EES4725 Digital Circuits and FPGA Design

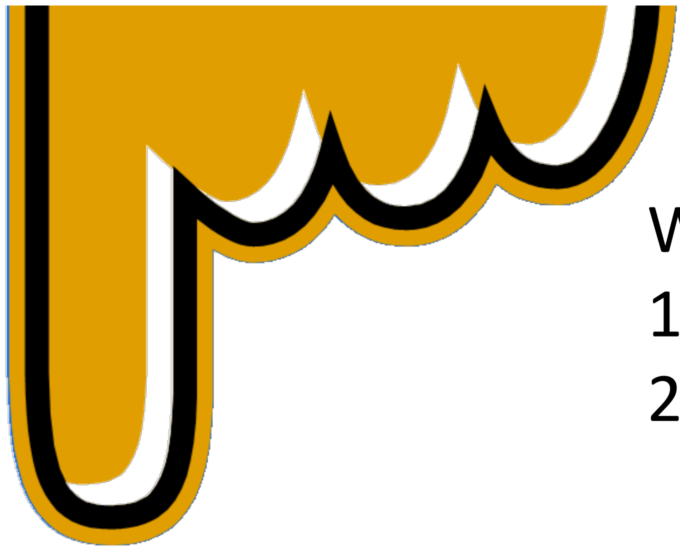
Chua Dingjuan
elechuad@nus.edu.sg

Lecture 4

SEQUENTIAL LOGIC - JK, D, T FLIP-FLOPS

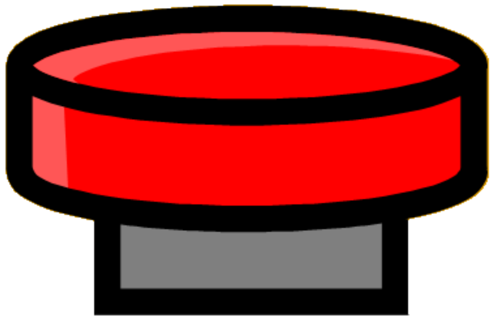
VERILOG FOR SEQUENTIAL LOGIC

Design a logic circuit to do this >>



When the button is pushed :

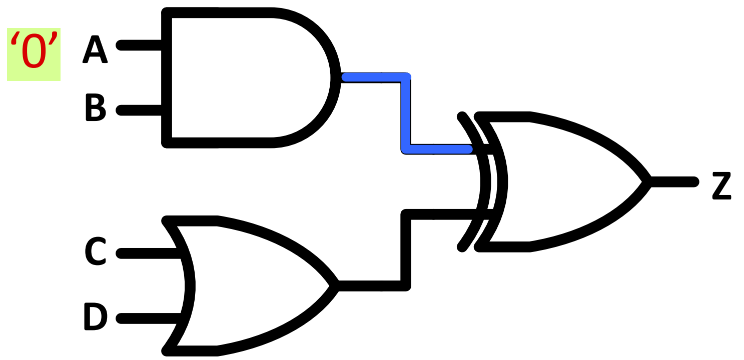
1) Turn On the light *if* it is Off
2) Turn Off the light *if* it is On



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:

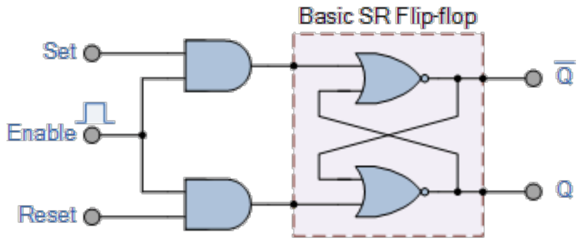
- Outputs depend on current inputs

Sequential Logic Circuits:

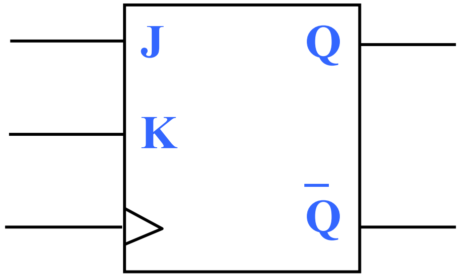
- Outputs depend on ***current and previous*** inputs ➔ Memory!
- Requires separation of previous, current, future : **states**
- 2 Types of sequential circuits:

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

JK Flip Flop



The universal JK flip flop is a commonly used flip flop in digital circuits.



