Lab. 06

Logic Design Lab.
Fall 2019

Prof. Sungjoo Yoo
(yeonbin@snu.ac.kr)
TA. Hyunsu Kim
(gustnxodjs@gmail.com)
TA. Hyunyoung Jung
(gusdud1500@gmail.com)

Contents

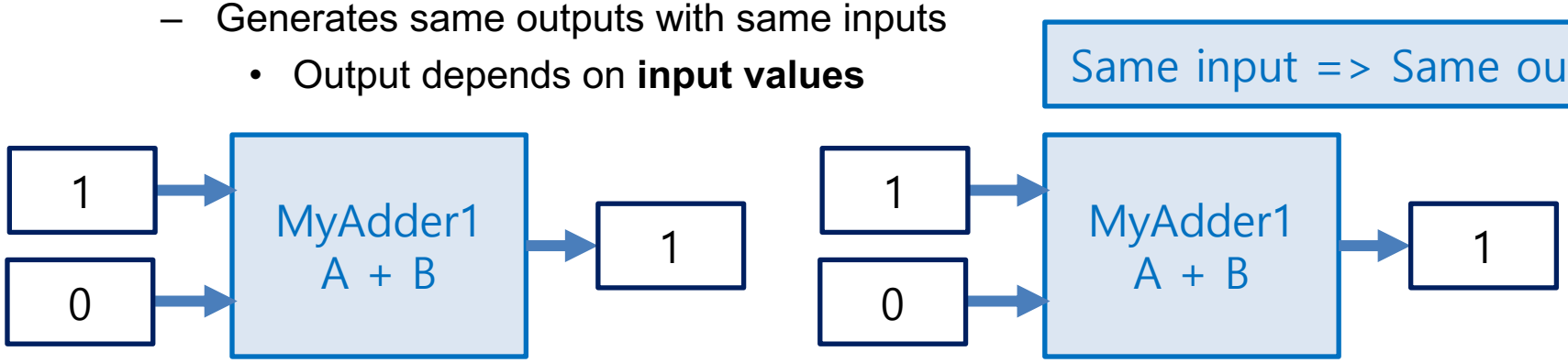
- **Sequential Logic**
- **Oscillator**

- **Latch & F/F**
- **RS Latch**
- **Gated RS Latch**
- **Master-Slave Latches**
- **Lab**

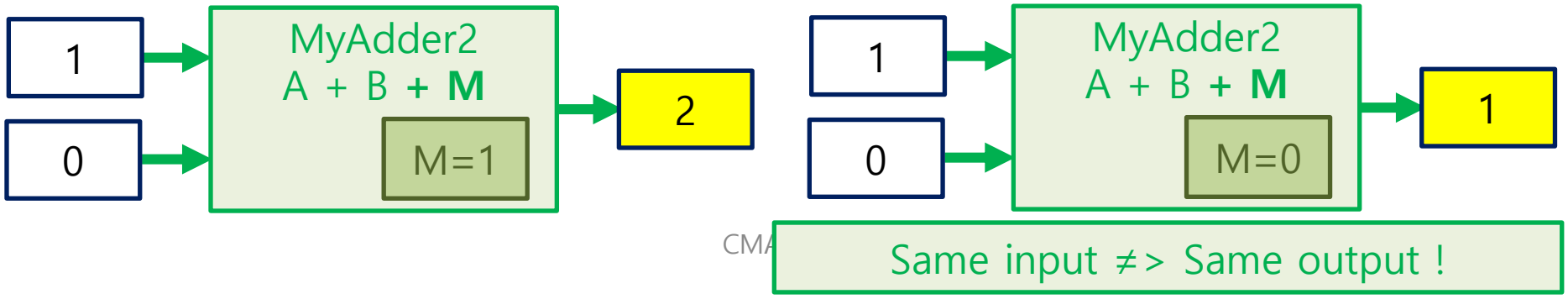
Sequential Logic

Sequential Logic

- Combinational Logic
  - Generates same outputs with same inputs
    - Output depends on **input values**

