

1. Goal

- Implement base of Float32 MUL+ACC
- Implement Adder, Multiplier, Fused multiplier (Verliog)

2. Implement details

2.1. Adder

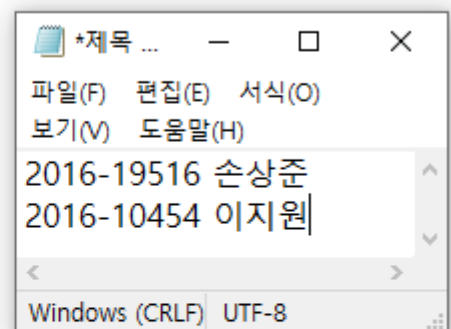
이 누산기는 2개의 input, 1개의 output, 1개의 overflow detecting bit 로 이루어져 있다.

같은 크기의 bitwidth를 가지는 2개의 input을 더했을 때, output의 bitwidth도 동일하기 때문에, overflow 가 발생할 수 있다.

Overflow detecting은 concatenation assign을 통해 구현하였다.

Overflow bit와 output의 총 길이는 bitwidth+1이고, 2개의 input을 더했을 때 나올 수 있는 최대 bit수는 bitwidth+1이기 때문에, 이를 통해 overflow를 detect 할 수 있다.

```
1  `timescale 1ns / 1ps
2
3  module my_add #(
4      parameter BITWIDTH = 32
5  )
6  (
7      input[BITWIDTH-1 : 0] ain,
8      input[BITWIDTH-1 : 0] bin,
9      output[BITWIDTH-1 : 0] dout,
10     output overflow
11 );
12
13     // concatenate (overflow, dout) & detecting overflow
14     assign {overflow,dout} = ain + bin;
15
16
17 endmodule
```



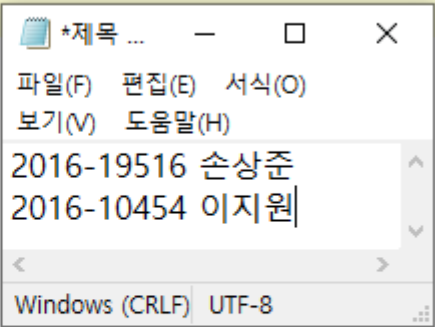
2.2. Multiplier

곱셈기는 2개의 input, 1개의 output으로 이루어져 있다. 곱셈기는 overflow가 발생할 일이 없다. 이는

$$(2^{\text{bitwidth}} - 1) \times (2^{\text{bitwidth}} - 1) = 2^{2 \times \text{bitwidth}} - 2^{\text{bitwidth}+1} + 1$$

에서 알 수 있듯이, 최대 bit수는 $2 \times \text{bitwidth}$ 이기 때문이다.

```
1  `timescale 1ns / 1ps
2
3  module my_mul#(
4      parameter BITWIDTH = 32
5  )
6  (
7      input[BITWIDTH-1 : 0] ain,
8      input[BITWIDTH-1 : 0] bin,
9      output[2*BITWIDTH-1 : 0] dout
10 );
11
12     assign dout = ain * bin;
13
14
15 endmodule
16
```



2.3. Fused Multiplier

단일 곱셈-누산기는 2개의 input, 1개의 output, *en* register, *clk* bit로 이루어져 있다.

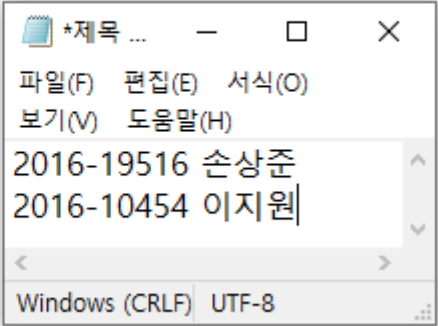
en register는 현재까지의 계산결과를 저장하고 있는 register이고, *clk*은 계산 및 누산이 수행되는 타이밍을 결정한다.

구현은 *clk*이 posedge일때, *en*과 2개의 input을 읽어 output을 계산하였다.

en 이 1인 경우, 기존의 output과 input 2개를 곱한 값을 output에 저장하였다.

en 이 0인 경우, output을 0으로 초기화하였다.

```
1  `timescale 1ns / 1ps
2
3  module my_fusedmult #(
4      parameter BITWIDTH = 32
5  )
6  (
7      input[BITWIDTH-1 : 0] ain,
8      input[BITWIDTH-1 : 0] bin,
9      input en,
10     input clk,
11     output reg [2*BITWIDTH-1 : 0] dout
12 );
13
14     // read en when clk is posedge
15     always @(posedge clk) begin
16         if(en == 1)
17             dout <= dout + ain * bin;
18         else
19             dout = 0;
20     end
21
22 endmodule
23
```



3. About Simulation results & conclusion

3.1. Adder

testbench에서 주어지는 ain, bin의 값은 $0 \sim 2^{31} - 1$ 까지의 임의의 값이다.

따라서, dout의 값은 최대 $2^{32} - 2$ 이고, 이는 32bit로 만들 수 있는 최댓값인 $2^{32} - 1$ 보다 작으므로 주어진 testbench에서는 overflow가 발생하지 않는다.

하지만, 실제로 ain, bin의 값을 증가시켜 overflow가 발생하도록 하였을 때, overflow bit가 1이 되어 detecting한다는 것을 알 수 있었다.

3.2. Multiplier

testbench에서 주어지는 ain, bin의 값은 $0 \sim 2^{31} - 1$ 까지의 임의의 값이다.

따라서, dout의 값은 최대 $2^{62} - 2^{32} + 1$ 이고, 이는 64bit로 만들 수 있는 최댓값인 $2^{64} - 1$ 보다 작으므로 주어진 testbench에서는 overflow가 발생하지 않는다.

또한, ain, bin 이 가질 수 있는 최댓값인 $2^{32} - 1$ 일때도 동일하게 overflow는 발생하지

않는다.

3.3. Fused Multiplier

testbench에서 clk, en값은 0으로 시뮬레이션 시작과 동시에 할당된다. 이후, 30ns에서 en값이 1로 변경된다. 또한, 주어지는 ain, bin의 값은 $0 \sim 2^{31} - 1$ 까지의 임의의 값이다. ain, bin의 값은 30ns에 할당되기 때문에, 30ns 이전에는 unknown으로 처리됨을 알 수 있었다

clk의 posedge는 35ns부터 10ns마다이기 때문에, dout의 값은 35ns이전에는 0(en이 0이기 때문에), 이후에는 단일 곱셈-누산기의 역할을 수행함을 확인 할 수 있었다.

특히, clk의 posedge에서만 en의 값이 읽히기 때문에, dout의 값이 0~5ns까지 unknown, 그 이후부터는 en의 값에 따른 정확한 결과값이 저장됨을 확인 할 수 있었다.