Lab 7. Fully-Connected Layer

Digital System Design and Experiment

Graduate School of Convergence Science and Technology

Seoul National University

Mobile Multimedia Systems Group

Goal of Lab 7

- Verilog를 이용하여 Neural Network의 주요 연산인 MAC 연산을 구현하고 최종적으로 Fully-Connected Layer를 설계하고 구현해본다.

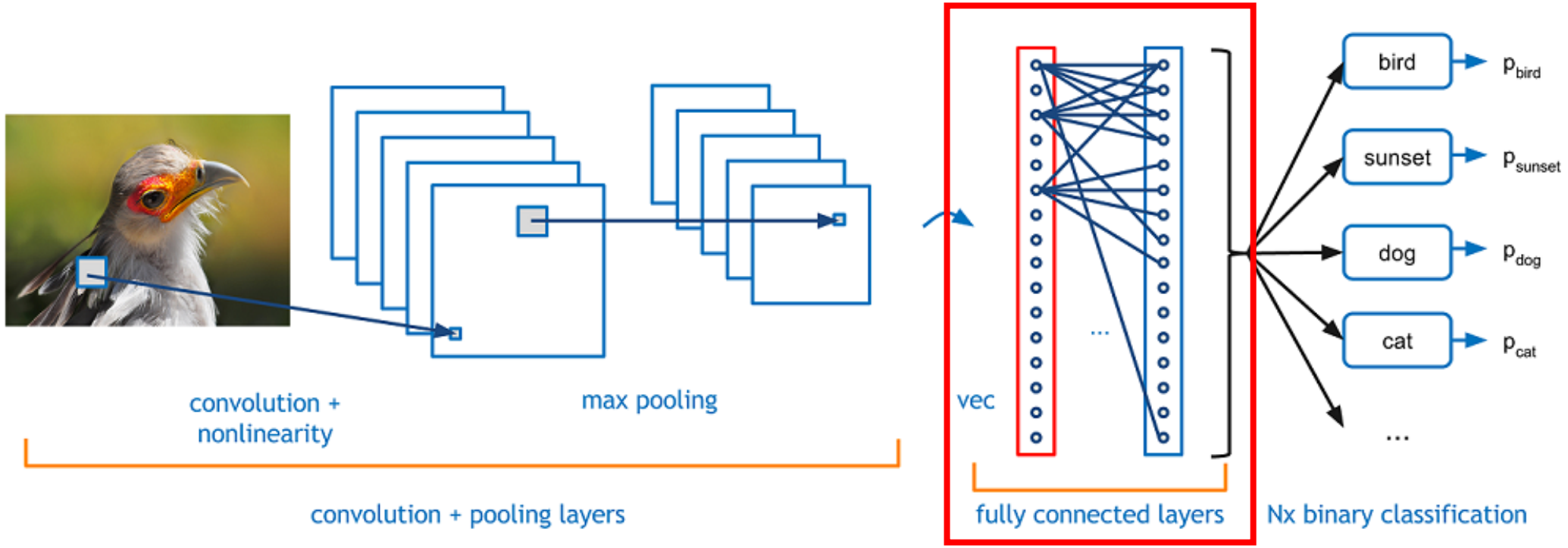
- Memory로 부터 원하는 parameter를 읽어 연산에 사용하는 방법을 익힌다.

1.

Background

Fully-Connected Layer in Neural Network

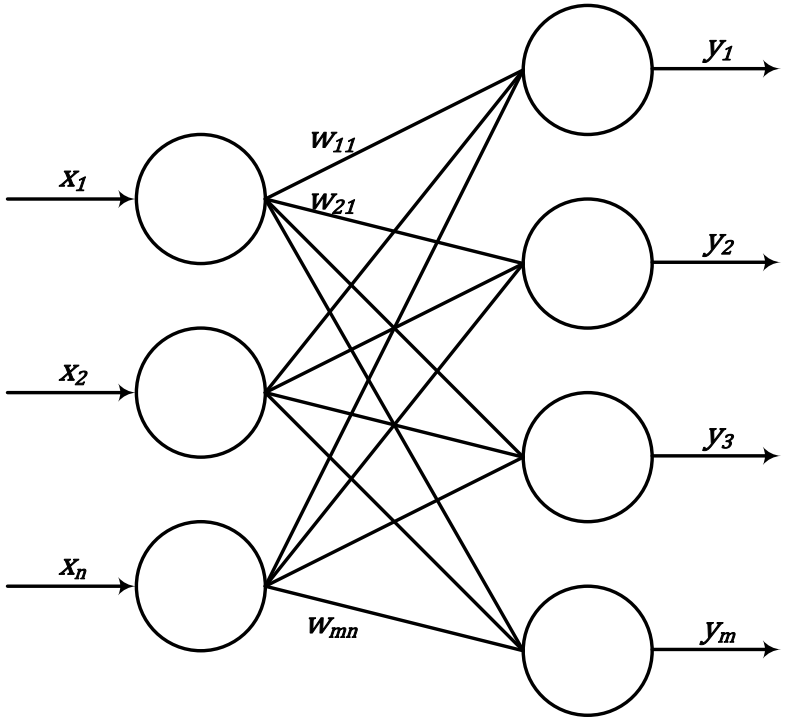
- FC layer in CNN for image classification

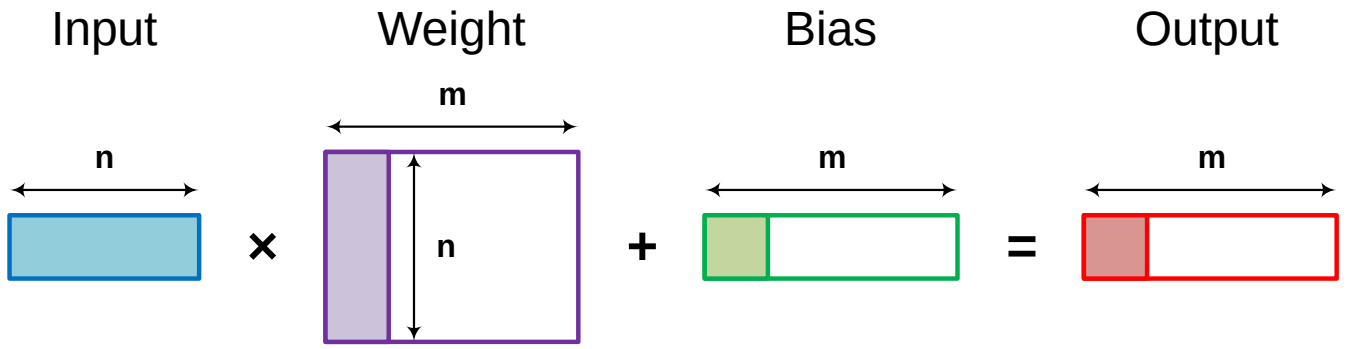


- 주로 network 끝단에 위치하여, convolution layer에 의해 추출된 feature를 통해 image를 분류하는 classifier 역할.

Reference: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Operation of FC Layer

