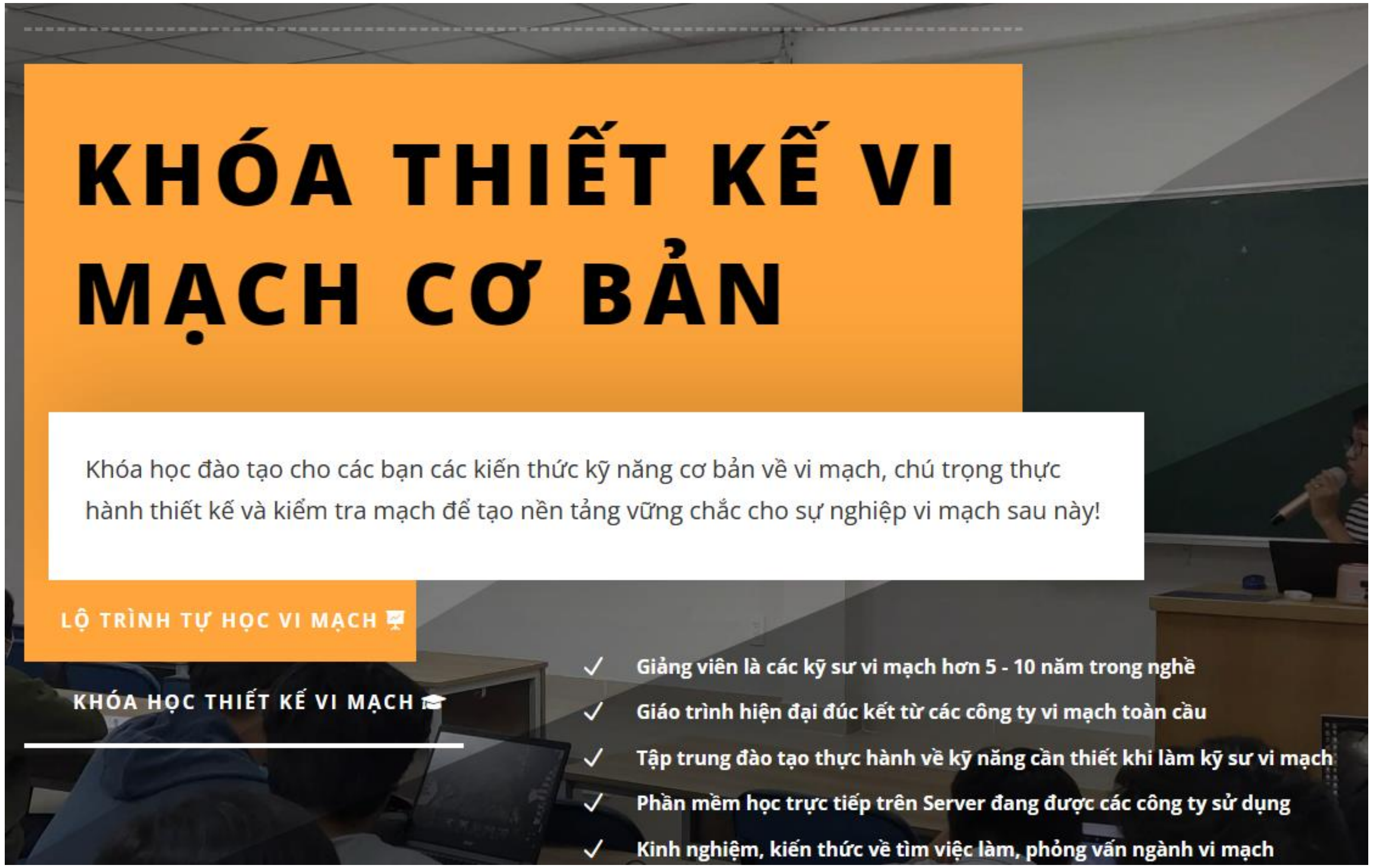
IC OVERVIEW
RTL DESIGN AND VERIFICATION

COURSE INTRODUCTION

[Khóa Học Thiết Kế Vi Mạch Cơ Bản - Trung Tâm Đào Tạo Thiết Kế Vi Mạch ICTC]



KHÓA THIẾT KẾ VI MẠCH CƠ BẢN

Khóa học đào tạo cho các bạn các kiến thức kỹ năng cơ bản về vi mạch, chú trọng thực hành thiết kế và kiểm tra mạch để tạo nền tảng vững chắc cho sự nghiệp vi mạch sau này!

LỘ TRÌNH TỰ HỌC VI MẠCH

KHÓA HỌC THIẾT KẾ VI MẠCH

- ✓ Giảng viên là các kỹ sư vi mạch hơn 5 - 10 năm trong nghề
- ✓ Giáo trình hiện đại đúc kết từ các công ty vi mạch toàn cầu
- ✓ Tập trung đào tạo thực hành về kỹ năng cần thiết khi làm kỹ sư vi mạch
- ✓ Phần mềm học trực tiếp trên Server đang được các công ty sử dụng
- ✓ Kinh nghiệm, kiến thức về tìm việc làm, phỏng vấn ngành vi mạch

COURSE INTRODUCTION

SUMMARY

HOMEWORK

QUESTION

SELF-LEARNING

Session 9: Verilog Fundamental – Part 4 Sequential Logic Design

1. Sequential logic design
2. Non-blocking assignment
3. Latch
4. Flip-flop

SEQUENTIAL LOGIC DESIGN

- A RTL design is incomplete without any register logic, need an element to store value, so the sequential logic holds the binary data.
- Sequential logic elements are latches and flip-flops and are used to design the sequential circuits for the given design functionality
- Sequential logic are used to non-blocking assignment (<=).

NON-BLOCKING ASSIGNMENT

- **Non-blocking** assignment are primarily used in sequential logic generation.
- LHS of non-blocking assignment must be reg type.
- All assignments in RHS run in parallel. Than means all variables within the same clock cycle are updated simultaneously.
- **Syntax**: y <= expression;
- **Example**: (1) b <= a + 1;

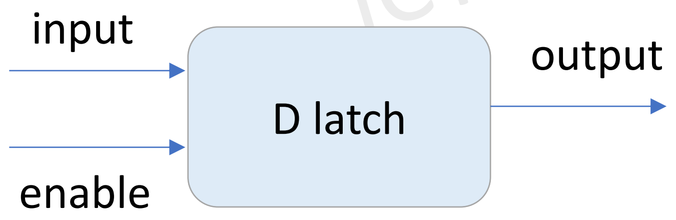
  (2) c <= b + 2;

  Assume that a = 1, b = 1 → (1) and (2) are executed in parallel, then b = 2, c = 3

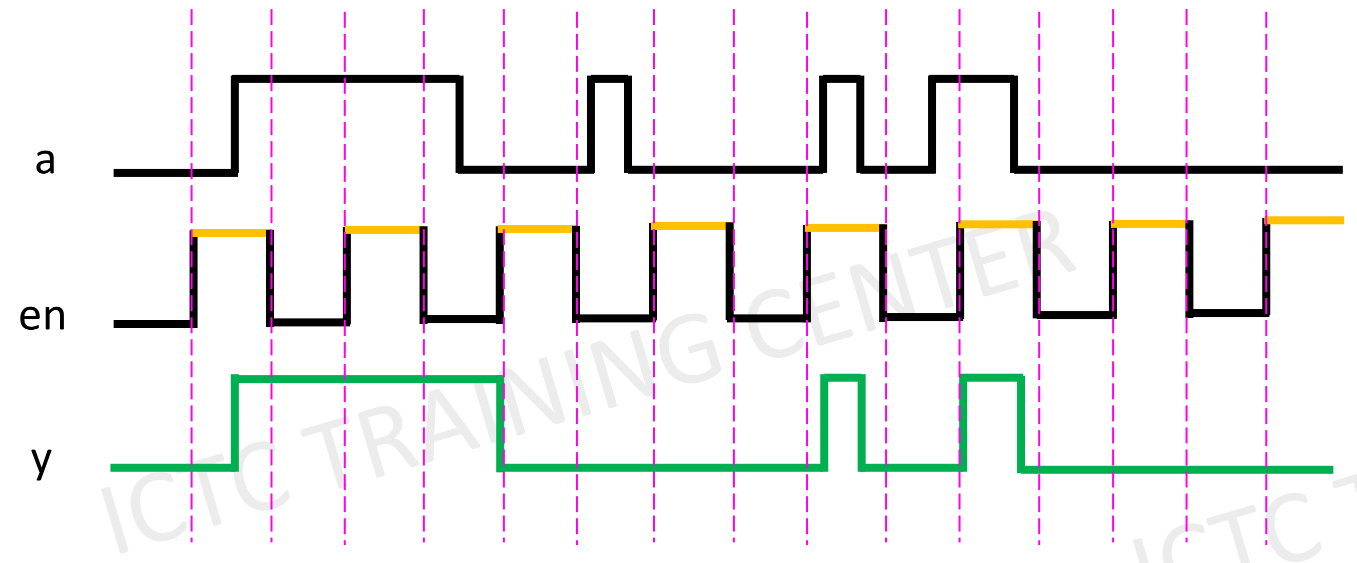
Always make sure that bit-width of LHS and RHS are equal to avoid unexpected issues

