**하드웨어 시스템 설계 4주차 실습 보고서**

2017-12751 컴퓨터공학부 이동학

Goal: Implement 32bit integer multiply adder, 32bit floating point multiply adder by IP catalog, and adder array consists of 4 adders by Verilog.

Code:

Tb\_multadd.v

module tb\_multadd();

reg [32-1:0] ain;

reg [32-1:0] bin;

reg [64-1:0] cin;

reg rst;

reg clk;

wire [64-1:0] res;

integer I;

initial begin

clk <= 0;

rst <= 0;

for (i=0 ; i<32 ; i=i+1) begin

ain = $urandom%(2\*\*31);

bin = $urandom%(2\*\*31);

cin = $urandom%(2\*\*63);

#20;

end

end

always #5 clk = ~clk;

xbip\_multadd\_0 UTU(

.CLK(clk),

.CE(1'b1),

.SCLR(rst),

.SUBTRACT(1'b0),

.A(ain),

.B(bin),

.C(cin),

.P(res)

);

Endmodule

Tb\_floating.v

module tb\_floating();

reg [32-1:0] ain;

reg [32-1:0] bin;

reg [32-1:0] cin;

reg rst;

reg clk;

wire [32-1:0] res;

wire dvalid;

integer i;

initial begin

clk <= 0;

rst <= 0;

for (i=0 ; i<32 ; i=i+1) begin

ain = $urandom%(2\*\*31);

bin = $urandom%(2\*\*31);

cin = $urandom%(2\*\*31);

#20;

end

end

always #5 clk = ~clk;

floating\_point\_0 UUT(

.aclk(clk),

.aresetn(~rst),

.s\_axis\_a\_tvalid(1'b1),

.s\_axis\_b\_tvalid(1'b1),

.s\_axis\_c\_tvalid(1'b1),

.s\_axis\_a\_tdata(ain),

.s\_axis\_b\_tdata(bin),

.s\_axis\_c\_tdata(cin),

.m\_axis\_result\_tvalid(dvalid),

.m\_axis\_result\_tdata(res)

);

Endmodule

Adder\_array.v

module adder\_array(

cmd, ain0, ain1, ain2, ain3, bin0, bin1, bin2, bin3,

dout0, dout1, dout2, dout3, overflow);

input [2:0] cmd;

input [31:0] ain0, ain1, ain2, ain3;

input [31:0] bin0, bin1, bin2, bin3;

output [31:0] dout0, dout1, dout2, dout3;

output [3:0] overflow;

wire [31:0] ain [3:0];

wire [31:0] bin [3:0];

reg [31:0] dout [3:0];

reg [3:0] overflowt;

wire [31:0] tempout [3:0];

wire [3:0] tempover;

assign {ain[0], ain[1], ain[2], ain[3]} = {ain0, ain1, ain2, ain3};

assign {bin[0], bin[1], bin[2], bin[3]} = {bin0, bin1, bin2, bin3};

assign {dout0, dout1, dout2, dout3} = {dout[0], dout[1], dout[2], dout[3]};

assign overflow = overflowt;

genvar i;

generate for (i=0 ; i < 4 ; i=i+1) begin

my\_add adder(.ain(ain[i]), .bin(bin[i]), .dout(tempout[i]), .overflow(tempover[i]));

end

endgenerate

always@(\*) begin

if(cmd == 3'b000) begin

dout[0] = tempout[0]; dout[1] = 0; dout[2] = 0; dout[3] = 0;

overflowt[0] = tempover[0]; overflowt[1] = 0; overflowt[2] = 0; overflowt[3] = 0;

end

else if(cmd == 3'b001) begin

dout[0] = 0; dout[1] = tempout[1]; dout[2] = 0; dout[3] = 0;

overflowt[0] = 0; overflowt[1] = tempover[1]; overflowt[2] = 0; overflowt[3] = 0;

end

else if(cmd == 3'b010) begin

dout[0] = tempout[0]; dout[1] = 0; dout[2] = tempout[2]; dout[3] = 0;

overflowt[0] = 0; overflowt[1] = 0; overflowt[2] = tempover[2]; overflowt[3] = 0;

end

else if(cmd == 3'b011) begin

dout[0] = 0; dout[1] = 0; dout[2] = 0; dout[3] = tempout[3];

overflowt[0] = 0; overflowt[1] = 0; overflowt[2] = 0; overflowt[3] = tempover[3];

