

**과 목 : 논리회로설계실험**

**과 제 명 : 기말 프로젝트**

**담당교수 : 전 일 용**

**학 과 : 전자전기공학부**

**학 년 : 2학년**

**2학년**

**2학년**

**학 번 : 2019310435**

**2019310896**

**2019314422**

**이 름 : 김준환**

**오진혁**

**허형무**

**제 출 일 : 2022/12/31**

**목차**

**1. 개요**

**2. 모듈**

(2-1) top.v

(2-2) controller.v

(2-3) memory.v

(2-4) display.v

(2-5) core.v

(2-5-a) 8\_bit\_multiplier.v

(2-5-b) accumulater.v

(2-5­-c) pe.v

(2-5­-d) single\_process\_array.v

(2-5-e) systolic\_array\_3\_by\_3.v

(2-5-f) systolic\_array\_2\_by\_2.v

**3. 결론**

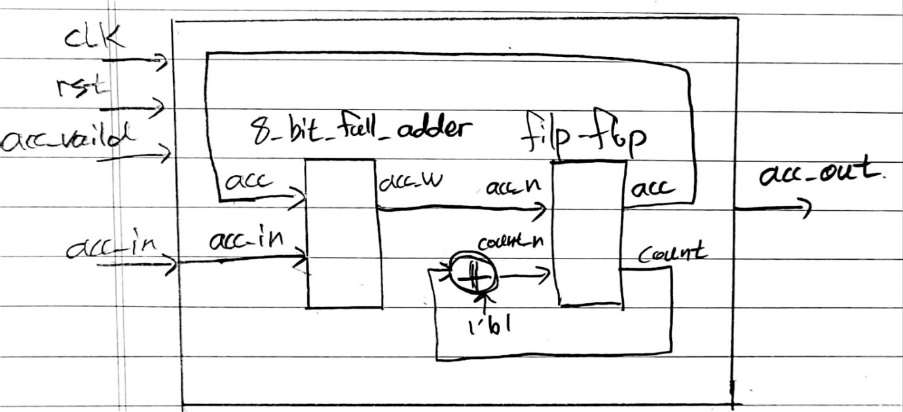
(3-1) 성능 비교 A single PE vs a systolic array

(3-2) 담당

(3-3) 참고자료

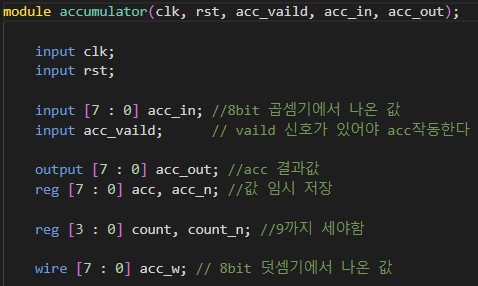
**1. 개요**

**(2-5-c) accumulater.v**

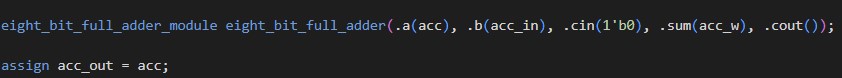


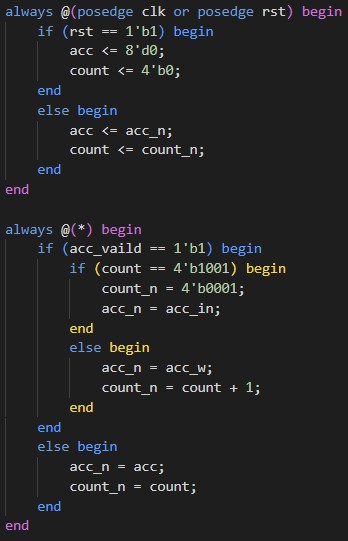


acc\_in은 accumulator로 누산하기 위해 들어오는 8bit 값이고 누산기는 acc\_vaild신호가 1’b1일 때만 작동한다. pe에 들어오는 mode의 값에 따라 pe안에 있는 누산기의 동작이 결정되는데 single\_process\_array가 작동할 때는 pe에 있는 누산기가 작동하며 systolic array가 작동할 때는 pe에 있는 누산기는 작동하지 않고 systolic array에 내장되어 있는 누산기가 작동하도록 설계했다.



