6.1

`timescale 1ns / 1ps

`default\_nettype none

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

// Company:

// Engineer:

//

// Create Date: 15:21:04 10/10/2016

// Design Name:

// Module Name: two\_one\_mux

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module two\_one\_mux(Y, A, B, S);

//declare output and input wires

output wire Y;

input wire A, B, S;

// Declare internal nets

wire notS; // WIre to carry not S

wire andA; // Wire to carry andA

wire andB; // Wire to carry andB

// instatiate gate-level modules

not not0(notS, S); // invert S

and and0(andA, notS, A); // and of notS and A

and and1(andB, S, B); // and of S and B

or or0(Y, andA, andB); // or andA and and B

endmodule

Experimept 2.1

`timescale 1ns / 1ps

`default\_nettype none

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

// Company:

// Engineer:

//

// Create Date: 15:52:35 10/10/2016

// Design Name:

// Module Name: four\_bit\_mux

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module four\_bit\_mux(Y,A,B,S);

// Declare output and input prots

output wire [3:0] Y; // Y output wire is 4 bit wide

input wire [3:0] A, B; // A and B are 4-bit wires

input wire S; // Select wire is 1 bit wide

// Initialize user-defined modules

two\_one\_mux MUX0(Y[0], A[0], B[0], S);

two\_one\_mux MUX1(Y[1], A[1], B[1], S);

two\_one\_mux MUX2(Y[2], A[2], B[2], S);

two\_one\_mux MUX3(Y[3], A[3], B[3], S);

Endmodule

6.3

`timescale 1ns / 1ps

`default\_nettype none

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

// Company:

// Engineer:

//

// Create Date: 16:06:23 10/10/2016

// Design Name:

// Module Name: full\_adder

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module full\_adder(S, Cout, A, B, Cin );

// Declare input and output ports

input wire A, B, Cin;

output wire S, Cout;

// Declare wires

wire andBCin, andACin, andAB; // add more

// Use dataflow to create gatelevel commands

assign S = A ^ B ^ Cin; // ^ is XOR

assign andAB = A & B;

assign andBCin = B & Cin;

assign andACin = A & Cin;

assign Cout = andAB | andBCin | andACin;

endmodule

6.4 Ripple carry adder

`timescale 1ns / 1ps

`default\_nettype none

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

// Company:

// Engineer:

//

// Create Date: 16:24:12 10/10/2016

// Design Name:

// Module Name: add\_sub

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module add\_sub(

//Declare output and input

output wire [3:0] Sum, // 4-bit reulst

output wire Overflow, // 1-bit wire for overflow

input wire [3:0] opA, opB, // 4-bit opperands

input wire opSel // if opSel =1, then subtractadd\_sub

);

// declare internal nets

wire [3:0] notB;

wire c0, c1, c2, c3;

// Create complement logic

assign notB[0] = opB[0] ^ opSel;

assign notB[1] = opB[1] ^ opSel;

assign notB[2] = opB[2] ^ opSel;

assign notB[3] = opB[3] ^ opSel;

// full adders to create a ripple carry adder

full\_adder adder0(Sum[0], c0, opA[0], notB[0], opSel);

full\_adder adder1(Sum[1], c1, opA[1], notB[1], c0);

full\_adder adder2(Sum[2], c2, opA[2], notB[2], c1);

full\_adder adder3(Sum[3], c3, opA[3], notB[3], c2);

// Overflow detection logic

assign Overflow = c2 ^ c3;

endmodule

Experiment 3

`timescale 1ns / 1ps

`default\_nettype none

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

// Company:

// Engineer:

//

// Create Date: 16:47:31 10/10/2016

// Design Name:

// Module Name: four\_bit\_alu

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module four\_bit\_alu(

output wire [3:0] Result, // 4-bit output

output wire Overflow,

input wire [3:0] opA, opB,

input wire [1:0] ctrl // 2 bit operation select

);

wire [3:0] Sum;

wire [3:0] andAB;

wire add\_sub0;

wire OverflowInit;

add\_sub addsub(Sum, OverflowInit, opA, opB, ctrl[1]);

assign Overflow = OverflowInit & ctrl[0];

assign andAB[0] = opA[0] & opB[0];

assign andAB[1] = opA[1] & opB[1];

assign andAB[2] = opA[2] & opB[2];

assign andAB[3] = opA[3] & opB[3];

four\_bit\_mux four\_bit\_mux0(Result,andAB,Sum,ctrl[0]);

/\* four\_bit\_mux1 four\_bit\_mux(Result[1],opA[1],opB[1],S);

four\_bit\_mux2 four\_bit\_mux(Result[2],opA[2],opB[2],S);

four\_bit\_mux3 four\_bit\_mux(Result[3],opA[3],opB[3],S);\*/

endmodule