**하드웨어 시스템 설계 6주차 실습 보고서**

2017-12751 컴퓨터공학부 이동학

Goal: Implement PE controller based on PE made in lab5

Code:

pe\_tb

module pe\_tb();

parameter L\_RAM\_SIZE = 4;

reg start;

wire done;

reg aclk;

reg aresetn;

wire [L\_RAM\_SIZE:0] rdaddr;

reg [31:0] rddata;

wire [31:0] wrdata;

reg [31:0] in [0:31];

initial begin

aclk <= 0;

start <= 0;

aresetn <= 0;

$readmemh("din.txt", in);

#10;

start <= 1;

aresetn <= 1;

#10;

start <= 0;

#3300;

aresetn <= 0;

end

always #5 aclk <= ~aclk;

always @(negedge aclk) begin

rddata <= in[rdaddr];

end

pe\_ctrl pe(

.start(start),

.done(done),

.aclk(aclk),

.aresetn(aresetn),

.rdaddr(rdaddr),

.rddata(rddata),

.wrdata(wrdata)

);

endmodule

pe\_ctrl

module pe\_ctrl#(

parameter VECTOR\_SIZE = 16,

parameter L\_RAM\_SIZE = 4

)

(

input start,

output done,

input aclk,

input aresetn,

output [L\_RAM\_SIZE:0] rdaddr,

input [31:0] rddata,

output reg [31:0] wrdata

);

wire [31:0] ain;

wire [31:0] din;

wire [L\_RAM\_SIZE-1:0] addr;

wire we;

wire global\_we;

wire valid;

wire dvalid;

wire [31:0] dout;

reg [31:0] global\_dout;

(\* ram\_style = "block" \*) reg [31:0] global\_buffer [0:VECTOR\_SIZE-1];

reg [3:0] state, state\_d;

wire LOAD\_END;

wire CALC\_END;

wire DONE\_END;

localparam S\_IDLE = 4'd0;

localparam S\_LOAD = 4'd1;

localparam S\_CALC = 4'd2;

localparam S\_DONE = 4'd3;

always @(posedge aclk)

if (!aresetn) begin

state <= S\_IDLE;

state\_d <= S\_IDLE;

end

else begin

state\_d <= state;

case (state)

S\_IDLE:

state <= (start) ? S\_LOAD : S\_IDLE;

S\_LOAD:

state <= (LOAD\_END) ? S\_CALC : S\_LOAD;

S\_CALC:

state <= (CALC\_END) ? S\_DONE : S\_CALC;

S\_DONE:

state <= (DONE\_END) ? S\_IDLE : S\_DONE;

default:

state <= S\_IDLE;

endcase

end

reg LOAD\_FLAG;

wire LOAD\_RESET = (!aresetn || LOAD\_END);

wire LOAD\_EN = (state == S\_LOAD) && (state\_d == S\_IDLE);

always @(posedge aclk)

if (LOAD\_RESET)

LOAD\_FLAG <= 'd0;

else

if (LOAD\_EN)

LOAD\_FLAG <= 'd1;

else

LOAD\_FLAG <= LOAD\_FLAG;

assign we = (LOAD\_FLAG && rdaddr[L\_RAM\_SIZE]) ? 'd1 : 'd0;

assign global\_we = (LOAD\_FLAG && !rdaddr[L\_RAM\_SIZE]) ? 'd1 : 'd0;

always @(posedge aclk)

if (global\_we) global\_buffer[addr] <= rddata;

else global\_dout <= global\_buffer[addr];

assign din = we ? rddata : 'd0;

reg CALC\_FLAG;

wire CALC\_RESET = (!aresetn || CALC\_END);

wire CALC\_EN = (state == S\_CALC) && (state\_d == S\_LOAD);

always @(posedge aclk)

if (CALC\_RESET)

CALC\_FLAG <= 'd0;

else

if (CALC\_EN)