CLK	J	K	Q+	
↑	0	0	Q	Hold
↑	0	1	0	Clear
↑	1	0	1	Set
↑	1	1	$\overline{Q}$	Toggle

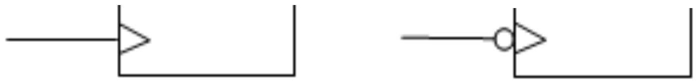
condensed characteristic table

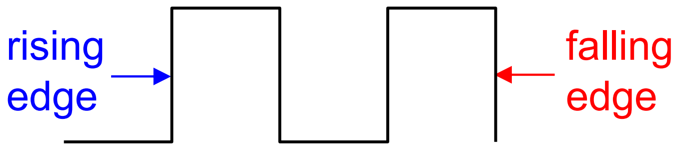
CLK	J	K	Q	Q^+
↑	0	0	0	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	1	0
↑	1	0	0	1
↑	1	0	1	1
↑	1	1	0	1
↑	1	1	1	0

characteristic table

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

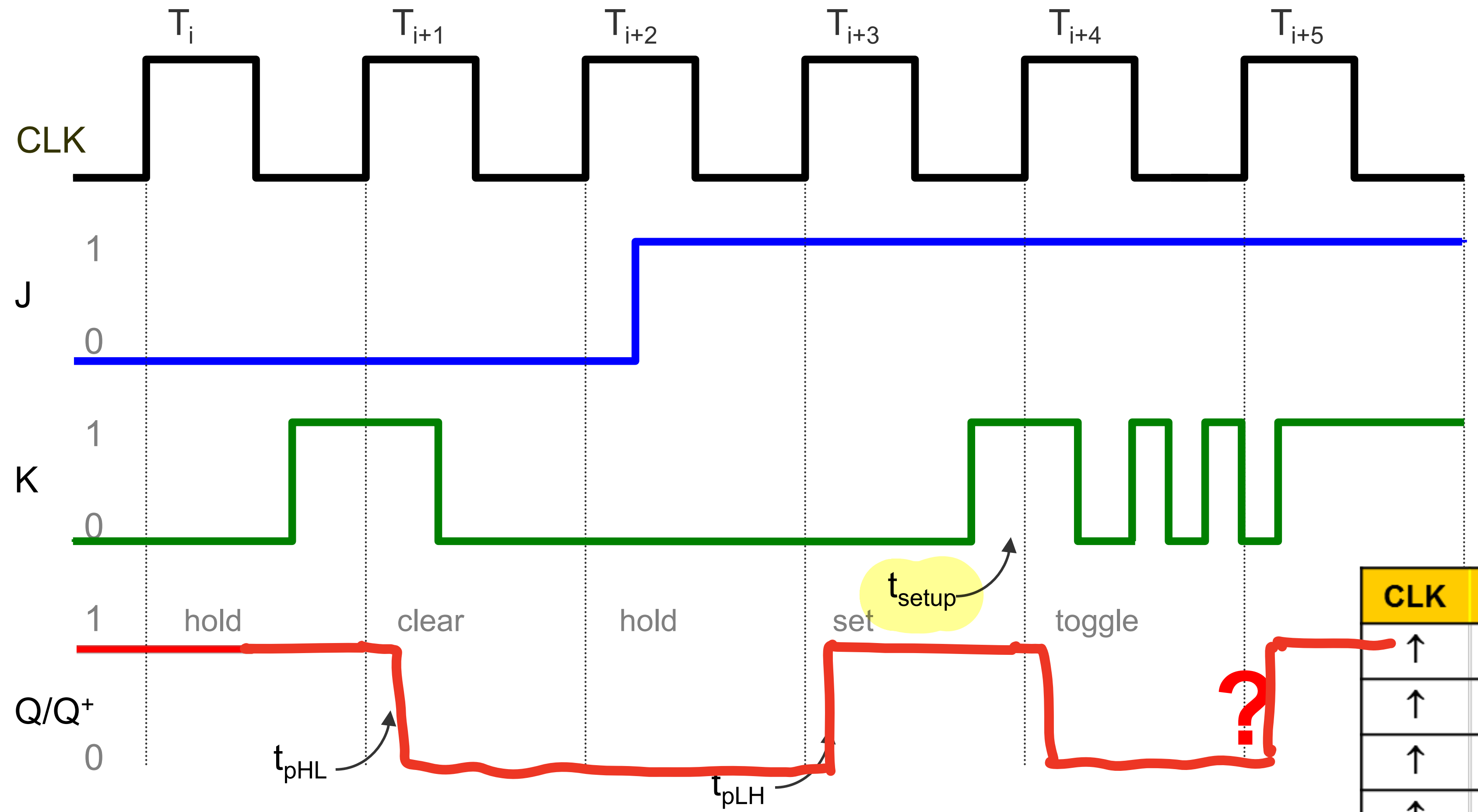
- *Clock input* is a controlling input.
  It specifies when circuit read inputs / change outputs.
- *Synchronous circuits* respond only at the active clock edges
  i.e., LOW → HIGH, HIGH → LOW transitions





