CSE471-A96 Snyder

Assignment 1

Verilog HDL Program For ALU

Background Reading: See Hennesey and Patterson, Section 4.5, pp. 182-198 for a description of the ALU to be designed. For information on the simulator and simulation language, see the class handouts on Verilog, and the "Quick Reference Manual for Verilog HDL" by Rajeev Madhavan, available on-line on the web page.

Goal: To acquire initial proficiency in writing Verilog behavioral descriptions, to gain experience running simulations, and to reacquaint ourselves with the basic ALU structure of a modern computer architecture.

Resources: Verilog is available on the "329 Machines," WOLF, GRIZZLY and LYNX. It is invoked by typing "verilog" followed by a sequence of file names containing the modules of your design, as in

```
wolf> verilog tester.v design.v
```

The subsequent behavior will be pretty self explanatory. There could be compile time errors, run time errors or the successive outputs of your simulation. At completion, a short synopsis is provided of the execution time used.

Additonally, there is a full behavioral description in Verilog of the 32 bit ALU given in Hennessy and Patterson appended below.

Tasks: It is suggested that as an exercise to familiarize yourself with the syntax of Verilog that you type the appended design in manually rather than exploiting its availability on-line.

>> As an exercise in understanding what the ALU design is doing incorporate comments into the Verilog program.

Statisfactory comments describe what a routine is doing, and as appropriate, explains specific subtleties in the statements. Once the typos are removed from the program, run it. Notice the control lines that configure the design, such as `Binvert`.

>> Prepare a page of output that exercises the design and establish that it works. This includes, as a minimum, executing all of the operations on both normal and "overflowing" operands.

The `fulladder()` in the appended design is a simple behavioral description.

>> Replace the fulladder design with a structural design based on the defining logic of binary addition (H&P, pp. 186-7). This new Verilog code should be formulated into two routines: `summer()` which computes the result bit (half add), and `carry()` which computes the carry out bit. Additionally the `fulladder()` routine will have to be changed to call these two new routines.

Submit: (a) Original program, commented, (b) Exercise code, with an explanation of what the tests are testing, (c) The three routines, `summer()`, `carry()` and `fulladder()` and (d) exercise output showing that the new adder works.

Due: Parts (a) and (b) are due by 5:00pm Friday, 18 October 1996, and parts (c) and (d) are due by 5:00pm Monday 21 October 1996.

-- fulladder--

```
module fulladder( CarryOut, Result, a, b, c);
input a, b, c;
output CarryOut, Result;
reg CarryOut, Result;

     always @(a or b or c)
       {CarryOut, Result} = a + b + c;
endmodule
```

-- mux2 --

```
module mux2( z, Select, in0,in1);
input Select, in0, in1;
output z;
reg z;

     always @(Select or in0 or in1)
        if (Select == 1'b1)
          z = in1;
     else if (Select == 1'b0)
          z = in0;
     else
          z = 1'bx;
endmodule
```

-- mux4 --

```
module mux4( z, Select, in0, in1, in2, in3);
input [1:0] Select;
input in0, in1, in2, in3;
output z;
reg z;

     always @(Select or in0 or in1 or in2 or in3)
        case (Select)
          2'b00: z = in0;
          2'b01: z = in1;
```

```
        2'b10: z = in2;
        2'b11: z = in3;
        default: z = 1'bx;
    endcase
endmodule
```

-- alu1 --

```
module alu1( a, b, CarryIn, Less, Result,
    CarryOut, Operation, Binvert);
input a, b, CarryIn, Less;
output Result, CarryOut;
input [1:0] Operation;
input Binvert;

wire t0, t1, t2;
wire bbar, bb;

and  g0( t0, a, b);
or   g1( t1, a, b);
not  g2( bbar, b);
mux2 g3( bb, Binvert, b, bbar);
fulladder g4( CarryOut, t2, a, bb, CarryIn);
mux4 g5( Result, Operation, t0, t1, t2, Less);

endmodule
```

-- alu1msb --

```
module alu1msb( a, b, CarryIn, Less, Result,
               CarryOut, Operation, Binvert,
               Set, Overflow);
input a, b, CarryIn, Less;
output Result, CarryOut;
input [1:0] Operation;
input Binvert;
output Set, Overflow;
reg Overflow, Set;

wire t0, t1, t2;
wire bbar, bb;

and  g0( t0, a, b);
or   g1( t1, a, b);
not  g2( bbar, b);
mux2 g3( bb, Binvert, b, bbar);
fulladder g4( CarryOut, t2, a, bb, CarryIn);
mux4 g5( Result, Operation, t0, t1, t2, Less);

always @(CarryIn or CarryOut or t2)
    begin
    Overflow = CarryIn ^ CarryOut;
    Set = Overflow ^ t2;
end
endmodule
```

