Computer Design and Organization

Problem Set #2

Due: Friday January 21

This short assignment is to get you familiar with the basics of Verilog. How to turnin your programs will be posted on the Web page.

Problem 1

You will design an 8 by 3 decoder. There are two inputs, in1, in2 respectively 2 and 6 bits long. The output out is 3 bits. You are strongly encouraged to use constructs such as case.

The output of the decoder should have the following values:

out takes the value 0 if in1 = 2 and in2 = 4, or 20, or 36, or 52.  
out takes the value 1 if in1 = 2 and in2 = 5, or 21, or 37, or 53.  
out takes the value 2 if in1 = 0  
out takes the value 2 if in1 = 2 and in2 = 0, or 16, or 32, or 48.  
out takes the value 6 if in1 = 1  
out takes the value 6 if in1 = 2 and in2 = 2, or 18, or 34, or 50.  
out takes the value 7 if in1 = 2 and in2 = 10, or 26, or 42, or 58.  
out is otherwise undefined.

You should test thoroughly your design by writing a test program in Verilog. The decoder module should be called dec8x3 and have parameters listed as: module dec8x3(in1,in2,out). Turnin both the module and the test program.

The code that you’ll write should be at a level similar to the following example:

```verilog
module decode2x4(in,z);
input [1:0]in;
output [3:0]z;
reg [3:0]z;
always @ (in)
  case(in)
    0: z= 4’b0001;
    1: z= 4’b0010;
    etc.....
  endcase
end
endmodule
```
Problem 2
Design, at the functional level, an ALU that performs the operations listed below. The inputs \(a\) and \(b\) should be 32 bits. The third input is the opcode which is 3 bits with:

- opcode = 0 means “and”
- opcode = 1 means “or”
- opcode = 2 means “add”
- opcode = 3 means “sub”
- opcode = 4 means “slt”
- all other opcodes are undefined and should return undefined values \(x\) in the result.

The outputs are a 32-bit result \(r\), an overflow bit \(v\), and a zero bit \(z\) which is equal to 1 if and only if all 32 bits of \(r\) are 0’s.

The module should be called alu32 and have parameters listed as: alu32(a,b,opcode,r,v,z). Turn in both the module and the test program.

The code that you’ll write should be at a level similar to the example below (which has not the same list of parameters, on purpose).

Of course you can (and will need to) use other statement types (e.g., case) and other operators (e.g., reduction operators).

You should also write, and turn in a test program which verifies your design. Be sure to include tests to see if the values generated for \(v\) and \(z\) (for example) are correct.

```verilog
module alusim(a,b,Carryin,Result,Overflow,Operation);

    input [width-1:0] a,b;
    input Carryin;
    output [width-1:0]Result;
    output Overflow;
    input Operation;
    reg Carryout, Overflow;
    reg [width-1:0]Result;

    always @(a or b or Carryin or Operation)
        if (Operation == 0)
            Result = a & b;
```
else if (Operation == 1)
    begin
        {Carryout,Result}= a + b + Carryin;
        Overflow = ....
    end
endmodule

Problem 3
Design and test a Verilog program for a 2-bit saturating counter (finite state machine) with the following state diagram.

States are labelled 0, 1, 2, 3. Transitions are indicated by the arrows and occur every time the input in is assigned a value of either 0 or 1.

The module should be called satc and have parameters listed as: module satc(in,state). Turn in both the module and the test program.