

# EE2026

# Digital Design

---

SEQUENTIAL LOGIC : FLIP-FLOPS

Chua Dingjuan [elechuad@nus.edu.sg](mailto:elechuad@nus.edu.sg)

# Ask Week 5 Questions here...

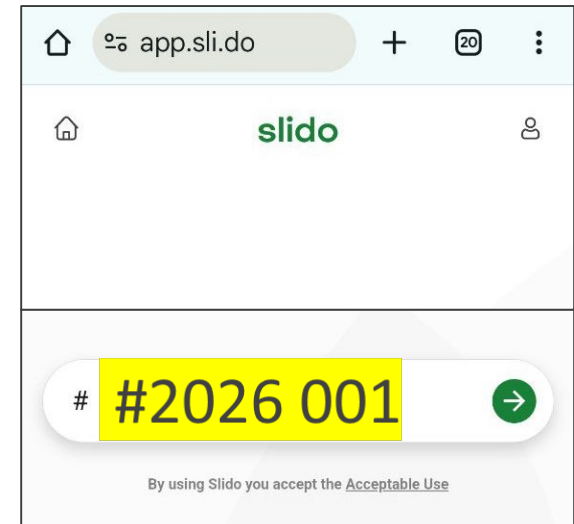
---

You can ask questions during the week using slido here : *(Slido remains open for the week)*

<https://app.sli.do/event/97WjnSxqUbJmcXLZJZkBgJ>

Or at slido.com + **#2026 001**

Or the tiny little QR :



# Course Organization (Refer Canvas)

Week	Lab (E4-03-06 Digital Electronics Lab)	Lecture (LT7A / LT6)	Tutorial
WK 1		✓	
WK 2	(B01 Friday Session one week ahead)	✓	Tutorial – 1
WK 3	Lab 1	✓	Tutorial – 2
WK 4	Lab 2	✓	Tutorial – 3
WK 5		✓	Tutorial – 4
WK 6	Lab 3	✓	Tutorial – 5 Mid-Semester Quiz 20 Sept Sat 3pm – 430pm, EA/LT7A
Recess Week			



# Course Organization (Refer Canvas)

---

Week	Lab	Lecture	Tutorial
WK 7	Project 1	✓	Tutorial – 6
WK 8	Project 2	✓	Tutorial – 7
WK 9	Project 3	✓ Guest Lecture	Tutorial – 8
WK 10			
WK 11	Verilog Evaluation		
WK 12		Final Quiz	
WK 13	Project Assessment and Demo		

No final exam 😊

# Module Lecture Structure

---

## Contents

### Part 1 (Combinational Logic)

- Number systems + Verilog
- Boolean Algebra and logic gates + Verilog
- Gate-level design and minimization + Verilog
- Combinational logic blocks and design + Verilog

### Part 2 (Sequential Logic)

- Introduction to Sequential Logic - Flip-flops + Verilog
- Counters + Verilog
- Combining combinational/sequential building blocks + Verilog
- Finite State Machine Design + Modeling of FSMs in Verilog




# Course Assessment

Component	Assessment Weight
<b>Quizzes</b>	<b>Total 40%</b>
○ Mid-Term Quiz	20%
○ <b>Weekly Canvas Quizzes</b> To id misconceptions / reinforce concepts MCQ, MRQ, FIB, etc 3 Attempts, Best Score Taken	5%
○ <b>Final Quiz</b>	15%
<b>Labs</b>	<b>Total 30%</b>
○ Lab Assignment 1	3%
○ Lab Assignment 2	5%
○ Lab Assignment 3	10%
○ Verilog Evaluation	12%
<b>Design Project – Team Work</b>	<b>Total 30%</b>
○ Project basic features (specified) ○ Enhanced features (open-ended)	30%

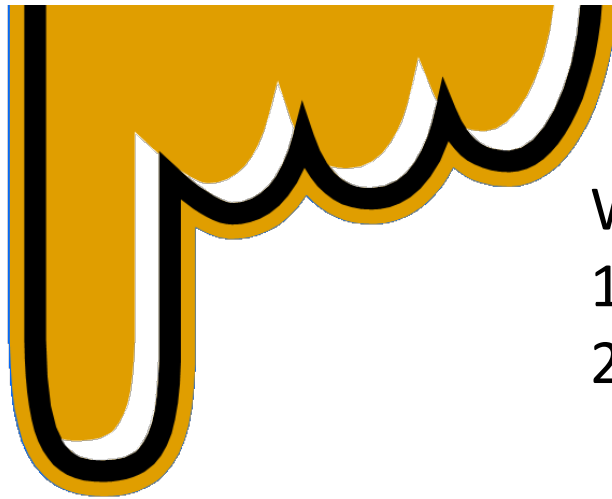


# SEQUENTIAL CIRCUITS - I



©COPYRIGHT CHUA DINGJUAN. ALL RIGHTS RESERVED.

# Design a circuit to do this >>



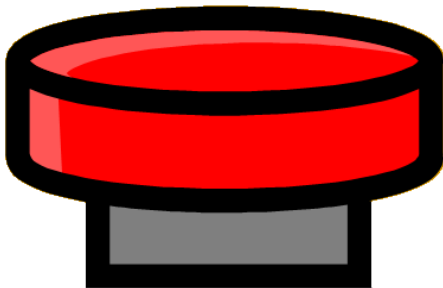
When the button is pushed :

- 1) Turn On the light *if* it is Off
- 2) Turn Off the light *if* it is On



*What is missing?*

- 1) Remembering the previous state of the bulb → *MEMORY*
- 2) Responding to an input *EVENT* (cf. input value)

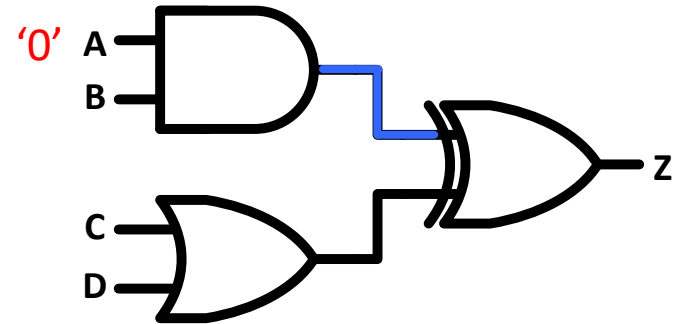




# Sequential Logic Circuits?

## Combinational Logic Circuits:

- Outputs depend on current inputs

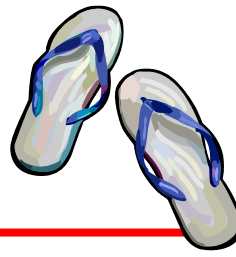


## Sequential Logic Circuits:

- Outputs depend on **current and previous** inputs → Memory!
- Requires separation of previous, current, future : states
- 2 Types of sequential circuits:

Synchronous	Asynchronous
Clocked: need a clock input	Unclocked
Responds to inputs at discrete time instants governed by a clock input	Responds whenever input signals change