end

else if(cmd == 3'b100) begin

dout[0] = tempout[0]; dout[1] = tempout[1]; dout[2] = tempout[2]; dout[3] = tempout[3];

overflowt[0] = tempover[0]; overflowt[1] = tempover[1]; overflowt[2] = tempover[2]; overflowt[3] = tempover[3];

end

end

endmodule

tb\_addarray.b

module tb\_addarray();

parameter BITWIDTH = 32;

reg [2:0] cmd;

reg [BITWIDTH-1:0] ain0, ain1, ain2, ain3;

reg [BITWIDTH-1:0] bin0, bin1, bin2, bin3;

wire [BITWIDTH-1:0] dout0, dout1, dout2, dout3;

wire [3:0] overflow;

integer i, j;

initial begin

for(i=0 ; i<5 ; i=i+1) begin

cmd = i;

for(j=0 ; j<4 ; j=j+1) begin

ain0 = $urandom%(2\*\*31);

ain1 = $urandom%(2\*\*31);

ain2 = $urandom%(2\*\*31);

ain3 = $urandom%(2\*\*31);

bin0 = $urandom%(2\*\*31);

bin1 = $urandom%(2\*\*31);

bin2 = $urandom%(2\*\*31);

bin3 = $urandom%(2\*\*31);

#10;