LATCH

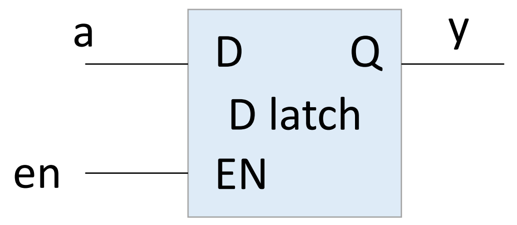
- A latch is a type of sequential logic circuit used to store information.
- There are many kinds of latch: SR Latch, JK Latch, D Latch, T Latch. We will focus on D-latch only in this course.
- D-latch has a single data input D and an enable input “enable”.
  - When EN is active, output Q follows the input D
  - When EN is inactive, the output retains its last update



LATCH



```
//code to generate a latch
always @ (a or en)
begin
   if( en == 1'b1) begin
      y = a;
   end
end
```



LATCH

- Avoid using latch in synchronous design as much as possible because the data changed asynchronously with clock.
- Take care of all branch conditions in if-else or case statements to avoid creating un-intentional latches.

```
//4:2 encoder
always @ (In_A) begin
   case(In_A)
      4'b0001 : Out_Y = 2'b00;
      4'b0010 : Out_Y = 2'b01;
      4'b0100 : Out_Y = 2'b10;
      4'b1000 : Out_Y = 2'b11;
endcase
end
```

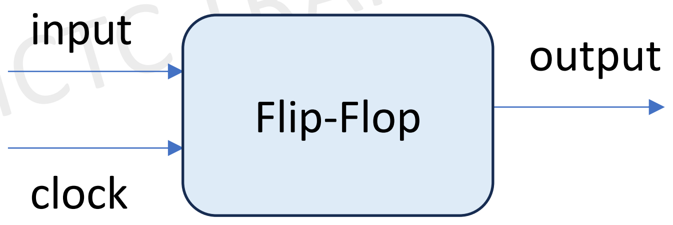
✗ This code will generate latches

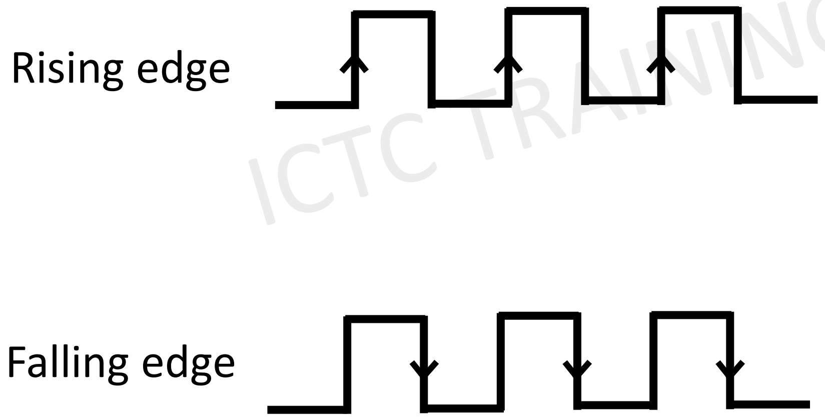
✓

```
//4:2 encoder
always @ (In_A) begin
   case(In_A)
      4'b0001 : Out_Y = 2'b00;
      4'b0010 : Out_Y = 2'b01;
      4'b0100 : Out_Y = 2'b10;
      4'b1000 : Out_Y = 2'b11;
      default  : Out_Y = 2'b00;
   endcase
end
```

FLIP-FLOP

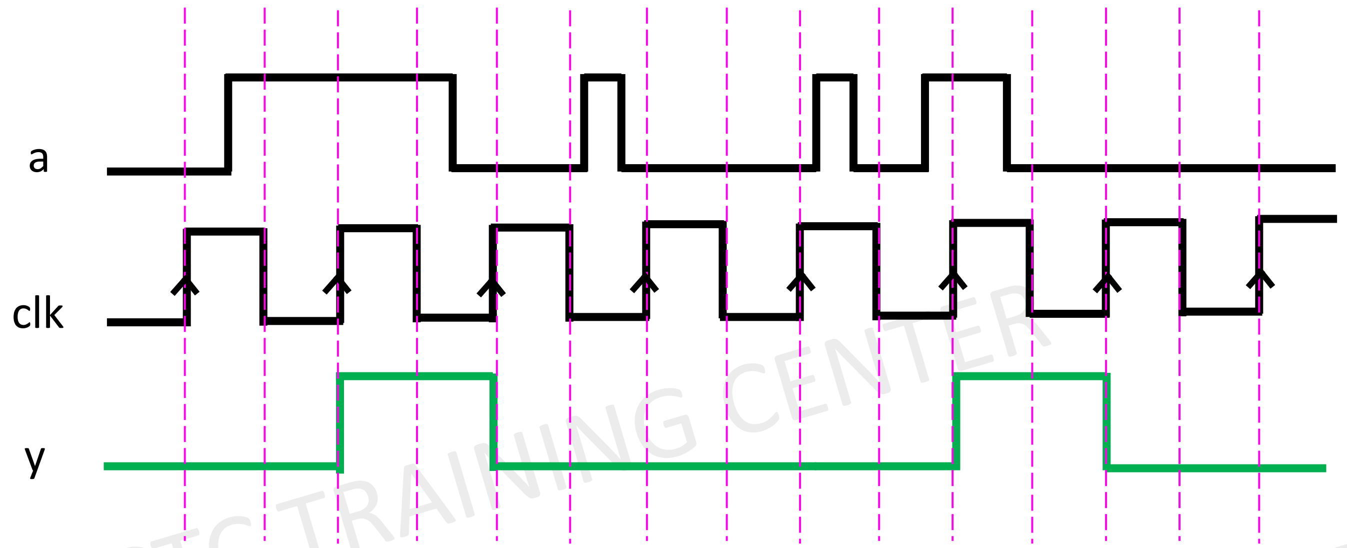
- A flip-flop is a type of sequential logic circuit used to store information.
- A flip-flop responds to the input level change at clock edge.
- There are 2 types of flip-flop:
  - Positive edge triggered
  - Negative edge triggered