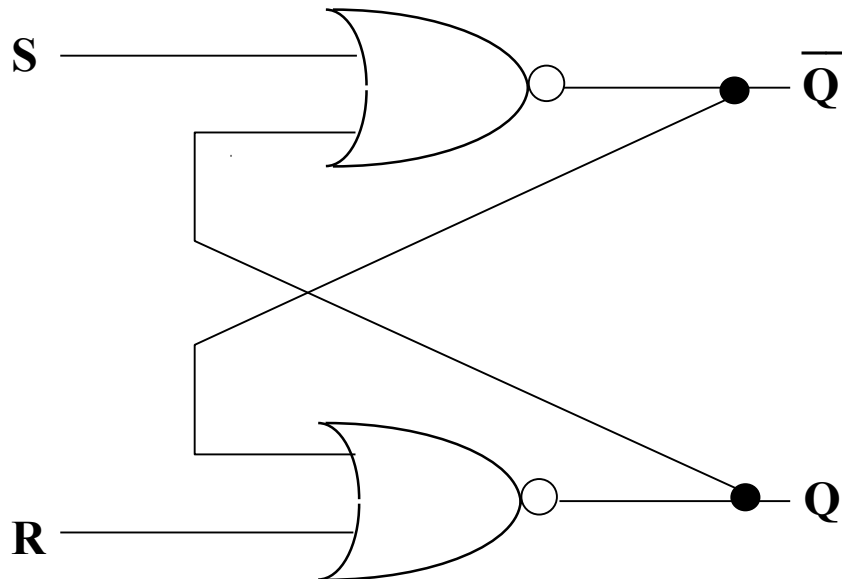
# SR Flip-flop (FF)



The simplest memory element has two stable states :

**Flip-Flop (FF)** → it can store 1 bit of information

Most basic FF : **Set-Reset** (SR) Flip-flop / Latch

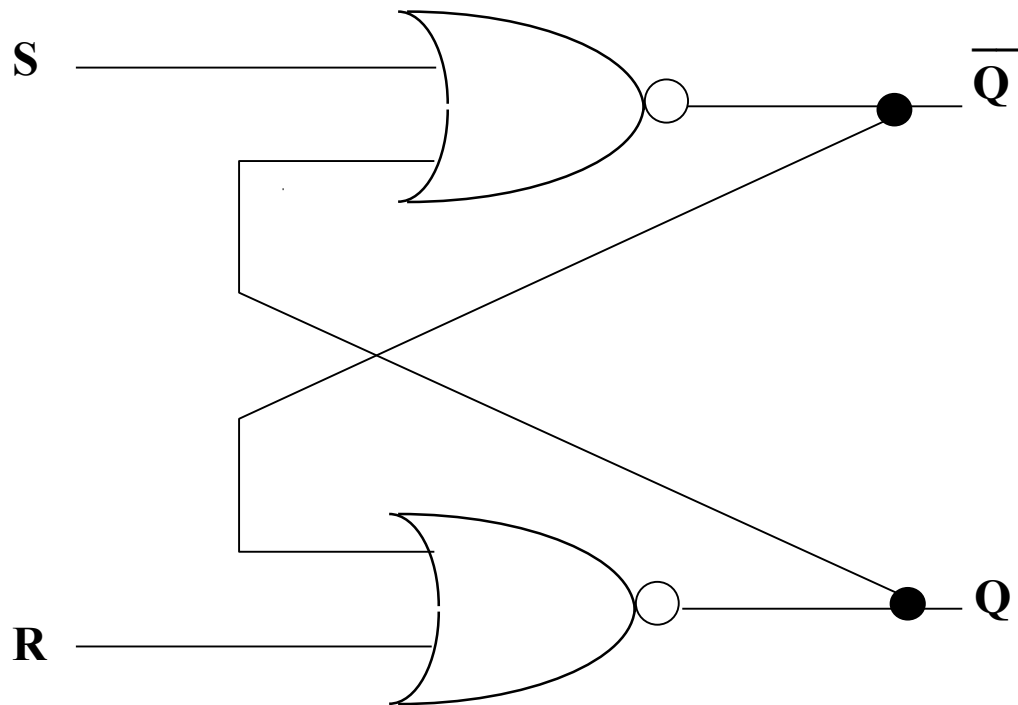


S	R	Output	Q+
0	0		
0	1		
1	0		
1	1		
0 0 is the <b>rest state</b>			

Implemented with NOR / NAND gates



# SR Flip-flop (FF)



NOR Implementation

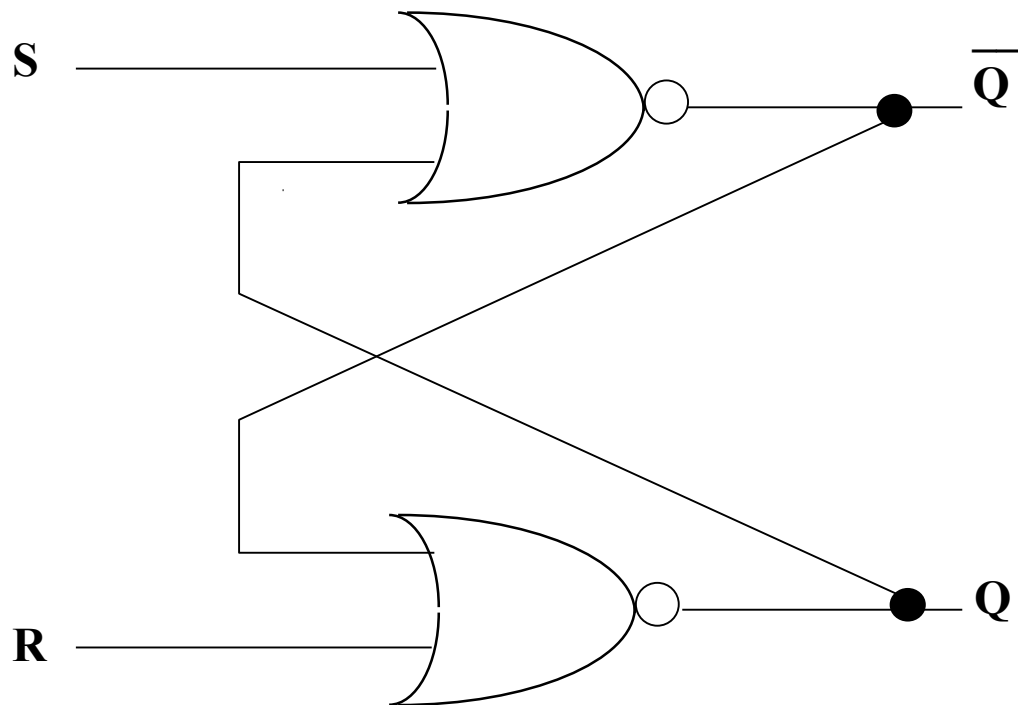
S	R	Output Q
0	0	
0	1	
1	0	
1	1	

A	B	NOR





# SR Flip-flop (FF)



NOR Implementation

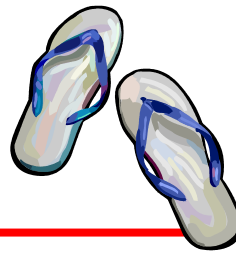
S	R	Output Q
0	0	
0	1	
1	0	
1	1	

A	B	NOR





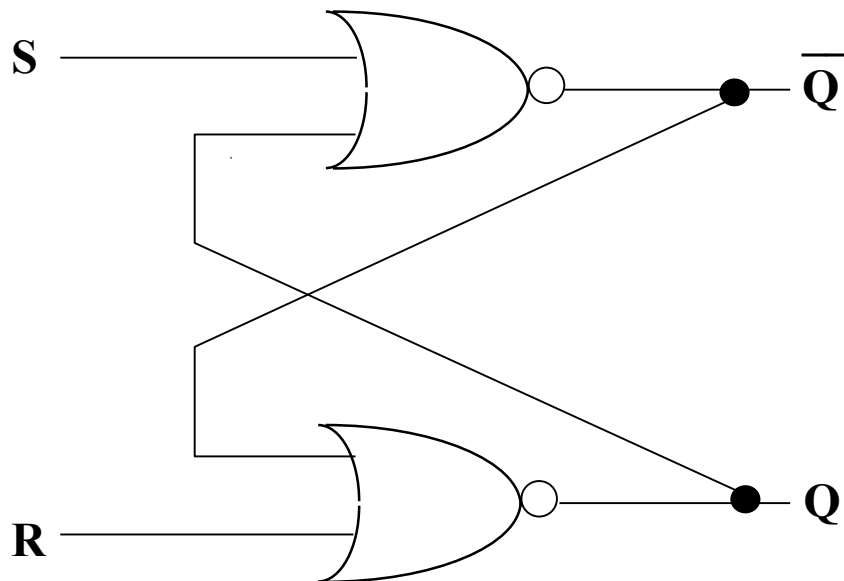
# SR Flip-flop (FF)