CALC\_FLAG <= 'd1;

else

CALC\_FLAG <= CALC\_FLAG;

always @(posedge aclk)

if (!aresetn)

wrdata <= 'd0;

else

if (CALC\_END)

wrdata <= dout;

else

wrdata <= wrdata;

reg valid\_pre, valid\_reg;

always @(posedge aclk)

if (!aresetn)

valid\_pre <= 'd0;

else

if (CALC\_EN || dvalid)

valid\_pre <= 'd1;

else

valid\_pre <= 'd0;

always @(posedge aclk)

if (!aresetn)

valid\_reg <= 'd0;

else if (CALC\_FLAG)

valid\_reg <= valid\_pre;

assign valid = valid\_reg;

assign ain = valid ? global\_dout : 'd0;

reg DONE\_FLAG;

wire DONE\_RESET = (!aresetn || DONE\_END);

wire DONE\_EN = (state == S\_DONE) && (state\_d == S\_CALC);

always @(posedge aclk)

if (DONE\_RESET)

DONE\_FLAG <= 'd0;

else

if (DONE\_EN)

DONE\_FLAG <= 'd1;

else

DONE\_FLAG <= DONE\_FLAG;

assign done = (state == S\_DONE);

reg [31:0] CNT;

wire CNT\_LD = (LOAD\_EN || CALC\_EN || DONE\_EN);

wire CNT\_EN = (LOAD\_FLAG || (CALC\_FLAG && dvalid) || DONE\_FLAG);

wire CNT\_RESET = (!aresetn);

always @(posedge aclk)

if (CNT\_RESET)

CNT <= 'd0;

else

if (CNT\_LD)

CNT <= 'd0;

else if (CNT\_EN)

CNT <= CNT + 1;

assign addr = (LOAD\_FLAG) ? rdaddr[L\_RAM\_SIZE-1:0] :

(CALC\_FLAG) ? CNT :

'd0;

assign rdaddr = (LOAD\_FLAG) ? CNT : 'd0;

assign LOAD\_END = (LOAD\_FLAG) && (CNT == 'd31);

assign CALC\_END = (CALC\_FLAG) && (CNT == 'd15) && dvalid;

assign DONE\_END = (DONE\_FLAG) && (CNT == 'd4);

my\_pe #(

.L\_RAM\_SIZE(L\_RAM\_SIZE)

) pe (

.aclk(aclk),

.aresetn(aresetn && (state != S\_DONE)),

.ain(ain),

.din(din),

.addr(addr),

.we(we),

.valid(valid),

.dvalid(dvalid),

.dout(dout)

);

endmodule

my\_pe

module my\_pe #(

parameter L\_RAM\_SIZE = 6

)

(

input aclk,

input aresetn,

input [31:0] ain,

input [31:0] din,

input [L\_RAM\_SIZE-1:0] addr,

input we,

input valid,

output dvalid,

output [31:0] dout

);

(\* ram\_style = "block" \*) reg [31:0] peram [0:2\*\*L\_RAM\_SIZE - 1];

reg [31:0] psum;

wire [31:0] buffer;

always @(posedge aclk) begin

if(we == 1) peram[addr] <= din;

if(~aresetn) psum <= 0;

if(dvalid == 1) psum = buffer;

end

assign dout = (dvalid == 1) ? buffer : 32'b0;