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

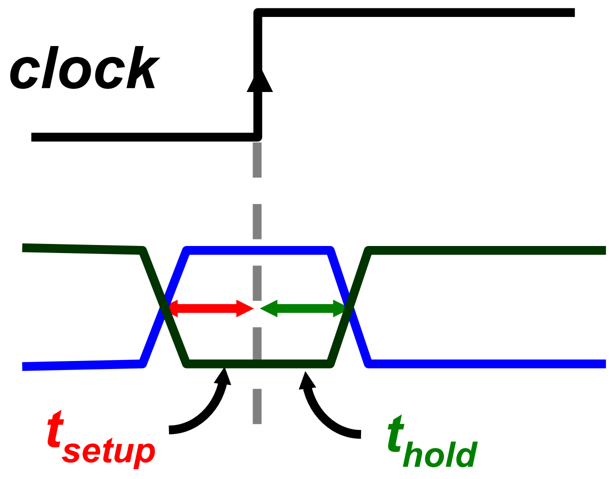
Respond @ Active Clock Edges



CLK	J	K	Q+
↑	0	0	Q
↑	0	1	0
↑	1	0	1
↑	1	1	$\bar{Q}$

- 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 unpredictable

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

http://www.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=sn74ls107a&fileType=pdf

🔹 t_setup（建立时间）

• 含义：
在“有效时钟边沿到来之前”，输入必须已经稳定的最短时间

• 直觉理解：
👉 别等铃响才交卷，得提前交

• 若违反：
FF 来不及“看清”输入，可能采样错误

🔹 t_hold（保持时间）

• 含义：
在“有效时钟边沿之后”，输入仍需保持不变的最短时间

• 直觉理解：
👉 铃刚响完，先别立刻改答案

• 若违反：
FF 刚采样完，输入就变了，结果可能出错

Maximum Clock Frequency

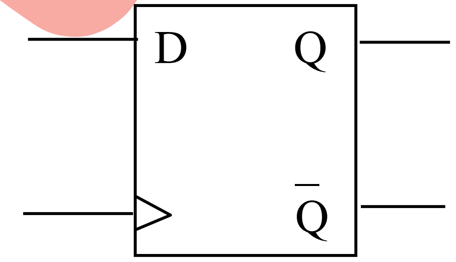
For a JK Flip-flop with the following parameters, what is the maximum theoretical clock speed?

$t_{setup} = 10ns$

$t_{pLH} = t_{pLH} = 20ns$

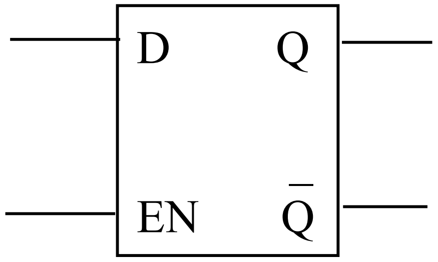
$$f_{clk,max} = \frac{1}{10ns + 20ns} = 33M$$

Other Flip-Flops…



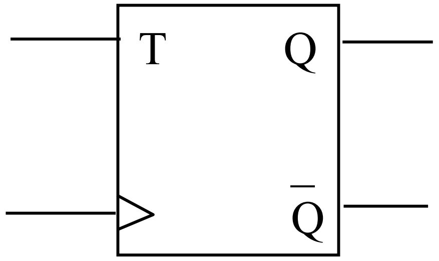
D Flip-Flop

CLK	D	Q⁺
↑	0	0
↑	1	1



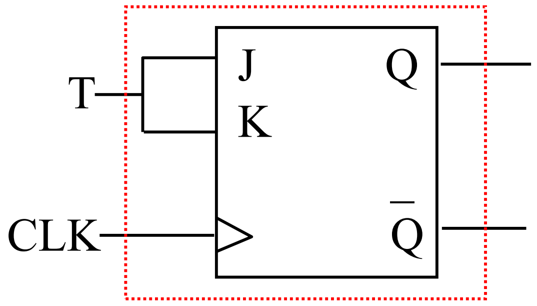
D Latch

EN	D	Q⁺
0	X	No change
1	0	0
1	1	1



T Flip-Flop

CLK	T	Q⁺
↑	0	Q
↑	1	$\bar{Q}$



A T Flip-flop made from a J-K Flip-flop

CLK	J	K	Q+
↑	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.

1️⃣ Latch 是什么（一句话）

Latch（锁存器）是“电平敏感”的存储元件：
👉 **只要使能信号有效，输入怎么变，输出就跟着变；失能后才“记住”最后的值。**

2️⃣ 和 Flip-Flop 的本质区别（重点）

对比点	Latch	Flip-Flop (FF)
触发方式	**电平触发（level-sensitive）**	**边沿触发（edge-triggered）**
何时更新	使能有效期间一直更新	仅在时钟边沿瞬间
时序分析	**复杂、容易出问题**	简单、可控
FPGA / 数字系统	**一般不推荐用**	**主流选择**

👉 **Latch = 门没关死**
👉 **FF = 门只在“咔哒”一瞬间开一下**

❌ 典型 “无意中生成 latch” 的代码

verilog 复制

```verilog
always @(*) begin
    if (en)
        q = d;
    // else 什么都没写
end
```