The simplest memory element has two stable states :

**Flip-Flop (FF)** → it can store 1 bit of information

Most basic FF : **Set-Reset** (SR) Flip-flop / Latch




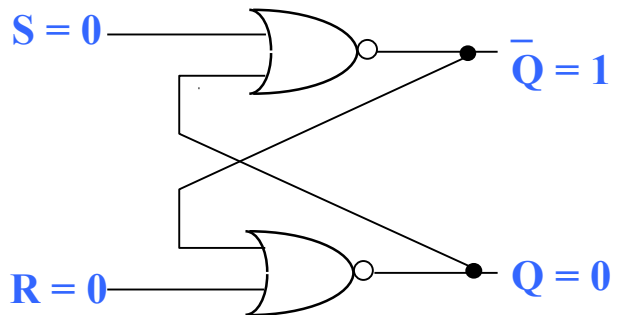
S	R	Output	Q+
0	0	Hold	Q
0	1	Clear	0
1	0	Set	1
1	1	Invalid	Invalid
0 0 is the <b>rest state</b>			

Implemented with NOR / NAND gates



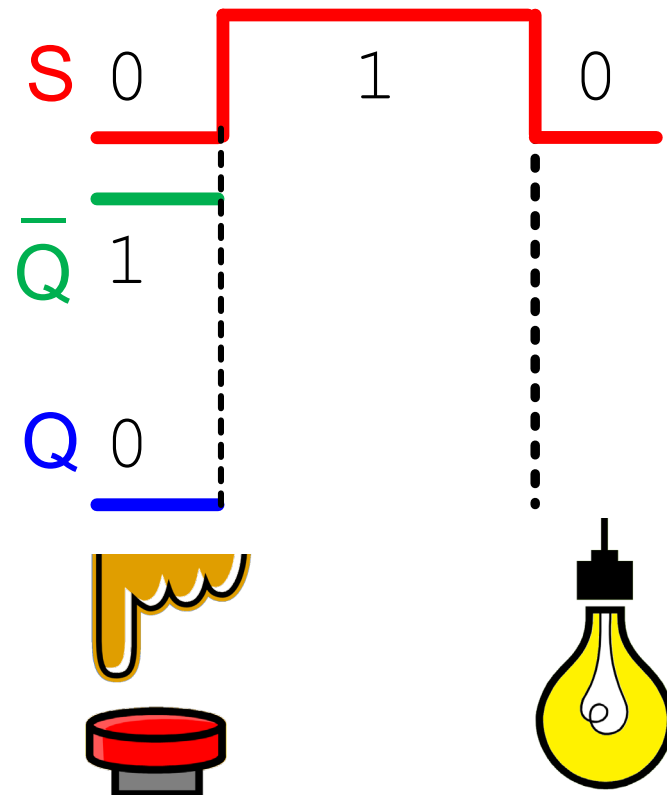
# SR Flip-flop (FF)

- FF can **record** and **store** transient events. 
- Switching is not instantaneous → **propagation delays**



S	R	Output
0	0	Hold
0	1	Clear
1	0	Set
1	1	Invalid
0 0 is the <b>rest state</b>		

Assume R=0 :

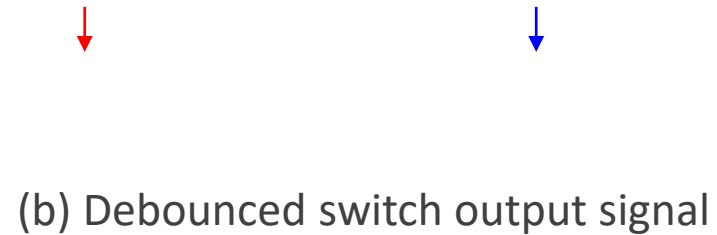
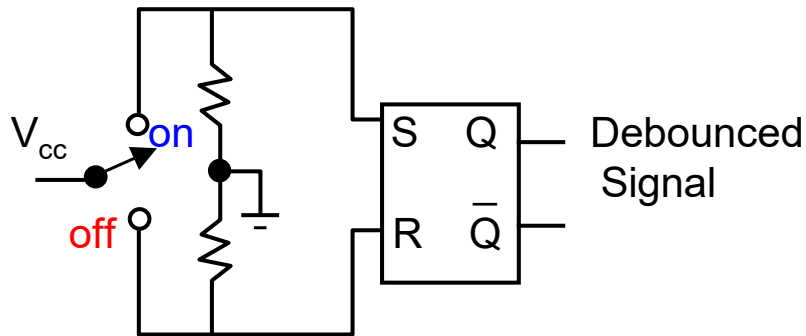
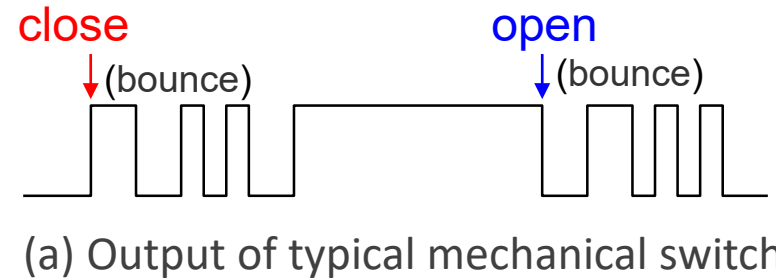
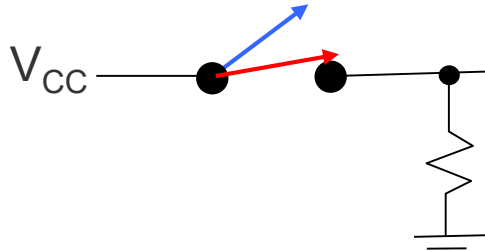
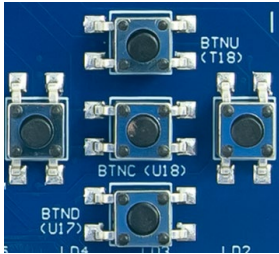


1) Assume that the *rest state* is:

$S = R = 0$ ; let  $Q = 0$ ,  $\bar{Q} = 1$

2) If  $S \rightarrow \text{pulse}$  while  $R = 0 \Rightarrow Q = 1$ ,  $\bar{Q} = 0$ ,  
i.e., the event (S going high) is recorded  
and stored as  $Q = 1$ .