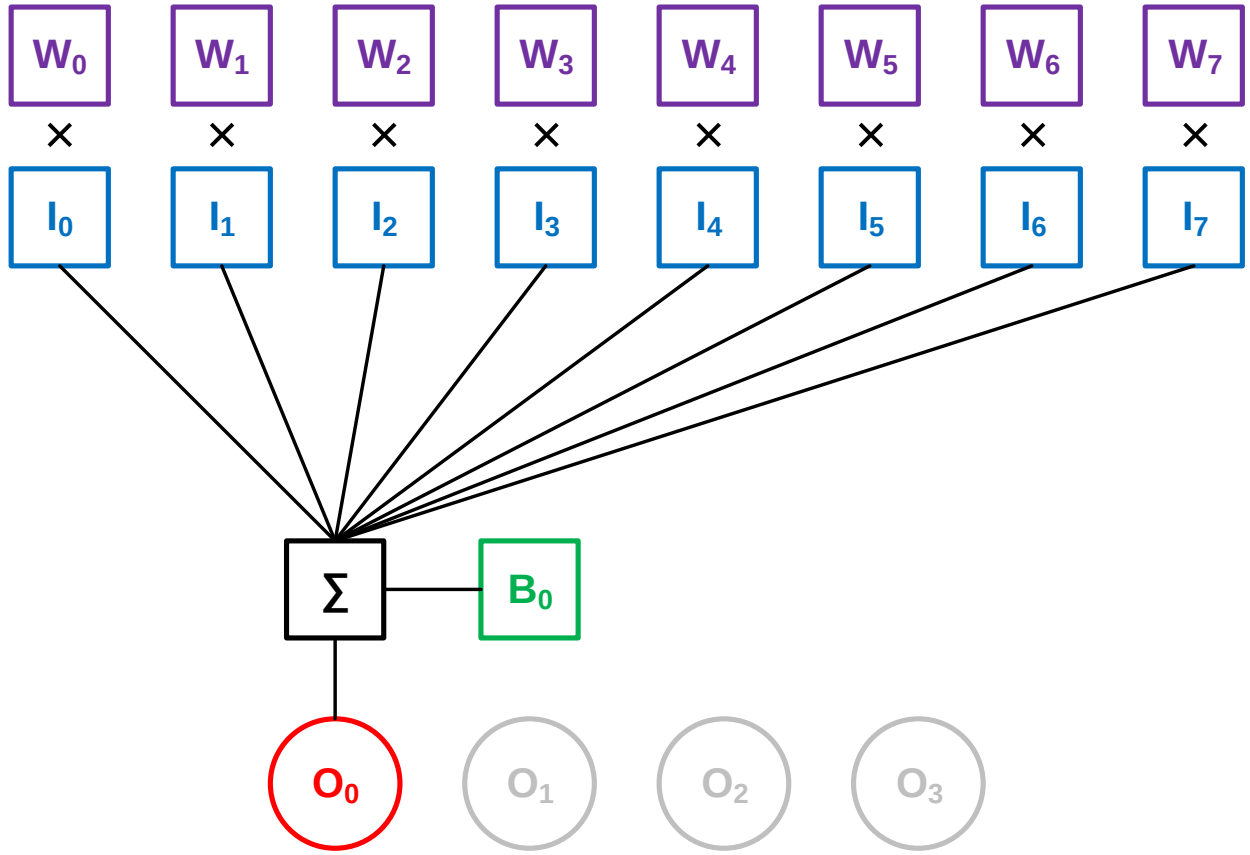




$$y_1 = x_1 \times W_{11} + x_2 \times W_{12} + x_3 \times W_{13} \cdots + x_n \times W_{1n} + b_1$$
$$y_2 = x_1 \times W_{21} + x_2 \times W_{22} + x_3 \times W_{23} \cdots + x_n \times W_{2n} + b_2$$
$$\vdots$$
$$y_m = x_1 \times W_{m1} + x_2 \times W_{m2} + x_3 \times W_{m3} \cdots + x_n \times W_{mn} + b_m$$

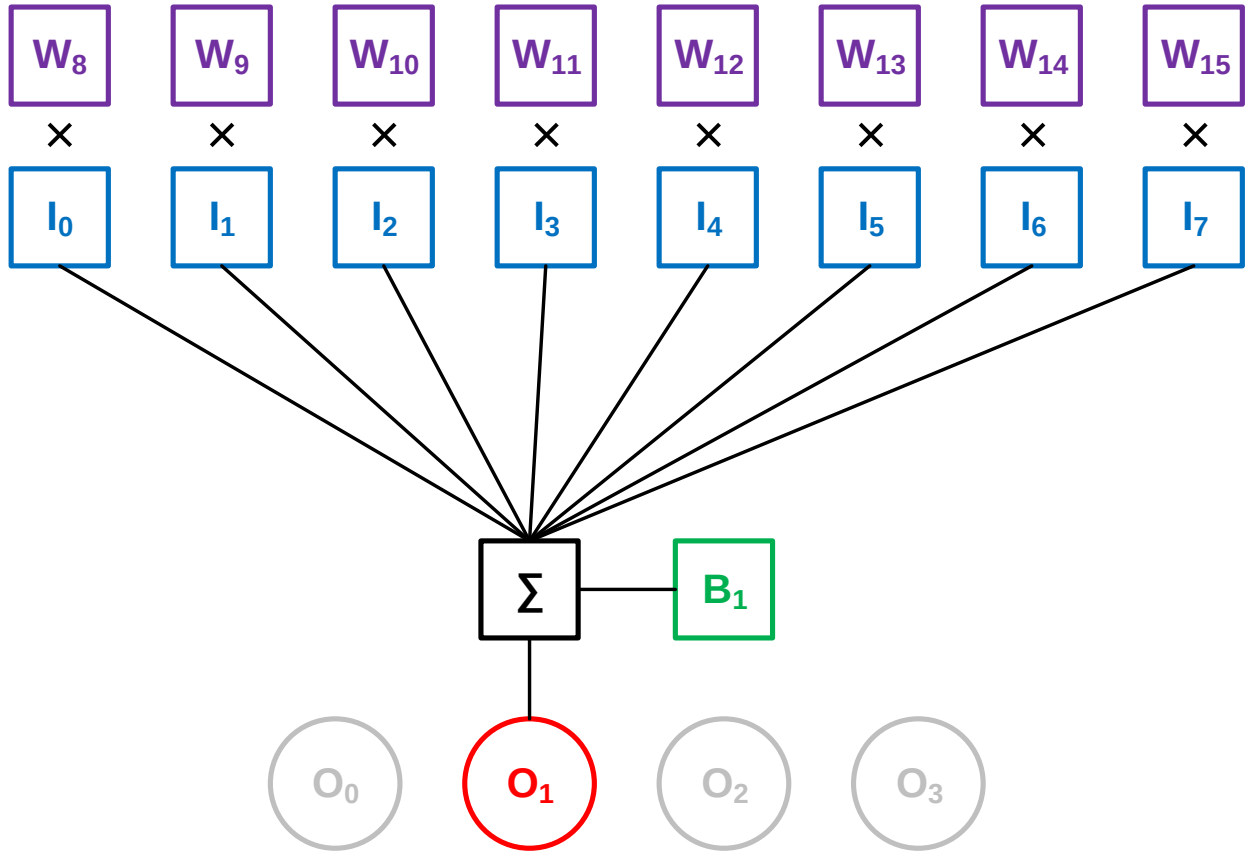
Example of FC Layer

- **8 X 4** fully connected layer
  - ‘I’nput Size = **8**
  - ‘O’utput Size = **4**
  - ‘W’eight Size = Input Size x Output Size = 8 x 4 = **32**
  - ‘B’ias Size = Output Size = **4**



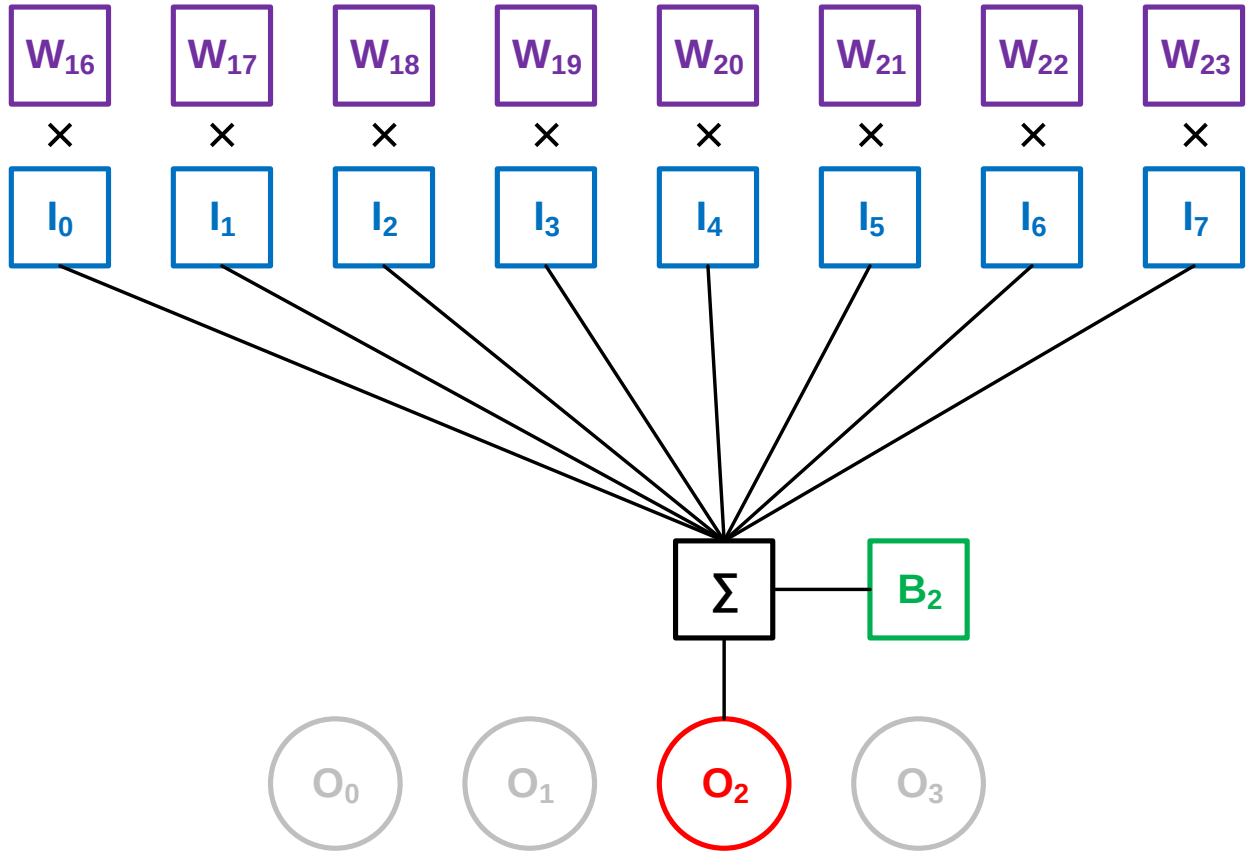
Example of FC Layer

- **8 X 4** fully connected layer
  - ‘I’nput Size = **8**
  - ‘O’utput Size = **4**
  - ‘W’eight Size = Input Size x Output Size = 8 x 4 = **32**
  - ‘B’ias Size = Output Size = **4**