综合器理解为：

> en=1 → q = d
> en=0 → q 必须记住上一次的值

👉 **这正是 latch 的定义**

✅ 如果你不想要 latch，必须这样

verilog 复制

```verilog
always @(*) begin
    if (en)
        q = d;
    else
        q = 1'b0;    // 或其他明确赋值
end
```

📌 **always @(*) 里：所有路径都必须赋值**

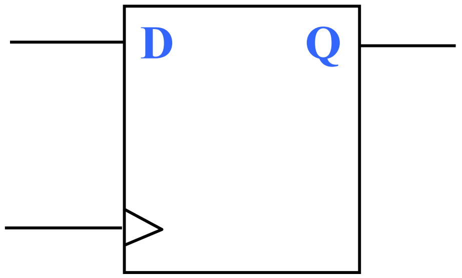
Verilog for Sequential Logic

In previous labs and lectures…

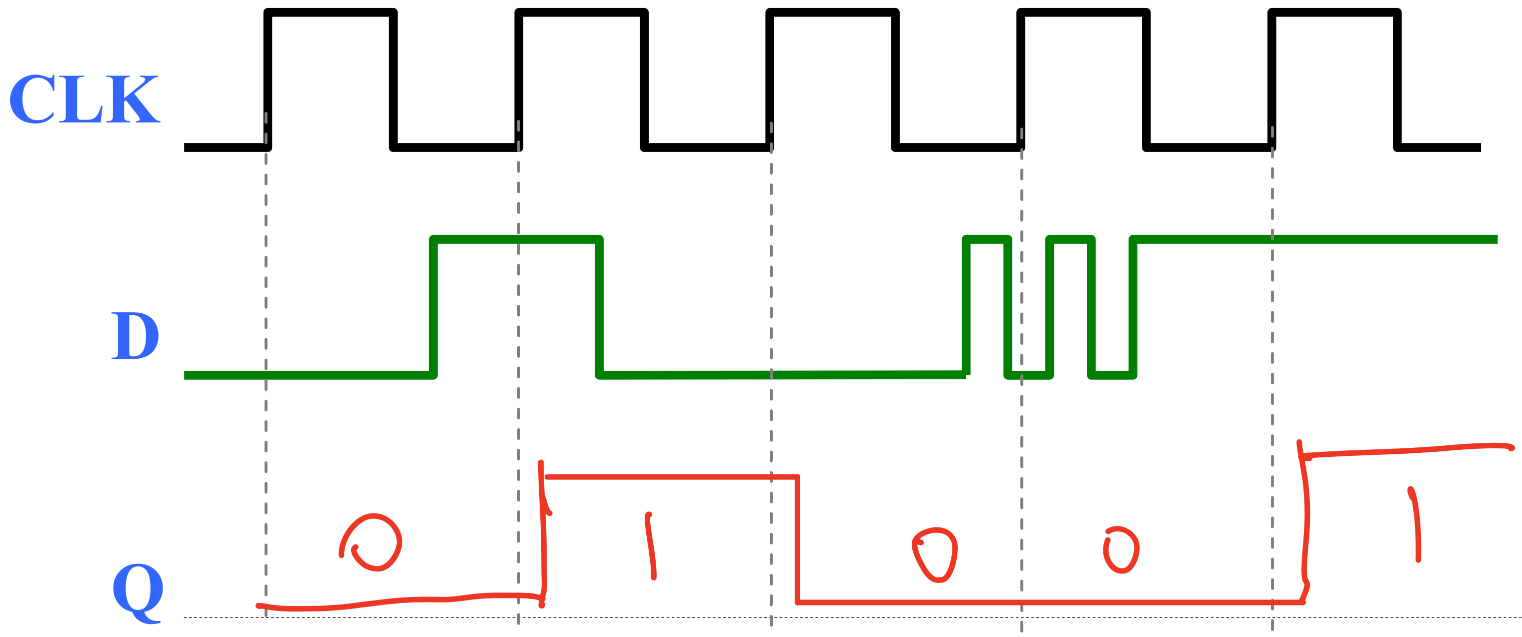
- module
- wire (net) , reg
- Logic gates (~, &, |, ^)
- Operators : +, -, ? :, { }
- assign
- always @ (*)
- structural modeling

Verilog Time! – D-FF

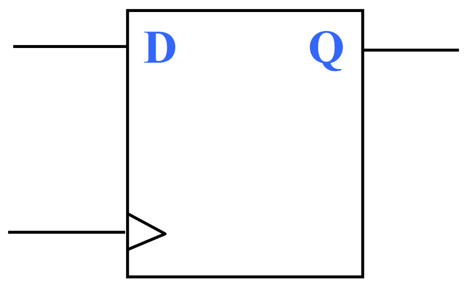
Let's start with a simple D FF!



