# **Hardware System Design**

# Term project V0 submission 이다운 2016-13919

#### 0. Overview

lab06 에서 V x V 연산을 위한 pe\_controller를 구현하였다. 이를 수정 및 보안해서 M x V 연산이 가능하게 만드는 것이 이번 Term project V0 submission의 목표이다.

## 1. Code

```
module pe_control#(
    parameter MSIZE = 4,
    parameter VSIZE =4
9
ί(
linput start,
input areset,
linput clk,
input [31:0] rddata,
butput reg [MSIZE+VSIZE:0] raddr,
output reg done,
output [31:0] wrdata0,
output [31:0] wrdatal,
output [31:0] wrdata2,
butput [31:0] wrdata3,
output [31:0] wrdata4,
butput [31:0] wrdata5,
output [31:0] wrdata6,
butput [31:0] wrdata7,
output [31:0] wrdata8,
out put
       [31:0] wrdata9,
butput [31:0] wrdata10,
output [31:0] wrdata11,
butput [31:0] wrdata12,
output [31:0] wrdata13,
```

위 사진은 모듈의 parameter와 port 코드 사진이다. parameter 에서 MSIZE, VSIZE는 각각 Matrix의 크기를 의미하는데 Matrix의 row가 2<sup>MSIZE</sup>, column이 2<sup>VSIZE</sup> 라는 의미이다. PORT는 lab06 과 비교하면 output만 다른데 M x V의 output으로 1 x 2<sup>VSIZE</sup> Matrix를 출력해야 한다. 그래서 1 x 2<sup>VSIZE</sup> Matrix 의 원소 값을 동시에 출력하게 여러 output port를 두었다. 이 report에서는 MSIZE, VSIZE 둘 다 4로 가정하고 할 것 이므로 16개의 wrdata output을 만들었다.

```
LOAD: begin

if(counterdata <= 2**(MSIZE+VSIZE) -1)begin

globalram[counterdata] <= rddata;

raddr <= counterdata;

end

if(counterdata > 2**(MSIZE+VSIZE) -1)begin

we <= 1;

raddr <= counterdata - 2**(MSIZE+VSIZE);

local_addr<= counterdata - 2**(MSIZE+VSIZE);

din <= rddata;

end

end
```

기본적인 뼈대는 lab06과 같다. LOAD단계에서 M의 원소 2<sup>MSIZE+VSIZE</sup> 개 받고 V의 원소 2<sup>VSIZE</sup>개를 받아 MY PE의 local 버퍼에 저장해준다.

```
|generate for(i=0; i<2**MSIZE; i=i+1) begin : pe
wire [31:0]douti;
my_pe # (.L_RAM_SIZE(VSIZE))
pe(
    .aclk(~clk),
    .aresetn(pe_areset),
    .ain(ain[i]),
    .din(din),
    .addr(local_addr),
    .we(we),
    .valid(valid),
    .dvalid(dvalid),
    .dout(douti)
);
end
endgenerate
```

그리고 행렬 곱셈의 각 행들의 연산을 동시에 할 수 있도록 generate for문으로 2<sup>MSIZE</sup>개의 MY\_PE모듈을 만들어줬다. 각 MY\_PE[I]의 output port인 douti는 전체 모듈 output port와 연결될 것이다.

### 3. Testbench & Simulation

```
start =1;
#10 start= 0;
#5;

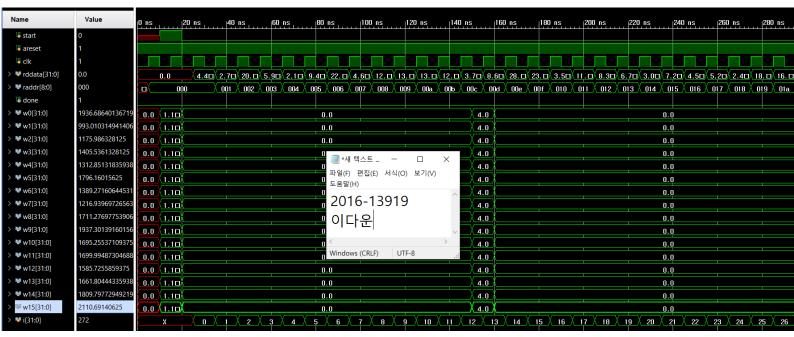
for(i=0; i<272; i=i+1)begin
    rddata = $urandom%(2**31);
    rddata = {7'b0100000, rddata[24:0]};
    #10;

end
rddata = 0;

#3000;
areset =0;</pre>
```

Simulation은 MV연산 크기를 (16x16) X 16 연산으로 진행할 것이다.

testbench는 start신호를 준 다음에 input으로 M,V의 원소 총 272개를 매 클럭마다 차례대로 입력해주었다. 이때 \$urandom 함수에서 너무 큰 수가 나오면 overflow가 나올 수 있으니 부 동소수점의 exp부분을 조정해 주었다.