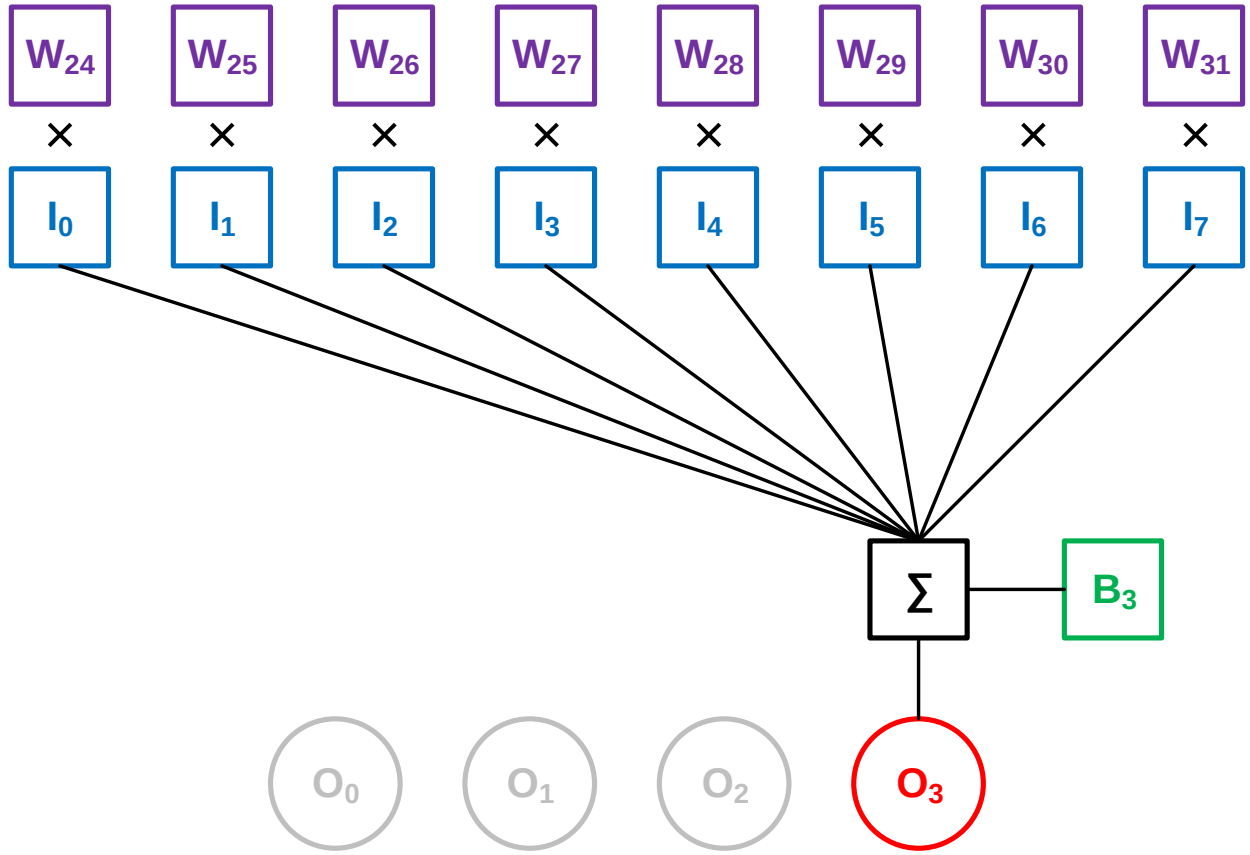
Example of FC Layer

- **8 X 4** fully connected layer
  - ‘I’nput Size = **8**
  - ‘O’utput Size = **4**
  - ‘W’eight Size = Input Size x Output Size = 8 x 4 = **32**
  - ‘B’ias Size = Output Size = **4**



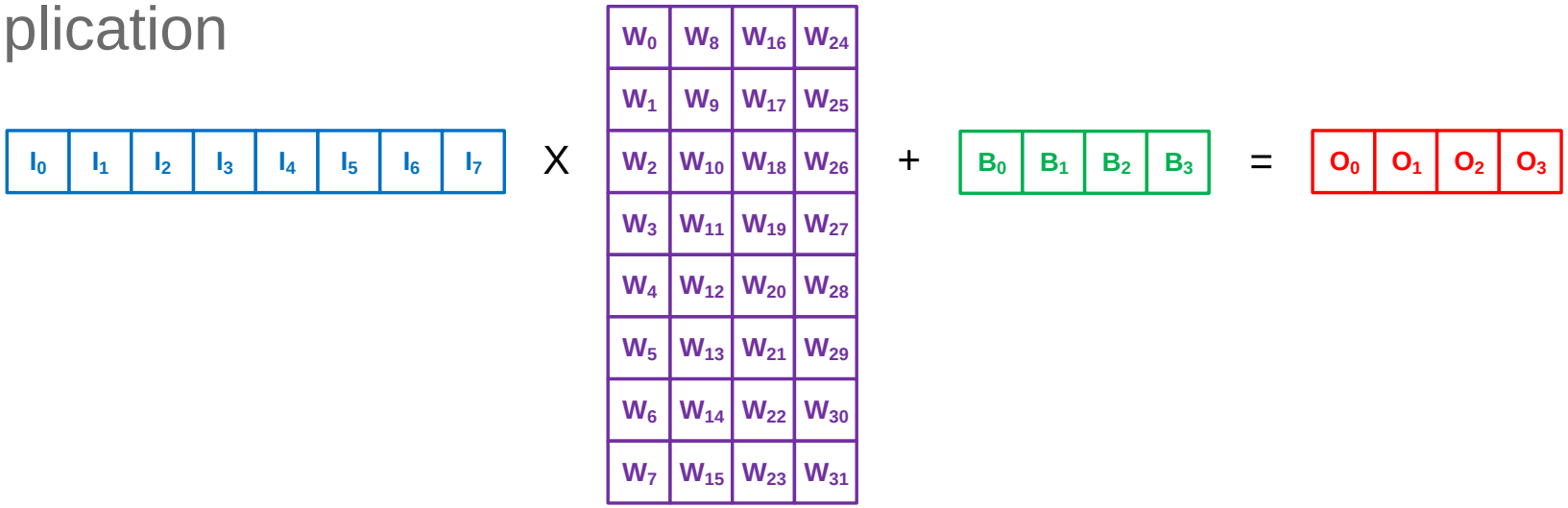
Example of FC Layer

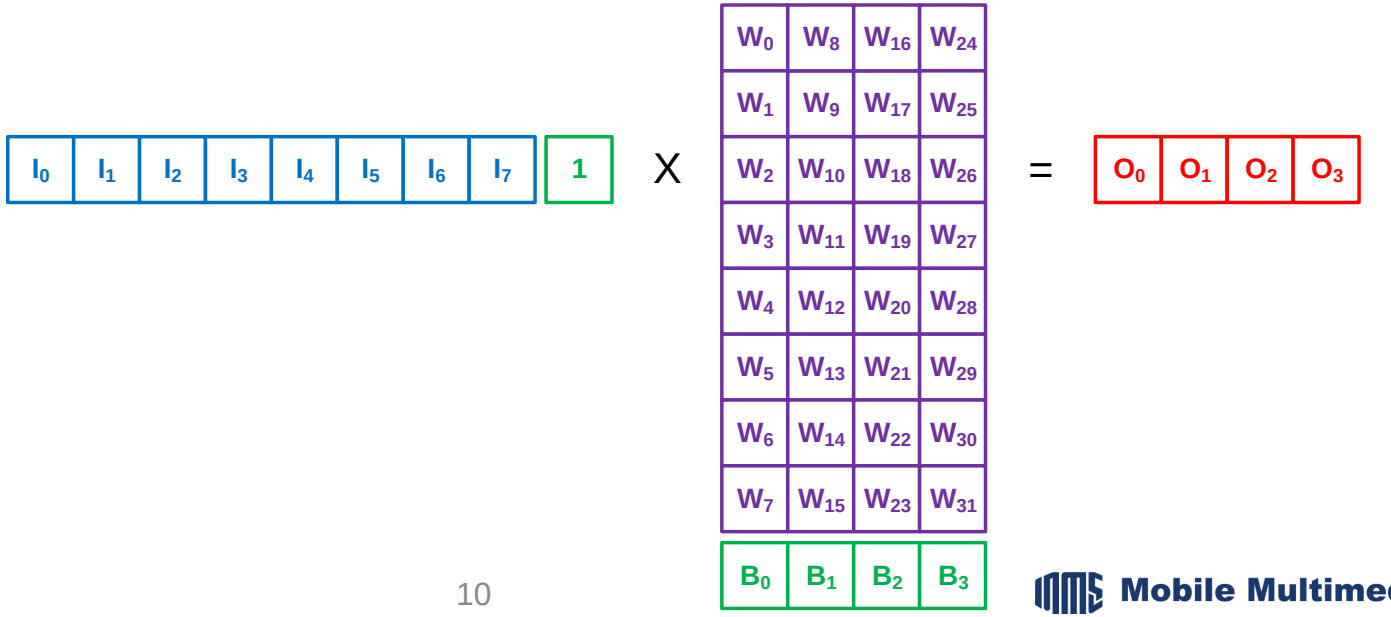
- **8 X 4** fully connected layer
  - ‘I’nput Size = **8**
  - ‘O’utput Size = **4**
  - ‘W’eight Size = Input Size x Output Size = 8 x 4 = **32**
  - ‘B’ias Size = Output Size = **4**