CLK	D	Q+
↑	0	0
↑	1	1



Verilog Time! – D-FF



CLK	D	Q+
↑	0	0
↑	1	1

```
↑ always @ (posedge clk)
↓ always @ (negedge clk)
```

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

always @ (posedge clk)

begin

    q = d;

end

endmodule
```

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

Conceptually, the **always** block runs *once* when a signal in *sensitivity list* changes value.

begin and **end** behave like parentheses/brackets for conditional statements.

Blocking & Non-blocking

Verilog supports two types of assignments within

always

= blocking assignment

- Sequential evaluation
- Immediate assignment

```
always @ (*)
begin
x = y;
z = ~x;
end
```

1) Evaluate y, assign result to x
2) Evaluate ~x, assign result to z

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

<= non-blocking asignment

- Sequential evaluation
- *Deferred* assignment

```
always @ (*)
begin
x <= y;
z <= ~x;
end
```

1) Evaluate y, defer assignment
2) Evaluate ~x, defer assignment
3) Assign x and z with new values

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

Example

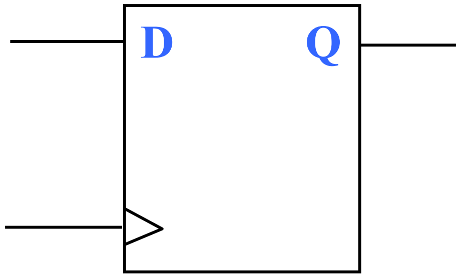
```
module example(input [2:0] A,
               output reg [2:0] V, Z, W);
always @ (A)
   begin

1  V = A | 3'b001;
2  Z <= V | 3'b100;
3  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	011	101	000	
Stmt 2	010	011	101	000	Z <= 111
Stmt 3	010	011	101	101	
Assignment	010	011	111	101	

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

- 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

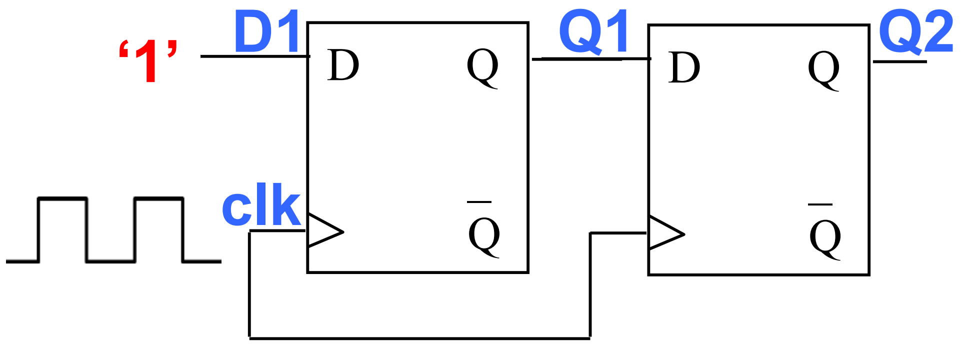


CLK	D	Q+
↑	0	0
↑	1	1

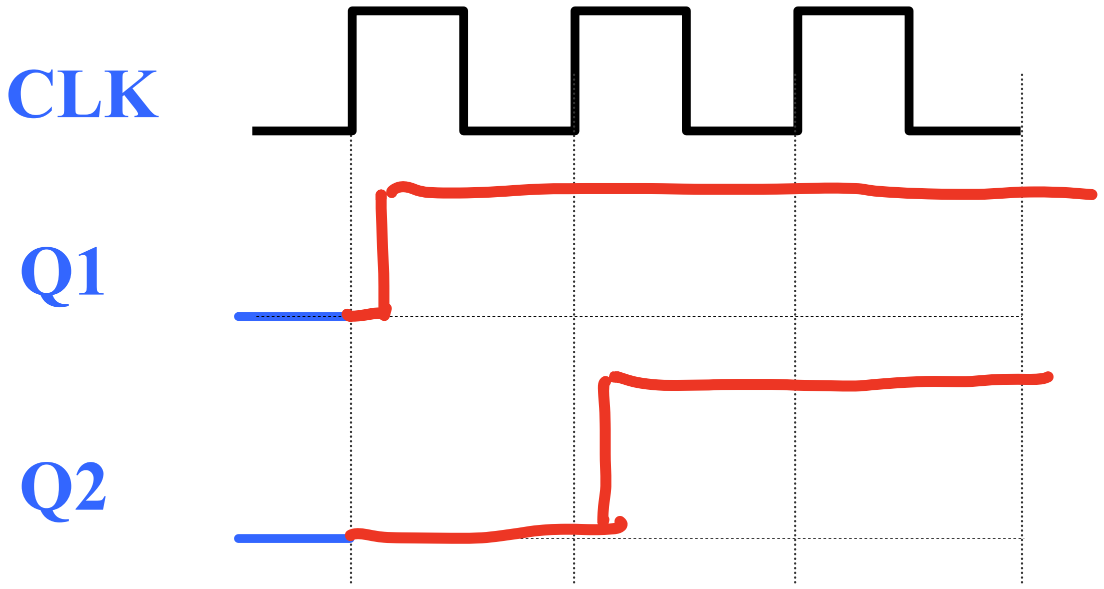
```
module dff ( input d, clk,
             output reg q);



always @ (posedge clk)

begin

     q <= d; ✓        or        q = d; ✓

end

endmodule
```

```
↑ always @ (posedge __)

↓ always @ (negedge __)
```

