COE 202, Term 201

Digital Logic Design

Assignment# 3 Solution

Due date: Saturday, Nov. 28, 2020

# You are required to design and implement an ALU with the following ports:

# A and B are inputs, and are n-bit signed numbers represented in 2’s complement

# F is 3-bit unsigned number for ALU function selector

# R is the signed ALU result represented in 2’s complement. You will have to determine the minimum number of bits required for R such that an overflow never occurs.

# Below is a table of ALU functions along with an illustration of its port interface

|  |  |  |
| --- | --- | --- |
| Function Code | ALU Result |  |
| F=000 | R=(A+B)/4 |
| F=001 | R=4\*(A+B) |
| F=010 | R=2\*A + B |
| F=011 | R=A – 2\*B |
| F=100 | R=A and B |
| F=101 | R= not(A) |
| F=110 | R=A or B |
| F=111 | R=A xor B |

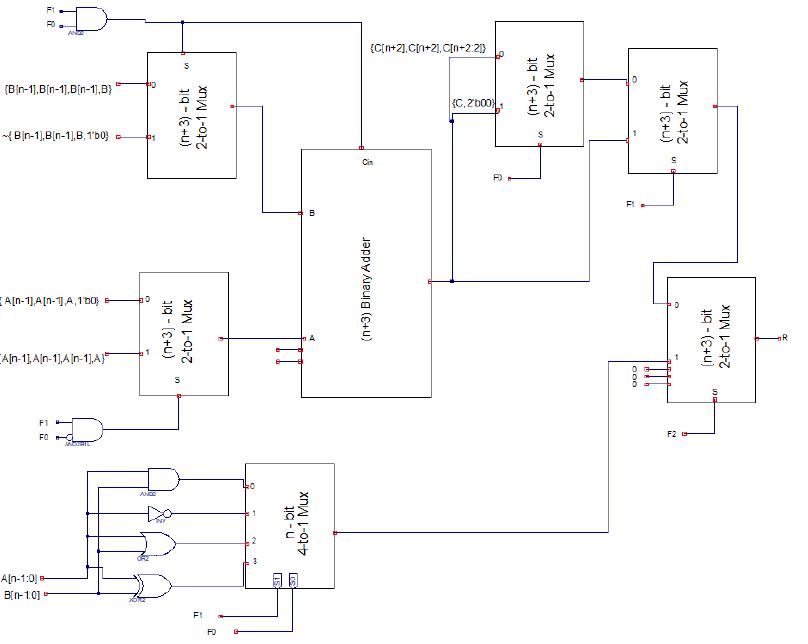
## (5 points) Determine the size of R in bits such that overflow can never occur.

The maximum value of R comes when A and B have the maximum positive value (2n-1-1) and F=001 🡺 F = 4\*2\*(2n-1-1) = 2n+2-8 🡺 needs n+3 bits to represent this value.

The minimum value of R comes when A and B have the minimum negative value (-2n-1) and F=001 🡺 F = 4\*2\*(-2n-1) = -2n+2 🡺 needs n+3 bits to represent this value.

🡺 to avoid overflow, R must have at least n+3-bits

## (10 points) Show the implementation of the ALU using MSI components and the minimum number of additional gates. Hint: you will have to use some form of extension (zero- or sign-extension).



## (10 points) Create behavioral Verilog modules for every MSI component you used in Part (b). Make sure the size of every module you design is parameterized, so that you can change your design easily during the testing phase.

module adder #(parameter w=4) (input [w-1:0] A,B, input Cin, output reg [w-1:0] S, output reg Cout);

always @(\*)

begin

{Cout,S}=A+B+Cin;

end

endmodule

module mux4to1 #(parameter w=4)( input [w-1:0] I0,I1,I2,I3,input [1:0] Sel, output reg [w-1:0] O);

always @(\*)

begin

case (Sel)

0: O=I0;

1: O=I1;

2: O=I2;

3: O=I3;

default: O=0;

endcase

end

endmodule

module mux2to1 #(parameter w=4) (input [w-1:0] I0,I1, input S, output [w-1:0] o);

assign o = (S)? I1:I0;

endmodule

## (5 points) Create a structural model for your ALU designed in Part (b) using the modules you derived in Part (c).

module ALU1 #(parameter n=4) (input [2:0] F, input [n-1:0] A,B, output [n+2:0] R);

wire [n+2:0] C, S, X, mux\_B, mux\_A;

wire [n-1:0] and\_AB, not\_A, or\_AB, xor\_AB, mux\_logic;

wire co;

mux2to1#(n+3) muxA ({A[n-1],A[n-1],A[n-1],A},{ A[n-1],A[n-1],A,1'b0},F[1]&~F[0],mux\_A);

mux2to1#(n+3) muxB ({B[n-1],B[n-1],B[n-1],B},~{ B[n-1],B[n-1],B,1'b0},F[1]&F[0],mux\_B);

adder#(n+3) ad1 (mux\_A,mux\_B,F[1]&F[0],C,co);

mux2to1#(n+3) muxC ({C[n+2],C[n+2],C[n+2:2]},{C[n:0],2'b00},F[0],S);