Example of FC Layer

- Represented as Matrix Multiplication
  : I · W + B = O



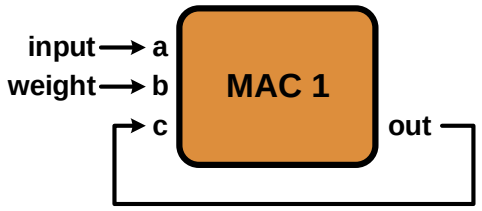


Multiply-Accumulate Operation (MAC)

- 두 수를 곱하고(**multiply**) 여기에 또 다른 값을 더하는(**accumulate**) 연산
  ➔ $Out = A \times B + C$

- 디지털 신호 및 멀티미디어 데이터 처리, 머신 러닝에서 핵심이 되는 연산

- MAC 연산의 반복적인 수행을 통해, fully-connected layer와 convolution layer의 동작을 수행할 수 있다.

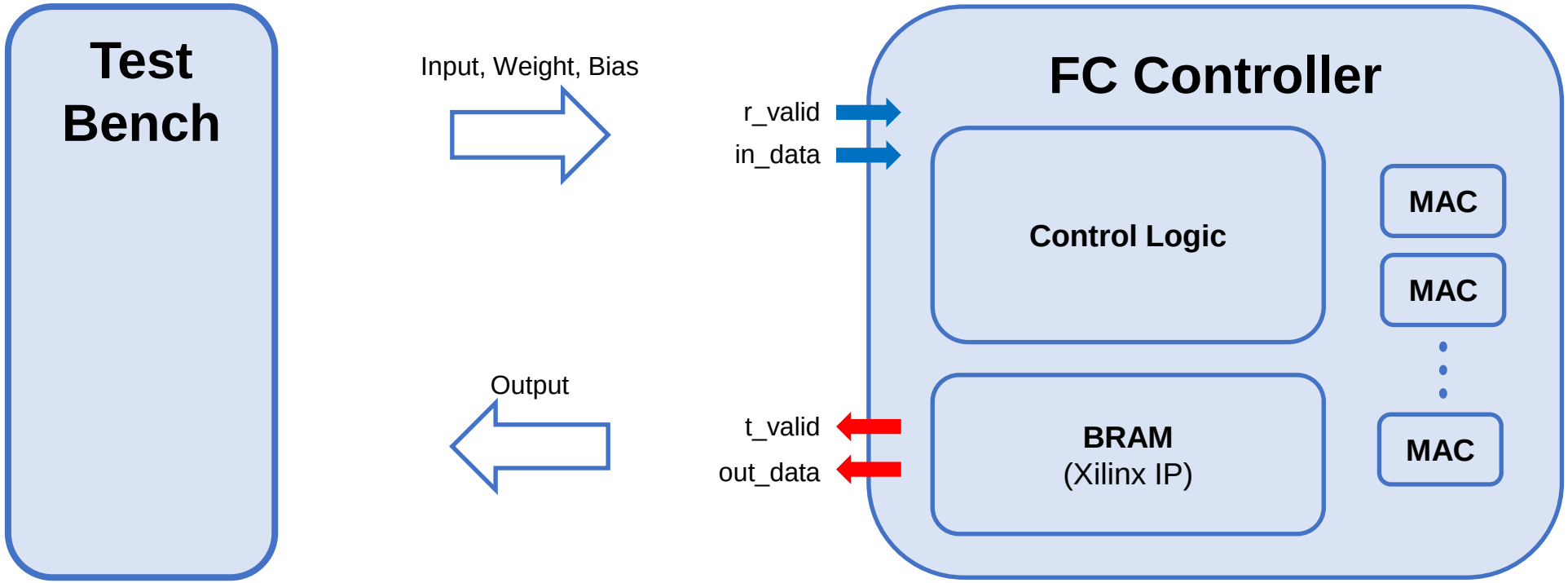


$$y_1 = x_1 \times W_{11} + x_2 \times W_{12} + x_3 \times W_{13} \cdots + x_n \times W_{1n} + b_1$$
$$y_2 = x_1 \times W_{21} + x_2 \times W_{22} + x_3 \times W_{23} \cdots + x_n \times W_{2n} + b_2$$
$$\vdots$$
$$y_m = x_1 \times W_{m1} + x_2 \times W_{m2} + x_3 \times W_{m3} \cdots + x_n \times W_{mn} + b_m$$

2.