Two D Flip-Flops…



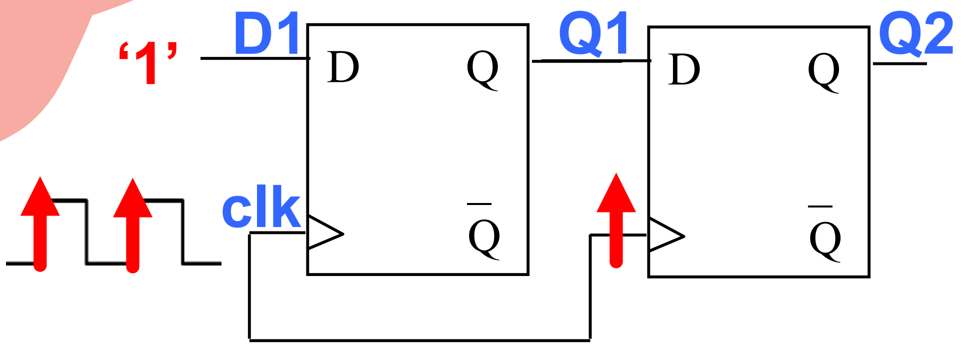
CLK	D	Q^+
↑	0	0
↑	1	1



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

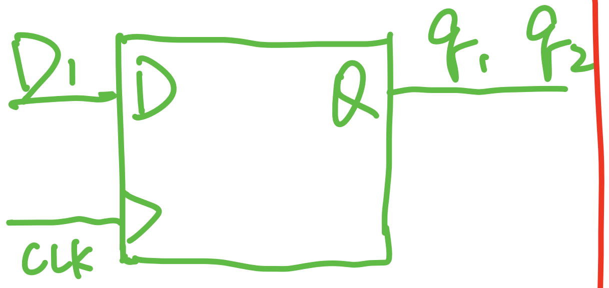
Behaviour	Q1	Q2
	0	0
After 1st rising edge	1	0
After 2nd rising edge	1	1

Two D Flip-Flops… and Verilog!



Behaviour	Q1	Q2
	0	0
After 1st rising edge	1	0
After 2nd rising edge	1	1

```
always @ (posedge clk)
begin

q1 = d1;
q2 = q1;

end
```



Behaviour	Q1	Q2
	0	0
After 1st rising edge	1	1
After 2nd rising edge	1	1

```
always @ (posedge clk)
begin

q1 <= d1;
q2 <= q1;

end
```

/ $q_2 = q_1$

Behaviour	Q1	Q2
	0	0
After 1st rising edge	1	0
After 2nd rising edge	1	1

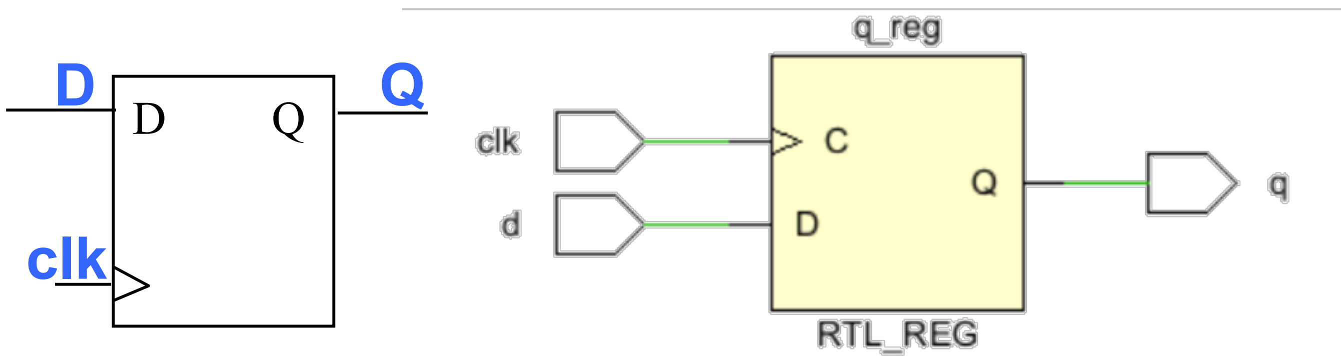
FFs on Artix-7 FPGA

D Flip-Flop… in Vivado?

