Homework4: CMPS 499 003 Embedded Systems Fall 2007

Date Assigned: Tuesday Oct 23, 2007
Date Due: Thursday, Oct 30, 2007

Objectives:

1. **Verilog modeling at DATA-FLOW level**
2. **Verilog modeling at GATE level**

Q1. Write a Verilog model for designing a 16-bit bit adder. The inputs to the adder are two 16-bit numbers and a 1-bit carry-in, while the outputs are 16-bit sum and a 1-bit carryout.

[Hint: Code for 4-bit adder is provided on the next two pages. Put the given two Verilog modules in two separate files, named "OneBitFullAdder.vl" and "Top.vl", respectively.
To compile: → iverilog –o mycompiledcode Top.vl OneBitFullAdder.vl
To run: → vvp mycompiledcode
]

Q2. We want to design our first simple ALU in Verilog. The inputs to the module are two 16-bit numbers, say A, and B; and also a 3-bit number, say *ControlCode*. Based on the ControlCode, the module performs operations, as follows:

Value of ControlCode (3 bit value)	Operation Performed
000	A+B (logical sum, not arithmetic sum)
001	A! + B!
010	A.B ('and' operation)
011	A exor B
Anyother value	No operation; display invalid message

The output, clearly, is a 16-bit number. [Hint: The code can be written with if-then or also case statements].

Q3. Develop a GATE-LEVEL Verilog design of the 1-bit adder in Q1 (the model in Q1 used data-flow level modeling). Test its operation for a 16-bit full adder.

Q4. Develop a GATE-LEVEL Verilog design for a 3-to-8 decoder and test its operation.

[Hint: A gate-level design of a 2:1 multiplexor is provided as an example]

```
/ CMPS 430 (UG)
// My first 1 bit full-adder in Verilog !!!
// The adder takes 3 1-bit numbers a, b and cin and produces Sum and CarryOut

// PLEASE NAME THIS FILE as "OneBitFullAdder.vl"


module OneBitFullAdder(a, b, cin, sum, cout);
input   a, b, cin;
output  sum, cout;


// use the logic equations for 1-bit full adder to compute sum and carryout

assign sum = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);


endmodule
```

```
// 4-bit Full Adder CMPS 430 (UG) Fall 2007
// The name of the file containing this code is Top.vl

module top();

reg [3:0] Ain;
reg [3:0] Bin;
reg  CarryIn;

wire [3:0] SumOut;
wire  c1, c2, c3, CarryOut;

OneBitFullAdder FBA0(Ain[0], Bin[0], CarryIn, SumOut[0], c1);
OneBitFullAdder FBA1(Ain[1], Bin[1], c1, SumOut[1], c2);
OneBitFullAdder FBA2(Ain[2], Bin[2], c2, SumOut[2], c3);
OneBitFullAdder FBA3(Ain[3], Bin[3], c3, SumOut[3], CarryOut);


initial
      begin
            Ain = 4'b1000;
            Bin = 4'b0001;
            CarryIn = 1'b0;

            #10;
            Ain = 4'b1000;
            Bin = 4'b0001;
            CarryIn = 1'b0;

            #10;
            Ain = 4'b1111;
            Bin = 4'b1111;
            CarryIn = 1'b1;

            #10;
            Ain = 4'b1001;
            Bin = 4'b1011;
            CarryIn = 1'b1;

 // you must do more assignments to test your adder better !!

            #10;
            $dumpflush;
      end


initial
      begin
      $monitor("Ain=%4b, Bin=%4b, Carryin=%b, SumOut=%4b, CarryOut=%b, time=
%t\n", Ain, Bin, CarryIn, SumOut, CarryOut, $time);
      $dumpfile("top.dump");
      $dumpvars(5, top);
end


endmodule
```

```
// The Verilog modules on this and the next page together demonstrate how to
// model a simple ALU using case statement


// alucode.vl
// CMPS 430 (UG)
// My first ALU Design!!
// The ALU takes 2 4-bit numbers A, B as well as a 1-bit control-code. If the
// control-code is 0 then A and B are ANDed else they are ORed by the ALU.



// You can name this file as "alucode.vl"


module alucode (Ain, Bin, Aluctl, Aluout);

input [3:0]Ain;
input [3:0]Bin;
input Aluctl;

output [3:0]Aluout;

reg [3:0] Aluout;

always @(Ain, Bin, Aluctl)
case (Aluctl)
0: Aluout <= Ain & Bin;
1: Aluout <= Ain | Bin;
default: $display ("invalid control code\n"); //print an error message
endcase

endmodule
```

```
//Top module, or testbench, for testing the ALU design
// You can name this file as "Top.vl"

module Top();

reg [3:0] Ainput;
reg [3:0] Binput;
reg ctlInput;

wire [3:0] whatisOut;


alucode MYALU(Ainput, Binput, ctlInput, whatisOut);

initial
      begin
            Ainput = 4'b1000;
            Binput = 4'b0001;
            ctlInput = 1'b0;

            #10;
            Ainput = 4'b1001;
            Binput = 4'b0011;
            ctlInput = 1'b0;

 // you must do more assignments to test better !!

            #10;
            $dumpflush;
      end


initial
begin
      $monitor("Ainput=%4b, Binput=%4b, ctlInput=%b, whatisOut=%4b,time=
%t\n", Ainput, Binput, ctlInput, whatisOut, $time);

end


endmodule
```

```
// CMPS 430 (UG)
// My first gate-level MUX Design!!
// The MUX takes 2 1-bit numbers Ain, Bin as well as a 1-bit control input
// ctl. If  ctl is 0 then output is the same as A, else it is the same as B .



// You can name this file as "muxcode.vl"


module muxcode (Ain, Bin, ctl, muxout);

input Ain;
input Bin;
input ctl;

output muxout;

//reg muxout;

wire w1;
wire w2;
wire w3;

not(w1, ctl);
and(w2, Ain, w1);
and(w3, Bin, ctl);
or(muxout, w2, w3);

endmodule
```

```
// You can name this file as "muxtop.vl"



module Muxtop();

reg Ainput;
reg Binput;
reg ctlInput;

wire muxOut;



muxcode MYMUX (Ainput, Binput, ctlInput, muxOut);


initial
      begin
            Ainput = 1'b1;
            Binput = 1'b0;
            ctlInput = 1'b0;

            #10;
            Ainput = 1'b1;
            Binput = 1'b0;
            ctlInput = 1'b1;

            #10;
            Ainput = 1'b1;
            Binput = 1'b0;
            ctlInput = 1'b0;

 // you must do more assignments to test better !!

            #10;
            $dumpflush;
      end


initial
begin
      $monitor("Ainput=%b, Binput=%b, ctlInput=%b, muxOut=%b,time=%t\n",
Ainput, Binput, ctlInput, muxOut, $time);

end


endmodule
```