# A Simple Application...



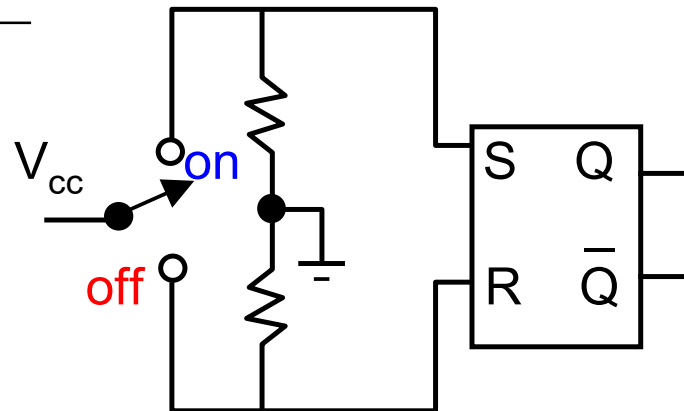
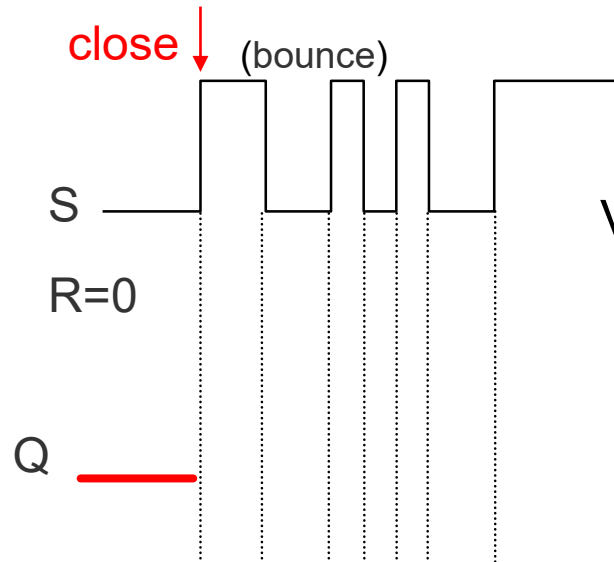
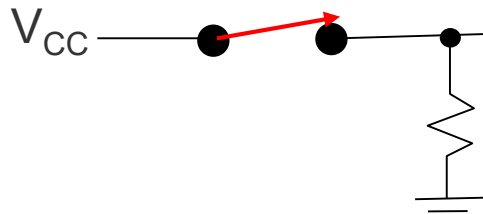
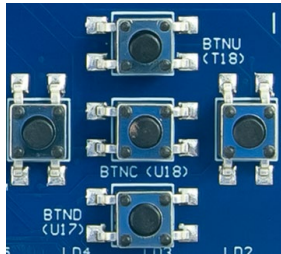
- Mechanical switches bounce before settling down which may cause problems as inputs.
- Switch **debouncing** is a common use of S-R FFs.

S	R	Output
0	0	Hold
0	1	Clear
1	0	Set
1	1	Invalid
0 0 is the <b>rest state</b>		





# A Simple Application...



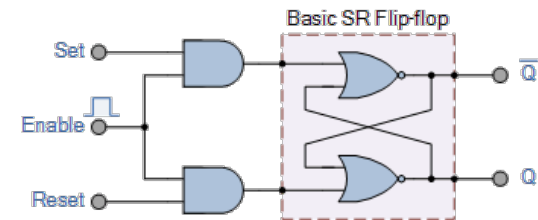
Debounced Signal

S	R	Output
0	0	Hold
0	1	Clear
1	0	Set
1	1	Invalid
0 0 is the rest state		

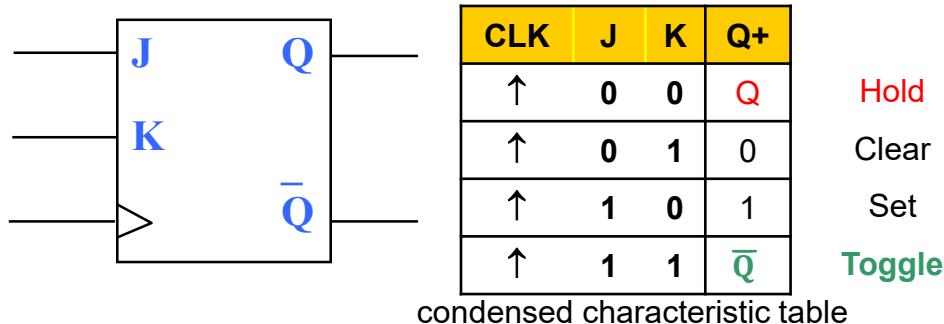




# JK Flip Flop



The JK FF is based on SR with 2 improvements : \_\_\_\_\_ & \_\_\_\_\_

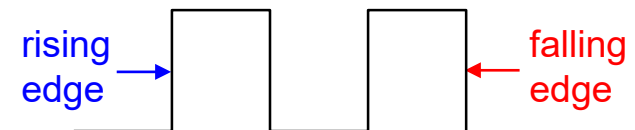


CLK	J	K	Q	Q+
↑	0	0	0	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	1	0
↑	1	0	0	1
↑	1	0	1	1
↑	1	1	0	1
↑	1	1	1	0

characteristic table

The JK FF is a **synchronous** circuit:

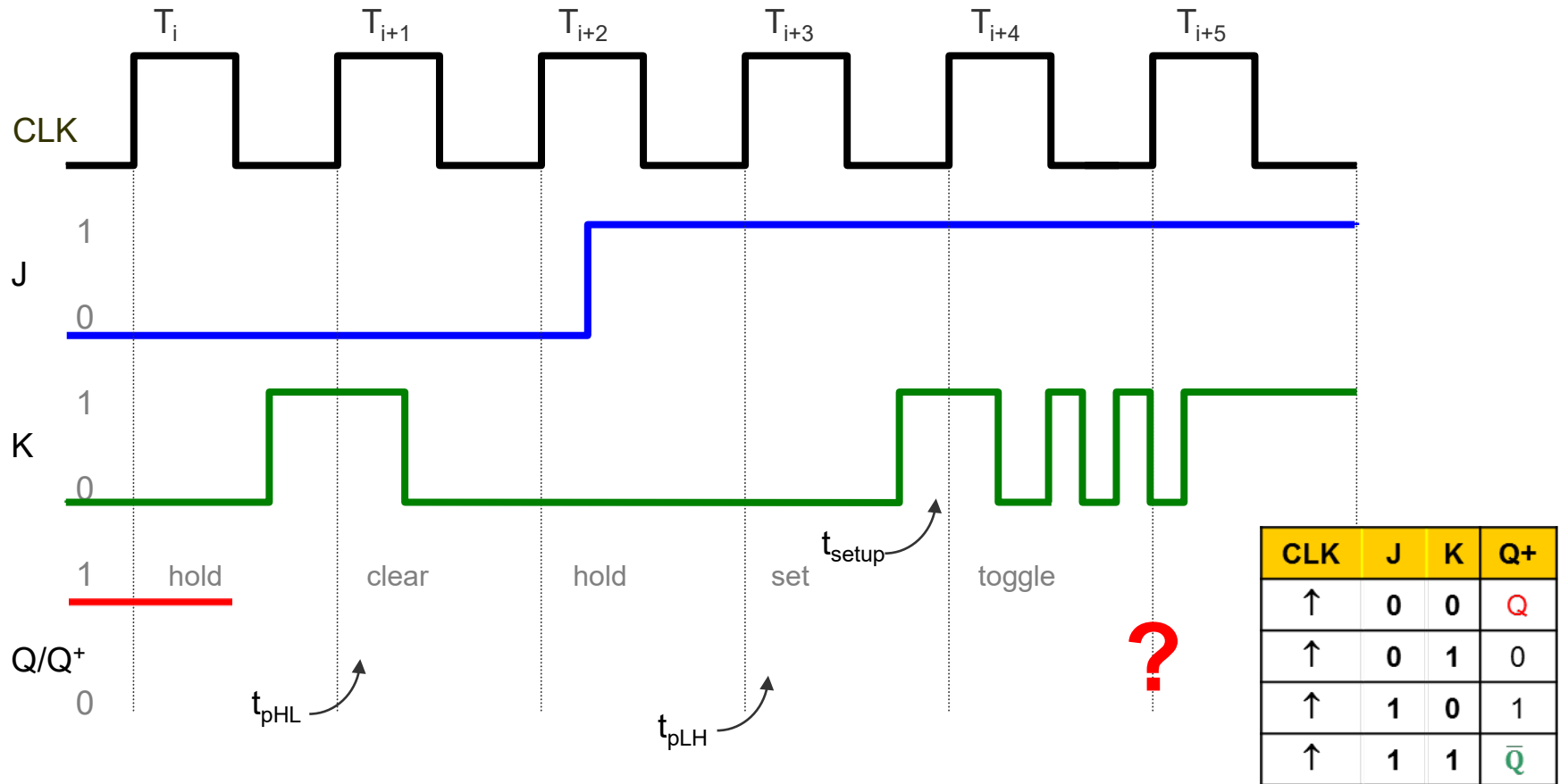
- *Clock input* is a controlling input.  
It specifies when circuit read inputs / change outputs.
- *Synchronous circuits* respond only at the \_\_\_\_\_ clock edges  
i.e., **LOW** → **HIGH**, **HIGH** → **LOW** transitions



At any other time, changing inputs have no effect on the output.