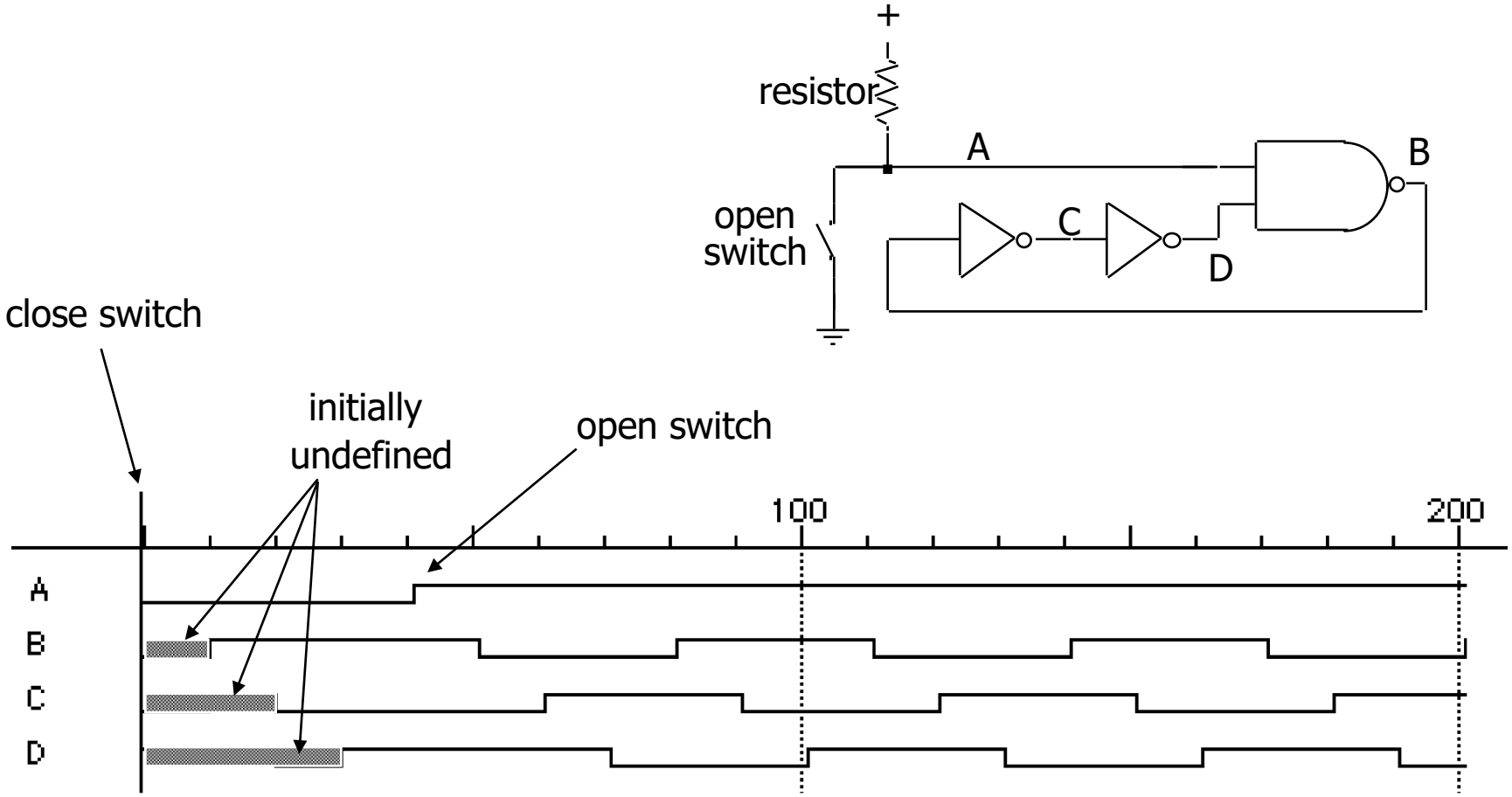


- Sequential Logic
  - Consist of **combinational logic** and **memory components**
  - Can generate different outputs with same inputs
    - Output depends on **input** and **memory values**



Oscillator

Oscillatory Behavior



Verilog Implementation (Oscillator)

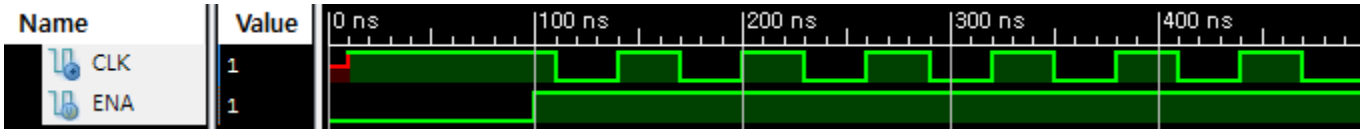
<Module>

```
1  `timescale 1ns / 1ps
2
3  module oscillator(
4      input ENA,
5      output CLK
6      );
7
8      wire tmp[1:0];
9
10     assign #10 CLK = ~(ENA & tmp[1]);
11     assign #10 tmp[0] = ~CLK;
12     assign #10 tmp[1] = ~tmp[0];
13
14 endmodule
```

<Sample Testbench>

```
1  `timescale 1ns / 1ps
2
3  module osc_test;
4
5     reg ENA;
6     wire CLK;
7
8     oscillator uut (
9        .ENA(ENA),
10       .CLK(CLK)
11    );
12
13    initial begin
14       ENA = 0;
15       #100;
16
17       ENA = 1;
18    end
19
20 endmodule
```

<Simulation>



Review – latches & F/Fs

Flip-Flops

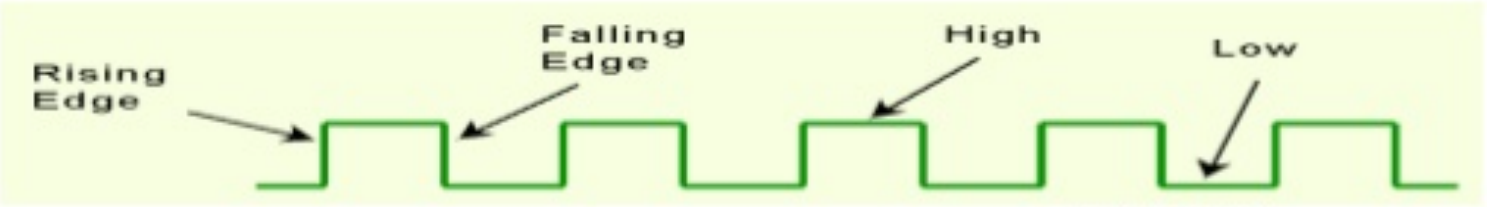
- A flip-flop is a bi-stable device: a circuit having 2 stable conditions (0 or 1)
- 3 classes of flip-flops
  - latches: outputs respond immediately while enabled (no timing control)
  - pulse-triggered flip-flops: outputs response to the triggering pulse
  - edge-triggered flip-flops: outputs responses to the control input edge

Difference between flip flop and latch

- A unique signal called "enable" is provided with latch.
- The output changes only when enable signal is active.
- No change in output take place when the enable signal is inactive.

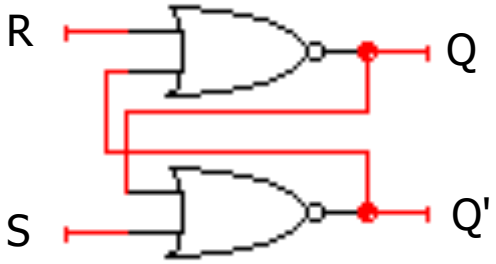
이 설명이 맞는 설명이다. rising edge , only flip flop when it it on the high level and then changes then that is about latches so they are different!!!

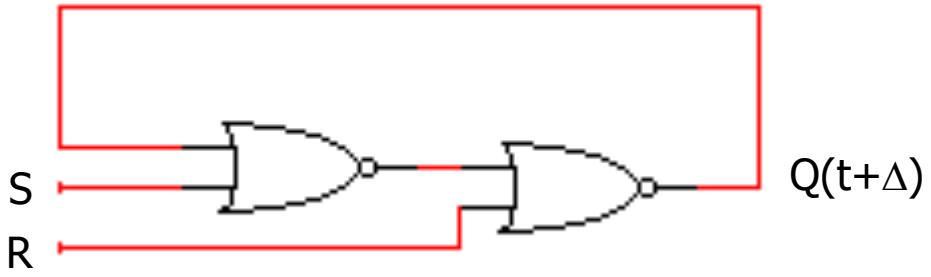
- Flip flop are edge trigger, while latches are level trigger.



