Group Members:

Mihir Gajjar (1401076)

Vidit Shah (1401078)

Project Name: 4 – Bit ALU using Verilog.

Code:

ALU (Top) Module:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer: Mihir Gajjar (1401076) and Vidit Shah (1401078)

//

// Create Date: 17:55:07 04/20/2015

// Design Name: 4 - Bit ALU.

// Module Name: Alu

// Project Name: 4 - Bit Alu using Verilog

// Target Devices: Nexys 2.

// Tool versions: Xillinx 14.2

// Description: This is a Verilog Code for 4 - Bit ALU.

//

// Dependencies: This Bit file is generated using Xillinx 14.2 verion in windows 8.If there is any conflict please tell us.

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments: Here the outputs of all the five modules are set according the Opcodes and hence Precise Operations

// will take place.Here the inputs of the modules will increase(because of the Opcode) but the inputs

// of the Mux will decrease.Also the output of the other four modules other then the Mux module will decrease.

//

//////////////////////////////////////////////////////////////////////////////////

//Here we are combining all the Modules and using the wire we are giving the output of one module as an input to the other module.

module Alu(A,B,Op,C,V,Carry);

input [3:0]A; //4 - Bit Inputs of A,B and Op.

input [3:0]B;

input [3:0]Op;

output [3:0]C; //4-Bit Output from the Mux.

wire of; //wire to connect the of(overflow) from adder subtractor block to the comparator block.

wire [3:0]Sum,out,Outp,ooutp; //wire to connect the appropriate modules.

wire Car; //wire to connect the carry from adder subtractor block to the Shifter Block and Mux.

output V,Carry; //1-Bit Output of Overflow and Carry.

//Here we are creating instances of all the 5 modules.

Addf Add(A,B,Op,Car,Sum,of);

Comparatorf Comparator(Sum,of,Outp,ooutp,Op);

Logicalf Logical(A,B,Op,Outp);

Shifterf Shift(A[1:0],B,Car,Op,out);

Mux muxf(C,out,Sum,Outp,ooutp,Op,Car,Carry);

assign V = ((of)&(Op[1])&(~Op[3])); //Here we need overflow only for adder subtractor block and for all other blocks it will be set to zero.

endmodule

Adder Subtractor Module:

//Four Bit Adder Subtractor.

module Test(A,B,S,Cout,V,Op); //Collection of Inputs and Outputs.

input [3:0]A; //4 - Bit Input of A.

input [3:0]B; //4 - Bit Input of B.

input [3:0]Op; //4 - Bit Input of Opcode.

reg [3:0]X; // A 4- Bit register to take 1's Complement of the Subtrahend(input B).

output [3:0]S; //4-Bit Output of Addition or Subtraction depending upon the Opcode.

output Cout; //Single Bit Output Carry.

output V; //Single bit output of overflow.

wire [2:0]w; //Taking 3 bit wire for passing the carry to the next stage.

always @(\*)

begin

if(Op == 4'b0110 || Op == 4'b0111 || Op == 4'b1101 || Op == 4'b1111) //These are the opcodes for subtraction,greater than and less than.

//In the Comparator Block for greater than and less than we are using the MSB but of subtraction and the overflow and hence the Opcodes for

//greater than and less than are written.

X = (~B); //Here it will take the 1's Complement of the input and store it in the reg X when the above Opcodes are for Subtraction or greater then or less then.

else

X = B; //else the inputs will not be complemented.

end

onebitfulladder a0(A[0],X[0],Op[2],S[0],w[0]); //here if the Op[2] that is for subtraction is 1 then 1 will be taken as an input carry and it will be added to the one's complements and hence 2's complement will take place.Else if Op[2] will be 0 that is for addition then neither the ones complement will take place and neither 1 will be taken as an input carry and hence no change will take Place.

onebitfulladder a1(A[1],X[1],w[0],S[1],w[1]); //Passing the Carry via wire.

onebitfulladder a2(A[2],X[2],w[1],S[2],w[2]);

onebitfulladder a3(A[3],X[3],w[2],S[3],Cout);

assign V = w[2] ^ Cout ; //If last both carries are different then there will be overflow , if both the carries are same then then there will be no overflow and hence we are using a xor gate.

endmodule

//One Bit Full Adder.

module onebitfulladder(a,b,cin,s,cout);

input a,b,cin; //Taking one bit inputs.

output reg s,cout; //Output of Sum and Carry of one bit.

always @\*

begin

{cout,s } = (a + b + cin); //Here the sum will be assigned to the sum(s) and if there is a carry then it will be assigned to the carry(cout).

end

endmodule

//Four Bit Adder Subtracter.