mux2to1#(n+3) muxS (S,C,F[1],X);

assign not\_A = ~A;

assign and\_AB = A&B;

assign or\_AB = A|B;

assign xor\_AB = A^B;

mux4to1#(n) muxLogic (and\_AB, not\_A, or\_AB, xor\_AB,F[1:0],mux\_logic);

mux2to1#(n+3) muxR (X,{3'b000,mux\_logic},F[2],R);

endmodule

## (5 points) Write a Verilog test bench that tests your ALU module in Part (d) assuming that A and B are 4-bits using the following input values:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | F | Expected output |
| 4 | 4 | 0 | R=2 |
| 2 | 1 | 1 | R=12 |
| -1 | -5 | 2 | R=-7 |
| -1 | -5 | 3 | R=+9 |
| -1 | -2 | 4 | R=+14 |
| -1 | -2 | 5 | R=0 |
| -1 | -2 | 6 | R=+15 |
| -1 | -2 | 7 | R=+1 |

## Allow a period of 20 ps between two consecutive test cases.

module ALU1tb();

reg [3:0] inA, inB;

reg [2:0] inF;

wire [6:0] outR;

ALU1 test\_m (inF,inA,inB,outR);

initial begin

inA=4; inB=4; inF=0;

#20 inA=2; inB=1; inF=1;

#20 inA=-1; inB=-5; inF=2;

#20 inA=-1; inB=-5; inF=3;

#20 inA=-1; inB=-2; inF=4;

#20 inA=-1; inB=-2; inF=5;

#20 inA=-1; inB=-2; inF=6;

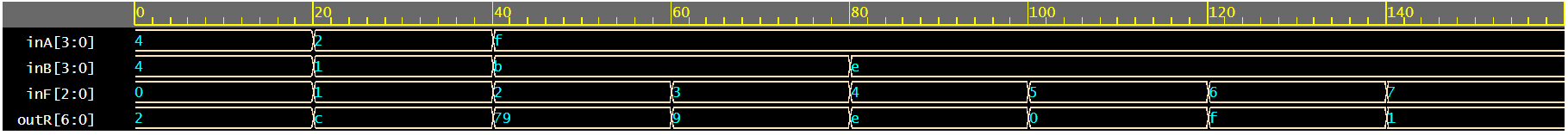
#20 inA=-1; inB=-2; inF=7;

#20 ;

end

endmodule

## The results of simulation are given below, which show that the correct output is obtained for all applied inputs:



## (5 points) Write a single behavioral Verilog module that models the given ALU.

module ALU2 #(parameter n=4) (input [2:0] F, input [n-1:0] A,B, output reg [n+2:0] R);

reg [n:0] add;

always@ (\*)

begin

add = {A[n-1], A} + {B[n-1], B};

case(F)

0: R= {add[n], add[n], add[n], add[n], add[n:2]};

1: R= {add[n:0],2'b00};

2: R= {A[n-1],A[n-1],A,1'b0}+ { B[n-1],B[n-1],B[n-1],B};

3: R= {A[n-1],A[n-1],A[n-1],A} - { B[n-1],B[n-1],B,1'b0};

4: R= A & B;

5: R= {3'b000,~A};

6: R= A | B;

7: R= A ^ B;

default: R=0;

endcase

end

endmodule

## (5 points) Write a Verilog test bench to test your behavioral ALU model in Part (f). Assuming that A and B are 4-bit, use the following input values:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | F | Expected output |
| 4 | 4 | 0 | R=2 |
| 2 | 1 | 1 | R=12 |
| -1 | -5 | 2 | R=-7 |
| -1 | -5 | 3 | R=+9 |
| -1 | -2 | 4 | R=+14 |
| -1 | -2 | 5 | R=0 |
| -1 | -2 | 6 | R=+15 |
| -1 | -2 | 7 | R=+1 |

## Allow a period of 20 ps between two consecutive test cases.

module ALU2tb();

reg [3:0] inA, inB;

reg [2:0] inF;

wire [6:0] outR;

ALU2 test (inF,inA,inB,outR);

initial begin

inA=4; inB=4; inF=0;

#20 inA=2; inB=1; inF=1;

#20 inA=-1; inB=-5; inF=2;

#20 inA=-1; inB=-5; inF=3;

#20 inA=-1; inB=-2; inF=4;

#20 inA=-1; inB=-2; inF=5;

#20 inA=-1; inB=-2; inF=6;

#20 inA=-1; inB=-2; inF=7;

#20 ;

end

endmodule

## The results of simulation are given below, which show that the correct output is obtained for all applied inputs:

## 

## (5 points) Submit a report (Word or PDF document) that should contain:

1. Problem description
2. Your answers to Parts (a) and (b)
3. All the behavioral models derived in Part (c), structural model in Part (d), and the test bench in Part (e)
4. The behavioral model in Part (f) and its test bench in Part (g)
5. The timing diagrams (waveforms) taken directly as snapshots from the simulator. Have as many snapshots as needed to cover all the test cases.

The assignment can be solved individually, or in groups of two students. Submit a soft copy of your solution on Blackboard in a zip file including your Verilog models.