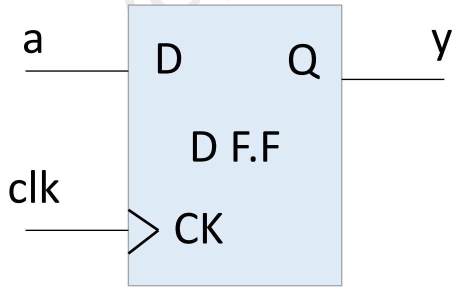


FLIP-FLOP

rising edge triggered

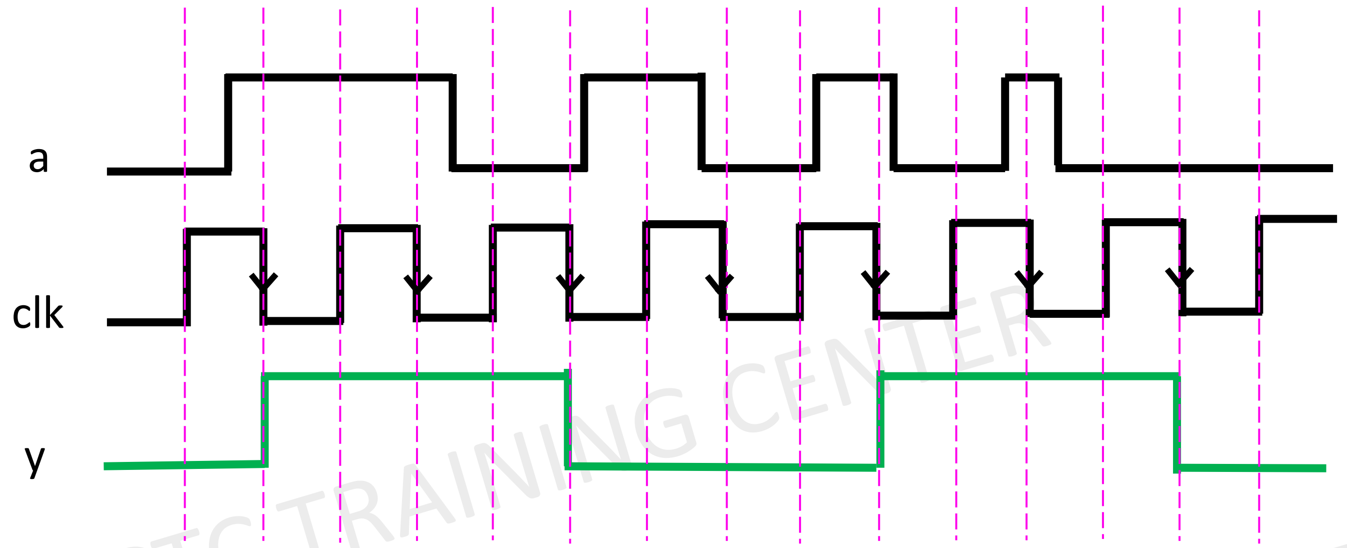


```
//code to generate a rising edge F.F
always @ (posedge clk)
begin
   y <= a; //non-blocking assignment
end
```

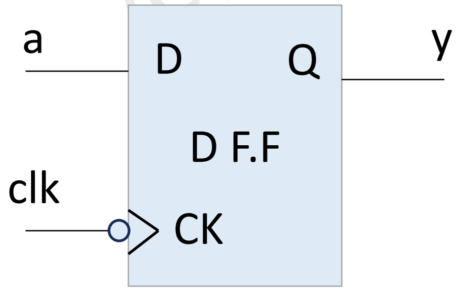


FLIP-FLOP

falling edge triggered

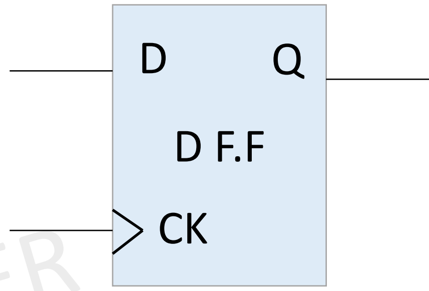


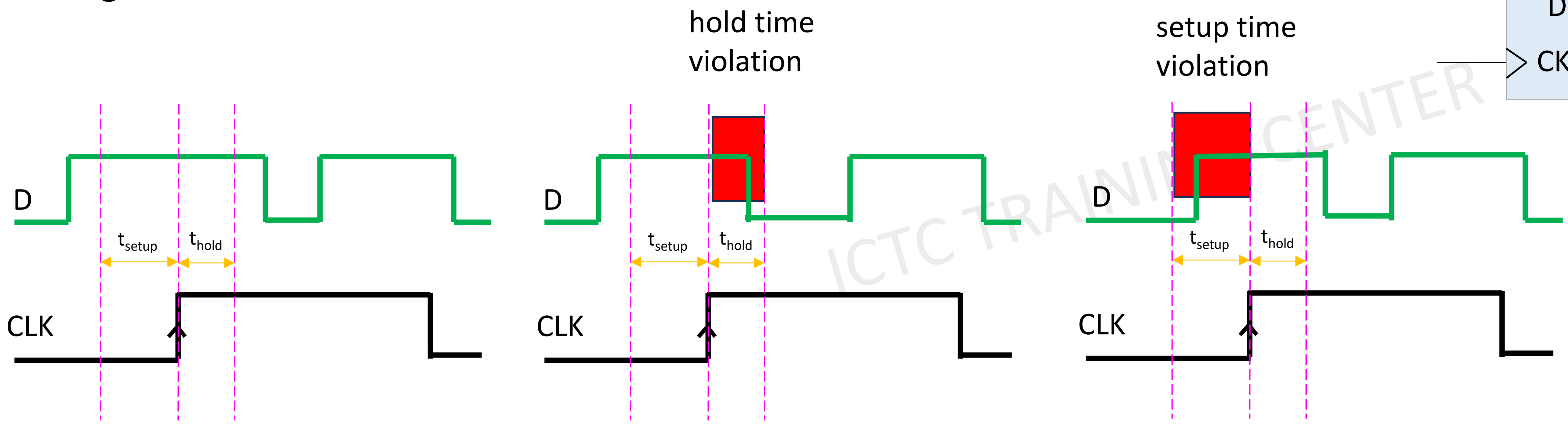
```
//code to generate a falling edge F.F
always @ (negedge clk)
begin
   y <= a; //non-blocking assignment
end
```



FLIP-FLOP SETUP & HOLD TIME

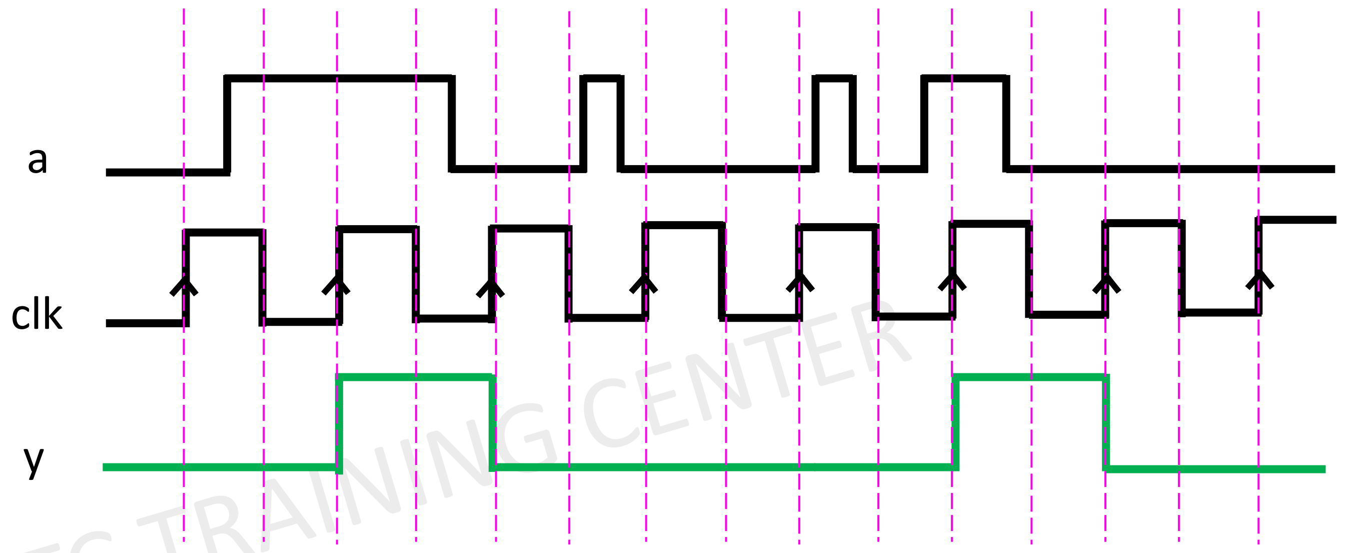
- Setup time: the time duration that the input should remain stable before the arrival of the clock edge.
- Hold time: the time duration that the input should remain stable after the clock edge.

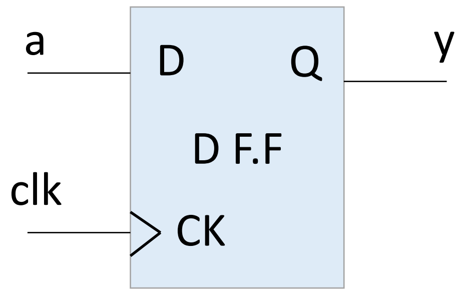




We may not get the correct output of Flip-flop (can not sample data correctly) when violation occurs !!!