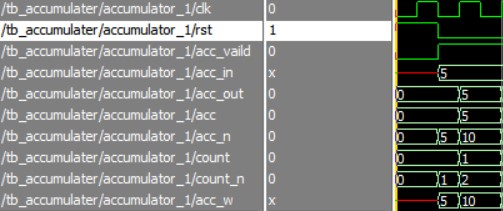
Input으로 clk, rst, acc\_vaild, acc\_in이 있고 output으로 acc\_out이 있다. acc\_in과 acc\_out만 8bit이고 나머지는 1bit이다. Flip flop을 사용하기 위해 input값을 저장 할 reg 8bit acc와 acc\_n이 필요하고 매 clk마다 몇 번 누산했는지 알려줄 4bit count와 count\_n이 필요하다 4bit인 이유는 9가 4’b1001이기 때문이다. 그리고 acc안에 있는 8-bit 덧셈기의 output을 reg인 acc\_n에 연결해줄 wire 8bit acc\_w가 필요하다.



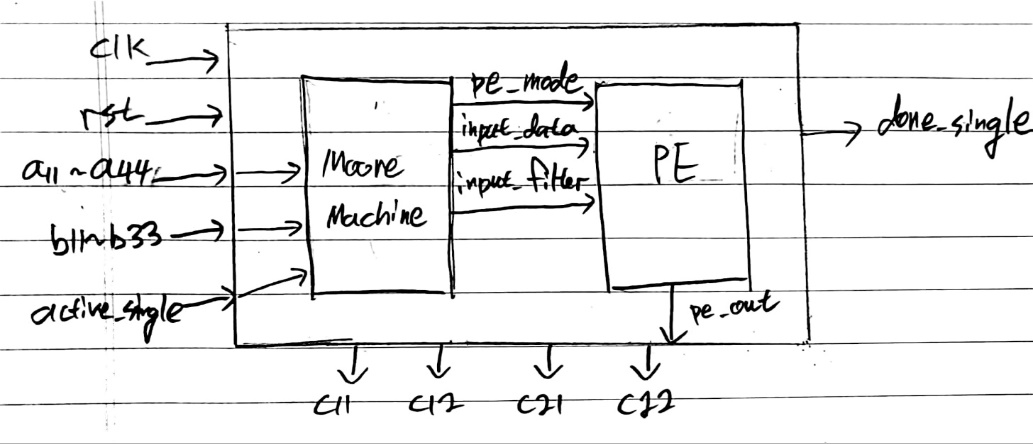


Sequential 구문으로 sensitivity list가 posedge clk와 posedge rst이고 rst가 1이 되면 acc와 count가 바로 0으로 초기화 된다. 아니면 매 clk마다 acc\_n값과 count\_n의 값이 acc와 count의 값이 된다.

Combination 구문으로 acc\_vaild가 1이면 acc\_n값은 8-bit-fulladder의 output이 되고 count\_n은 count에서 1이 더해진다. 만약 count가 9가 된다면 count\_n이 1이 되어 다음 clk에 바로 count가 1이 되게 해서 다시 누산하는 것을 시작하고 acc\_n이 acc\_in이 되어 다음 clk에 acc값을 acc\_in값과 같게 만들어 누산하던 값을 초기화 시킨다. 8-bit-fulladder에는 acc와 acc\_in이 인풋으로 들어가고 output은 acc\_w로 acc\_n에 연결된다. Output인 acc\_out은 acc로 assign하는데 결과적으로 acc\_in이 들어온 다음 clk에 output이 나온다. 이유는 acc\_in이 들어온 순간 acc\_n값은 acc\_w값(8-bit fulladder의 계산값)과 같아지고 flip flop으로 다음 clk에서 acc가 acc\_n값이 되어 output은 acc\_in이 들어간 다음 clk부터 나온다. Count는 처음 값이 들어오고 결과값이 나온 순간부터 1이 된다. 따라서 count가 9일 때 acc\_out은 9번 누산된 값이고 single에서는 pe를 거쳐 single의 무어머신에서 c11, c12, c21, c22에 할당한다.