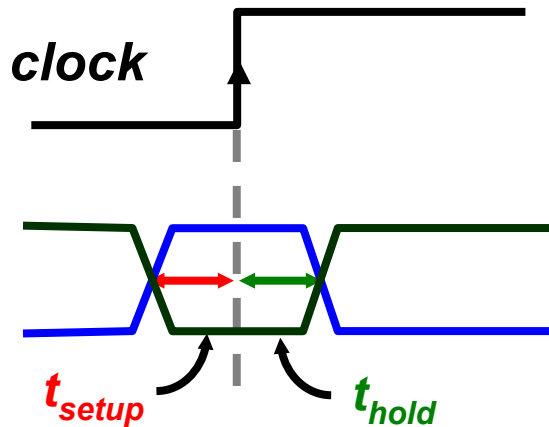


# Respond @ Active Clock Edges



- When inputs don't change → FF outputs don't change.
- If inputs change → FF output changes state only at active clock edge.

# FF Timing Parameters



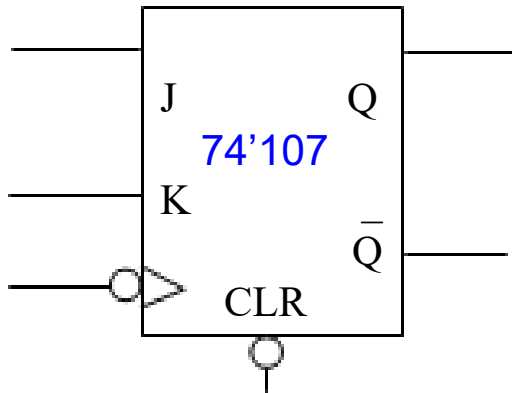
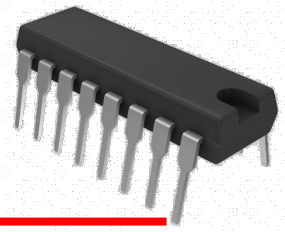
- $t_{setup}$  : minimum time before the *active* clock edge by which FF inputs must be stable.
- $t_{hold}$  : minimum time inputs must be stable after *active* clock edge
- $t_{pHL}$  : time taken for FF output to change state from High to Low.
- $t_{pLH}$  : time taken for FF output to change state from Low to High.

What happens if inputs change state right at the *active* clock transition?

Answer: output is \_\_\_\_\_

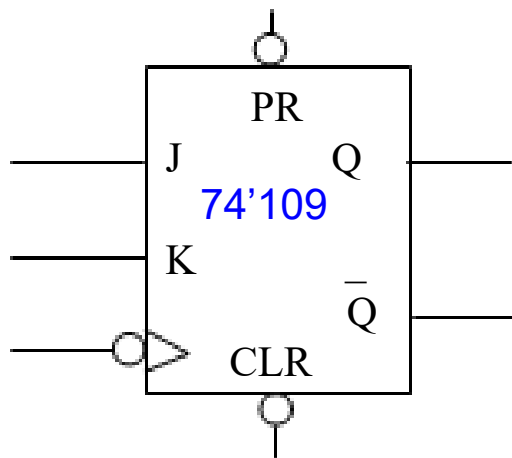
Thus, input changes must meet required *setup* & *hold* times of device  
== Operating Speed of device

# Commercially Available JK FFs



74'107 with asynchronous clear

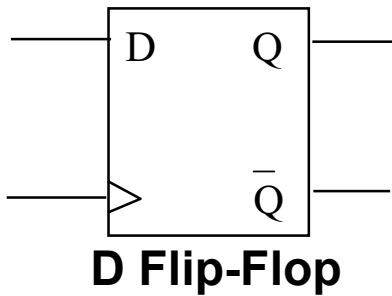
CLK	CLR	J	K	Q <sup>+</sup>
X	L	X	X	L
↓	H	L	L	Q
↓	H	L	H	L
↓	H	H	L	H
↓	H	H	H	$\bar{Q}$



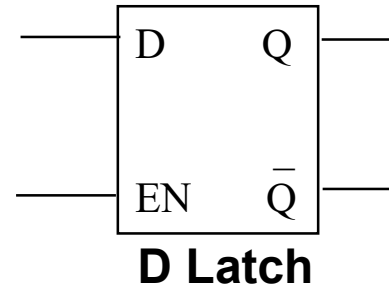
74'109 with direct set & direct clear

CLK	PR	CLR	J	K	Q <sup>+</sup>
X	L	H	X	X	H
X	H	L	X	X	L
X	L	L	X	X	not allowed
↓	H	H	L	L	Q
↓	H	H	L	H	L
↓	H	H	H	L	H
↓	H	H	H	H	$\bar{Q}$

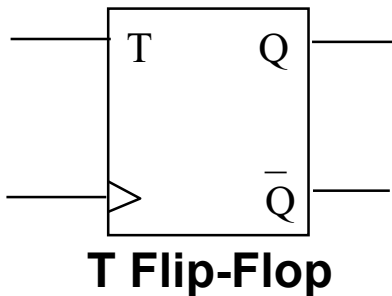
# Other Flip-Flops...



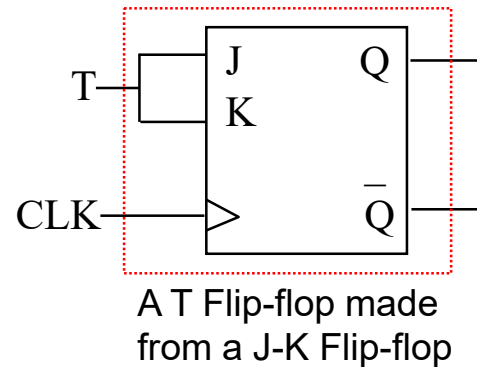
CLK	D	Q <sup>+</sup>
↑	0	0
↑	1	1



EN	D	Q <sup>+</sup>
0	X	No change
1	0	0
1	1	1



CLK	T	Q <sup>+</sup>
↑	0	Q
↑	1	$\bar{Q}$



CLK	J	K	Q <sup>+</sup>
↑	0	0	Q
↑	0	1	0
↑	1	0	1
↑	1	1	$\bar{Q}$

Since **T Flip-flops** are easy to construct from other FFs, they are not often used commercially.

# Verilog for Sequential Logic

---

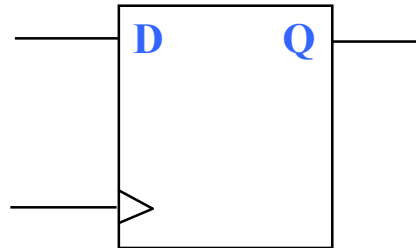
# In previous labs and lectures...