FLIP-FLOP PROPAGATION DELAY

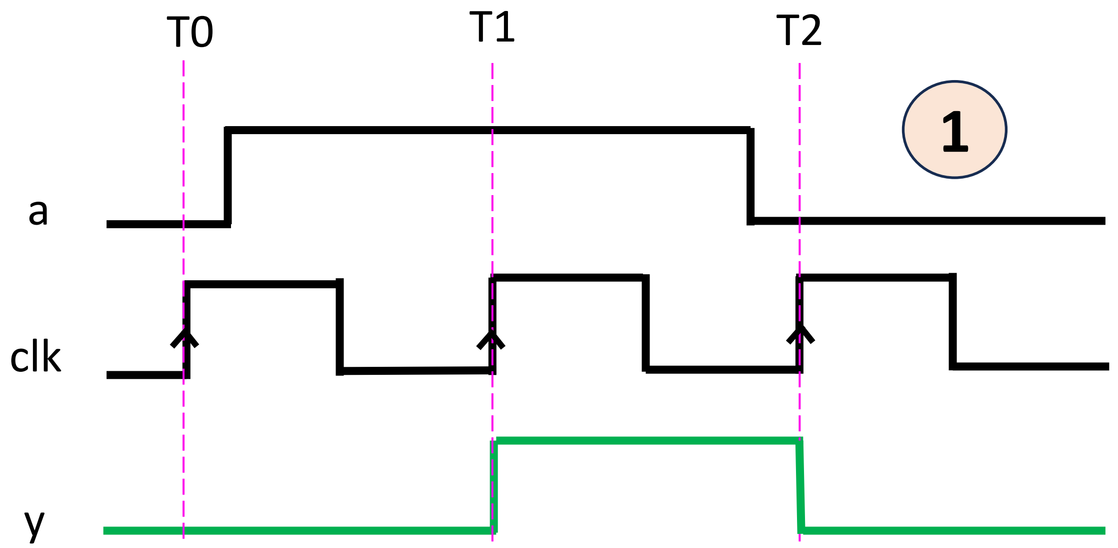


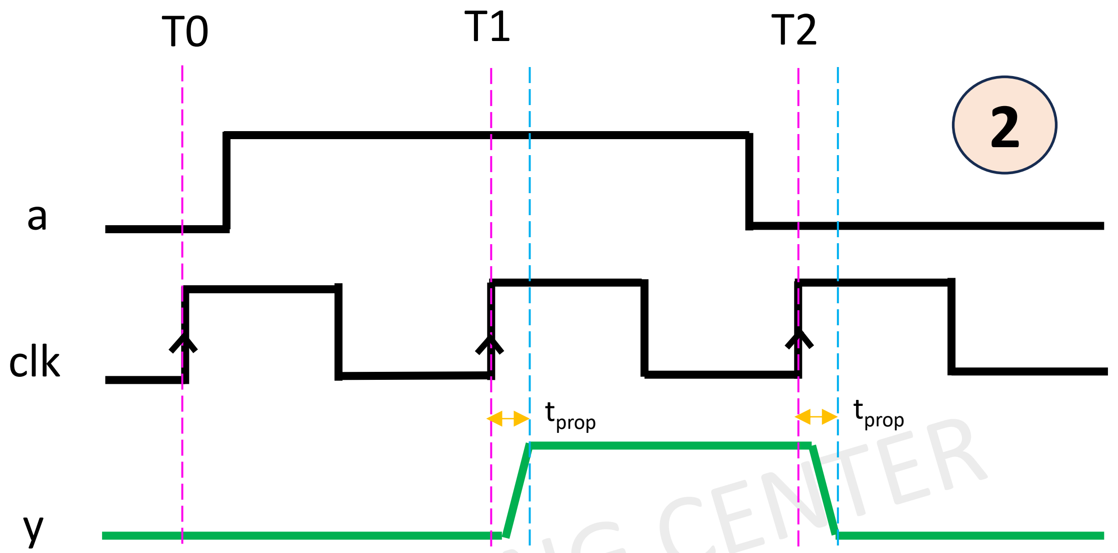


So far, we assumed that at clock edge, the F.F samples the input and changes its value immediately. But actually, the FF requires a certain time to response to the input change. Therefore, the change in the output of FF will appear after a certain delay.

FLIP-FLOP PROPAGATION DELAY

But actually, the FF requires a certain time to response to the input change.
Therefore, the change in the output of FF will appear after a certain delay.





- What you see on (1) is actually (2).
- (1) is just an ideal case where propagation time is 0, or it's too small to see on the waveform.
- You will see (1) on any EDA's waveform, but you need to think that it is (2).

sig	T0	T1	T2
a	0	1	0
y	0	0	1

FLIP-FLOP PROPAGATION DELAY

You will see some waveform in document like below, where signals do not change on clock edges due to propagation delay

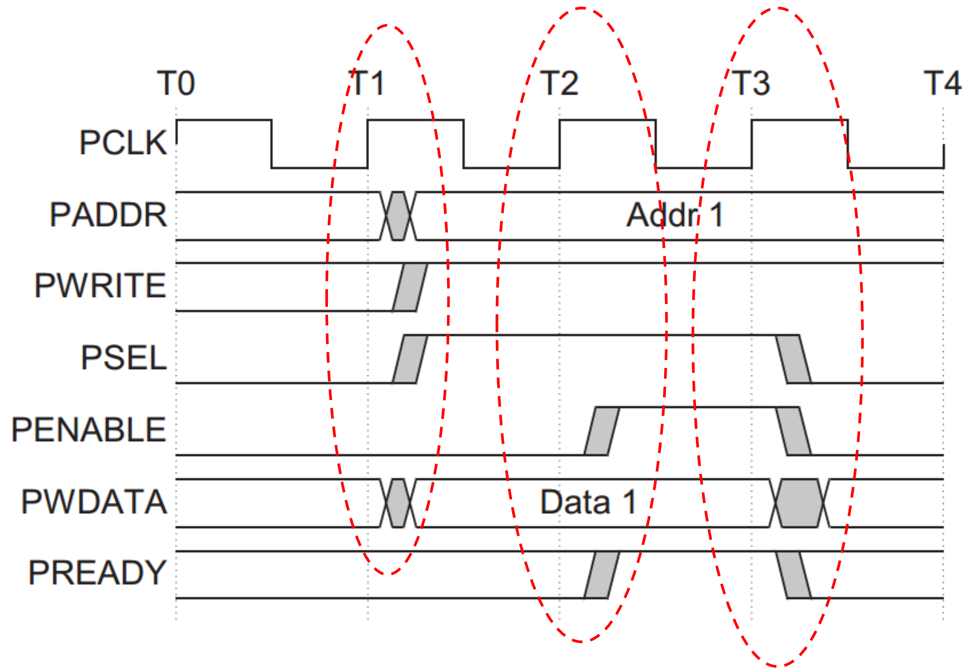
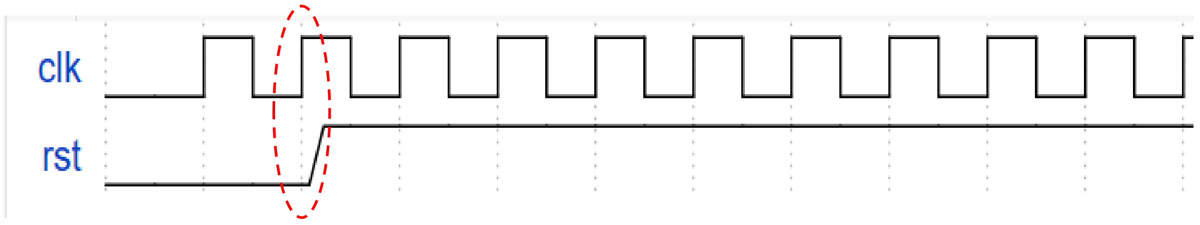
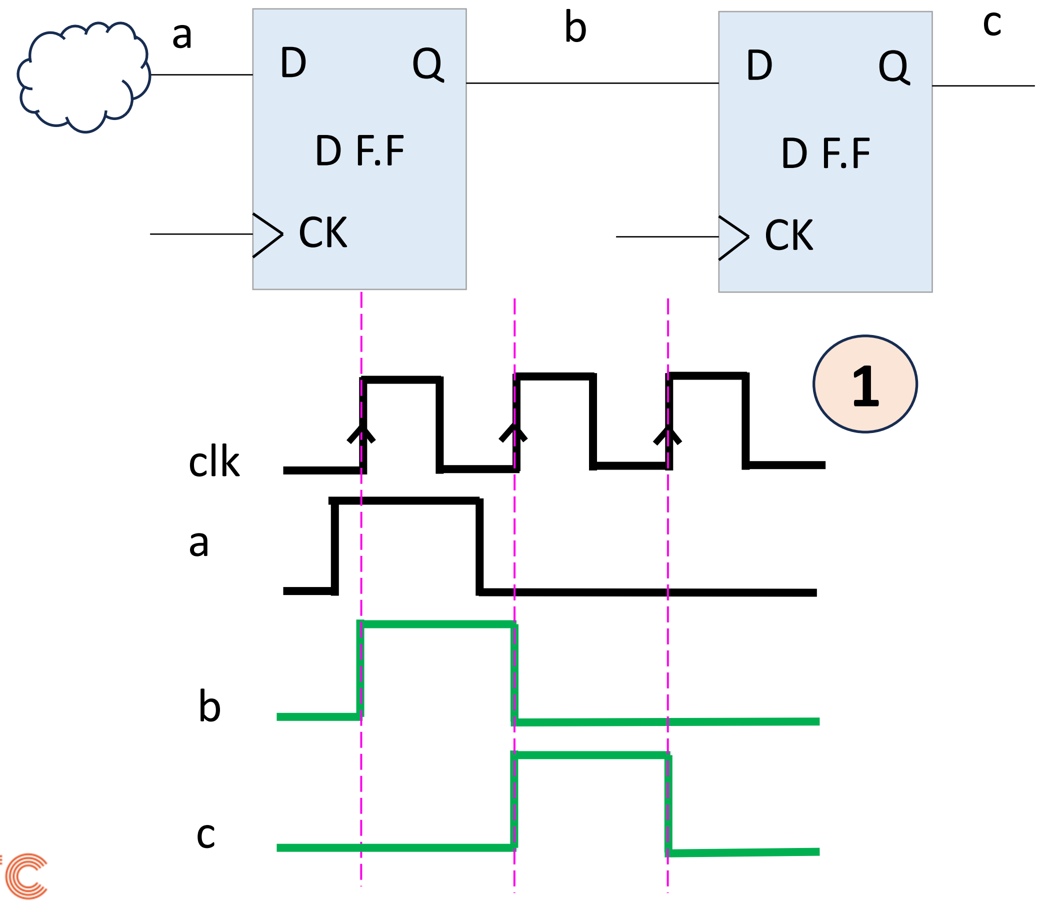