**(2-5­-e) single\_process\_array.v**



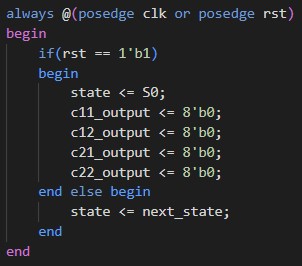
Single\_process\_array는 한 개의 pe를 사용하여 c11, c12, c21, c22를 구해야 한다. Single\_process\_array는 input으로 16개의 a11~a44값과 9개의 b11~b33값을 받아온다. 이 값들을 single의 무어머신을 통해 한 state마다 16개의 data와 9개의 filter값 중에서 하나씩 pe에 보내 누산을 시키고 매 clk이 지날 때마다 state를 다음 state로 옮겨 또 pe에 각각 하나씩 값을 집어넣는다. Pe로 9번째 누산한 값이 나오는 state에서는 c에 pe\_out을 저장하는 코드를 추가해 주면 c11, c12, c21, c22에 9번씩 누산한 값들이 들어갈 것이다. 이때 filter값은 b11부터 순서대로가 아닌 뒤집어서 b33부터 b11까지 넣어야 한다.

Single\_process\_array로 들어오는 active\_single 신호는 controller에서 주는 값으로 single이 연산을 시작하라는 신호이다. done\_single은 c22까지 모든 값이 나왔고 Single\_process\_array의 동작이 끝났다는 것을 알려준다.

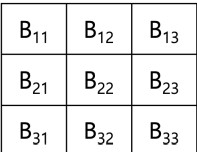
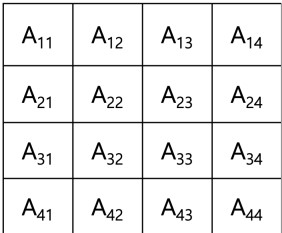


Single\_process\_array는 pe한 개만 inst한다. 각각의 input값은 무어머신의 state마다 달라지고 output은 항상 필요한 것이 아니라 9번째 누산된 값만 중요하다. Pe\_mode는 single이 동작하면 값이 0으로 바뀌어 pe에 single연산 중이라는 것을 알려준다. Activate는 1일 때만 pe가 작동하게 끔 설정을 했지만 항상 1이어도 결과를 내는데 상관이 없어 그대로 두었다. Pe\_out은 누산된 값이 나오는 것이고 pe\_in\_o는 pe가 받은 input\_data값을 옆에있는 다른 pe에 전달해 주는 것인데 single에서는 필요가 없다.

Reg 8bit input\_data와 input\_filter는 pe로 들어갈 값으로 무어머신으로 각각의 state마다 값 한 개씩 pe에 넣어주기 위해 설정되었다. Pe\_mode는 single에 active\_single신호가 들어와 작동을 시작할 때 pe의 mode를 알려주어 pe가 mode에 맞게 동작하도록 설정된다. State와 next\_state는 무어머신의 상태를 알려주기 위한 것이고 c11\_output등은 pe의 output중 9번째 누산 값만 저장해 마지막에 assign해주기 위한 것이고 done\_singel\_w는 single의 연산이 종료되었을 때 1이 되어 done\_single에 assign해준다. parameter들은 각각의 state를 나타내기 위해 설정되었다.

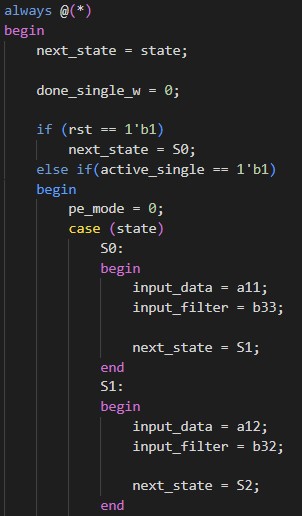
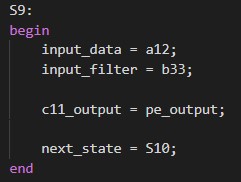
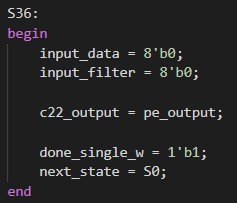


Sequential 구문에 sensitivity list로 posedge clk와 posedge rst가 있고 rst가 1일 때 state는 S0으로 초기화 되고 모든 output이 0으로 초기화 된다. Rst가 1이 아닐 때 다음 state가 현재 state로 바뀌며 매 clk마다 state가 다음으로 넘어간다.