RS Latch

RS Latch analysis

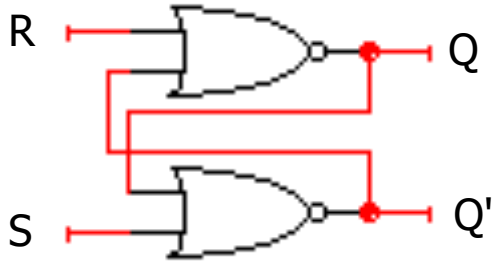


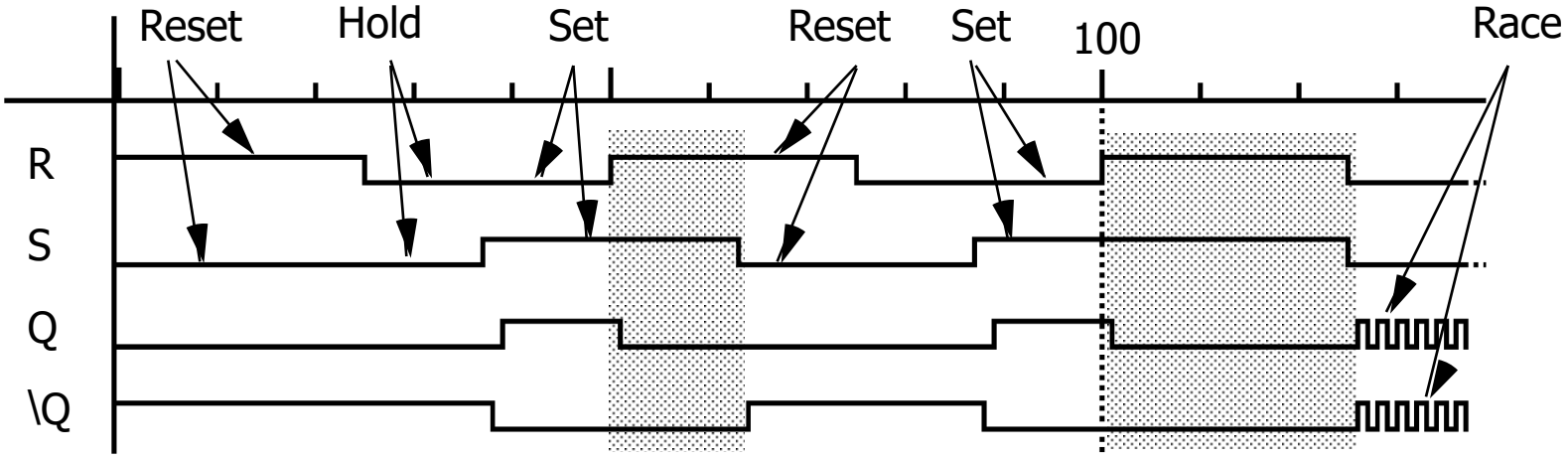


S	R	Q(t)	Q(t+Δ)	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	not allowed
1	1	1	X	

Q(t) \ S,R	0,0	0,1	1,1	1,0
0	0	0	X	1
1	1	0	X	1

Timing Behavior





Verilog Implementation (RS Latch)

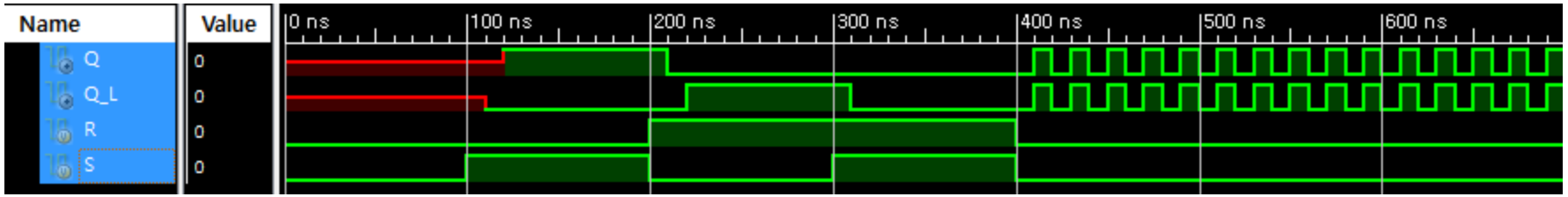
<Module>

```
1  `timescale 1ns / 1ps
2
3  module rslatch(
4      input R,
5      input S,
6      output Q,
7      output Q_L
8      );
9
10     assign #10 Q = ~(Q_L | R);
11     assign #10 Q_L = ~(Q | S);
12
13 endmodule
```

<Sample Testbench>

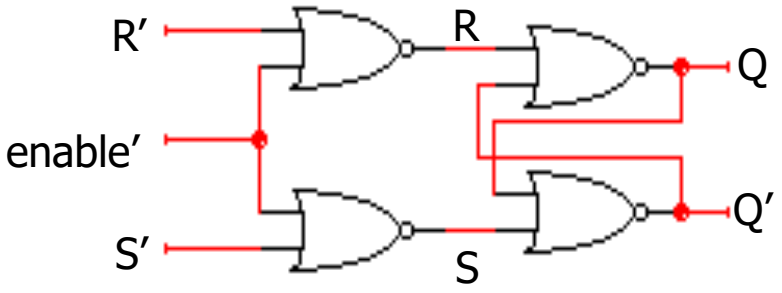
```
1  `timescale 1ns / 1ps
2
3  module rs_test;
4     reg R;
5     reg S;
6     wire Q;
7     wire Q_L;
8
9     rslatch uut (
10       .R(R),
11       .S(S),
12       .Q(Q),
13       .Q_L(Q_L)
14    );
15
16    initial begin
17       R = 0;
18       S = 0;
19       #100;
20
21       R = 0;
22       S = 1;
23       #100;
24
25       R = 1;
26       S = 0;
27       #100;
28
29       R = 1;
30       S = 1;
31       #100;
32
33       R = 0;
34       S = 0;
35    end
36
37 endmodule
```

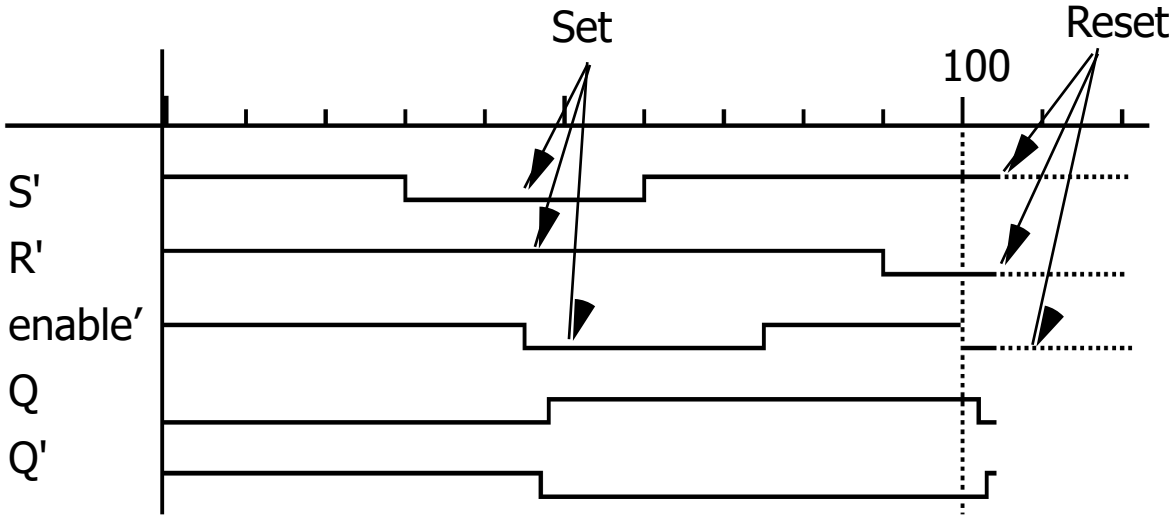
<Simulation>



Gated RS Latch

Gated RS Latch Analysis





Verilog Implementation (Gated RS Latch)