end

end

end

adder\_array adder(

.cmd(cmd),

.ain0(ain0),

.ain1(ain1),

.ain2(ain2),

.ain3(ain3),

.bin0(bin0),

.bin1(bin1),

.bin2(bin2),

.bin3(bin3),

.dout0(dout0),

.dout1(dout1),

.dout2(dout2),

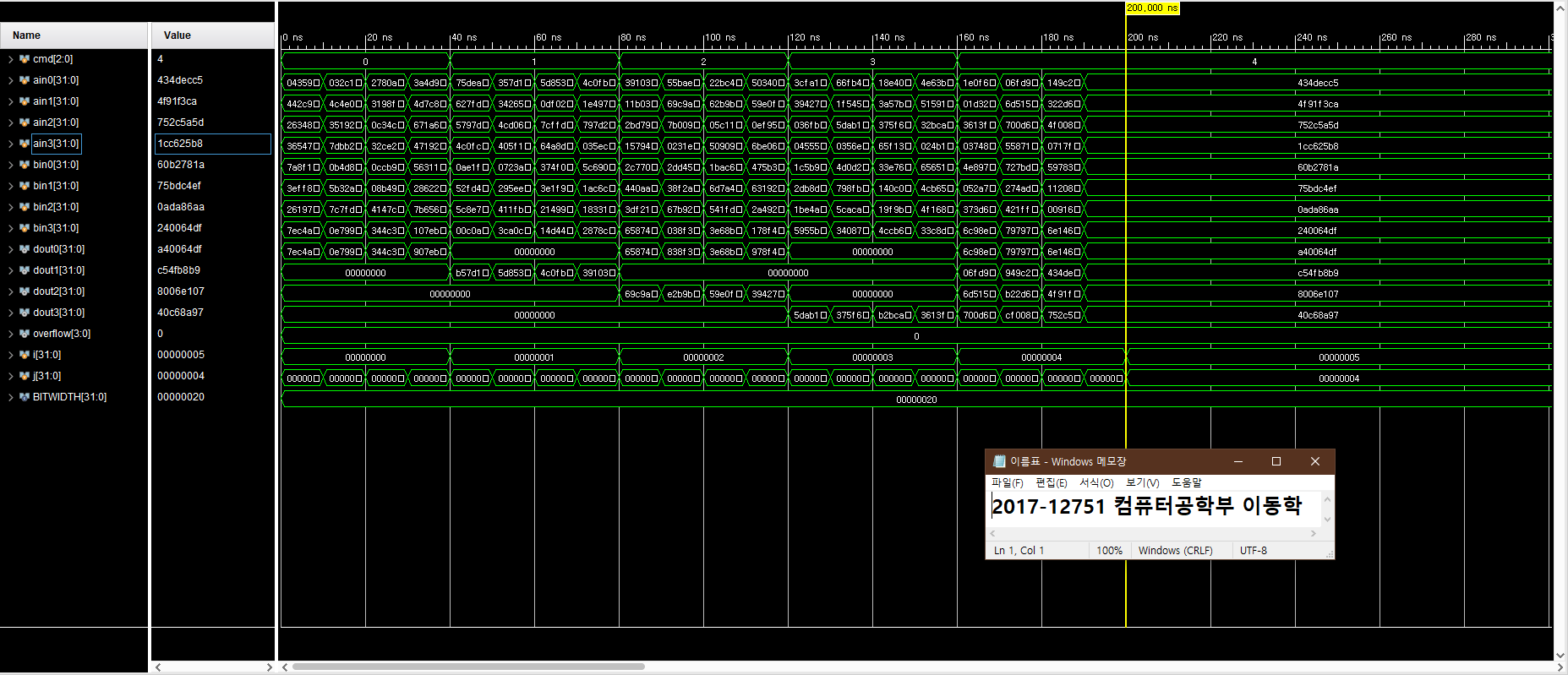
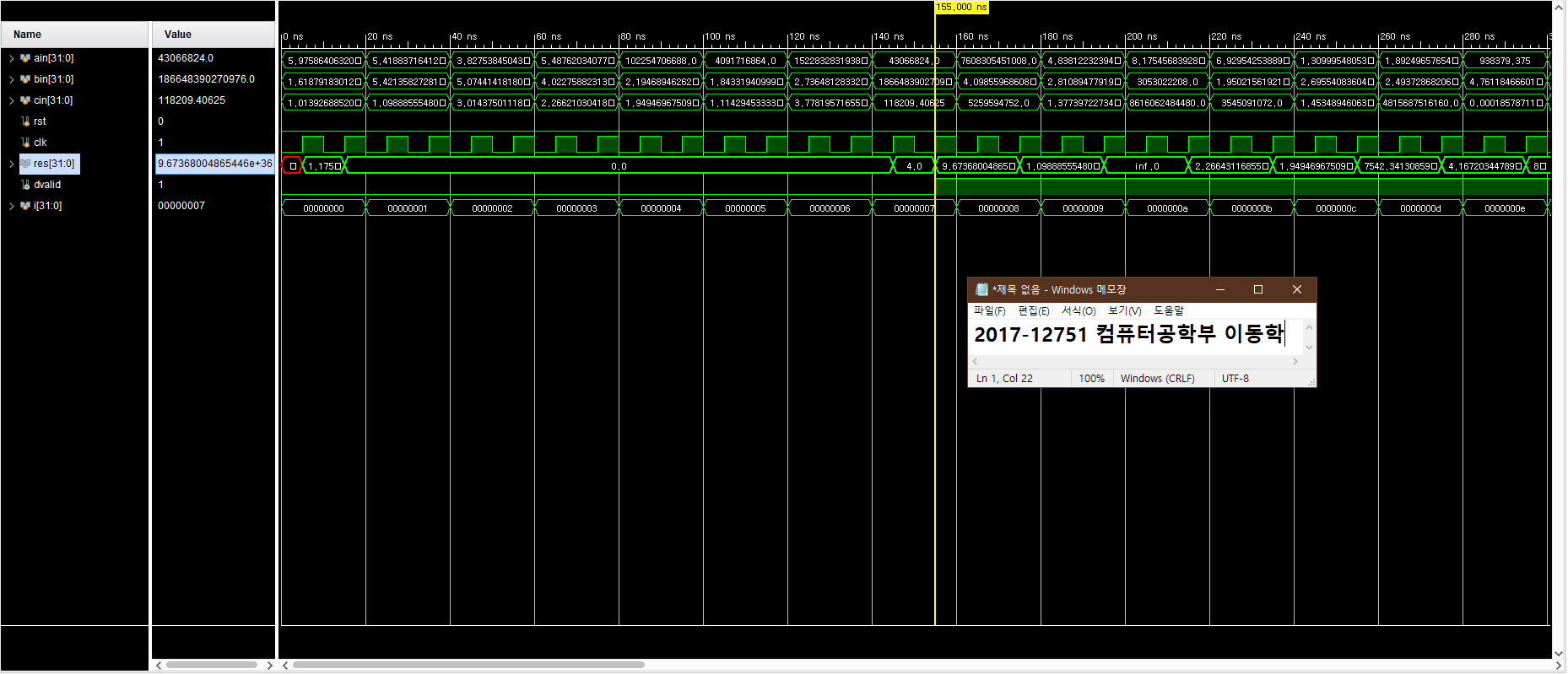
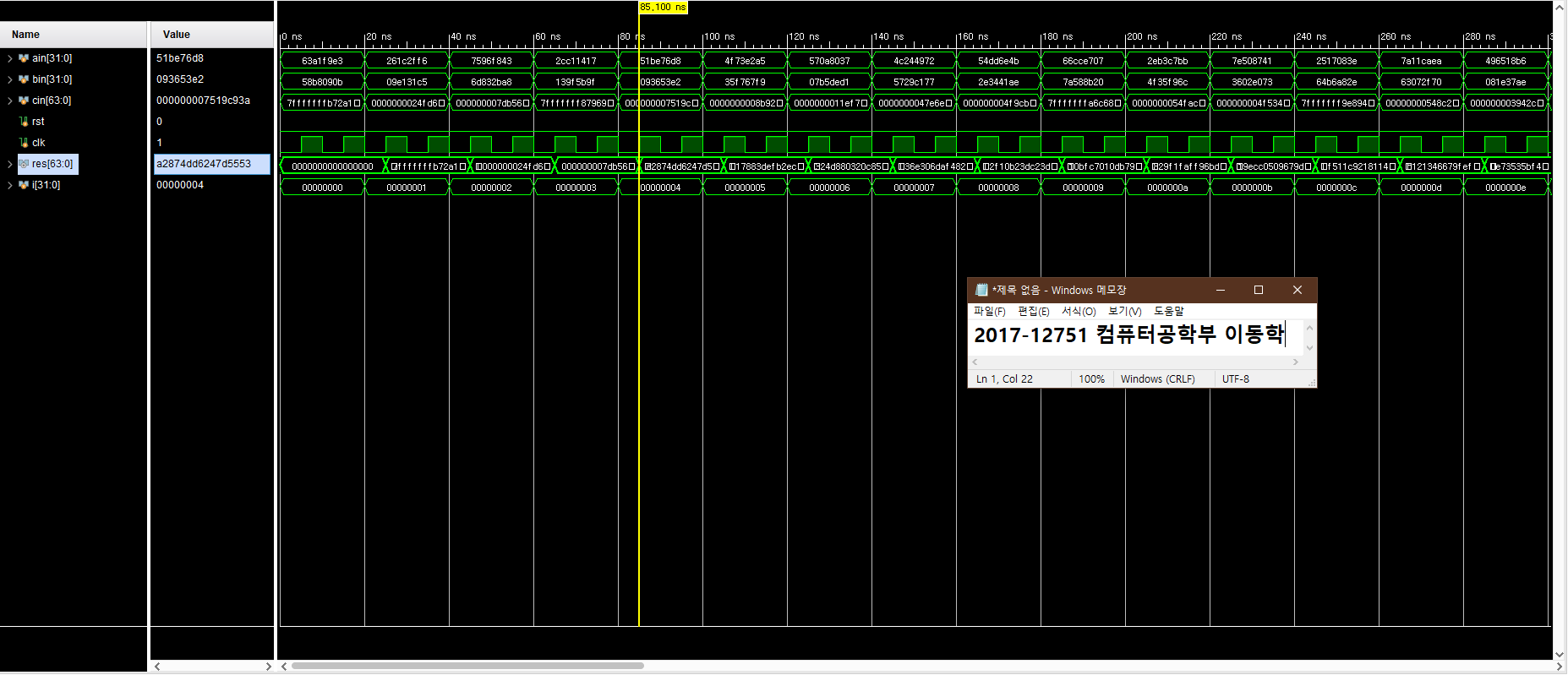
.dout3(dout3),

.overflow(overflow)

);

endmodule

Result: 32bit integer multiply adder / 32bit floating point multiply adder / adder array



Discussion: floating point multiply adder의 경우에는 튜토리얼을 참고하여 구현한 결과, 입력이 주어지고 15cycle 후에 계산 결과가 나왔습니다. Integer multiply adder의 경우에는 A와 B는 unsigned 32bit, C와 P는 unsigned 64bit로 설정하였고 A:B – P Latency와 C – P Latency는 둘다 -1로 설정한 결과, C의 값이 2cycle 후에, A\*B+C의 값이 8cycle 후에 나왔습니다.

Adder array의 경우, 임시 wire와 reg 배열을 선언하여 input과 output을 generate for문으로 연산하기 쉽게 해주었습니다. 그 후 always문을 이용하여 cmd값에 따라 dout과 overflow 값을 필터링 해주었습니다. For문 안에서 cmd값을 이용해 필터링을 편하게 할 수 있는 방법을 고민했으나, generate문과 always문, module instantiation 사이에 충돌이 발생하여 일일이 필터링을 하였습니다. 각 cmd 값마다 4개의 벡터를 이용하여 테스트해본 결과, 정상 작동 하는 것을 알 수 있습니다.