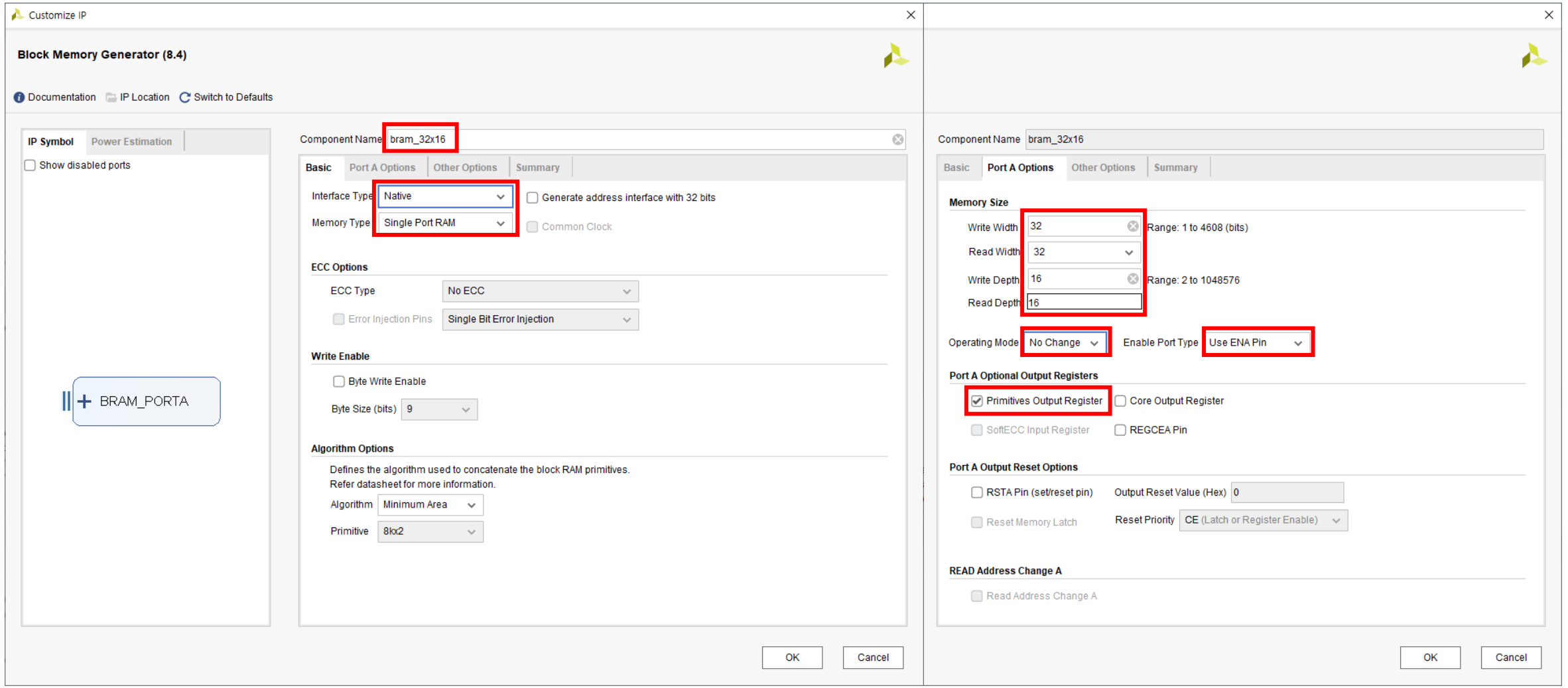
FC Layer Operation

Architecture of FC Layer in Lab7



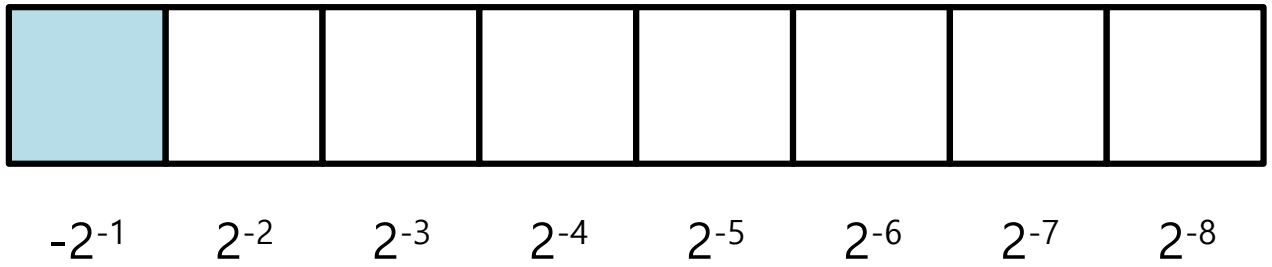
BRAM IP Generation

- 32bit width x 16 depth의 single port RAM을 사용



Number Representation Used In Lab7

- 8-bit 2's fixed point
  - 소수점 위치가 특정 위치에 고정되어 있다고 가정.
  - 2's complement representation 형태를 따른다.
  - 이번 lab에서는 $-0.5 \leq x \leq 0.49609375$ 의 범위를 가지는 데이터를 사용.

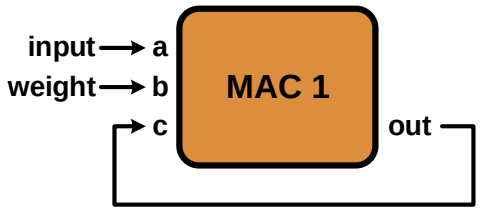


- Example
  - $00100110 = 2^{-3} + 2^{-6} + 2^{-7} = (2^5 + 2^2 + 2^1) \times 2^{-8} = 0.1484375$
  - $10011001 = -2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} = (-2^7 + 2^4 + 2^3 + 2^0) \times 2^{-8} = -0.40234375$
  - $10000000 = -2^{-1} = (-2^7) \times 2^{-8} = -0.5$

8bit 2's Fixed Point MAC Unit

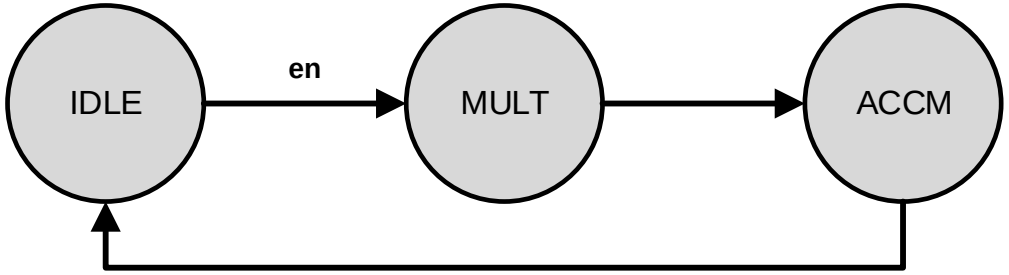
- 8bit 2's fixed point data인 data_a, data_b를 곱하고 더 큰 bit의 data인 data_c를 더하여 결과를 출력
  - 곱셈과 덧셈을 하면 할수록 bit width가 증가하므로 data_c와 out의 bit width는 충분히 크게 설정하여야 한다.
  - 각 data는 parameterized 되어 있다. Output에 필요한 bit width가 어느 정도 될지 확인하고 OUT_BITWIDTH를 선택하여 구현한다.(C_BITWIDTH = OUTBITWIDTH-1)

- MAC 내부의 adder를 이용하여 덧셈 연산만을 따로 수행할 수 있도록 구현
  - 'add' signal이 켜지면, data_a와 data_c의 덧셈을 수행한다.
  - 이 때, data_a는 data_c와 fixed point 위치를 일치하기 위해, 일정 bit 수를 shifting 시켜야 한다.



$$y_1 = x_1 \times W_{11} + x_2 \times W_{12} + x_3 \times W_{13} \cdots + x_n \times W_{1n} + b_1$$
$$y_2 = x_1 \times W_{21} + x_2 \times W_{22} + x_3 \times W_{23} \cdots + x_n \times W_{2n} + b_2$$
$$\vdots$$
$$y_m = x_1 \times W_{m1} + x_2 \times W_{m2} + x_3 \times W_{m3} \cdots + x_n \times W_{mn} + b_m$$

8bit 2's Fixed Point MAC Unit



- MAC unit operation

TO DO

1. '***en***'이 1이 되었을 때, 각 data의 bf에 data를 capture하고 MAC연산을 시작한다. (STATE_IDLE)
2. '***add***'가 0이면 곱셈을 수행하고 1 이면 덧셈을 위한 shifting 을 수행하여 '***out_temp***'에 넣어준다. (STATE_MULT)
   - capture 된 값을 이용
3. 덧셈을 수행하여 최종 계산 값 출력. 이때 '***done***'을 1로 올려주고, STATE_IDLE 로 이동 (STATE_ACCM)
   - '***done***'은 **1 cycle 동안**만 1로 올려주고, 이후로는 0으로 유지해야 한다.

- 주어진 FSM은 간단하게 구현해볼 수 있도록 제공된 것이며, 개인의 구현에 따라 자유롭게 변경해도 된다.

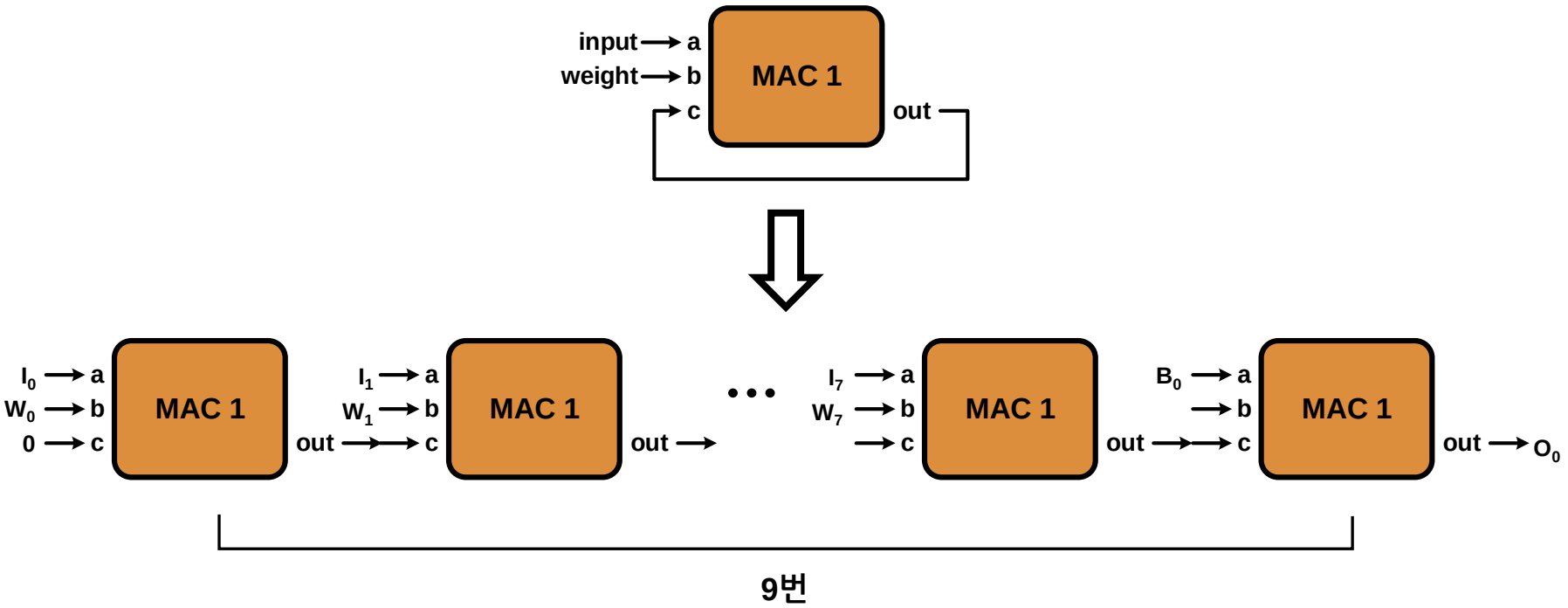
How To Use MAC Units

- FC layer 내 MAC optimization 방법
  - MAC을 reuse 하여 area를 줄인다.
  - MAC의 수를 늘려서 parallelism 을 높인다.
  - 여러 MAC을 pipelining으로 사용한다.