Figure 2-1 Write transfer with no wait states

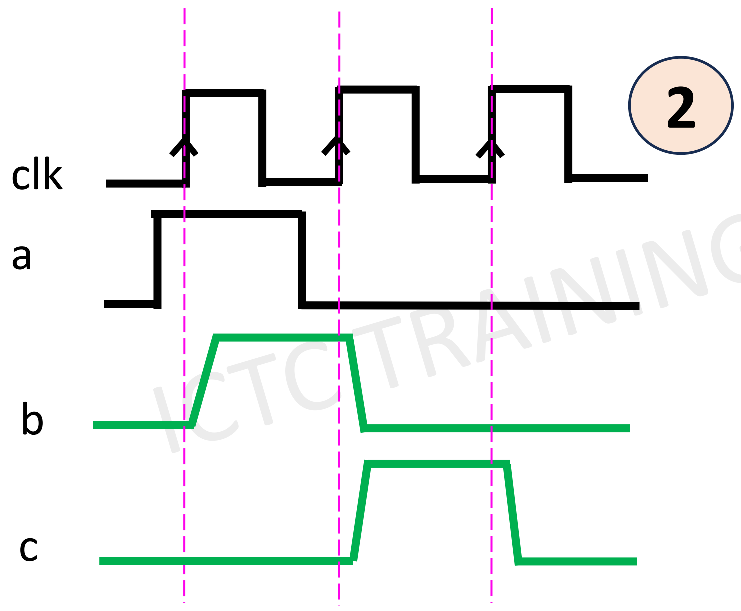


FLIP-FLOP PROPAGATION DELAY

Example: for the below circuit, you will see (1) in the simulation waveform, but actually it is (2). That means, if you read the value of b at the first clock edge, the returned value is 0. If you read b on the second clock edge, b is 1.



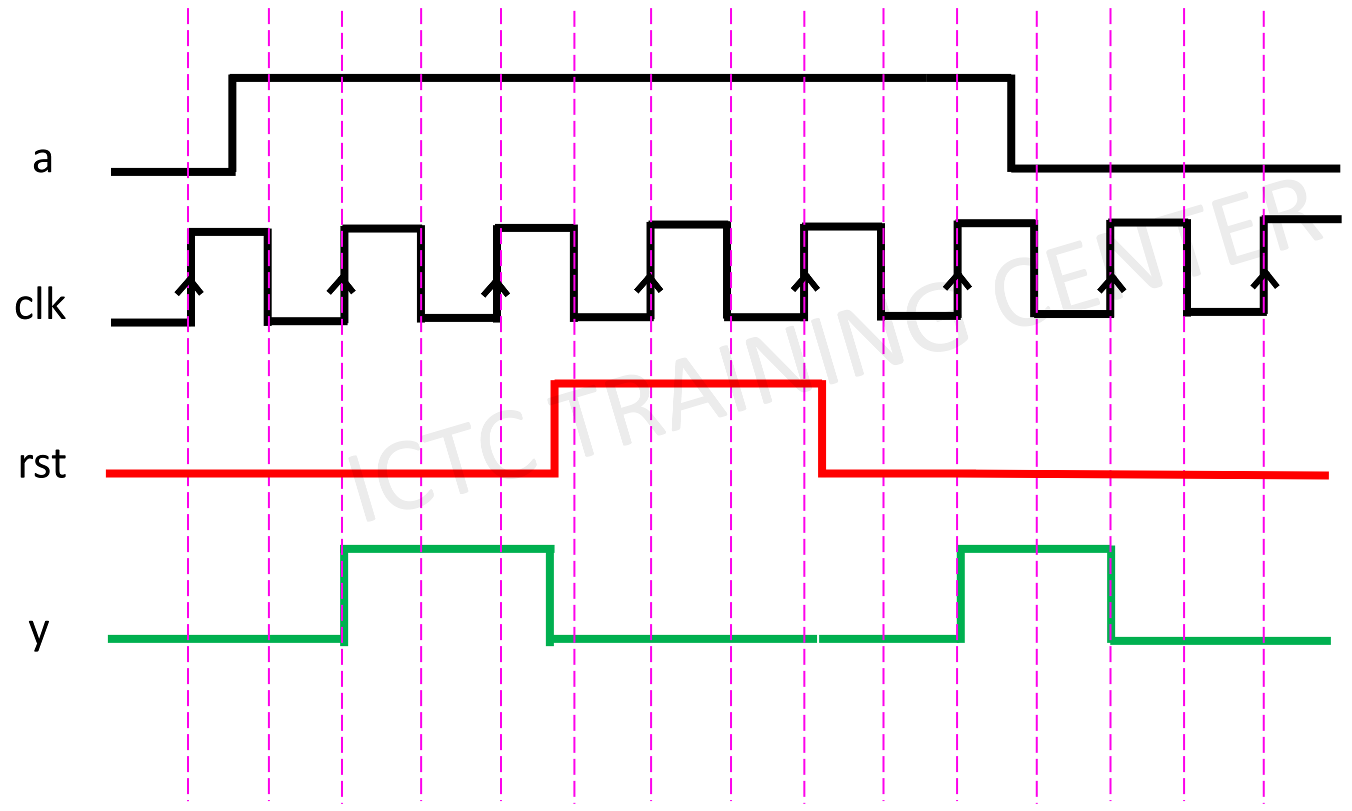
Let confirm this in the practice session later !!!