다음 코드를 설명하기 앞서 single\_process\_array의 동작을 설명해야 한다. Input값을 2-d convolution하려고 하면 filter인 b11~b33을 상하좌우 뒤집어야 한다. 그런 다음 input에 filter의 값을 하나씩 곱하면 된다. C값 하나만 예를 들면

C11 = a11\*b33 + a12\*b32 + a13\*b31 + a21\*b23 + a22\*b22 + a23\*b21 + a31\*b13 + a32\*b12 + a33\*b11 이다. 첫 state인 S0에서부터 a11과 b33을 pe에 넣고 다음 state인 S1에서는 a12와 b32를 pe에 넣는다. 이 과정에서 중요한 점은 pe에 들어간 값의 결과가 다음 clk에 나온다, 즉 다음 state에서의 pe의 output이 그 전 state에서 pe로 들어간 값들까지의 누산 값이다. S8에서 pe에 a33과 b11이 pe로 들어가는데 S9에서의 pe의 output이 우리가 구하고 싶은 9번 누산한 c11값이 나온다. 이 값을 reg인 c11\_output에 저장하고 마지막에 c11에 assign해주면 된다.

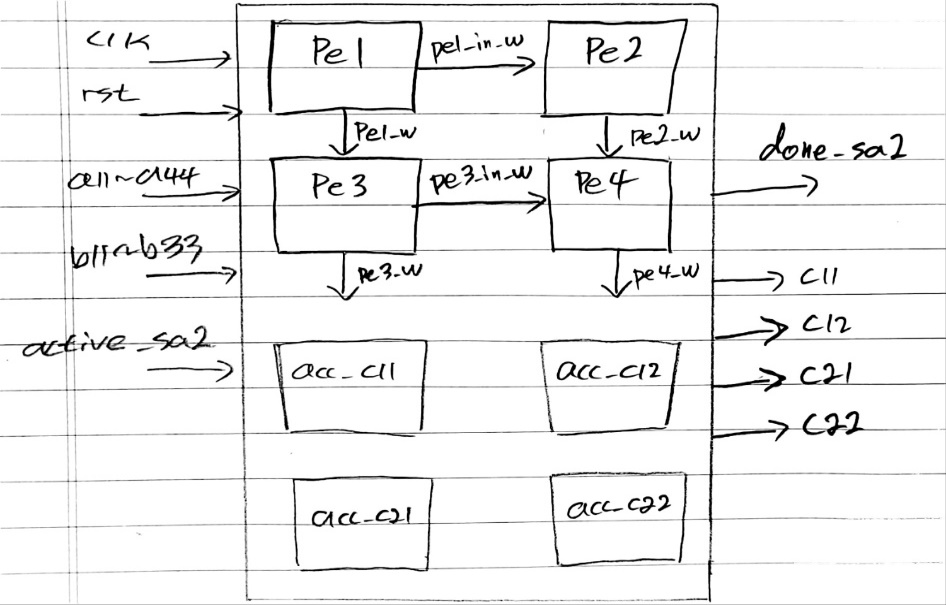
  

Combination 구문으로 active\_single이 1일 때 pe\_mode가 0이 되어 pe에게 single연산을 하고 있다는 것을 알려주고 위에서 설명한 것처럼 한 state마다 pe에 값을 각각 하나씩 넣어준다. 그러면 S9에서 처음 C11값이 나오고 S36에는 C22값이 나온다. 이때 done\_single이 1이 되게 해주어 single연산이 끝났다는 것을 알려준다. 이 코드가 잘 작동하는지 알아보기 위해 tb을 돌렸다. A11부터 A44까지 1에서 16이고 b11부터 b33까지 1에서 9를 할당했다.



모델심으로 돌려보니 c11 = 192, c12 = 237, c21 = 116, c22 = 161이 나왔다. C11을 직접 계산해 보면 1\*9 + 2\*8 +3\*7 +5\*6 +6\*5 +7\*4 +9\*3 +10\*2 +11\*1 = 192으로 값이 알맞게 나왔고 c22는 2\*9 + 3\*8 +4\*7 +5\*6 +7\*5 +8\*4 +10\*3 +11\*2 +12\*1 = 237로 맞게 나온 것을 확인할 수 있다. 나머지 두개도 직접 계산한 값과 모델심을 통해 구한 값이 같은 것을 확인할 수 있다. 그리고 마지막 값이 나오는 것과 동시에 done\_single신호가 1이 되는 것도 확인할 수 있다.