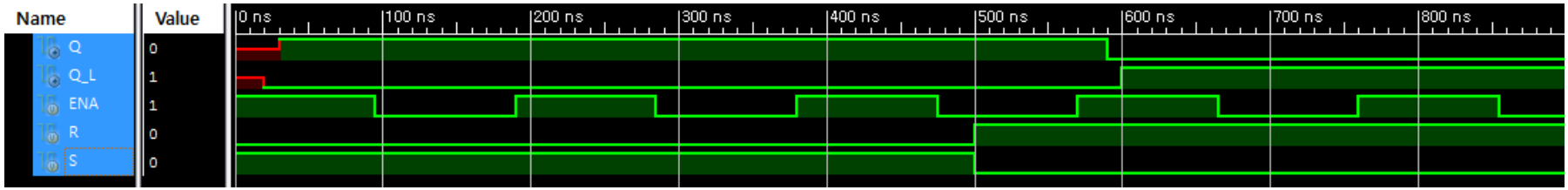
<Module>

```
1  `timescale 1ns / 1ps
2
3  module gatedrslatch(
4      input ENA,
5      input R,
6      input S,
7      output Q,
8      output Q_L
9      );
10
11     wire r_tmp, s_tmp;
12
13     assign #10 r_tmp = R & ENA;
14     assign #10 s_tmp = S & ENA;
15
16     rslatch RS_0(.R(r_tmp), .S(s_tmp), .Q(Q), .Q_L(Q_L));
17
18 endmodule
```

<Sample Testbench>

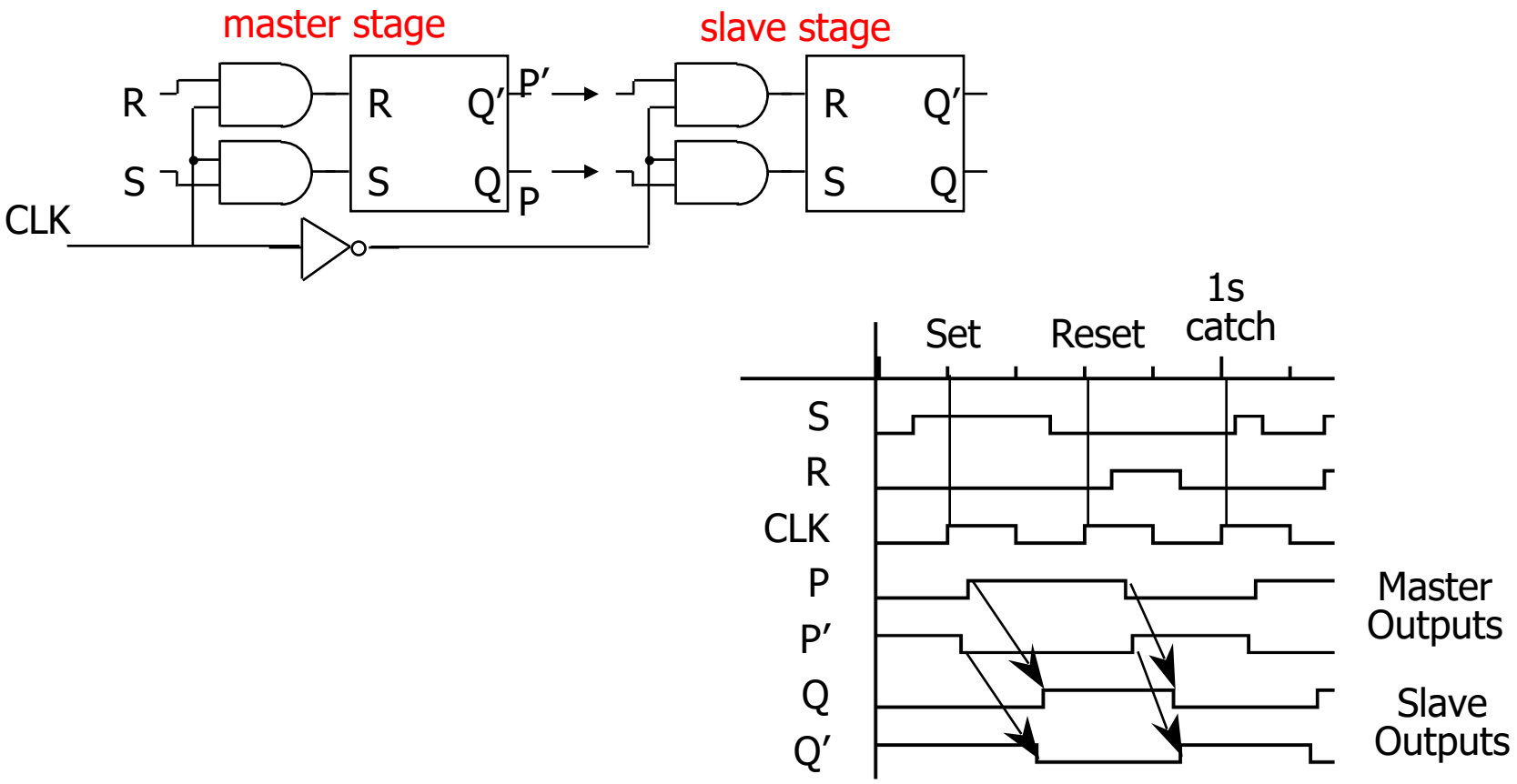
```
1  `timescale 1ns / 1ps
2  module grs_test;
3
4     reg ENA;
5     reg R;
6     reg S;
7
8     wire Q;
9     wire Q_L;
10
11    gatedrslatch uut (
12       .ENA(ENA),
13       .R(R),
14       .S(S),
15       .Q(Q),
16       .Q_L(Q_L)
17    );
18
19    initial begin
20       ENA = 1;
21       R = 0;
22       S = 1;
23       #500;
24       R = 1;
25       S = 0;
26       #500;
27       R = 0;
28       S = 0;
29       #500;
30       R = 1;
31       S = 1;
32    end
33
34    always
35       #95 ENA = ~ENA;
36
37 endmodule
```