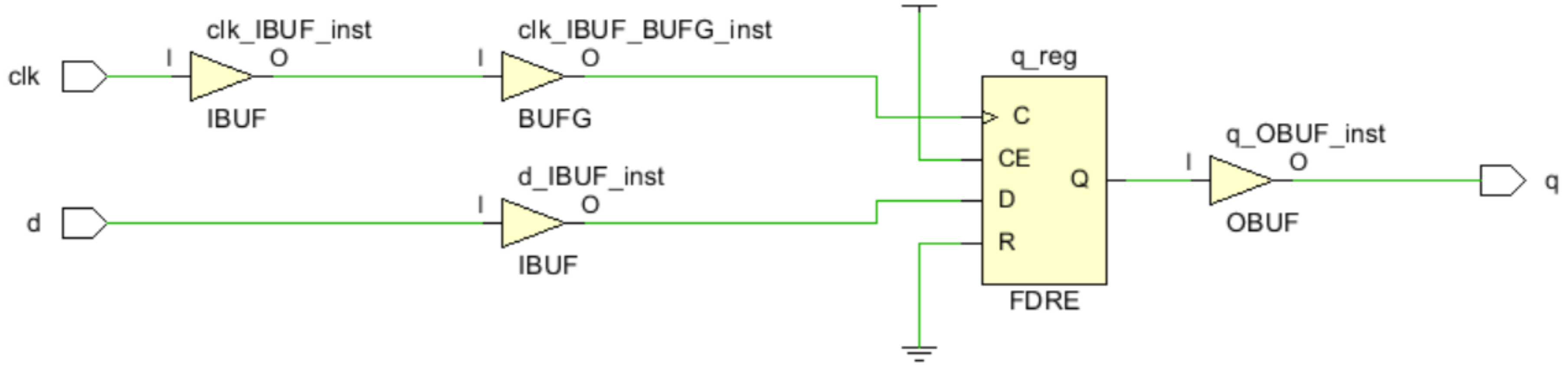
Verilog Code

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

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

Vivado RTL Schematic



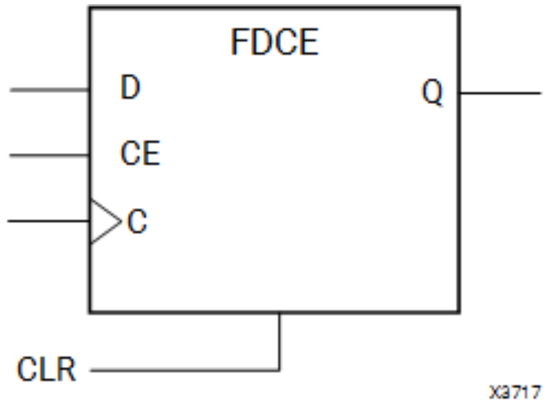
Vivado Synthesized Schematic



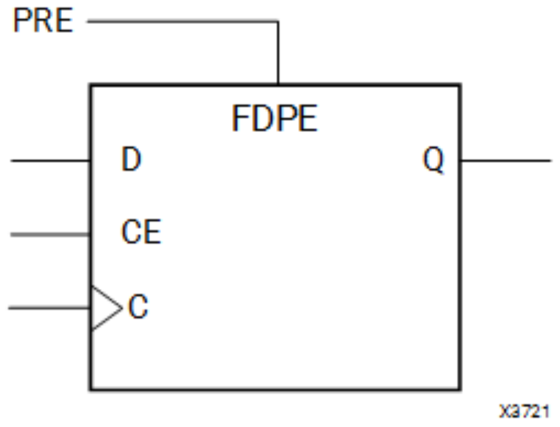
D Flip-Flop with Clock Enable and Synchronous Reset

FDXX Primitives in 7series FPGA

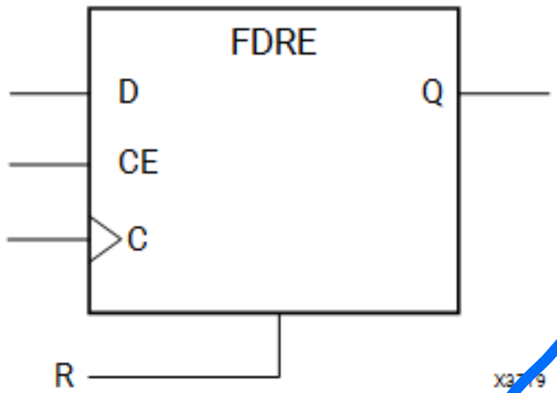
1 D Flip-Flop with Clock Enable and Asynchronous Reset