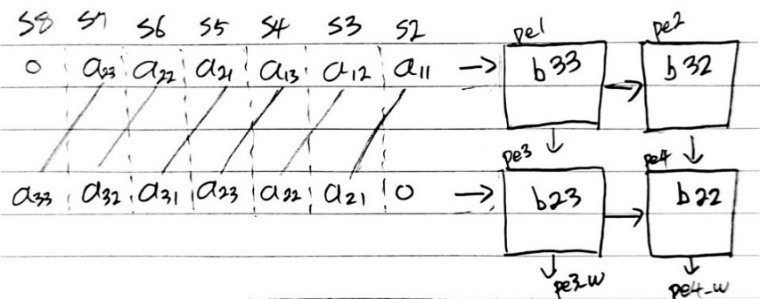
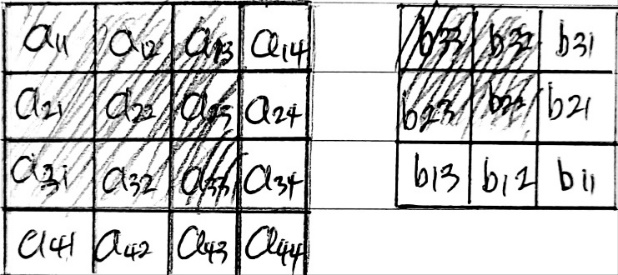
**(2-5-g) systolic\_array\_2\_by\_2.v**



systolic\_array\_2\_by\_2는 4개의 pe를 이용해 c11, c12, c21, c22의 값을 구한다. systolic\_array\_2\_by\_2는 먼저 4개의 pe에 먼저 9개의 filter값중 4개를 저장한 후 시작한다. 이때 pe에 저장되어 있는 filter의 값에 따라 input으로 들어가는 값에 차이가 있다. 위 그림과 같이 pe를 배열하고 각 pe에 1,2,3,4 숫자를 부여한다. 누산기도 총 4개가 필요한데 각각 c11, c22, c33, c44값을 누산으로 구한다. 각각의 누산기에 들어갈 input값이 5번이라서 9번 마다 값을 초기화 시키는 동작은 무시한다. 또한 pe내에 있는 acc는 pe의 mode를 조절해 동작하지 않도록 했다. systolic\_array\_2\_by\_2에서는 pe의 8bit 곱셈기와 덧셈기의 기능만을 사용한다. Pe 4개를 이으면 결과값은 pe3\_w와 pe4\_w에 나오는데 각각 pe3, pe4에서 나오는 pe\_out이다. 이 값들을 해당하는 acc에 넣어 누산을 시켜 최종 값은 acc에서 나온다.

systolic\_array\_2\_by\_2에서는 필터저장과 연산에 따라 pe의 mode가 다르고, 필터저장을 총 3번 하니 state마다 pe의 모드를 바꾸는데 유념해야 한다. 또한 필터를 저장할 때 상하좌우 반전을 해야 한다.

먼저 pe1, pe2, pe3, pe4에 각각 b33, b32, b23, b22를 저장한다. 이 값들은 pe의 data에 저장된다. 저장하는데 2clk가 걸리니 input값이 들어오는 state는 S2부터이다. b33, b32, b23, b22는 필터의 좌측상단 4개이다.



모든 input값을 pe에 넣을 필요가 없는데, 우리가 정한 필터 값들과 곱해지는 input값들만 pe에 넣으면 된다. 따라서 input값도 좌측상단 9개의 값만 필요하고 나머지 7개의 값은 pe에 넣어도 acc로 누산 될 필요가 없어서 뺀다.