-- alu32 --

```
module alu32( a, b, r, Overflow,
          Op, Bneg, AllZero);
input [31:0] a, b;
output [31:0] r;
input Bneg;
input [1:0] Op;
output Overflow, AllZero;
reg AllZero;

wire [32:1] c;
wire Set;

alu1 alu_0( a[0], b[0], Bneg, Set, r[0], c[1], Op, Bneg);
alu1 alu_1( a[1], b[1], c[1], 1'b0, r[1], c[2], Op, Bneg);
alu1 alu_2( a[2], b[2], c[2], 1'b0, r[2], c[3], Op, Bneg);
alu1 alu_3( a[3], b[3], c[3], 1'b0, r[3], c[4], Op, Bneg);
alu1 alu_4( a[4], b[4], c[4], 1'b0, r[4], c[5], Op, Bneg);
alu1 alu_5( a[5], b[5], c[5], 1'b0, r[5], c[6], Op, Bneg);
alu1 alu_6( a[6], b[6], c[6], 1'b0, r[6], c[7], Op, Bneg);
alu1 alu_7( a[7], b[7], c[7], 1'b0, r[7], c[8], Op, Bneg);
alu1 alu_8( a[8], b[8], c[8], 1'b0, r[8], c[9], Op, Bneg);
alu1 alu_9( a[9], b[9], c[9], 1'b0, r[9], c[10], Op, Bneg);
alu1 alu_10( a[10], b[10], c[10], 1'b0, r[10], c[11], Op, Bneg);
alu1 alu_11( a[11], b[11], c[11], 1'b0, r[11], c[12], Op, Bneg);
alu1 alu_12( a[12], b[12], c[12], 1'b0, r[12], c[13], Op, Bneg);
alu1 alu_13( a[13], b[13], c[13], 1'b0, r[13], c[14], Op, Bneg);
alu1 alu_14( a[14], b[14], c[14], 1'b0, r[14], c[15], Op, Bneg);
alu1 alu_15( a[15], b[15], c[15], 1'b0, r[15], c[16], Op, Bneg);
alu1 alu_16( a[16], b[16], c[16], 1'b0, r[16], c[17], Op, Bneg);
alu1 alu_17( a[17], b[17], c[17], 1'b0, r[17], c[18], Op, Bneg);
alu1 alu_18( a[18], b[18], c[18], 1'b0, r[18], c[19], Op, Bneg);
alu1 alu_19( a[19], b[19], c[19], 1'b0, r[19], c[20], Op, Bneg);
alu1 alu_20( a[20], b[20], c[20], 1'b0, r[20], c[21], Op, Bneg);
alu1 alu_21( a[21], b[21], c[21], 1'b0, r[21], c[22], Op, Bneg);
alu1 alu_22( a[22], b[22], c[22], 1'b0, r[22], c[23], Op, Bneg);
alu1 alu_23( a[23], b[23], c[23], 1'b0, r[23], c[24], Op, Bneg);
alu1 alu_24( a[24], b[24], c[24], 1'b0, r[24], c[25], Op, Bneg);
alu1 alu_25( a[25], b[25], c[25], 1'b0, r[25], c[26], Op, Bneg);
alu1 alu_26( a[26], b[26], c[26], 1'b0, r[26], c[27], Op, Bneg);
alu1 alu_27( a[27], b[27], c[27], 1'b0, r[27], c[28], Op, Bneg);
alu1 alu_28( a[28], b[28], c[28], 1'b0, r[28], c[29], Op, Bneg);
alu1 alu_29( a[29], b[29], c[29], 1'b0, r[29], c[30], Op, Bneg);
alu1 alu_30( a[30], b[30], c[30], 1'b0, r[30], c[31], Op, Bneg);
alu1msb alu_31( a[31], b[31], c[31], 1'b0, r[31], c[32], Op, Bneg,
          Set, Overflow);

always @(r)
     AllZero = (r == 32'b0);

endmodule
```

-- tester --

```
module tester;

reg [31:0] a, b;
reg [1:0] Operation;
reg Bnegate;
wire [31:0] Result;
wire Overflow, AllZero;

alu32 alu( a, b, Result, Overflow, Operation, Bnegate, AllZero);

initial begin

     a = 'h00000000; b = 'h80000001; Bnegate = 1; Operation = 2;
     $strobe( "a=%h b=%h r=%h Bnegate=%h v=%h op=%h z=%h",
               a, b, Result, Bnegate, Overflow,
               Operation, AllZero);

#1
     a = 'h00000000; b = 'h80000000; Bnegate = 1; Operation = 2;
     $strobe( "a=%h b=%h r=%h Bnegate=%h v=%h op=%h z=%h",
               a, b, Result, Bnegate, Overflow,
               Operation, AllZero);
#1
     a = 'h00000001; b = 'hffffffff; Bnegate = 0; Operation = 2;
     $strobe( "a=%h b=%h r=%h Bnegate=%h v=%h op=%h z=%h",
               a, b, Result, Bnegate, Overflow,
               Operation, AllZero);
#1
     $finish;
end
endmodule
```