//Main Module of Adder Subtractor Block.

module Addf(A,B,Op,Carry,C,V);

input [3:0]A; //Four Bit Inputs of A, B and Opcode.

input [3:0]B;

input [3:0]Op;

output Carry,V; // Output of 1 bit of Carry and overflow.

output [3:0] C; // Output of four bit of the Addition or Subtraction depending upon the Opcode.

Test fa(A,B,C,Carry,V,Op); //Creating an instance of the Test module.

Endmodule

Logical Module:

// This is the Logical Module.

//Main Module of Logical Module.

module Logicalf(A,B,Op,Outp); //Collection of inputs and outputs.

input [3:0]A; //4 - Bit inputs

input [3:0]B;

input [3:0]Op;

output [3:0]Outp; //4 - bit Output

reg [3:0]temp; //Taking a temporary register temp to temporary store the values.

always @ (\*)

begin

if(Op == 4'b1000) //OpCode for And Operation.

temp = (A&B);

else if(Op == 4'b1010) //OpCode for Or Operation.

temp = (A|B);

else if(Op == 4'b1100 || Op == 4'b1011 || Op == 4'b1001 || Op == 4'b1101 || Op == 4'b1111) //Here we have taken the Opcodes for Xor as well as the Opcodes for Comparator block because we need the Xor as an input in the comparator block.

temp = (A^B);

else if(Op == 4'b1110) //OpCode for Nor Operation.

temp = (~(A|B));

else

temp = 4'b0000; //Else the output will be zero for any other Opcodes.

end

assign Outp = temp; //Assigning our output Outp to temp which was used for temporary holding the values.

endmodule

Comparator Module:

//This is the Comparator Block.Here we have used the xor output from the logical block and the subtractor output and the overflow output from the adder subtractor block to compare A and B.

//Main Module of comparator block

module Comparatorf(Subtract,of,Xor,outp,Op); //Collections of Inputs and outputs

input [3:0]Subtract,Xor,Op; //Taking 4-bit input of the opcode and one input from logical block and other from Adder Subtractor Block

input of; //1 Bit input of Overflow from Adder Subtractor Block.

reg AgB; //1 Bit Register for A>B Condition

reg AlB; //1 Bit Register for A<B Condition

reg AeB; //1 Bit Register for A=B Condition

reg AneB; //1 Bit Register for A!=B Condition

reg Nor; //1 Bit register for storing nor value of Xor.

output reg [3:0]outp; //4 - Bit Output.

reg compare; //1 Bit register for storing the xor(ed) value of the MSB of the Subtractor input and the overflow.

always @(\*)

begin

compare = (of^Subtract[3]); //XOR operation of Overflow and MSB of Subtractor input.

Nor = ~|Xor; //Nor Operation of all the four bits of the Xor input.

if(Nor == 1'b1) //If the Nor value is 1 then A will be equal to B hence AequalB is 1 and all other registers are 0.

begin

AgB = 0;

AlB = 0;

AeB = 1;

AneB =0;

end

else //Else AnotequalB will be 1.

begin

AeB =0;

AneB =1;

AgB = 0;

AlB = 0;

end

if(AneB && compare) //Now if A is Not equal to B and compare is 1 then A will be less then B.

begin

AlB = 1;

AeB =0;

AneB =1;

AgB = 0;

end

else if (AneB && (~compare)) //Now if A is Not equal to B and compare is 0 then A will be greater then B.

begin

AlB = 0;

AeB =0;

AneB =1;

AgB = 1;

end

if(Op == 4'b1001) //Now here depending upon the Opcode we are giving the outputs.We are assigning the zeroth bit to the required output and all other bits to 0.

begin

outp[0] = AeB;

outp[3:1]=3'b000;

end

else if(Op == 4'b1011)

begin

outp[0] = AneB;

outp[3:1]=3'b000;

end

else if(Op == 4'b1101)

begin

outp[0] = AgB;

outp[3:1]=3'b000;

end

else if(Op == 4'b1111)

begin

outp[3:1]=3'b000;

outp[0] = AlB;

end

else //Else in every other Opcode the output will be zero.

outp = 4'b0000;

end

endmodule

Shifter Module:

//There is only One module for shifter Block.Here we are shifting the input B.0th bit and 1th bit of input A is taken as Sa.

//Here we are shifting 0 bit for Sa=00 , 1 bit for Sa=01 , 2 bit for Sa = 10 and 3 bit for Sa = 11.