9개의 input이 모든 pe에 한번씩 들어가는 것이 아니라 c11, c12, c21, c22의 부분합이 나타날 수 있도록 넣어야 한다. 예를 들어 b33과 b32는 a31, a32, a33과 곱해질 일이 없어 첫번째 pe줄에 a31, a32, a33이 input으로 들어가지 않고 b23과 b22는 a11, a12, a13과 곱해질 일이 없어 두번째 pe줄에 a11, a12, a13이 input으로 들어가지 않는다. S2에 a11과 0이 각각 pe1과 pe3에 들어가 pe에 저장된 b33과 b23에 곱해지고 S3에 a12와 a21이 각각 pe1과 pe3에 들어간다. 이때 pe3에서는 pe1에서 전달받은 a11\*b33이 pe\_filter로 들어오고, a21과 pe3에 저장된 b23이 곱해진 값과 더해진다. S4에 a11\*b33 + a21\*b23이라는 값이 pe3\_w으로 나오는데, 이는 C11 = a11\*b33 + a12\*b32 + a13\*b31 + a21\*b23 + a22\*b22 + a23\*b21 + a31\*b13 + a32\*b12 + a33\*b11에 포함되어 있다. 따라서 S4에서 a11\*b33 + a21\*b23값을 acc\_c11에 넣어 누산시키게 하고 이때 누산을 시키기 위해 active\_acc1신호가 1이되어 acc에 acc\_vaild로 들어가서 동작하게 한다. 이러한 C값의 부분합을 구하기 위해 input값을 넣는 타이밍이 중요하고 어느 state에 pe3\_w와 pe4\_w에서 부분합이 나올 지 정확히 알아야 한다. 다음은 어떤 state에 pe3\_w와 pe4\_w중 어느 곳에서 C의 부분합이 나올지 표로 작성했다.

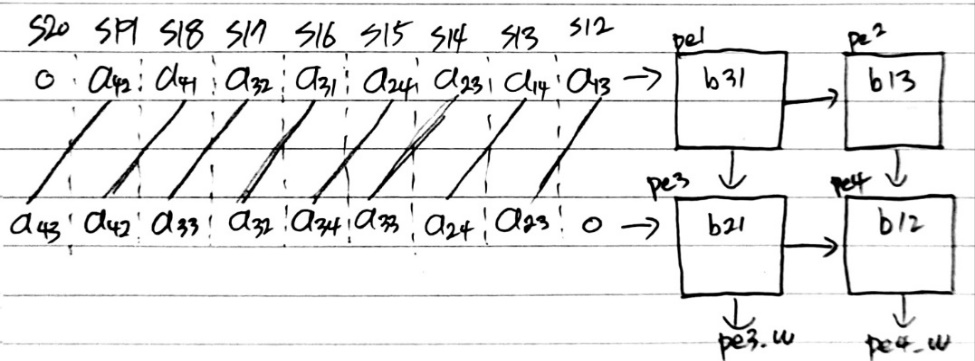
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Pe3\_w | | Pe4\_w | |
| C11 | a11\*b33 + a21\*b23 | S4 | a12\*b32 + a22\*b22 | S6 |
| C12 | a12\*b33 + a22\*b23 | S5 | a13\*b32 + a23\*b22 | S7 |
| C21 | a21\*b33 + a31\*b23 | S7 | a22\*b32 + a32\*b22 | S9 |
| C22 | a22\*b33 + a32\*b23 | S8 | a23\*b32 + a33\*b22 | S10 |

따라서 해당하는 state에 알맞은 acc를 active시키고 pe3\_w나 pe4\_w를 acc에 넣어 누산시킨다.

두번째로 pe1, pe2, pe3, pe4에 각각 b31, b13, b21, b12를 저장한다. 이 값들은 pe의 data에 저장된다. S10에서 c22의 부분합이 acc로 들어가지만 이때 모드는 연산모드가 아니라 필터저장 모드인데, 이유는 pe는 값을 저장하고 있다가 다음 clk에서 내보내는데, S9에서 연산 mode이고 저장된 값이 한 clk 뒤인 S10에 나오게 설계되어 있어 S10부터 필터 저장 모드로 바꾸어도 된다. 따라서 input값은 pe에 S12부터 들어온다.