How To Use MAC Units

- MAC을 reuse 하여 area를 줄이는 방법

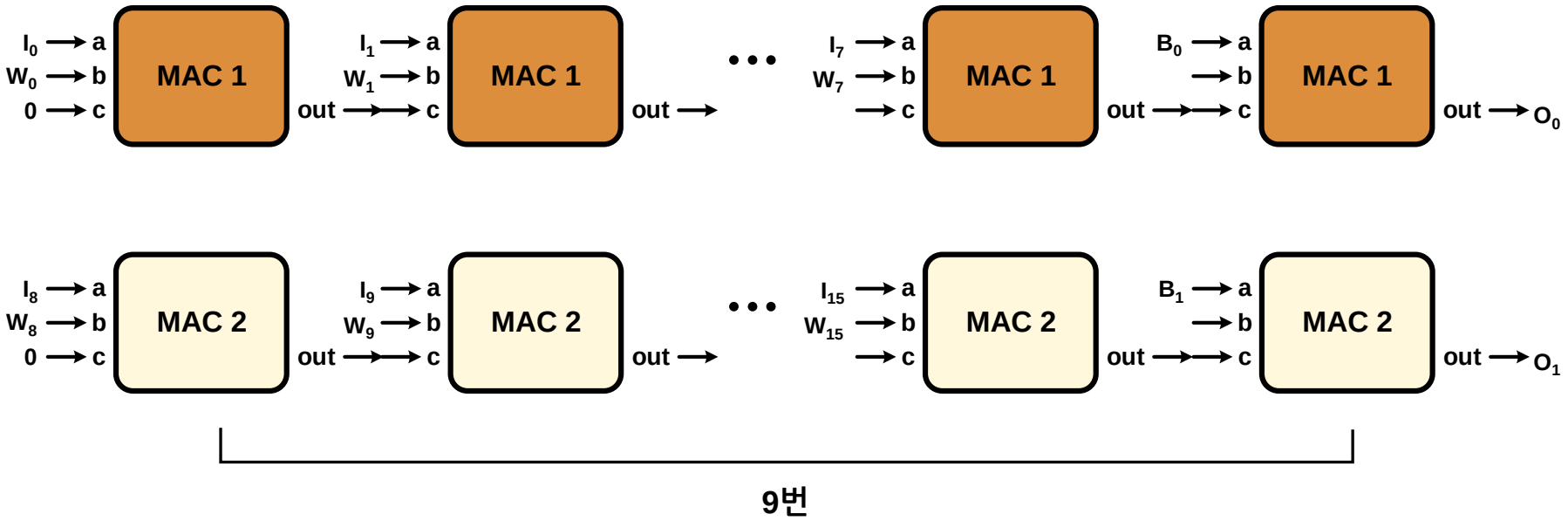
- MAC 하나 사용 시
  - 그림과 같이, input * weight를 누적(accumulate)하여 더한다.
  - 마지막에는 bias를 더해준다.



How To Use MAC Units

- MAC의 수를 늘려서 parallelism 을 높이는 방법
  - 속도는 증가하지만 area도 증가하는 trade-off 가 존재

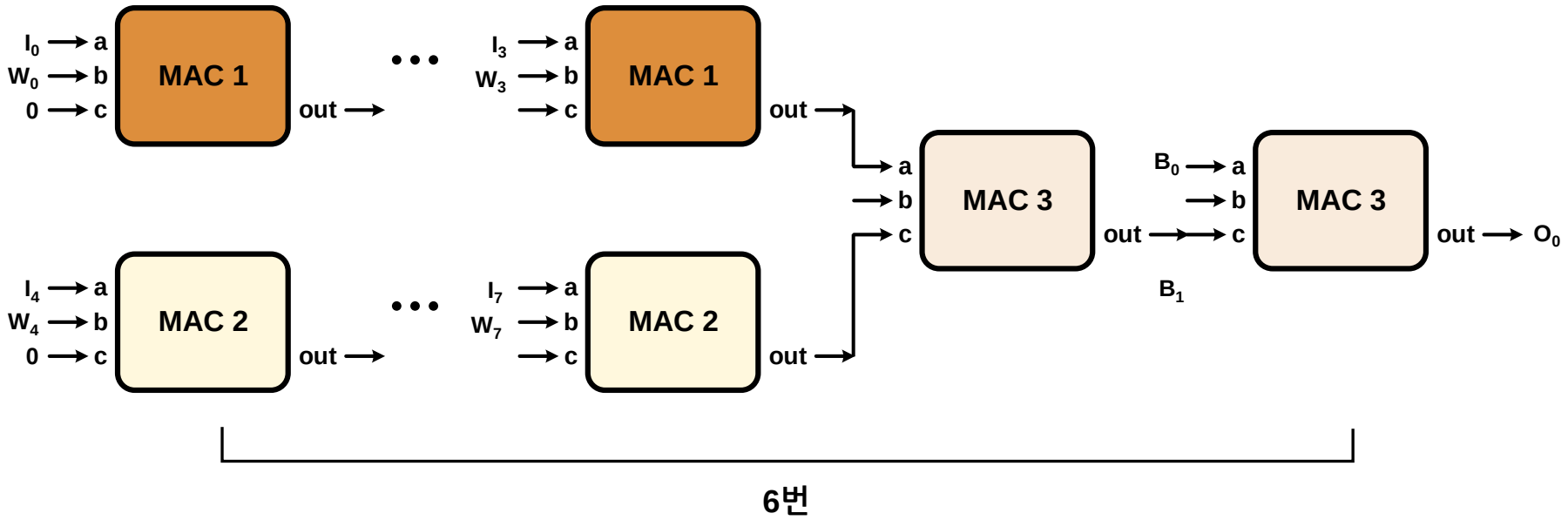
- MAC의 수를 늘려 독립적으로 parallel 하게 사용



How To Use MAC Units

- MAC의 수를 늘려서 parallelism 을 높이는 방법
  - 속도는 증가하지만 area도 증가하는 trade-off 가 존재

- MAC의 수를 늘려 Add-Tree를 만들어 Add를 빠르게 실행



Control Flow of FC Controller

- Testbench로 부터 input, weight, bias의 데이터를 받아 BRAM의 각 지정된 위치에 저장
  - Skeleton Code의 **STATE_DATA_RECEIVE**에 구현되어 있다. (**Not allowed to change**)

- **TO DO 1: BRAM에서 데이터를 적당히 Read 하여 주어진 reg에 저장**
  - reg: input_feature, weight, weight_bf, bias (불러올 수 있는 **size가 고정됨**. **Not allowed to change**)
  - 불러올 데이터의 크기가 너무 커, 데이터를 module **내부의 register에 전부 저장할 수 없는 상황**을 modeling

- **TO DO 2: input_data, weight, bias 의 값들을 적절히 MAC에 넣어가며 주어진 fc layer의 계산을 진행하고 결과를 저장**

- 계산이 완료되었으면, result를 output port인 out_data에 넣고 t_valid를 1로 set한다.

Operation Rule of FC Controller

- FC layer model: input size = 8 / output size = 4
- tb에서 data를 한꺼번에 받아 BRAM의 정해진 주소 범위에 저장
  - input: INPUT_START_ADDRESS / weight: WEIGHT_START_ADDRESS / bias: BIAS_START_ADDRESS
- 계산은 BRAM에서 read하여 주어진 reg 변수에 저장한 뒤에 진행
  - BRAM에서 input / weight / bias 값들을 읽어와서 저장
- reg 변수의 크기는 고정
  - input과 bias의 경우 BRAM에 저장된 값을 모두 reg에 저장 가능
  - weight는 reg의 크기가 전체 weight의 크기 보다 작다.
- Quantization Rule
  - Layer 내부의 각각의 MAC 연산 단계에서는 quantize 하지 않음
  - 마지막 연산 결과 (output) : 8-bit 2’s fixed point로 quantize
- out_data (32-bits)
  - 4개의 quantize된 연산 결과를 저장
  - t_valid 를 1로 바꾸어 test bench에 결과값을 내보낸다.

