2=%d, equal=%b, lesser=%b,
greater=%b",num1,num2, equal, lesser, greater);
```

```
        //num1 > num2
        num1 = 8'b11001100;
        num2 = 8'b01010101;
        #10;
        $display("Case 3: num1=%d, num2=%d, equal=%b, lesser=%b,
greater=%b",num1,num2, equal, lesser, greater);
        $finish;
    end
endmodule
```

OUTPUT:

```
[Running] lab3_3.v
Case 1: num1=100, num2=100, equal=1, lesser=0, greater=0
Case 2: num1= 56, num2=170, equal=0, lesser=1, greater=0
Case 3: num1=204, num2= 85, equal=0, lesser=0, greater=1
lab3_3.v:55: $finish called at 30 (1s)
[Done] exit with code=0 in 0.12 seconds
```