(확대하면 보입니다.)

simulation의 초기 부분이다. start신호가 1이 되고 rddata 포트로 Matrix의 원소가 될 값들이 입력되고 있는 것을 확인할 수 있다.

Name	Value		3,300 ns	3,350 ns	3,400 ns	3,450 ns	3,500 ns	3,550 ns	3,600 ns	3,650 ns	3,700 ns	3,750 ns	3,800 ns	3,850 ns	3,900 ns	3,950 ns	4,000 ns
₩ start	0																
↓ areset	1																
<sup>1</sup> clk	0																lll
> 💆 rddata[31:0]	0.0									0.0							
> W raddr[8:0]	003	002	003		Х			004		χ'		005			006		007
le done	0																
> <b>W</b> w0[31:0]	130.364013671875	XeX	130.364	013671875	X		154.7106170	6543		161.	750137329102	\( \)	X-X	412.54165	3494141		478.0070800 <b></b> □
> ₩w1[31:0]	144.749786376953	X <b>a</b> X	144.749786376953		XX	172.331817626953		26953	_>(a)(a)(_	182.349670410156		)	*	375.362609863281			388.5394897□
> <b>W</b> w2[31:0]	123.248641967773	X <b>a</b> X	123.248641967773		X0(0	134.037078857422			150.	150.15283203125		3(0)	206.535171508789			224.7012786🗆	
> <b>W</b> w3[31:0]	340.139556884766	X <b>6</b> (	340.139556884766		X	366.747314453125			398.65478515625		)		609.840637207031			642.0602416🗆	
> <b>W</b> w4[31:0]	106.511184692383	XaX	106.511184692383		XX	126.144996643066			159.363250732422		) (	*	299.898986816406			332.8870544□	
> <b>W</b> w5[31:0]	198.852035522461	XeX	198.852035522461		X_		207.933624267578			244.5244140625		)	) D	833.038818359375			840.8777465🗆
> ₩ w6[31:0]	264.063232421875	X <b>a</b> X	264.063232421875		XX	293.128082275391		_>(a)(a)(_	317.9	317.976318359375		*	599.358276367188			616.5344238□	
> <b>W</b> w7[31:0]	152.073059082031	XeX	152.073	152.073059082031		223.747055053711		53711		297.314849853516		)(ε	3(0)	365.01513671		71875	
> <b>W</b> w8[31:0]	202.326202392578	X <b>a</b> X	202.326	202392578	X_0X_0		233.82803344	17266	_>(0)	<i>■</i> *새	텍스트		×	994.54321	2890625		1082.838134🗆
> <b>W</b> w9[31:0]	120.73722076416	X6<	120.73	722076416	X_		216.13394165	50391		파일(F)	편집(E) 서식	(O) 보기(V)		890.84002	855469		897.9789428🗆
> <b>W</b> w10[31:0]	213.876113891602	X <b>&amp;</b>	213.876	113891602	XX_		308.49932861	3281		도움말(	H)			1142.8218	941406		1187.984252🗆
> <b>W</b> w11[31:0]	229.924758911133	X6X	229.924758911133		X_	320.194732666016		66016		2016-1391		19 ^		738.707153320313		824.31286620	
> ₩ w12[31:0]	101.833084106445	X <b>&amp;</b>	101.833	084106445	XX_		111.59487915	50391			1	)		386.30154	1189453		397.9769897□
> <sup>©</sup> w13[31:0]	239.285369873047	X	239.285	369873047	X <b>&amp;</b> G		267.31015014	16484	_X9X9X	이디	나운			457.23513	7939453	X6X6X.	490.5262145🗆
> <b>W</b> w14[31:0]	301.023651123047	X <b>&amp;</b>	301.023	651123047	X_		310.44485473	36328			· ·		_ ~	944.57904	1527344		958.8571166□
> <b>W</b> w15[31:0]	252.052124023438	X <b>&amp;</b>	252.052	124023438	XX		309.99978637	76953		Window	vs (CRLF)	UTF-8	´ .	910.72271	7285156		958.5233154□
> 💆 i[31:0]	272									272	is (SILLI)	011.0					

CALC state 중간의 스크린샷으로 raddr의 값에 해당하는 위치까지의 결과를 확인할 수 있다.