<Simulation>

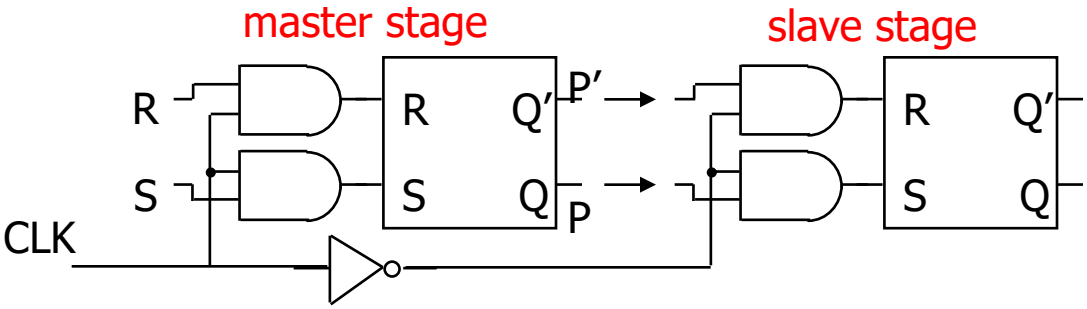


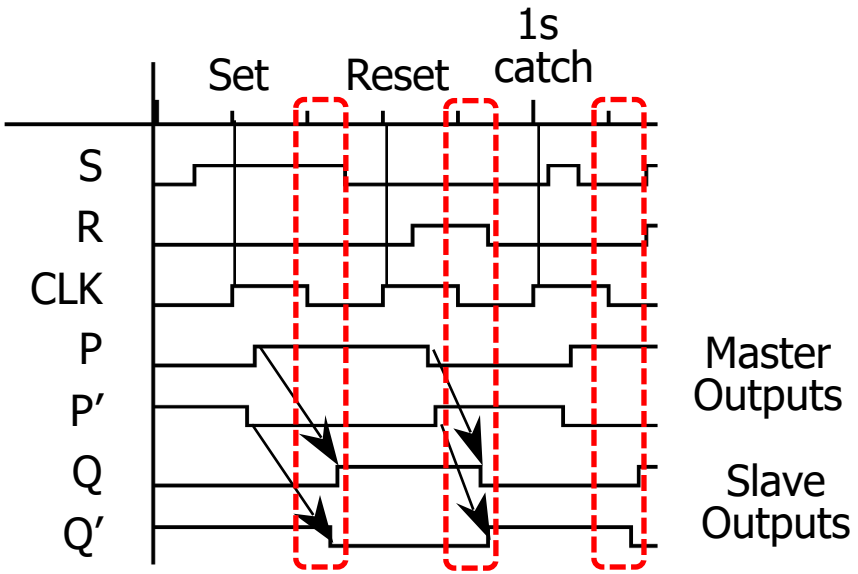
Master-Slave Latches

Master-Slave structure & The 1s catching Problem



Master-Slave structure & The 1s catching Problem





Verilog Implementation (Master-Slave Latches)

<Module>

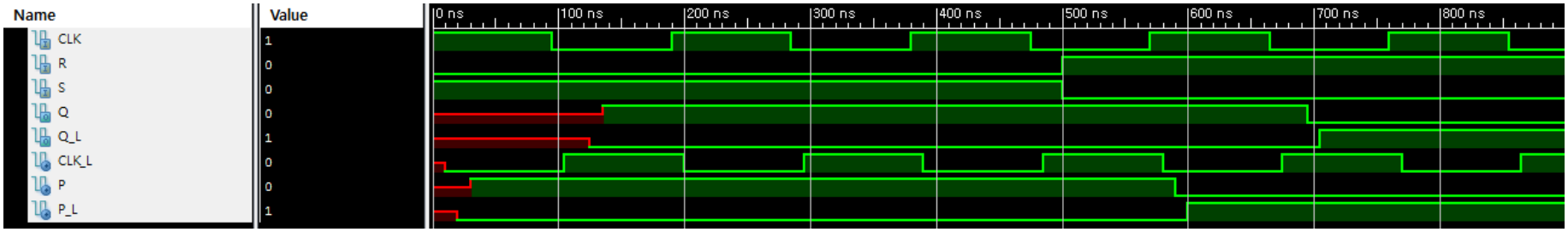
```
1  `timescale 1ns / 1ps
2
3  module msflipflop(
4      input CLK,
5      input R,
6      input S,
7      output Q,
8      output Q_L
9      );
10
11
12     wire CLK_L;
13     assign #10 CLK_L = ~CLK;
14
15     wire P, P_L;
16     gatedrslatch GRS_0(.ENA(CLK), .R(R), .S(S), .Q(P), .Q_L(P_L));
17     gatedrslatch GRS_1(.ENA(CLK_L), .R(P_L), .S(P), .Q(Q), .Q_L(Q_L));
18
19 endmodule
```

<Sample Testbench>

```
1  `timescale 1ns / 1ps
2
3  module msff_test;
4
5      reg CLK;
6      reg R;
7      reg S;
8      wire Q;
9      wire Q_L;
10
11     msflipflop uut (
12         .CLK(CLK),
13         .R(R),
14         .S(S),
15         .Q(Q),
16         .Q_L(Q_L)
17     );
18
```

```
19     initial begin
20         CLK = 1;
21         R = 0;
22         S = 1;
23         #500;
24         R = 1;
25         S = 0;
26         #500;
27         R = 0;
28         S = 0;
29         #500;
30         R = 1;
31         S = 1;
32     end
33
34     always
35         #95 CLK = ~CLK;
36
37 endmodule
```

<Simulation>



Lab

Lab & HW

1. Implement in Verilog, (1) an oscillator, (2) an RS latch, (3) a gated RS latch, and (4) master-slave latches. Simulate their behavior.

2. For master-slave latches, make a Verilog Testbench that clearly shows the 1s catching problem.

(HW0) Discuss what is the 1s catching problem and why it occurs.

→ Hint. Master-slave latches introduced on this lab operate as **?**

→ In this point of view, discuss what is the 1s catching problem, why it occurs and why it is called as 'problem'.

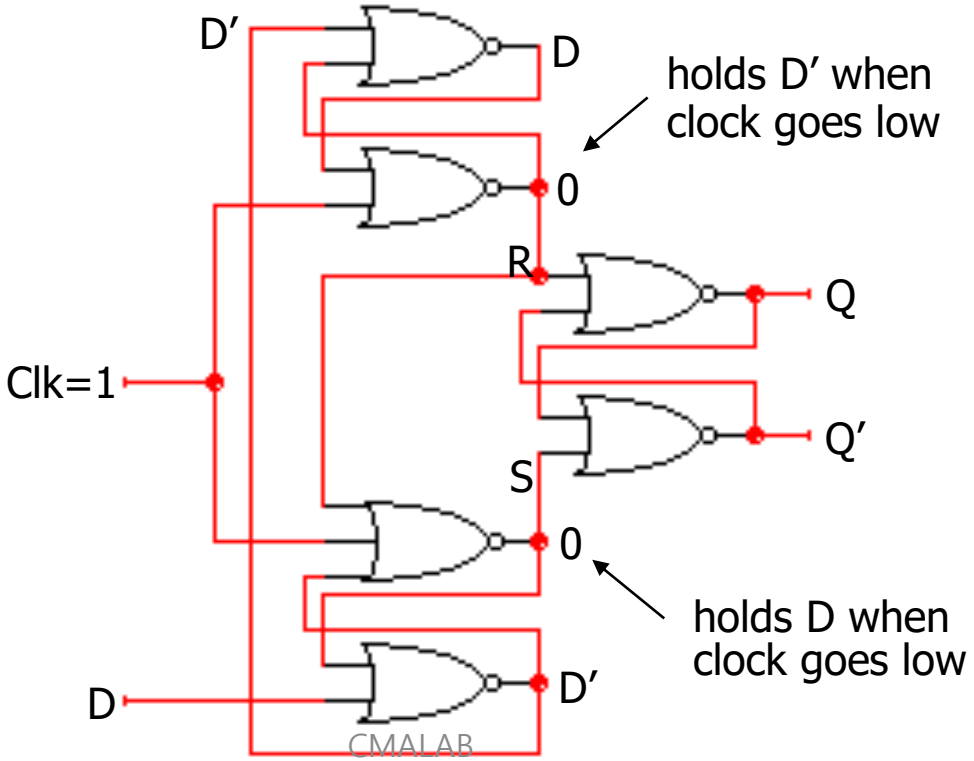
initial block으로 초기화하면 안된다. 내부적으로
그리고 behavior로 짜는 것이 좋다 ㅎㅅㅎ