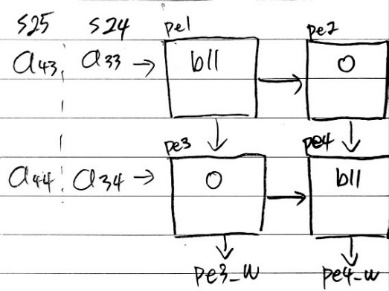
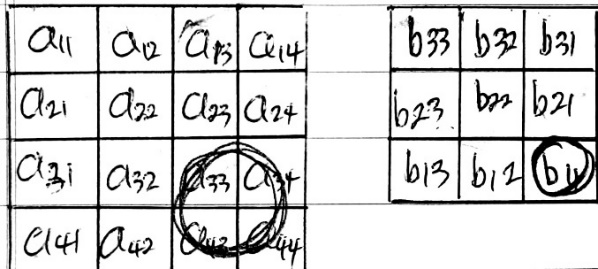
마찬가지로 모든 값이 input으로 들어오지 않아도 된다. 부분합으로 누산될 수 있는 것만 고르면 아래와 같이 된다.



다음은 어떤 state에 pe3\_w와 pe4\_w중 어느 곳에서 C의 부분합이 나올지 표로 작성했다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Pe3\_w | | Pe4\_w | |
| C11 | a13\*b31 + a23\*b21 | S14 | a31\*b13 + a32\*b12 | S19 |
| C12 | a14\*b31 + a24\*b21 | S15 | a32\*b13 + a33\*b12 | S20 |
| C21 | a23\*b31 + a33\*b21 | S16 | a41\*b13 + a42\*b12 | S21 |
| C22 | a24\*b31 + a34\*b21 | S17 | a42\*b13 + a43\*b12 | S22 |

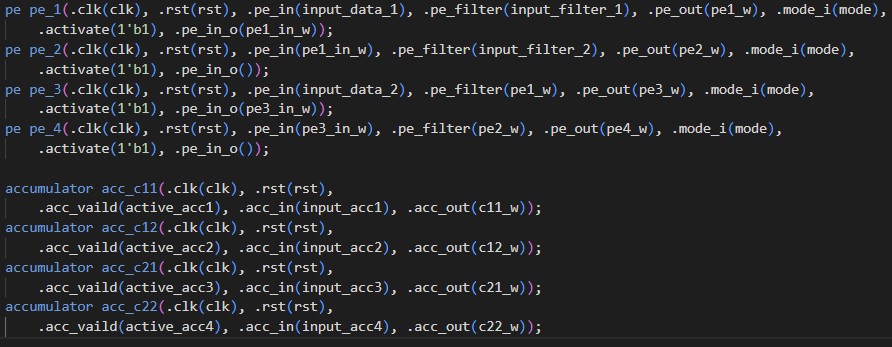
마지막으로 남은 b11과 a33, a34, a43, a44를 곱해야 한다.



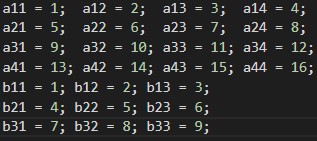
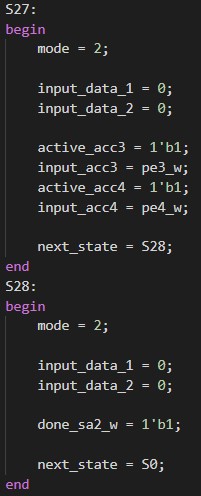
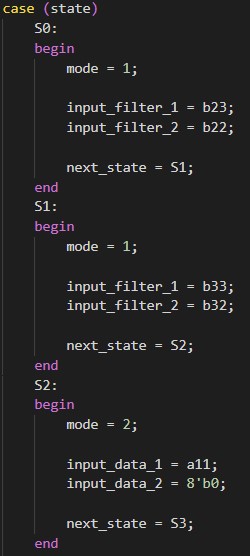
더 빠른 연산을 위해 pe1과 pe4에 b11을 저장하고 input값을 딜레이 없이 바로 넣는다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Pe3\_w | | Pe4\_w | |
| C11 | a33\*b11 | S26 |  |  |
| C12 |  |  | a34\*b11 | S26 |
| C21 | a43\*b11 | S27 |  |  |
| C22 |  |  | a44\*b11 | S27 |

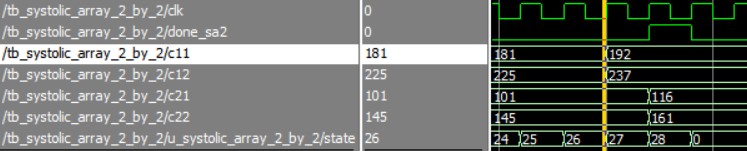
지금까지 구한 모든 부분합들을 누산하면 c11, c12, c21, c22를 구할 수 있다.