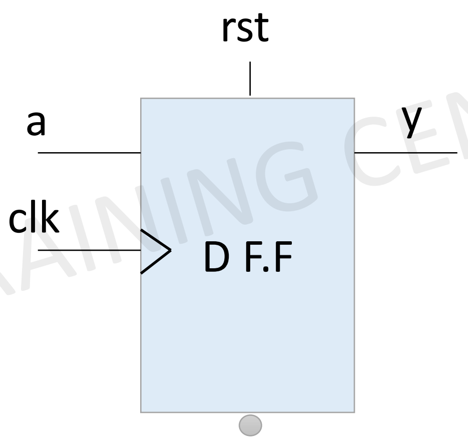
ASYNCHRONOUS SET/RESET

The output of F.F can be set to 1 or reset to 0 asynchronously to clock signal.

Below is the active high asynchronous reset.



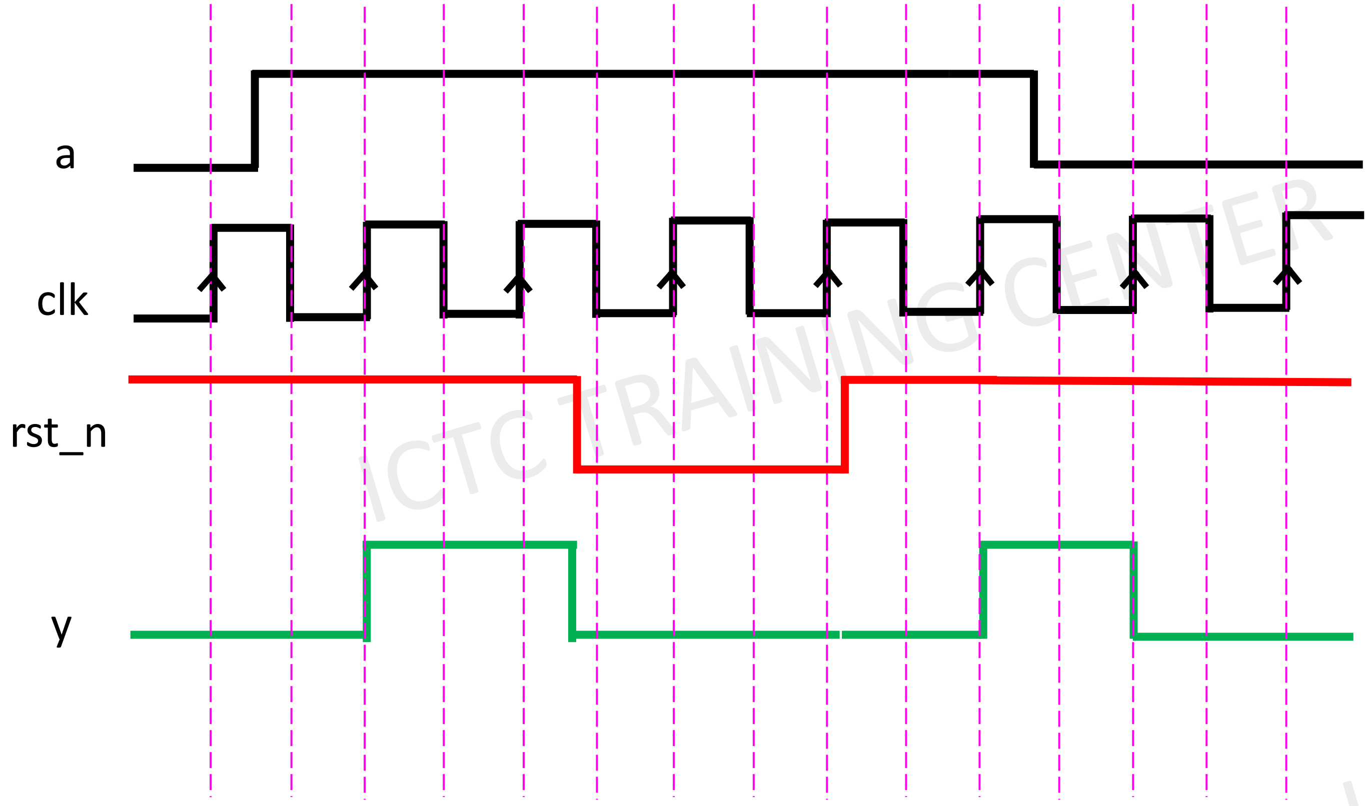
```
//code to generate a rising edge F.F with
positive edge asyn. reset
always @ (posedge clk or posedge rst)
begin
   if( rst == 1'b1)
      y <= 1'b0;
   else
      y <= a;
end
```



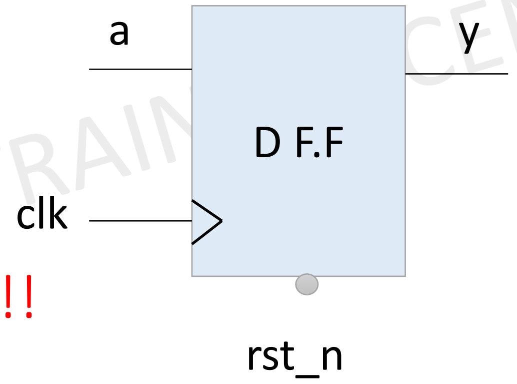
ASYNCHRONOUS SET/RESET

The output of F.F can be set to 1 or reset to 0 asynchronously to clock signal.

Below is the active low asynchronous reset.



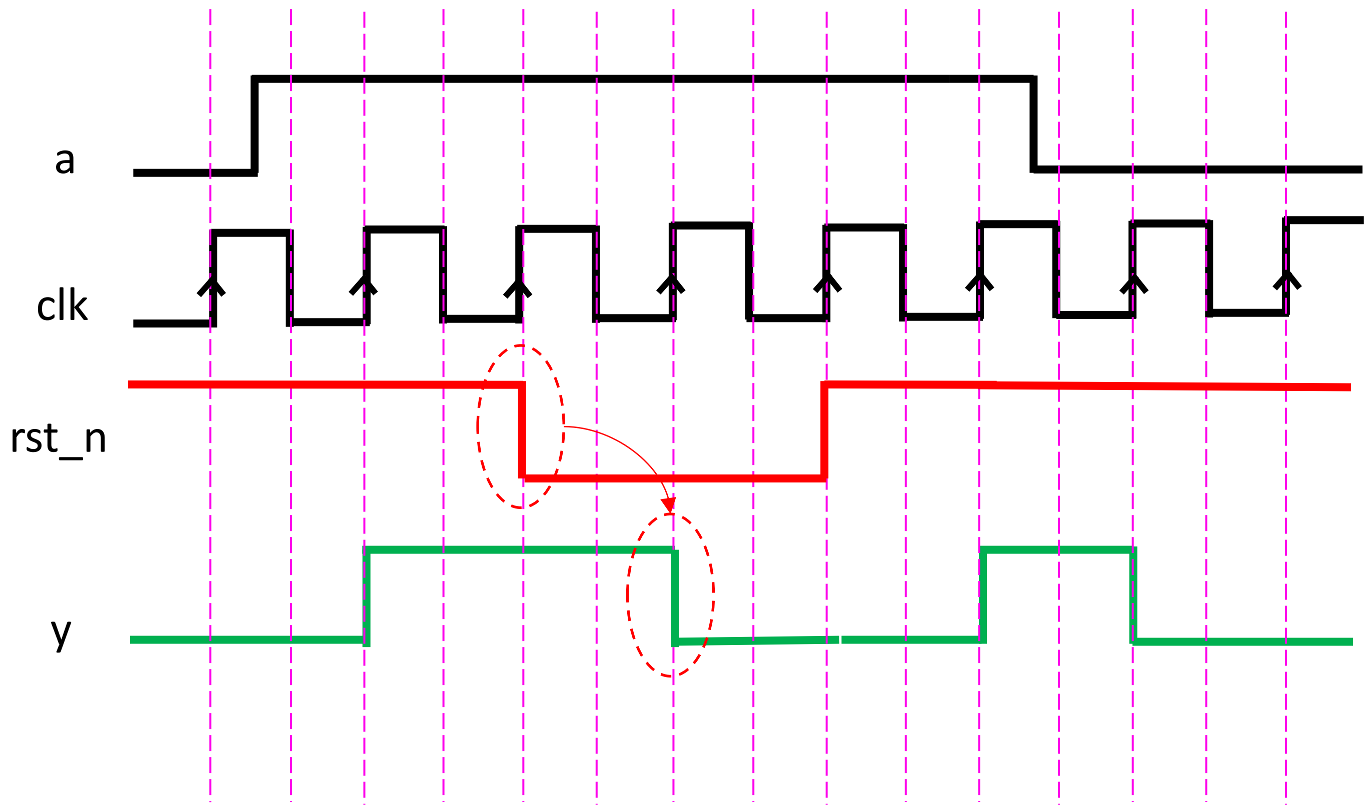
```
//code to generate a rising edge F.F with
negative edge asyn. reset
always @ (posedge clk or negedge rst_n)
begin
   if( rst_n == 1'b0)
      y <= 1'b0;
   else
      y <= a;
end
```