Homework

1.1. Implement the (negative) edge-triggered D flip-flop in Verilog
- Gate level(diagram below) or RTL or Behavioral

1.2. Simulate its behavior

1.3. Discuss the similarities and differences between D F/F and Master-Slave latches



Homework

2.1. Implement the RS F/F in Verilog
  - Gate level or RTL or Behavioral

2.2. Simulate its behavior

2.3. Discuss the similarities and differences between SR F/F and D F/F

Homework

3.1. Implement the JK F/F in Verilog, using SR F/F from HW 2

3.2. Simulate its behavior

3.3. Discuss the similarities and differences between SR F/F and JK F/F

3.1. Might be tricky!

Hint. You can use external inputs to initialize the states

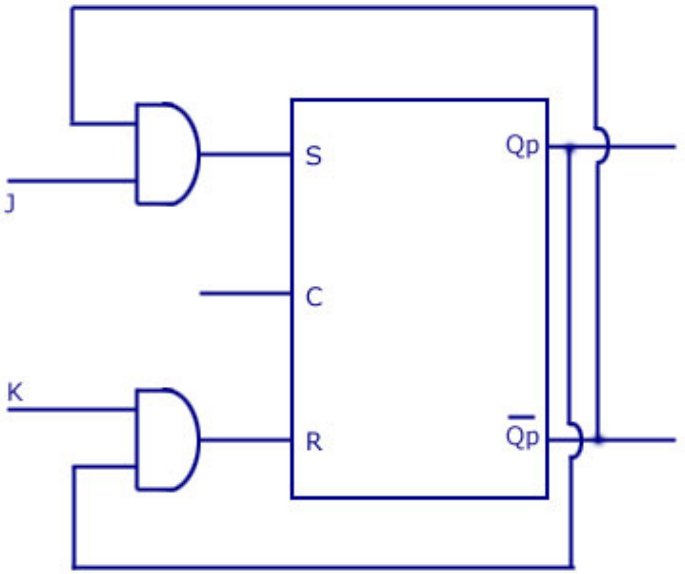
Do not forget HW 0 on slide 23 !

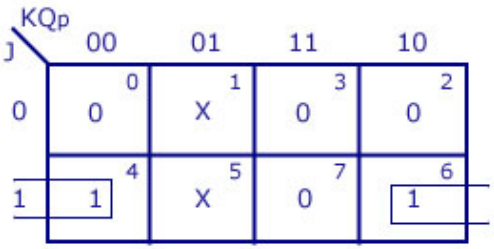
S-R Flip Flop to J-K Flip Flop

Conversion Table

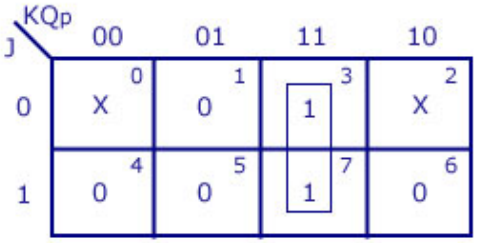
J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram





$S = \overline{J}Qp$



$R = KQp$

K-Map

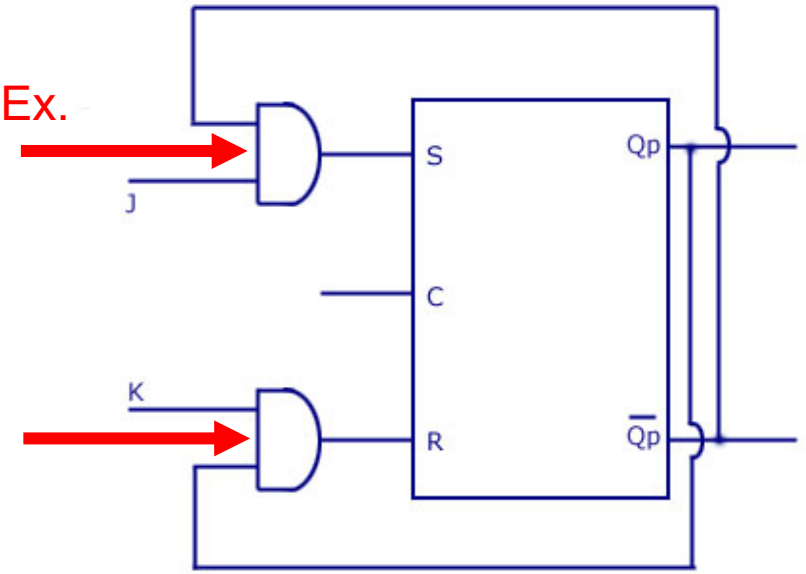
www.CircuitsToday.com

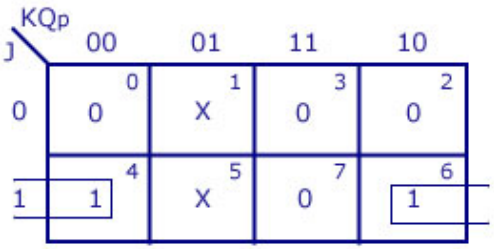
S-R Flip Flop to J-K Flip Flop

Conversion Table

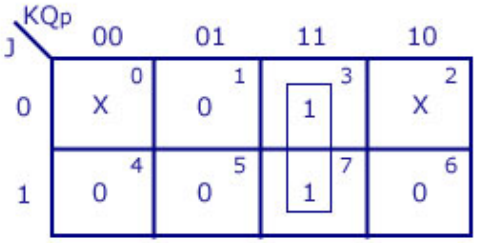
J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram





$S = \overline{J}Q_p$



$R = KQ_p$

K-Map

www.CircuitsToday.com

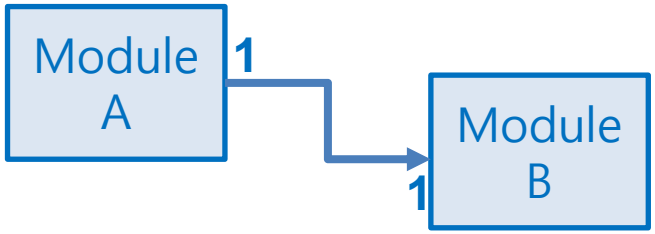
Report

- Write a report
  - # of pages doesn't matter
  - All the files should be compressed to ZIP format
  - **Your codes and simulations for every experiment should be included**
  - The file size should be less than 15MB
  - **Due : 28 Oct. (Before class begin at 7:00pm)**