Quantization Rule

- 각 input, weight, bias를 **8-bit 2's fixed point 로 받아** 계산 후, output으로 **8-bit 2's fixed point 을 출력**

- 그러나 Layer 내부에서 **각각의 MAC 계산 단계에서는 Quantize를 하지 않는다.**
  - 데이터 손실 최소화

- 마지막으로 나온 MAC 연산 결과만 다시 8-bit 2's fixed point 로 quantize
  - 단, Quantize를 하면서 **overflow 조건**을 고려해야 한다.

- 이를 위해, layer 내부 MAC의 OUT_BITWIDTH를 계산에서 손실이 없도록 크게 잡는다.
  - 이번 lab에서는 8bits인 두 값의 곱 8개와 8bits 값 1개가 더해져 결과가 나온다.
    ➔ $16 + \lceil \log_2 9 \rceil = 20\text{bits}$
  - 따라서 OUT_BITWIDTH=20bits (C_BITWIDTH=19bits)로 구현하는 방법을 추천 (Skeleton_2)

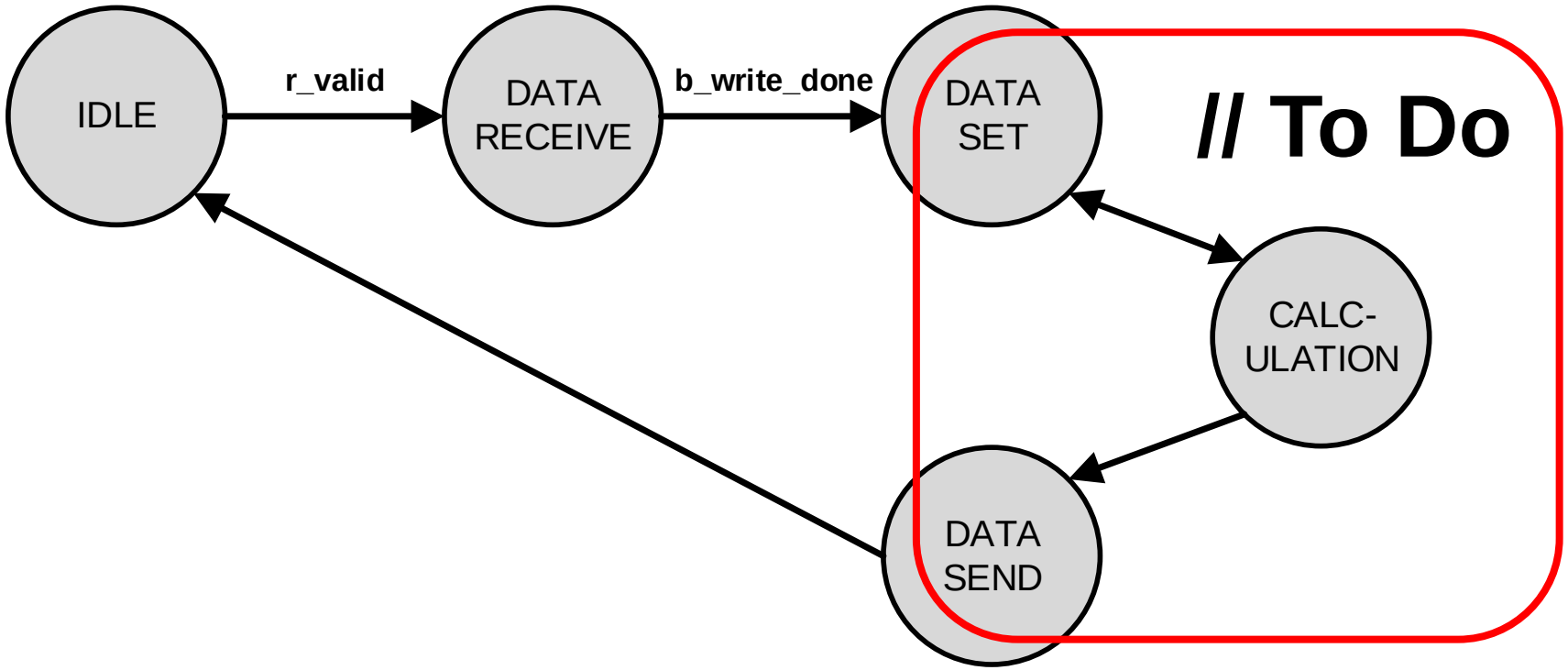
Quantization Rule

- **20-bit 2's fixed-point를 8-bit 2's fixed-point로 변환**
  - [19] : sign bit of 2's complement
  - **[18:15] : overflow check bits** ➔ positive overflow: **8'b01111111** / negative overflow: **8'b10000000**
  - [14:8] : 8-bit 2's fixed-point로 변환하였을 때, 실제 데이터 부분
  - [7:0] : 버리는 부분
  - Example
    - 20'b1_1111_1101100_10000001 ➔ 8'b11101100
    - 20'b0_0000_0010011_10111100 ➔ 8'b00010011
    - 20'b1_1110_1110100_01101011 ➔ 8'b10000000

- Why do quantization?
  - 데이터의 크기를 줄이고 통신을 빠르게 하기 위함
  - 계산할 때 마다 8-bits로 quantize 를 하면 accuracy 가 줄어 든다.
  - accuracy 의 감소를 최소화하기 위해 내부 계산에서는 quantize를 하지 않는다.

- 해당 Quantization 관련 문제는 알고리즘의 HW 설계에서 반드시 검증이 필요!

Basic FSM of FC Controller



Buffer 관련 wire, reg 설명(fc_controller.v)

이름	설명
input_feature	FC 계산을 위해 BRAM의 어느 정도의 input 값을 임시로 저장해두는 buffer
weight	FC 계산을 위해 BRAM의 어느 정도의 weight 값을 임시로 저장해두는 buffer
weight_bf	FC 계산과 BRAM 에서 data 를 read 하는 것을 parallel 하게 진행하기 위해 BRAM에서 읽어 온 weight data를 임시로 저장해둔 buffer.
bias	FC 계산을 위해 BRAM의 어느 정도의 weight 값을 임시로 저장해두는 buffer

BRAM 관련 wire, reg 설명(fc_controller.v)

이름	설명
r_valid	TestBench가 Data를 보낼 준비가 됐다는 Flag
bram_state	BRAM operation 를 위한 state
bram_addr	BRAM 에 read/write 할 address
bram_din	BRAM 에 write 할 data
bram_dout	BRAM 에 read 한 data
bram_en	1 이어야 BRAM 에 read/write 가 가능하다.
bram_we	1 이면 write, 0 이면 read
latency	BRAM read 할 때 address를 넣을 때와 그 address 에 해당하는 data 를 읽을 때 생기는 timing의 간극을 메우기 위한 latency
bram_counter	BRAM 에 read/write 할 때 특정한 개수 만큼 read/write 하기 위한 counter
b_write_done	Test bench 로 부터 받은 data 를 BRAM 에 모두 write 했다는 신호
input_set_done	BRAM 에서 일정량의 input 값을 read 하여 그 값들을 ‘input_feature’ 에 모두 저장했다는 신호
bias_set_done	BRAM 에서 일정량의 bias 값을 read 하여 그 값들을 ‘bias’ 에 모두 저장했다는 신호
weight_set_done	BRAM 에서 일정량의 weight 값을 read 하여 그 값들을 ‘weight_bf’ 에 모두 저장했다는 신호 (bram_state 내부 신호)
weight_counter	BRAM 에 weigh를 read 할 때 몇 번째 output의 weight를 read 하고 있는지 알려주는 counter

FC 관련 wire, reg 설명(fc_controller.v)

이름	설명
mac_state	FC calculation 을 위한 state
mac_en	MAC 의 enable 신호
mac_add	MAC 의 add operation 신호
data_a	MAC 의 a * b + c 에서의 a
data_b	MAC 의 a * b + c 에서의 b
data_c	MAC 의 a * b + c 에서의 c
mac_done	MAC 의 계산이 끝났음을 알려주는 신호
mac_result	MAC 의 결과값
mac_counter	STATE_ACCUMULATE 의 반복 횟수를 세는 counter. 이번 실험에서는 한 번에 8 번의 반복을 해야함.
output_counter	현재 몇 번째 output 의 계산 중인지를 알려주는 counter
partial_sum	FC를 계산할 때의 누적되는 중간 결과값. MAC의 output으로 나오고 MAC의 input으로 넣어준다.
result_q	*'mac_result'* 를 8-bits fixed point 로 quantize 한 값
t_valid	Testbench에 output data를 보내겠다는 신호

Control Flow of FC Controller (skeleton2의 fc_controller.v)

- BRAM Operation
  1. '***r_valid***' 가 1 이면 Test bench 가 데이터를 보낼 준비가 됐다는 뜻이므로 '***bram_state***' 를 STATE_DATA_RECEIVE 로 바꾼다. (STATE_IDLE)
  2. Test bench 로 부터 input, weight, bias를 받아서 내부 BRAM에 각자 지정된 주소에 저장(write)한다. (STATE_DATA_RECEIVE)
  3. BRAM에서 데이터를 읽어, 주어진 register인 '***input_feature***' 에 값을 저장한다. (STATE_INPUT_SET)
  4. BRAM에서 데이터를 읽어, 주어진 register인 '***bias***' 에 값을 저장한다. (STATE_BIAS_SET)

TO DO (STATE_WEIGHT_SET)

  5. BRAM에서 데이터를 읽어, 주어진 register인 '***weight_bf***' 에 값을 저장한다. weight 는 개수가 많아 한 번에 setting 할 수 없으므로 저장 가능한 weight (8 bytes)만 받아서 저장하고 '***weight_set_done***'을 1로 바꾼다.
  6. mac_state가 STATE_IDLE 이면 mac이 사용 가능하다는 뜻이므로 '***weight_bf***'의 값을 '***weight***'에 옮기고 '***weight_set_done***'을 0으로 바꾼다.
  7. 읽을 weight 이 남아 있다면 다음 weight을 준비하고, 없다면 STATE_IDLE 로 이동한다.