floating\_point\_0 mac(

.aclk(aclk),

.aresetn(aresetn),

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

.s\_axis\_a\_tvalid(valid),

.s\_axis\_b\_tdata(peram[addr]),

.s\_axis\_b\_tvalid(valid),

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

.s\_axis\_c\_tvalid(valid),

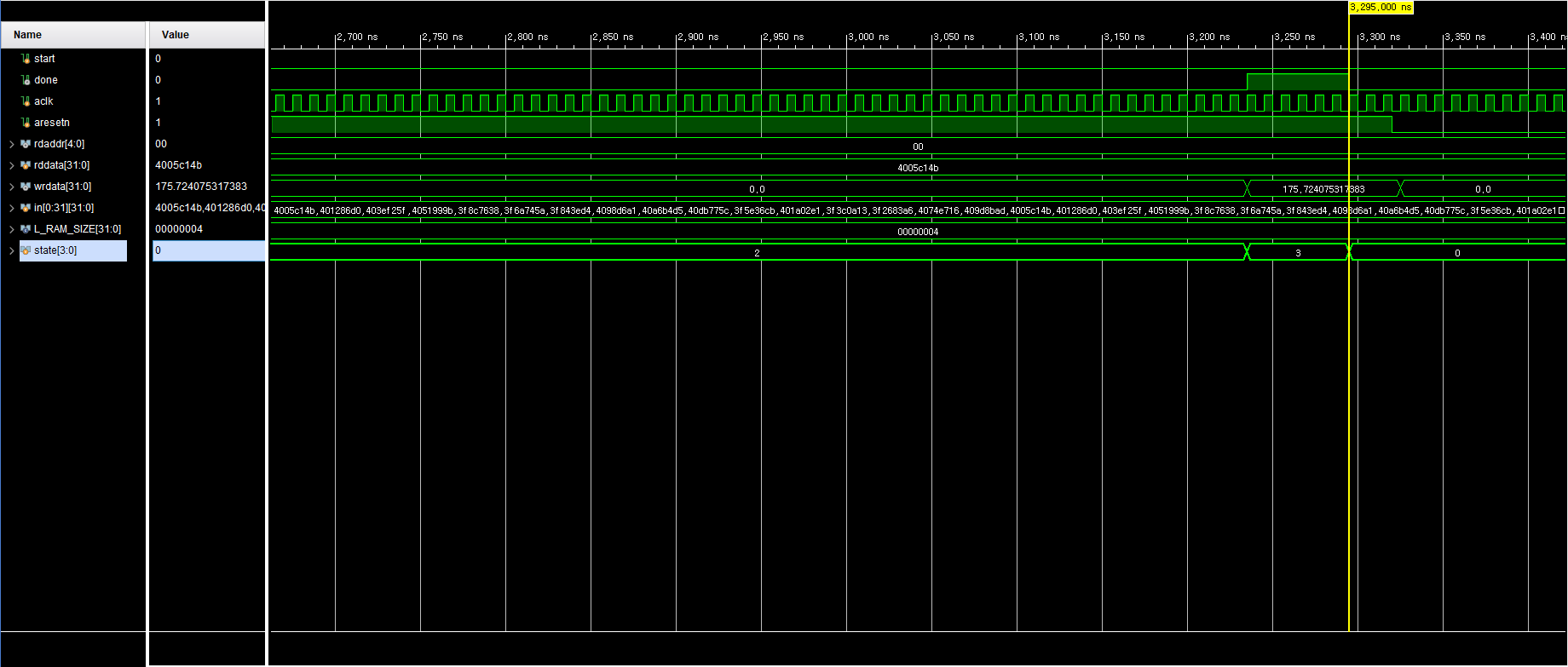
.m\_axis\_result\_tdata(buffer),

.m\_axis\_result\_tvalid(dvalid)

);

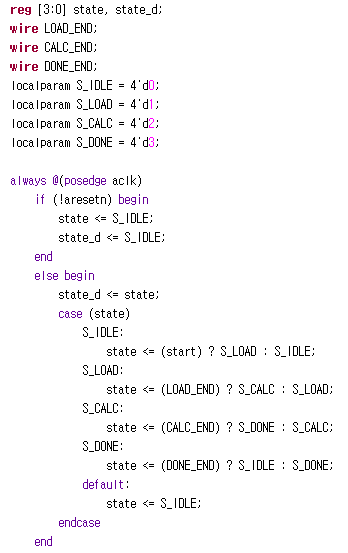
endmodule

Result: pe\_ctrl



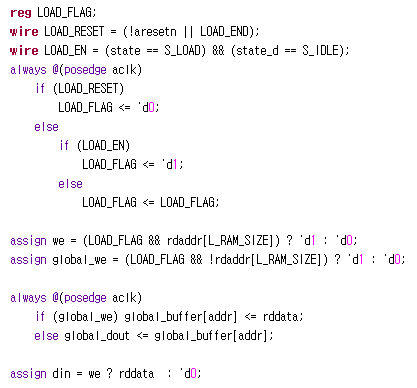
Discussion: 제가 구현한 pe\_ctrl을 설명드리도록 하겠습니다.