---

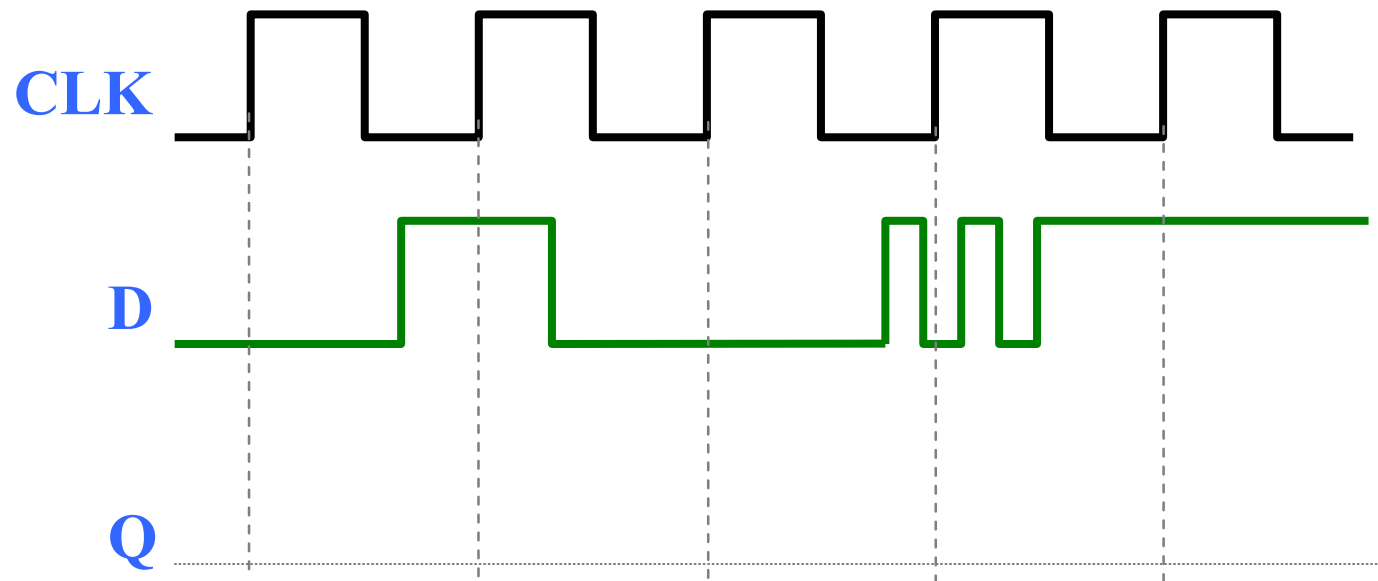
- module
- wire (net) , reg
- Logic gates ( $\sim$ ,  $\&$ ,  $|$ ,  $\wedge$  )
- Operators : +, -, ? :, { }
- Dataflow
  - Continuous Assignments : assign
- Behavioural
  - Procedural Assignments : always @ (\*)
  - Blocking (=) and Non-Blocking (<=)
- Structural modeling



# Verilog Time! – D-FF

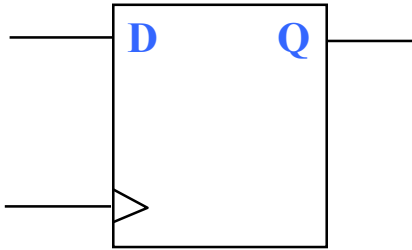


CLK	D	Q+
↑	0	0
↑	1	1





# Verilog Time! – D-FF



CLK	D	Q+
↑	0	0
↑	1	1



↑ `always @ (posedge __)`

↓ `always @ (negedge __)`

```
module dff ( input d, clk,  
            output reg q);
```

Anything assigned in an **always** block must be declared as type **reg**

```
always @ (posedge clk)
```

```
begin
```

```
q = d;
```

```
end
```

```
endmodule
```

Conceptually, the **always** block runs once when **sensitivity list** changes value. **posedge** captures the 0 → 1 change in clk.

If posedge / negedge is used in the sensitivity list, ALL signals must be used with posedge / negedge.

# Blocking & Non-blocking

Verilog supports two types of assignments within

**always**

**=** blocking assignment

- Sequential evaluation
- Immediate assignment

**always @ (\*)**

**begin**

**x = y;** 1) Evaluate y, assign result to x  
**z = ~x;** 2) Evaluate ~x, assign result to z  
**end**

Behaviour	x	y	z
Initial Condition	0	0	1
y changes			
x = y			
z = ~x			

**<=** non-blocking assignment

- Sequential evaluation
- Deferred assignment

**always @ (\*)**

**begin**

**x <= y;** 1) Evaluate y, defer assignment  
**z <= ~x;** 2) Evaluate ~x, defer assignment  
**end** 3) Assign x and z with new values

Behaviour	x	y	z	Deferred
Initial Condition	0	0	1	
y changes				
x <= y				
z <= ~x				
Assignment				

# Example

```
module example(input [2:0] A,  
               output reg [2:0] V, Z, W);
```

```
always @ (A)  
begin
```

```
    V = A | 3'b001;  
    Z <= V | 3'b100;  
    W = Z;
```

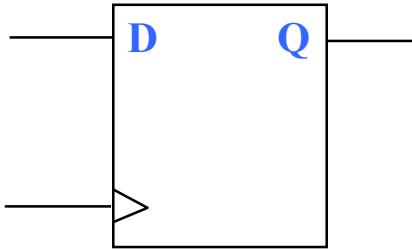
```
end  
endmodule
```

Behaviour	A	V	Z	W	Deferred
Initial Condition	000	001	101	000	
A changes	010	001	101	000	
Stmt 1	010		101	000	
Stmt 2	010	011		000	
Stmt 3	010	011	101		
Assignment					

An event occurs on **A** at simulation time :

- Stmt 1 is executed and V is assigned immediately
- Stmt 2 is executed and defer assignment to Z
- Stmt 3 is executed using old value of Z.
- Z is assigned.

# Verilog Time! – D-FF



CLK	D	Q+
↑	0	0
↑	1	1



↑ always @ (posedge \_\_)

↓ always @ (negedge \_\_)

```
module dff ( input d, clk,  
             output reg q);
```

```
always @ (posedge clk)
```

```
begin
```

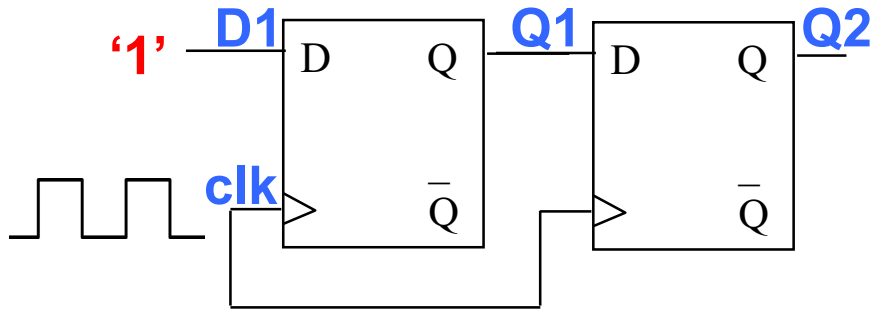
```
q <= d;           or           q = d;
```

```
end
```