Appendix

wire & reg

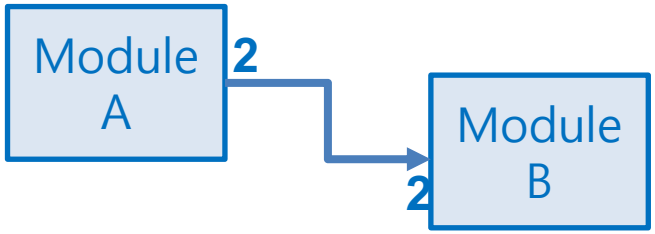
- Wire
  - Physical connections between modules
  - Propagate data **at all time**
  - Values are **always** updated **as soon as** the output of source component changes



- Reg
  - Memory component
  - Propagate data **at all time**
  - **Values are updated on specific condition**
  - **Values should be updated on specific condition**

wire & reg

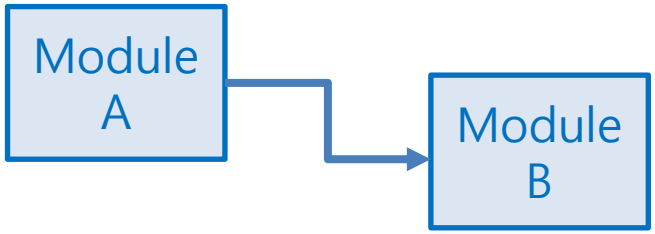
- Wire
  - Physical connections between modules
  - Propagate data **at all time**
  - Values are **always** updated **as soon as** the output of source component changes



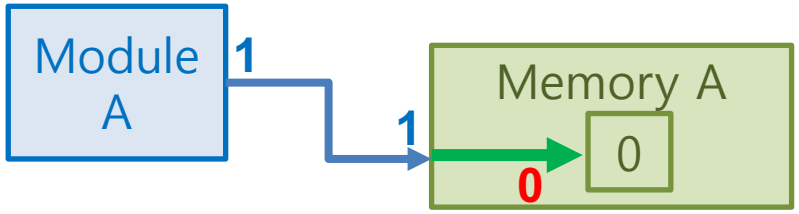
- Reg
  - Memory component
  - Propagate data **at all time**
  - **Values are updated on specific condition**
  - **Values should be updated on specific condition**

wire & reg

- Wire
  - Physical connections between modules
  - Propagate data **at all time**
  - Values are **always** updated **as soon as** the output of source component changes

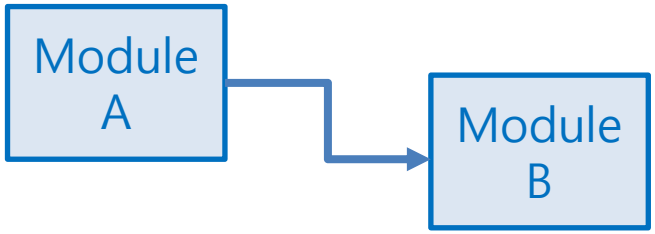


- Reg
  - Memory component
  - Propagate data **at all time**
  - **Values are updated on specific condition**
  - **Values should be updated on specific condition**

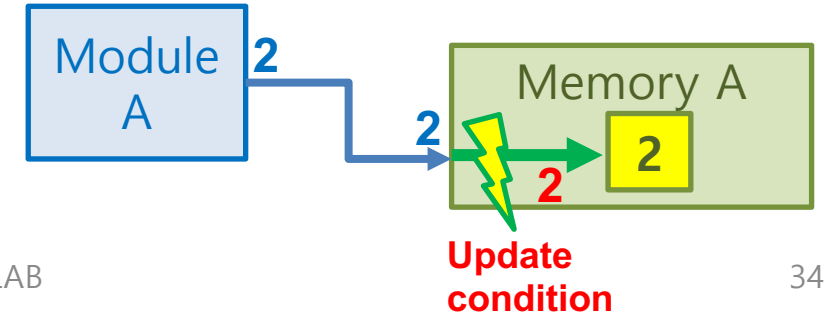


wire & reg

- Wire
  - Physical connections between modules
  - Propagate data **at all time**
  - Values are **always** updated **as soon as** the output of source component changes

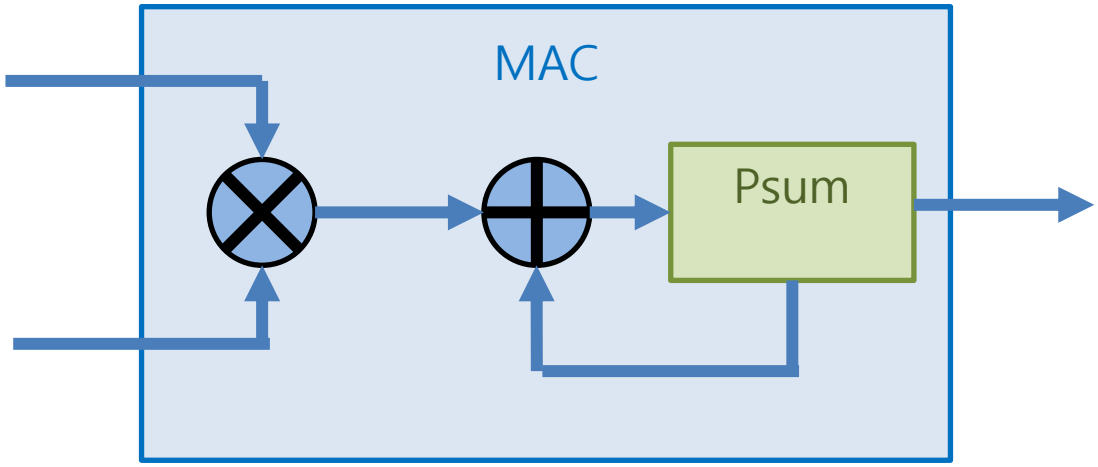


- Reg
  - Memory component
  - Propagate data **at all time**
  - **Values are updated on specific condition**
  - **Values should be updated on specific condition**