1. FSM



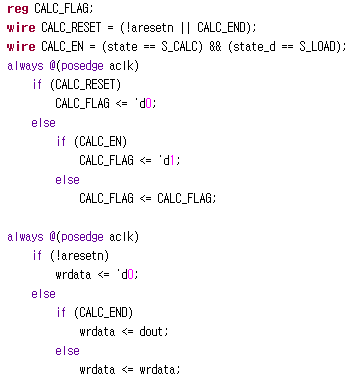
각 state에 진입한 직후인지 판단하기 위해 state와 state\_d 두개의 변수를 만들었습니다. reset 하면 둘다 S\_IDLE이고 start하거나 각 state가 끝나면 다음 state로 변합니다.

2. LOAD

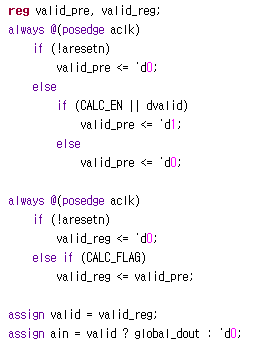


각 state마다 flag, reset, en의 3개의 변수가 있습니다. flag는 현재 state가 무엇인지, reset은 외부에서 reset이 들어오거나 state가 끝났는지, en은 state에 진입한 직후인지 판단합니다. load 상태일 때 rdaddr가 0~15면 global\_we를, 16~31이면 we를 활성화합니다. global\_we 상태이면 global\_buffer에 rddata의 값을 아니라면 global\_dout에 global\_buffer의 값을 저장합니다. we 상태이면 din에 rddata 값을 저장합니다.

3. CALC

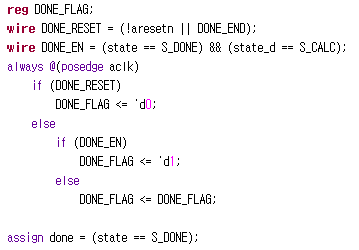


계산이 끝나면 wrdata, 즉 출력 값에 dout의 값을 저장합니다.



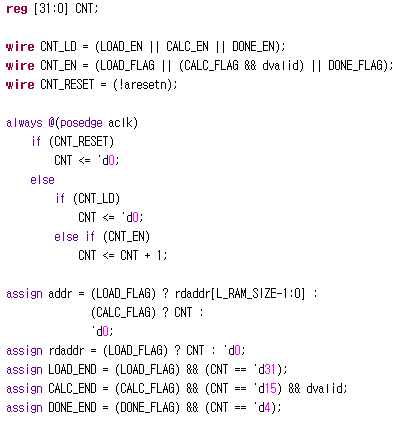
CALC 상태에 진입한 직후 혹은 dvalid가 활성화 된 상태면 valid\_pre를 활성화하고, 그 다음 cycle에 valid를 활성화하여 ain에 global\_dout의 값을 저장합니다.

4. DONE



DONE 상태에 진입하면 done을 활성화합니다.

5. CTR



LD는 각 상태에 진입한 직후인지, EN은 CNT가 enable한 상황인지, RESET은 리셋이 들어왔는지 판단합니다. RESET이거나 LD이면 CNT를 0으로 초기화하고 LOAD와 DONE은 매 cycle마다, CALC는 매 연산 마다 CNT를 1 증가시킵니다. addr은 LOAD일땐 rdaddr값을, CALC일땐 CNT값을 저장해줍니다. rdaddr은 LOAD일 때 CNT값을 저장해줍니다. LOAD는 32 cycle 후에, CALC는 16번 연산 후에, DONE은 5cycle 후에 각 상태를 종료합니다.

그렇게 해서 LOAD에 32+1 = 33cycle, CALC에 16\*18+1 = 289cycle, DONE에 5+1 = 6cycle, 총 328 cycle을 소모합니다.