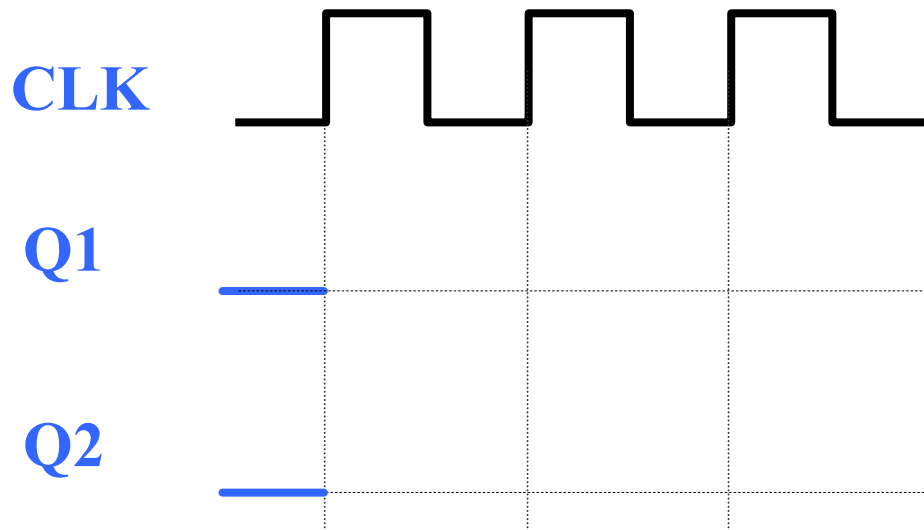
```
endmodule
```



# Two D Flip-Flops...



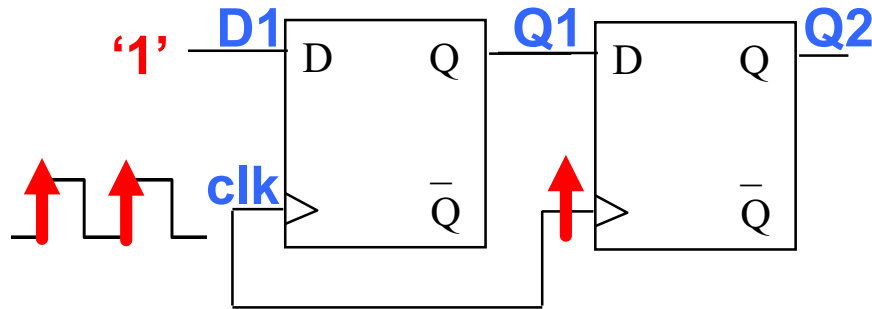
CLK	D	Q <sup>+</sup>
↑	0	0
↑	1	1



Assume initial outputs of FFs is '0' and D1 is '1'.

Behaviour	Q1	Q2
	0	0
After 1 <sup>st</sup> rising edge		
After 2 <sup>nd</sup> rising edge		

# Two D Flip-Flops... and Verilog!



Behaviour	Q1	Q2
	0	0
After 1 <sup>st</sup> rising edge	1	0
After 2 <sup>nd</sup> rising edge	1	1

```
always @ (posedge clk)
begin
```

```
q1 = d1;
q2 = q1;
```

```
end
```

Behaviour	Q1	Q2
	0	0
After 1 <sup>st</sup> rising edge		
After 2 <sup>nd</sup> rising edge		

```
always @ (posedge clk)
begin
```

```
q1 <= d1;
q2 <= q1;
```

```
end
```

Behaviour	Q1	Q2
	0	0
After 1 <sup>st</sup> rising edge		
After 2 <sup>nd</sup> rising edge		

# Discussion Example 1

---

```
always @ (posedge clk)
begin
```

```
q1 = d1;
q2 = q1;
```

```
end
```

```
always @ (posedge clk)
begin
```

```
q1 <= d1;
q2 <= q1;
```

```
end
```



# Discussion Example 1

---

```
module ff(input clk,  
d1,d2, output reg q1);
```

```
always @ (posedge clk)  
begin  
q1 <= d1;  
end
```

```
always @ (posedge clk)  
begin  
q1 <= d2;  
end
```

```
endmodule
```





# Basic Guidelines...

---

#1: When modeling sequential logic, use nonblocking assignments.

#2: When modeling simple combinational logic, use continuous assignments (assign).

#3: When modeling complex combinational logic, use blocking assignments in an always block.

#3: When modeling both sequential and combinational logic within the same always block, use nonblocking assignments.

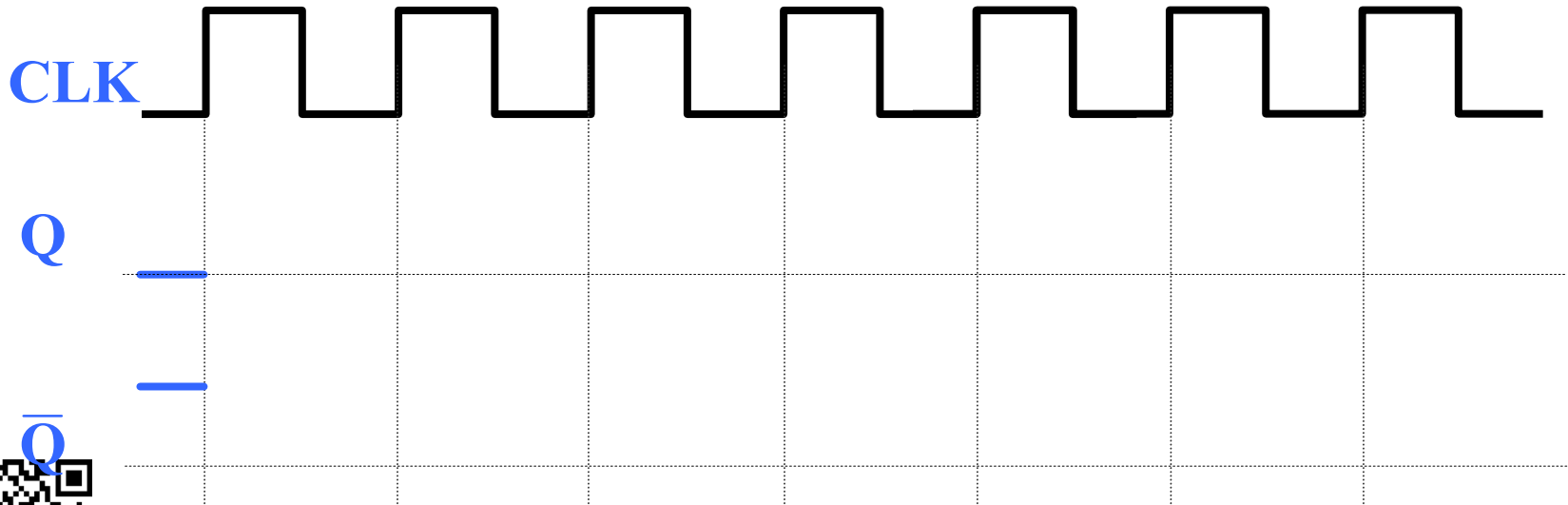
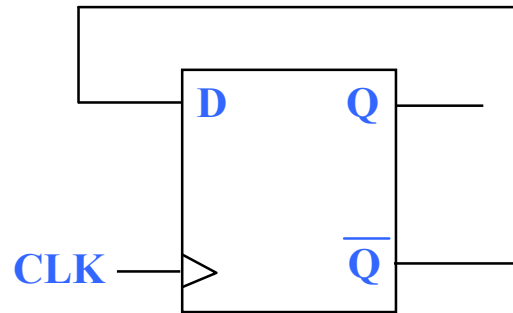
#4: Do not mix blocking and nonblocking assignments in the same always block.

#5: Do not make assignments to the same variable from more than one always block.



# Discussion Example

Given the circuit diagram below, complete the timing diagram below by filling in Q and  $\bar{Q}$ . Assume that the initial value of Q is '0' and include all propagation delays.