//Main Module.

module Shifterf(Sa,B,Cin,Op,fout); //Collection of inputs and outputs.

input [1:0]Sa; //Here Sa is of 2 bits for selecting the number of bits we want to shift.

input [3:0]B; //4-bit input B and Opcode.

input [3:0]Op;

input Cin; //Cin from the adder Subtractor Block.

reg [3:0] X; //A 4 bit register for storing the flipped value of B temporarily.

reg [3:0] out1; //A 4 bit registers for temporarily storing the output.

reg [3:0] outp;

reg [3:0] out;

output [3:0]fout; //4-Bit Shifted Output.

reg temp; //A temporary 1 bit register for selecting the value which we want to fill in the (Shifted) vacancies.

//Flipping the inputs for right shifting

always @ (\*)

//Flipping the inputs if we want to right shift else storing the same value in the temporary register X.

begin

if(Op[2] == 0) //For Left Shift Op[2] will be 0.

X = B;

else if(Op[2] == 1) //For Right Shift Op[2] will be 1.

begin

X[3] = B[0];

X[2] = B[1];

X[1] = B[2];

X[0] = B[3];

end

//Storing the value to be filled in the shifted vacancies to temp.

if(Op == 4'b0101) //For arithmetic storing the MSB of the input B.

temp = X[0]; //Here for arithmetic we have to fill the MSB of the input B.But the inputs are flipped for arithmetic right shift and hence we have used X[0] which is the MSB of the input B.

else

temp = Cin; //else carry will be filled in the shifted vacancies.

//Left Shift

if(Sa == 2'b00) //Shifting the input as per Sa.

out = X;

if(Sa == 2'b01)

begin

out[0] = temp;

out[1] = X[0];

out[2] = X[1];

out[3] = X[2];

end

if(Sa == 2'b10)

begin

out[0] = temp;

out[1] = temp;

out[2] = X[0];

out[3] = X[1];

end

if(Sa == 2'b11)

begin

out[0] = temp;

out[1] = temp;

out[2] = temp;

out[3] = X[0];

end

//Here again if the Opcode is for right shift then we are flipping the left shifted output to perform the right shifting.

if(Op[2] == 1)

begin

outp[3] = out[0];

outp[2] = out[1];

outp[1] = out[2];

outp[0] = out[3];

end

else if(Op[2] == 0) //If the Opcode is for Left shift then no flipping will take place.

begin

outp = out;

end

end

assign fout = outp; //Assigning the outp to the final Output fout.

Endmodule

MUX Module:

//This is the module of MUX.

//Here we are setting our final output according the opcodes.

//The output of all the 4 blocks are set according to the Opcodes and hence we need to set the final output according to the blocks.

module Mux(outp,shifter,adder,logical,comparator,Op,Cin,Cout);

input [3:0]shifter,adder,logical,comparator,Op; //Taking the 4-Bit Outputs from all the four blocks and also the Opcode of 4 Bits as Inputs.

input Cin; //Taking 1 bit Carry as Input.

output reg Cout; // 1 - bit output register of Carry.

output reg [3:0]outp; //4 - bit otuput register of final Output.

//According to the Opcodes the Final Output and the Carry will be assigned.

always @(\*)

begin

if(Op == 4'b0000 || Op == 4'b0001 || Op == 4'b0100 || Op == 4'b0101) //Opcodes for Shifter Block

begin

outp = shifter;

Cout = Cin;

end

else if(Op == 4'b0010 || Op == 4'b0011 || Op == 4'b0110 || Op == 4'b0111) //Opcodes for Adder Subtractor Block

begin

outp = adder;

Cout = Cin;

end

else if(Op == 4'b1000 || Op == 4'b1010 || Op == 4'b1100 || Op == 4'b1110) //Opcodes for Logical Block

begin

outp = logical;

Cout = 0;

end

else if(Op == 4'b1001 || Op == 4'b1011 || Op == 4'b1101 || Op == 4'b1111) //Opcodes for Comparator Block

begin

outp = comparator;

Cout = Cin;

end

else //Else the Output and the Carry will be zero.

begin

outp = 4'b0000;

Cout = 0;

end

end

endmodule

UCF:

NET "A[0]" LOC = G18;

NET "A[1]" LOC = H18;

NET "A[2]" LOC = K18;

NET "A[3]" LOC = K17;

NET "B[0]" LOC = L14;

NET "B[1]" LOC = L13;

NET "B[2]" LOC = N17;

NET "B[3]" LOC = R17;

NET "C[0]" LOC = J14;

NET "C[1]" LOC = J15;

NET "C[2]" LOC = K15;

NET "C[3]" LOC = K14;

NET "Carry" LOC =E17;

NET "V" LOC =P15;

NET "Op[0]" LOC = B18;

NET "Op[1]" LOC = D18;

NET "Op[2]" LOC = E18;

NET "Op[3]" LOC = H13;