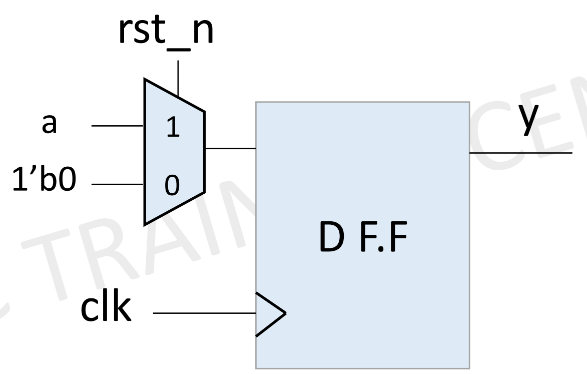
We often use active low asynchronous reset in actual design !!!

SYNCHRONOUS SET/RESET

F.F can use synchronous reset to initialize the data pin, the reset need to be synchronized with clk.



```
//code to generate a rising edge F.F with low
active sync. reset
always @ (posedge clk) begin
   if( rst_n == 1'b0)
       y <= 1'b0;
   else
       y <= a;
end
```

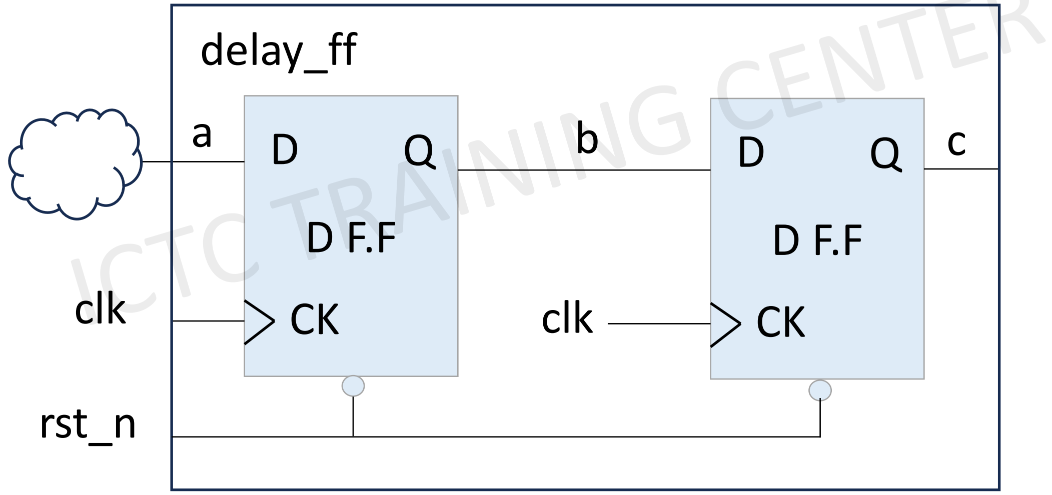


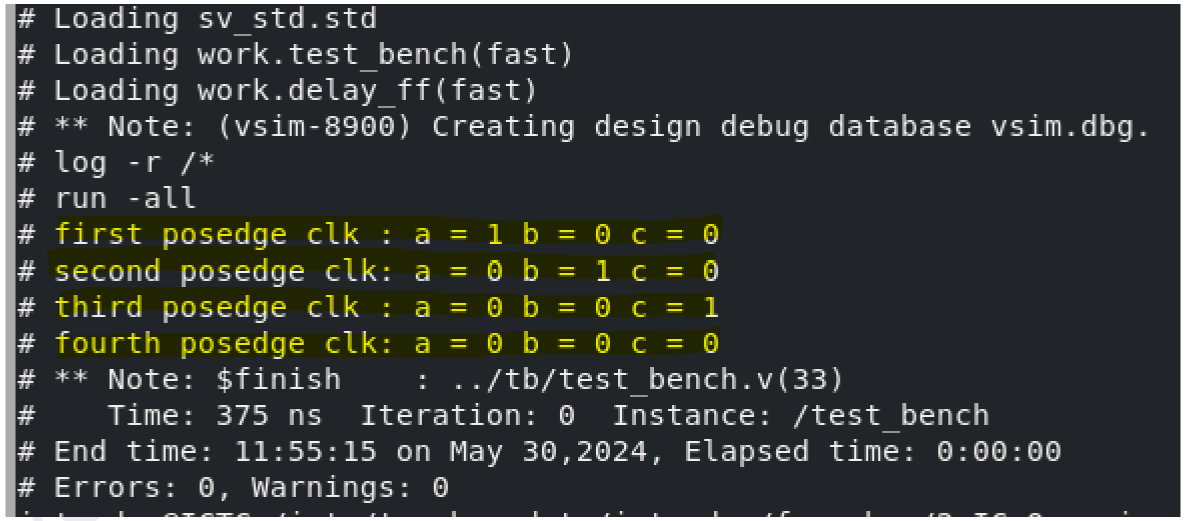
PRACTICE

Practice: Design a delay circuit as below, and confirm the previous example.

"If you read the value of b at the first clock edge, the returned value is 0. If you read b on the second clock edge, b is 1."

Copy 09_ss9/delay_ff in /ictc/student-data/share/teacher folder and create delay_ff.v under rtl.





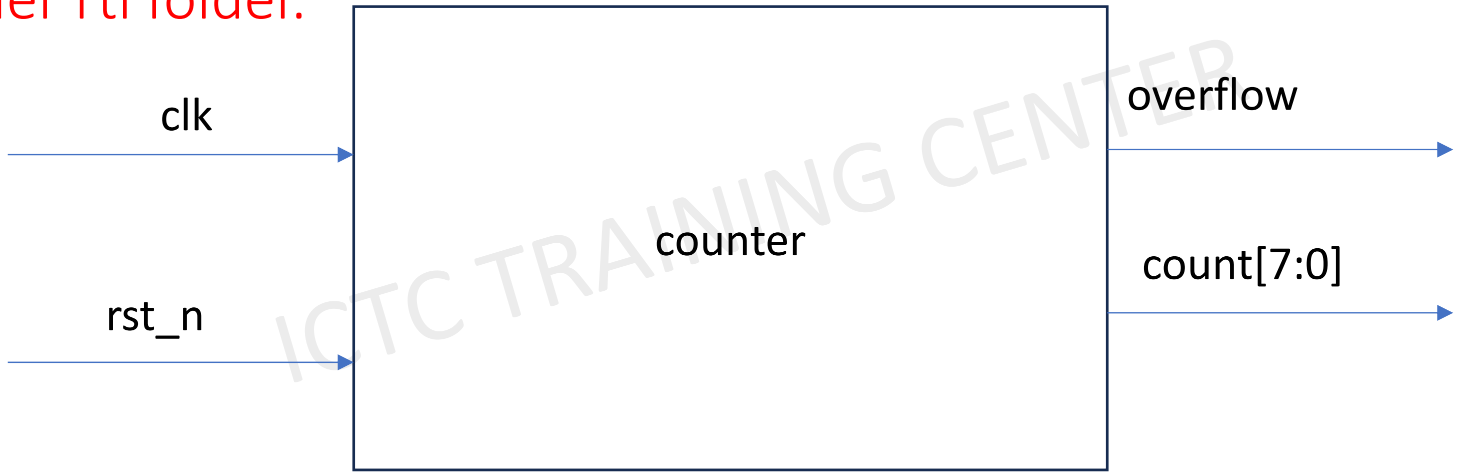
```
# Loading sv_std.std
# Loading work.test_bench(fast)
# Loading work.delay_ff(fast)
# ** Note: (vsim-8900) Creating design debug database vsim.dbg.
# log -r /*
# run -all
# first posedge clk : a = 1 b = 0 c = 0
# second posedge clk: a = 0 b = 1 c = 0
# third posedge clk : a = 0 b = 0 c = 1
# fourth posedge clk: a = 0 b = 0 c = 0
# ** Note: $finish    : ../tb/test_bench.v(33)
#    Time: 375 ns  Iteration: 0  Instance: /test_bench
# End time: 11:55:15 on May 30,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
```

After running, the log should appears like this. Open waveform for checking.