Name	Value	5,300	ns   5,350 ns	5.400 ns	5,450 ns	5,500 ns	5,550 ns	5,600 ns	5,650 ns	5,700 ns	5,750 ns	5,800 ns	5,850 ns	5,900 ns	5,950 ns	6,000 ns	6,05
↓ start	0																
areset	0																
⊌ clk	0						10000			10000							Ш
> W rddata[31:0]	0.0								0.0								
> <b>W</b> raddr[8:0]	000	00e					X9X			000							
¹⊌ done	0																
> ® w0[31:0]	1945.10766601563	10	17.58825683594	_X6X6X	1568.	48547363281	(_	KeX	1936.6864	136719	X6X6	1945.10766600	3	8.421	26941680908		
> <b>W</b> w1[31:0]	1042.82214355469	74	8.130310058594	_X0\0	913.	6650390625	X_	<b>X</b>	993.010314	1941406		1042.82214350	*	49.81	18133544922		
> <b>W</b> w2[31:0]	1194.91149902344	82	6.052612304688	X0<0<	1119.	95068359375	X	<b>X</b> 0	1175.986	328125		1194.91149900	×	18.92	51327514648		
> <b>W</b> w3[31:0]	1421.76928710938	12	32.69995117188	_X0\0	1327.	78588867188	X_	(a)	1405.5361	328125		1421.76928710	*	16.23	32038879395		
> <b>W</b> w4[31:0]	1358.11804199219		1151.4453125	_X0\0	1273.	33923339844	<u> </u>	Ke/	1312.8513	835938		1358	테 <b>스</b> 트 —		4091797		
> <b>W</b> w5[31:0]	1801.93530273438	119	90.59826660156	X0<0<	1362.	9521484375	Xc	(a)	1796.160	15625		1801			1376953		
> <b>W</b> w6[31:0]	1429.73571777344	10	68.60021972656	_X0\0	1241.	68969726563	X_	<b>X</b>	1389.2716	1644531		파일(F) 1429 도움말(H	편집(E) 서식(0 n	O) 보기(V)	1328125		
> <b>W</b> w7[31:0]	1240.29443359375	10-	47.17797851563		1091.	7614746094	, Xc	K-X	1216.9396	726563		1240.		^	^ <b>/2546387</b>		
> <b>W</b> w8[31:0]	1716.7216796875	15	86.26586914063		1654.	37768554688	X_	KD	1711.2769	7753906		<sub>1716</sub> 201	6-1391	9	10722656		
> <b>W</b> w9[31:0]	1946.43139648438	14	50.35363769531	_X6/6/C	1555.	0107421875	X_	KD	1937.3013	160156		1946. O C	유		2467041		
> <b>W</b> w10[31:0]	1713.29125976563	14	90.68054199219	_X6X	1648.	59838867188	X_c	(o)	1695.2553	7109375					v 1061 6455		
> <b>W</b> w11[31:0]	1712.65734863281	11	52.49597167969		1642.	27941894531	,X_	(a)	1699.9948	7304688				>	18447266		
> <b>W</b> w12[31:0]	1657.76232910156	98	6.974792480469		1530.	99548339844	, Xc	KD .	1585.7255	859375		1657 Window	s (CRLF) U	TF-8	5698242		
> <b>W</b> w13[31:0]	1708.21435546875	13	17.9013671875		1593.	7797851563	X_		1661.8044	1335938	X0X0	1708.21435540		46.40	98815917969		
> <b>W</b> w14[31:0]	1825.09948730469	17	02.70642089844	_X0/0/_	1769.	39392089844	,	(a)	1809.7977	2949219	X0X0	1825.09948730	-\	15.30	17730712891		
> <b>W</b> w15[31:0]	0.0	15	0.96008300781		1745.	40747070313	X_	(a)	2110.691	40625	X_			0.0			
> 💆 i[31:0]	272								272								

마지막으로 CALC state가 끝나고 DONE state로 들어가는 것을 확인할 수 있다. 모든 연산이 끝난 뒤 DONE 시그널이 1이 된다. done이 1일때의 output값이 MxV의 연산 결과 값이다.

### 4. Discussion

MxV 원소를 읽어올 때 readmemh 함수를 사용하면 훨씬 빠르게 LOAD단계가 끝날 것이다. 이를 사용하여 수행시간을 줄일 수 있을 것 이다.

Submission에서는 MxV의 연산만 보이면 돼서 최적화에 대해서는 고려하지 않았다.

output을 각 포트마다 출력해야 하는지 writememh함수를 사용 해야 하는지 아직 확실하게 모르겠어서 일단 port로 출력하는 방식을 택했다.