Control Flow of FC Controller (skeleton2의 fc_controller.v)

- **FC Calculation Operation**

TO DO

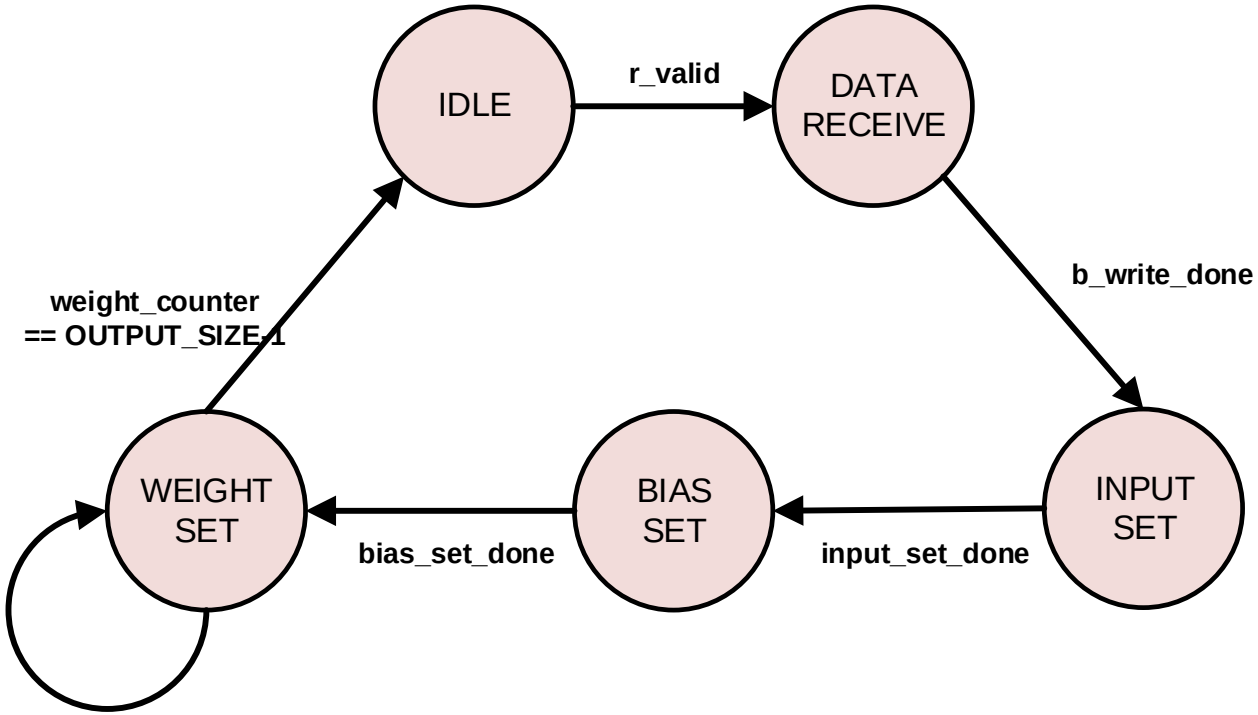
1. *'**weight_set_done**'* 이 1 이면 *'**state**'* 를 STATE_ACCUMULATE 바꾼다. (STATE_IDLE)

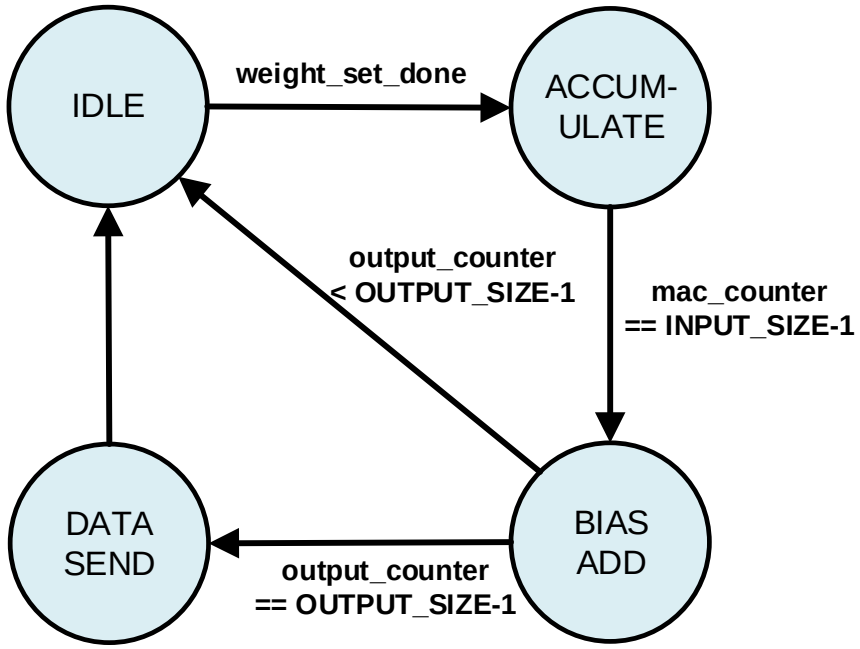
2. 하나의 MAC unit을 사용하여, *'**input_data**'*, *'**weight**'* 에서 각각 하나(1 byte)씩 MAC의 a, b에 넣고 그 결과를 c, 그 다음 input과 weight을 a, b에 넣어 MAC을 reuse 한다. 이것을 하나의 output 을 구하기 위해 8번을 반복한다. (STATE_ACCUMULATE)

3. 위의 결과와 '***bias***'값을 MAC에 넣어 연산을 수행한다. (STATE_BIAS_ADD)

4. 2~3 번을 4번을 반복하고 모든 output이 계산되면 8-bit로 변환하고 *'**t_valid**'* 를 1 로 바꾸어 *'**out_data**'* 를 test bench 로 보낸다.

FSM of FC Controller (skeleton2º| fc_controller.v)





3.

Overview of Lab 7

Description of Lab 7

- **mac.v: Skeleton code for MAC module**
  - TODO 부분을 구현 또는 원하는 구현 방향으로 수정 가능.
- **fc_controller.v: Skeleton code for fc controller module**
  - Skeleton_1: 주어진 rule만 지키고 나머지는 원하는 방법으로 구현. 구현의 자유도를 위해 제공.
  - Skeleton_2: 정해진 state와 동작에 맞춰 구현.
- tb_fc.v: Testbench for testing implemented FC controller
- Nexys4DDR_Master.xdc: Constraint file for post-synthesis simulation

- **Not allowed to change** 부분은 수정하면 안된다.

- 그 외의 변수 이름, state 이름, state 수, wire/reg 전환, 구현 방법 등 TO DO가 아닌 부분 또한 수정해도 무관하다.

Post-synthesis Simulation Result Check

- 구현 후 synthesis를 수행하고 post-synthesis simulation까지 완료
- 두 번의 case에 대하여 테스트를 진행. Tcl console을 통해 테스트 통과 여부를 확인 가능

```
////////////////////////////////////////////////////////////
////////////////////// First Test Start //////////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
////////////////// First Result is correct! //////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
////////////////////// Second Test Start //////////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
///////////////// Second Result is correct! ////////////////
////////////////////////////////////////////////////////////
```

Pass

```
////////////////////////////////////////////////////////////
////////////////////// First Test Start //////////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
////////////////// First Result is wrong! //////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
////////////////////// Second Test Start //////////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
///////////////// Second Result is wrong! ////////////////
////////////////////////////////////////////////////////////
```

Fail

Submission

- 구현한 source file(***mac.v, fc_controller.v***)과 post-synthesis simulation 화면 및 Tcl console 창의 pass 부분을 같이 capture한 이미지 파일을 압축하여 eTL에 제출

- File name: “학번_이름_lab7.zip”
  - ex) 2020-12345_홍길동_lab7.zip

- Deadline: **10월 31일 월요일 23:59분까지**