Pe와 acc를 inst한 것을 보여준다. Pe는 위에서 받는 input을 pe\_filter라고 하고 왼쪽에서 받는 input을 pe\_in이라고 하고 오른쪽으로 나가는 output을 pe\_in\_o, 아래로 나가는 output을 pe\_out이라고 한다. 이점을 유념해서 pe4개를 wiring하면 된다. Pe의 activate를 항상 1로 해도 결과값이 달라지지 않아 1로 유지하고 있다. Pe3\_w 아니면 pe4\_w의 값이 input\_acc에 할당되고 acc의 input으로 들어간다. Active\_acc는 1일 때만 acc가 작동하게 해준다. C\_w는 번호에 맞는 acc의 output으로 acc에서 누산되고 있는 값들을 보여준다.



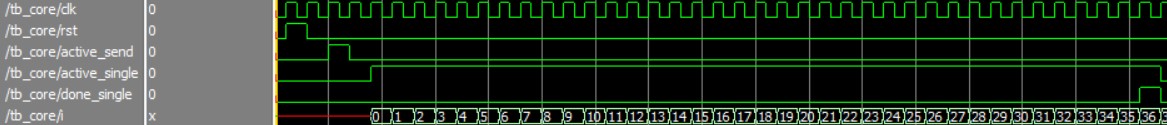
첫번째 사진은 필터저장모드를 하는데 2 state필요하다는 것을 나타내고 input\_filter\_1은 왼쪽줄에 들어갈 필터 값이고 input\_filter\_2는 오른쪽줄에 들어갈 필터 값이다. 이때 모드는 1이고 필터는 위에서 설명한 것과 같이 pe에 저장된다. 중간과정은 이미 위에서 그림으로 다 설명했으니 타이밍에 맞게 pe의 결과를 acc에 잘 넣으면 되고 마지막부분에 S27에서 최종적으로 C의 부분합들이 acc로 들어가고 한 clk뒤에 결과가 나온다. 따라서 마지막 state인 S28에서 acc에서 결과가 나오고 동시에 done\_sa2가 1이 되게 하였다. 총 29개의 state가 사용되었으며 37개의 state를 사용한 single보다 적은 시간이 걸리지만 17개의 state만 사용하는 3 by 3보다는 많은 시간이 걸리는 것을 알 수 있다.

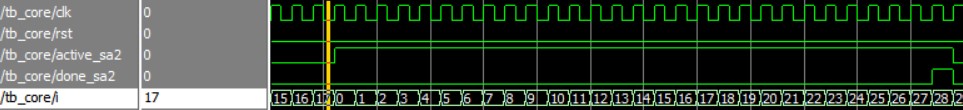


Testbench에 오른쪽 사진과 같이 single에서 넣은 값들과 같은 수를 넣으면 최종 값이 같은 것을 확인할 수 있고 S28에 모든 C값이 나오며 동시에 done\_sa2신호가 1이 된다.

**3. 결론**

**(3-1) 성능 비교** **A single PE vs a systolic array**

****

****

Single : 37clk, systolic\_array\_3\_by\_3 : 17clk, systolic\_array\_2\_by\_2 : 29clk

**3-1-a) Single VS systolic\_array\_3\_by\_3**

Speedup = = = 2.176

**3-1-b) Single VS systolic\_array\_3\_by\_3**

Speedup = = = 1.276

**(3-2) 담당**

**김준환 :** 8\_bit\_full\_adder.v, 8\_bit\_multiplier.v, pe.v,

systolic\_array\_3\_by\_3.v, core.v, 와 해당 tb제작

**오진혁 :** accumulater.v, single\_process\_array.v, pe.v,

systolic\_array\_2\_by\_2.v 와 해당 tb제작

**허형무 :**  top.v, memory.v, display.v, controller.v

와 해당 tb제작

**(3-3) 참고 자료**

Mahmood Naderan, systolic array, <https://youtu.be/cmy7LBaWuZ8>

Shift를 이용한 8\_bit 곱셈기

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=narabaljeon&logNo=220706158442

Charles Roth, Lizy Kurian John, Byeong Kil Lee 지음, Verilog를 이용한 디지털 시스템 설계