Report breakdown

//Report A

Joseph Thornton

Professor Faroughi

Section #3

Project 2: 2s compliment add/subtract

Spring 2017

**//Report C all below //Report B**

A7:0 B7:0 c-1

**//2 hybrid 8bit CLA CPA adder**

8bit CLA

module cla8bit(

input ci,

A3:2 B3:2

A1:0 B1:0

input [7:0] a,b,

output c6,c7,

C0,C1

2bit CLA

2bit CLA

output [7:0] s

);

wire [7:0] cs;

C3,C2

cla2bit cla1(ci,a[1:0],b[1:0],cs[0],cs[1],s[1:0]);

cla2bit cla2(cs[1],a[3:2],b[3:2],cs[2],cs[3],s[3:2]);

cla2bit cla3(cs[3],a[5:4],b[5:4],cs[4],cs[5],s[5:4]);

A5:4 B5:4

A7:6 B7:6

cla2bit cla4(cs[5],a[7:6],b[7:6],cs[6],cs[7],s[7:6]);

2bit CLA

2bit CLA

assign c6=cs[6];

C5,C4

assign c7=cs[7];

endmodule

S7:0 C7 C6

**//1.D combine to make 2bit cla //Report B**

A1B1 A0B0 c-1

2bit CLA

module cla2bit(

input ci,

2bit PG Unit

input [1:0] a,b,

output c0,cn,

G1P1 G0 P0

output [1:0] s

);

2bit CG unit

wire [1:0] p,g,c;

pgu p1(a,b,p,g);

C1 C0

cgu c1(ci,p,g,c);

2bit Sum Unit

au a1(ci,p,c,s);

assign c0=c[0];

S1S0 C1 C0

assign cn=c[1];

endmodule

**//1.c 2bit cgu**

|  |  |  |  |
| --- | --- | --- | --- |
| G0 | P0 | c-1 | C0 |
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **0** |
| 0 | 1 | 0 | **0** |
| 0 | 1 | 1 | **1** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **1** |
| 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | **1** |

Sop(co)={3,4,5,6,7} co(bar)={0,1,2} = ~P0~G0+~G0~C-1 Pos(co)=(P0+G0)(c-1+G0)used Kmap to simplify.

Sop(c1)=g1+g0p1+P1P0C-1

module cgu(

input ci,

input [1:0] p,g,

output [1:0] c

);

wire [3:0] r;

assign r[0] = p[0]|g[0];

assign r[1] = g[0]|ci;

assign c[0] = r[0]&r[1];

assign r[2] = ci&p[0]&p[1];

assign r[3] = p[1]&g[0];

assign c[1] = g[1]|r[2]|r[3];

endmodule

**//1.b 2bit pgu**

module pgu(

input [1:0] a,b,

output [1:0] p,g

);

xlogic x1(a[0],b[0],p[0]);

xlogic x2(a[1],b[1],p[1]);

and a1(g[0],a[0],b[0]);

and a2(g[1],a[1],b[1]);

endmodule

**//1.a xor gate**

|  |  |  |
| --- | --- | --- |
| A0 | B0 | F |
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **0** |

Sop (f)={1,2} f(bar) = {0,3} = A(not)B(not)+AB POS(f)=(A+B)(A(not)+B(not))

module xlogic(

input a,b,

output f

);

//xor logic pos implimented

wire o1, o2, o3, o4;

nor nor1(o1,a,b);

not n1(o2,a);

not n2(o3,b);

nor nor2(o4,o2,o3);

nor nor3(f,o4,o1);

endmodule

**//1.b 2 bit su unit**

module au(

input ci,

input [1:0] p,c,

output [1:0] s

);

assign s[0]=ci^p[0];

assign s[1]=c[0]^p[1];

endmodule

A7:0 B7:0 m

**//3.b 2s compliment adder/subtractor //Report B**

module complimentaddsub(

B7:0

input ci,m,

Inverter

input [7:0] a,b,

output [7:0] answer,

output ovf

A7:0 Bi7:0

);

CI

8bit CLA

wire [7:0] bi;

wire c6,c7;

inverter inv1(b,m,bi);

C7,C6

XOR

cla8bit cla8(ci,a,bi,c6,c7,answer);

xlogic xlog6(c6,c7,ovf);

endmodule

Result OVF

**//3.a 8bit inverter**

module inverter(

input [7:0] invert,

input m,

output reg [7:0] inverted

);

reg [7:0] i;

always@\*

for(i=8'b0;i<8'b1000;i=i+1'b1)begin

if(m==1)begin

inverted[i]=~invert[i];

end else begin

inverted[i]=invert[i];

end

end

endmodule

**//Test bench code**

`include "/gaia/class/student/thorntjl/csc137/proj2/cla8bit.v"

module testbench();

reg [7:0] a,b;

wire [7:0] answer;

reg m;

wire ovf;

complimentaddsub t1(m,m,a,b,answer,ovf);

initial begin

$display("\n\nJoseph Thornton Section3\n\tA\tB\tM\tOVF\tANSWER");

$monitor("\t\t\t\t%b\t%8b",ovf,answer);

a=8'hFF;b=8'h01;m=1'b0; $display("%8b %8b\t%b",a,b,m);

//test1

#10

a=8'h7F;b=8'h01;m=1'b0; $display("%8b %8b\t%b",a,b,m);

//test2

#10

a=8'h01;b=8'hFF;m=1'b0; $display("%8b %8b\t%b",a,b,m);

//test3

#10

a=8'h55;b=8'hAA;m=1'b0; $display("%8b %8b\t%b",a,b,m);

//test4

#10

a=8'h80;b=8'h01;m=1'b1; $display("%8b %8b\t%b",a,b,m);

//test5

#10

a=8'h6C;b=8'hCA;m=1'b1; $display("%8b %8b\t%b",a,b,m);

//test 6

#10

$finish;

end

endmodule

**//Results**

[thorntjl@athena:22]> simv

Chronologic VCS simulator copyright 1991-2014

Contains Synopsys proprietary information.

Compiler version I-2014.03-2; Runtime version I-2014.03-2; Apr 1 16:40 2017

Joseph Thornton Section3

A B M OVF ANSWER

11111111 00000001 0

0 00000000

01111111 00000001 0

1 10000000

00000001 11111111 0

0 00000000

01010101 10101010 0

0 11111111

10000000 00000001 1

1 01111111

01101100 11001010 1

1 10100010

$finish called from file "benchtest.v", line 29.

$finish at simulation time 60

V C S S i m u l a t i o n R e p o r t

**Report D)**

//by hand

11111111 01111111 00000001 01010101 100000000 01101100

00000001 00000001 11111111 10101010 000000001 11001010

100000000 10000000 100000000 11111111 10100010 010100010

111111110 011111110 111111110 000000001 100000001 011100011

Result C6,C7(matching no overflow) Other carry

Even with arithmetic adding over edge of a register the overflow flag will not be set this is because this only knows cares about signed arithmetic and not unsigned.