# Summary

---

- SR Flip Flop & Applications
- JK Flip Flop
- FF Timing Parameters
- Commercial JK Flip Flops
- Verilog description of D Flip Flop
- Blocking and Non-blocking procedural assignments
- Modeling of multiple D Flip-flops



# FFs on Artix-7 FPGA

---

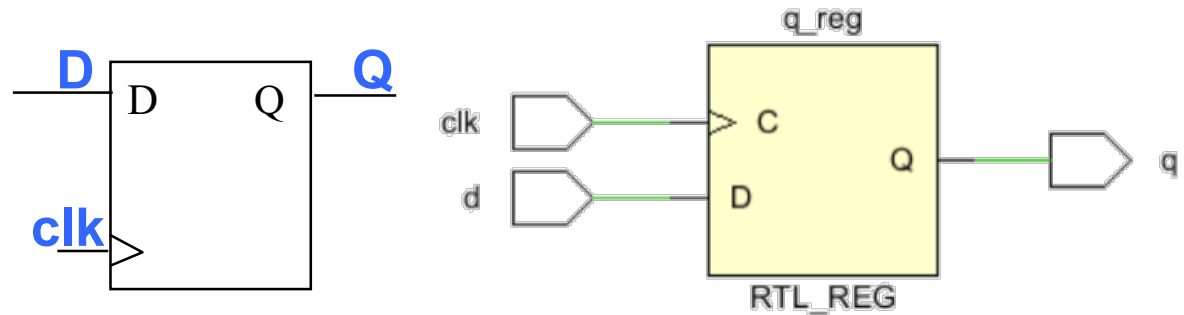
# D Flip-Flop... in Vivado?

## Verilog Code

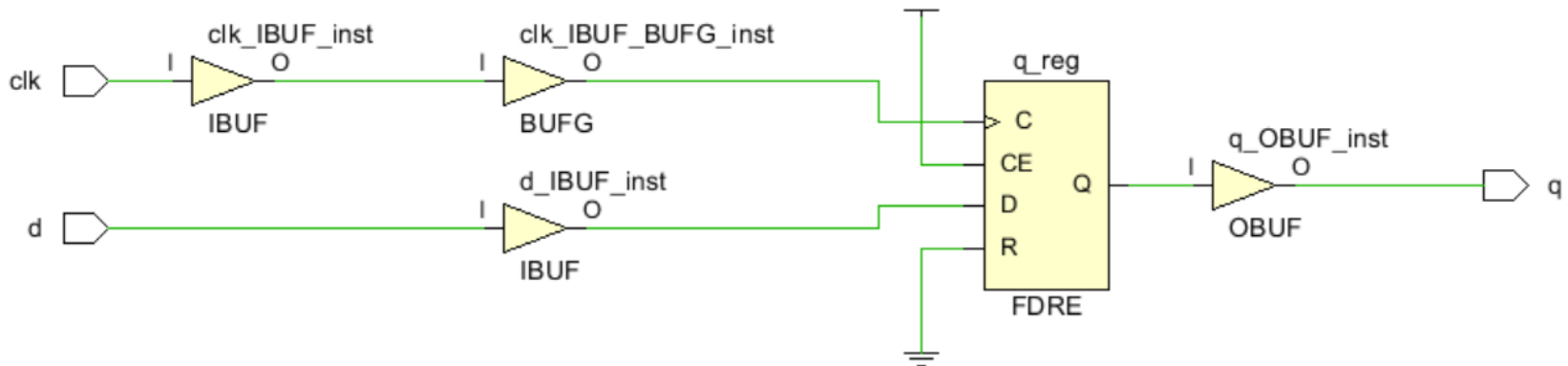
```
module dff1 (input d,clk,
output reg q);

always @ (posedge clk)
begin
q <= d;
end
endmodule
```

## Vivado RTL Schematic



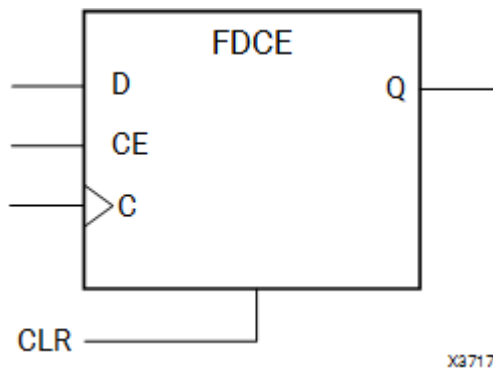
## Vivado Synthesized Schematic



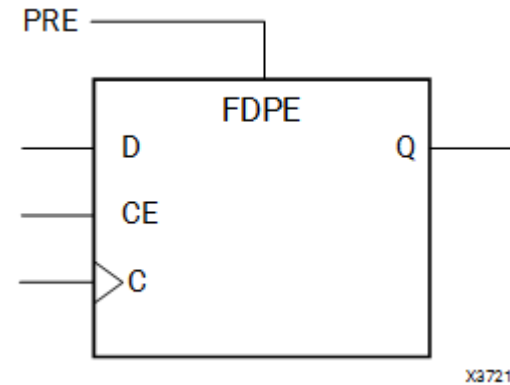
D Flip-Flop with Clock Enable and Synchronous Reset

# FDXX Primitives in 7series FPGA

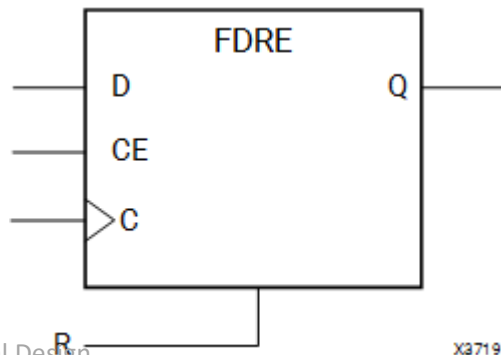
D Flip-Flop with Clock Enable and  
**Asynchronous** Clear



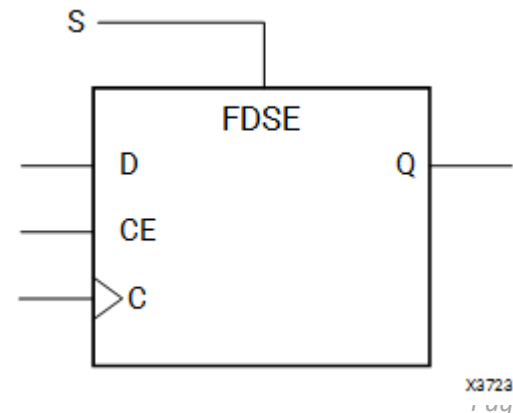
D Flip-Flop with Clock Enable and  
**Asynchronous** Preset



D Flip-Flop with Clock Enable and  
**Synchronous** Reset



D Flip-Flop with Clock Enable and  
**Synchronous** Set



# Try this!

---

- |  |                       |   |
|--|-----------------------|---|
| <code>assign</code> is used for                            | <input type="radio"/> | <input type="radio"/> <code>always @.</code>                    |
| In continuous assignments, the code is executed            | <input type="radio"/> | <input type="radio"/> when any RHS signal changes               |
| The code in the <code>always</code> block is executed when | <input type="radio"/> | <input type="radio"/> <code>module</code>                       |
| <code>always</code> is used for                            | <input type="radio"/> | <input type="radio"/> sequentially.                             |
| <code>&lt;=</code> is a                                    | <input type="radio"/> | <input type="radio"/> non-blocking procedural assignment.       |
| <code>endmodule</code> is always paired with               | <input type="radio"/> | <input type="radio"/> continuous assignments.                   |
| The sensitivity list follows the                           | <input type="radio"/> | <input type="radio"/> procedural assignments.                   |
| Code in <code>always</code> block is executed              | <input type="radio"/> | <input type="radio"/> a signal in the sensitivity list changes. |