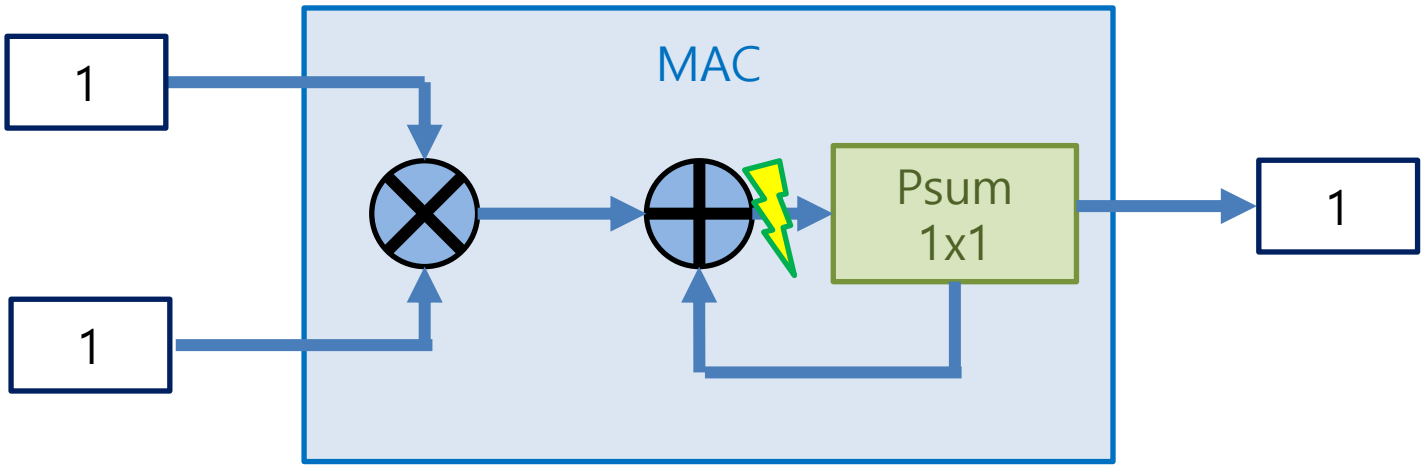
wire & reg

- Example: MAC (Multiplier-accumulator)
  - Operation: calculate “1x1 + 2x2 + 3x3 + …”
  - Use one memory component for partial sum (temporal output)



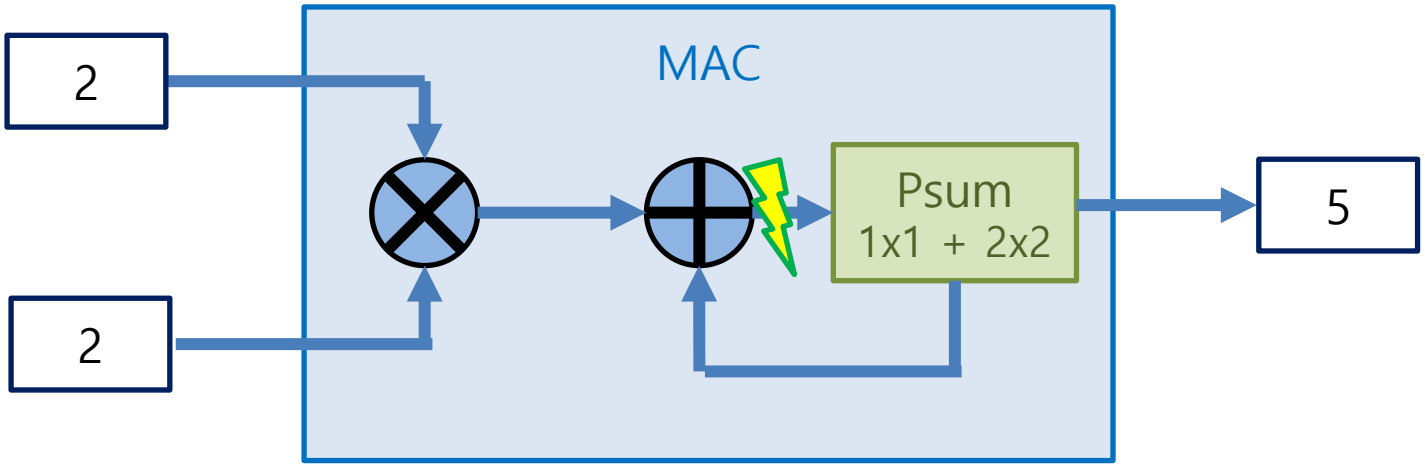
wire & reg

- Example: MAC (Multiplier-accumulator)
  - Step 1
    - Input: 1
    - Output: 1x1



wire & reg

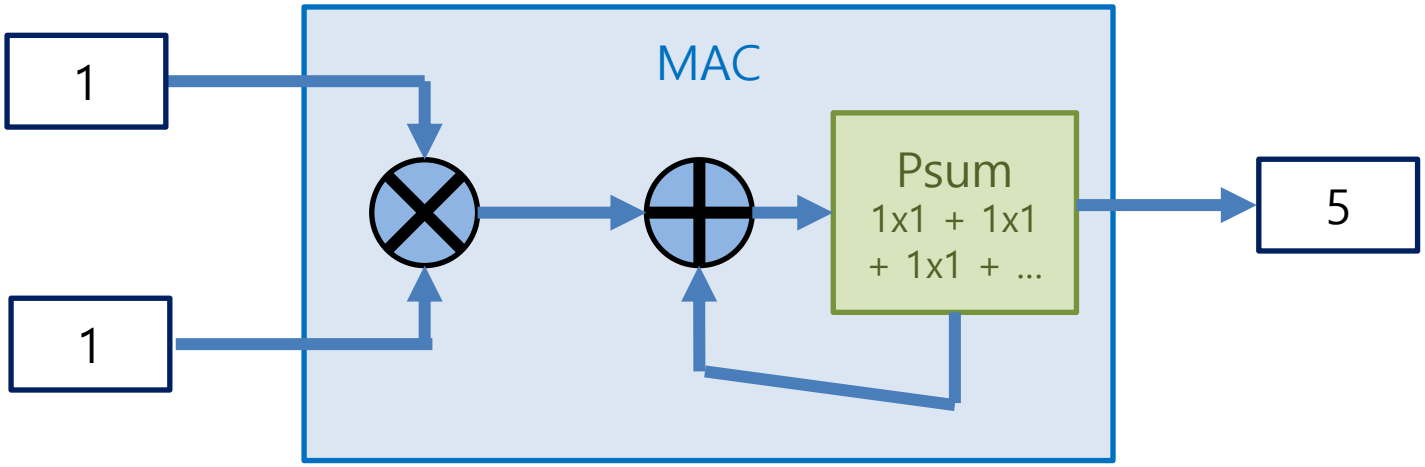
- Example: MAC (Multiplier-accumulator)
  - Step 2
    - Input: 2
    - Output: 1x1 + 2x2



- …And so on!

`wire` & `reg`

- Example: MAC (Multiplier-accumulator)
  - We have to implement Psum as `reg`
  - Or, Psum value will be explode!
  - Because, wire updates its value as soon as the source changes



wire & reg

- Example: MAC (Multiplier-accumulator)
  - That's why we use `always` block

```
always@(update_condition) begin
    Psum <= Psum + A * B;    //update Psum only update_condition met
    …
end
```