PRACTICE

<u>Practice</u>: Design a 8-bit counter

- 8-bit counter using D-FF, low active async reset, init value = 0.
- When counter reach max value, it overflowed (count_overflow = 1) and count again.
- Count_overflow is assert only when counter is overflowed and negate after that
- Draw waveform and logic diagram first
- Use the /ictc/student-data/share/teacher/09_ss9/counter for testing and create counter.v under rtl folder.



SESSION 9

SUMMARY

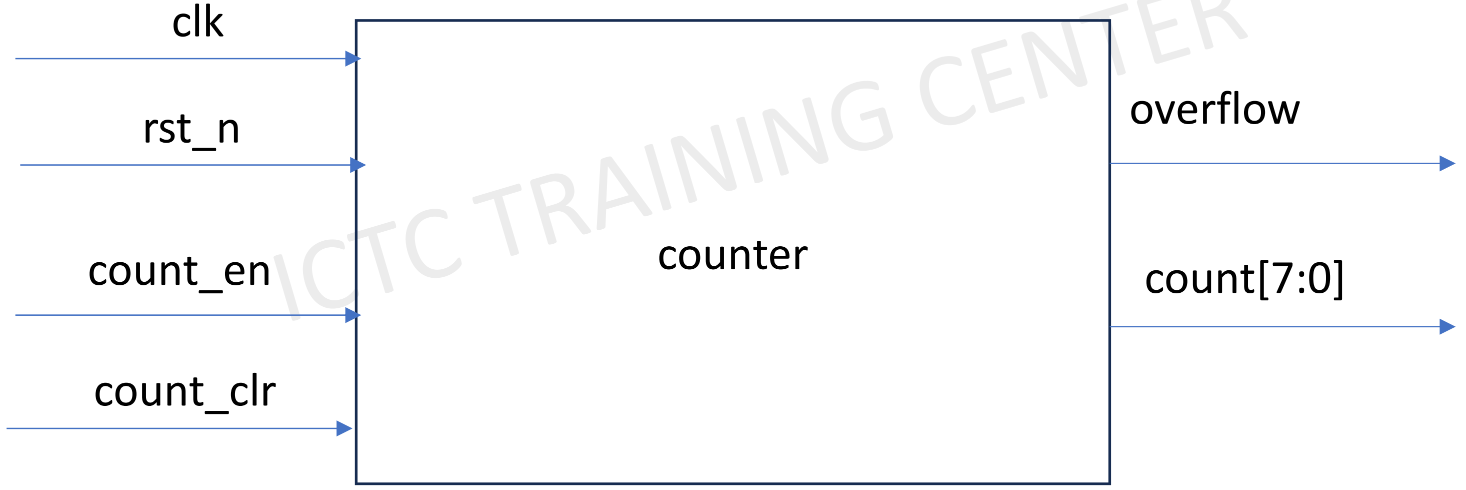
SUMMARY:

- ❑ Sequential logic elements are latches and flip-flops, can hold data.
- ❑ Need to use non-blocking assignment to generate sequential logic.
- ❑ Need to design carefully when coding for combinational logic so that not to generate un-intentional latches.
- ❑ Flip-flop has setup/hold time and propagation delay.

HOMEWORK

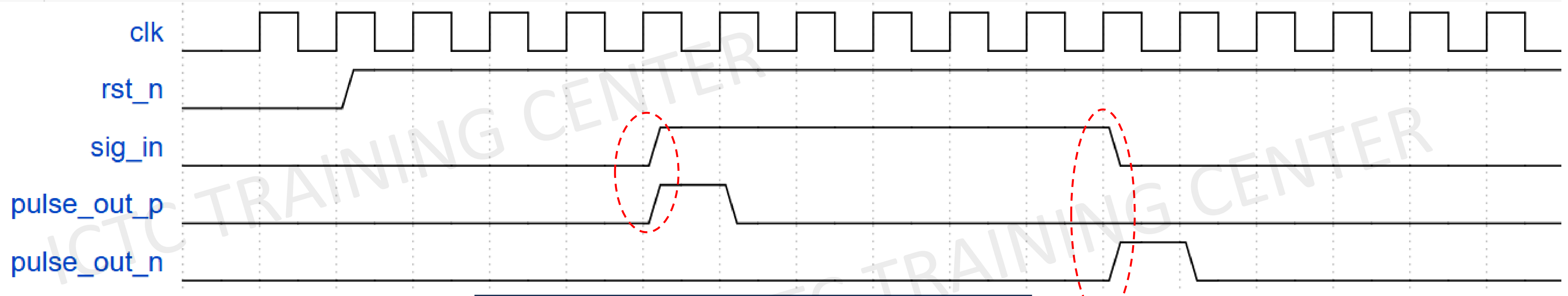
Homework1: Add more functions to 8bit counter

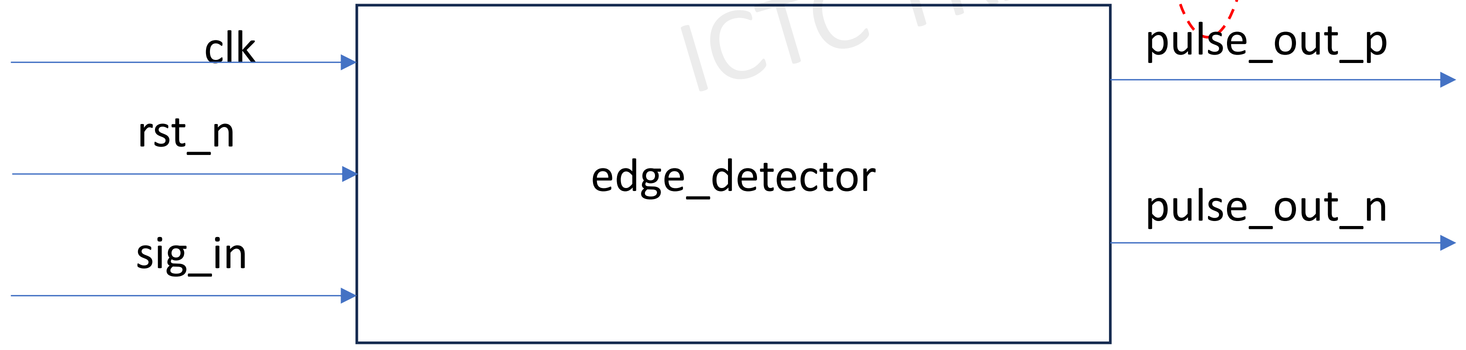
- 8-bit counter using D-FF, low active async reset.
- Reset (initial) value is 8'h00
- When counter reach max value, it overflowed (overflow = 1) and count again.
- "overflow" is assert only when counter is overflowed and negate after that
- Counter only start counting when input "count_en" is High. Otherwise, keep current value.
- Counter is cleared to its initial value (8'h00) when "count_clr" is High regardless of count_en. Note that count_clr is not a reset signal, it's just a data signal.
- Draw waveform and logic diagram first
- Use the /ictc/student-data/share/teacher/09_ss9/counter_hw for testing and create counter.v under rtl folder.



HOMEWORK

Homework2(*): Design a rising edge detector and falling endge detector following below waveform
pulse_out_p: indicates rising edge on sig_in signal.
pulse_out_n: indicates falling edge on sig_in signal
Draw logic diagram and design this module.
Create edge_detector folder under your 09_ss9 folder. Copy any rtl/sim/tb folder from any previous practice.
Draw logic diagram (2points), design the edge_detector.v (3points) and write testbench to verify for it (5points).


clk
rst_n
sig_in
pulse_out_p
pulse_out_n


clk
rst_n
sig_in
edge_detector
pulse_out_p
pulse_out_n