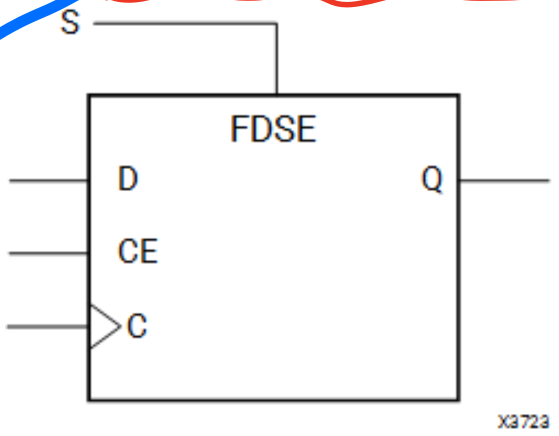
2 D Flip-Flop with Clock Enable and Asynchronous Set



3 D Flip-Flop with Clock Enable and Synchronous Reset



4 D Flip-Flop with Clock Enable and Synchronous Set

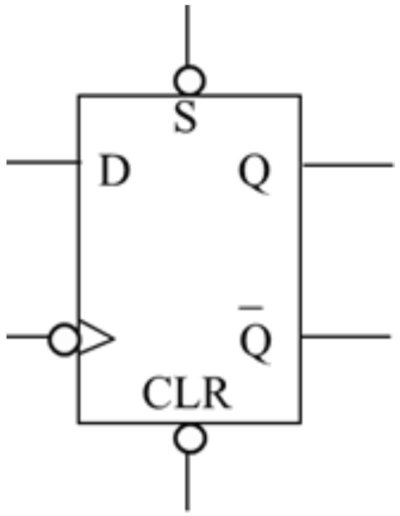


Example - Practice Question

The 74'74B is an integrated circuit containing negative-edge triggered D flip-flops with synchronous set and asynchronous reset inputs. The D flip-flop receives four 1-bit input signals, D, CLK, S, CLR and produces two 1-bit output signals, Q and QB. Write a Verilog program that implements the D flip-flop according to the characteristic table shown below.

CLK	CLR	S	D	Q⁺
X	0	X	X	0
↓	1	0	X	1
↓	1	1	0	0
↓	1	1	1	1

Reset
Set
DFF



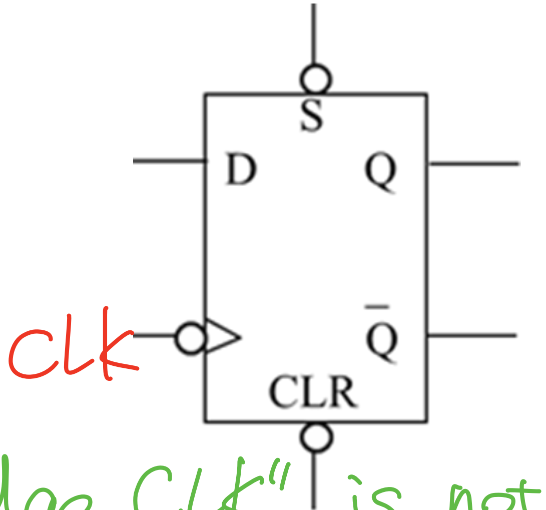
```
always @ ( negedge CLK, negedge CLR )
```

高有效 → posedge

低有效,异步清零

Practice Question

```
module dff ( input D, CLK, CLR, S, output reg Q, output QB);

always @ (negedge CLK, negedge CLR )

begin
if ( CLR == 0 )
    Q <= 1'b0;

else // CLR == 1
begin
    if ( S == 0 )
    Q <= 1'b1;
else Q <= D;
end
end

    assign QB = ~Q;

endmodule
```

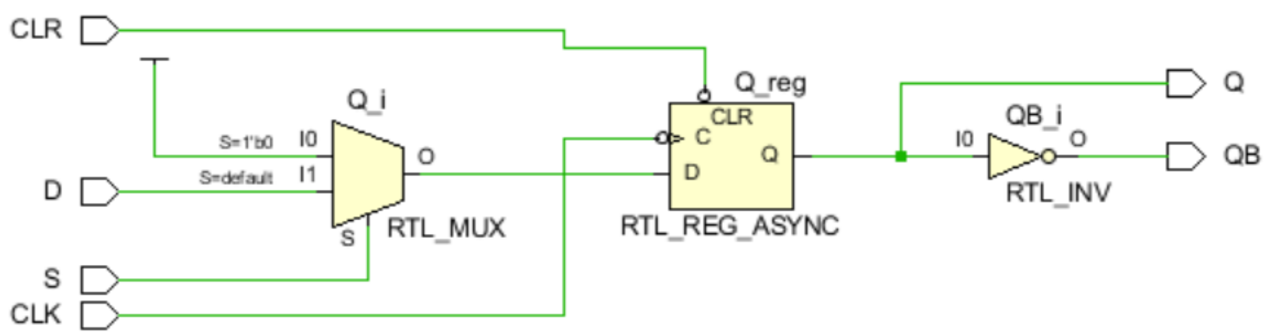


Only "negedge CLK" is not enough

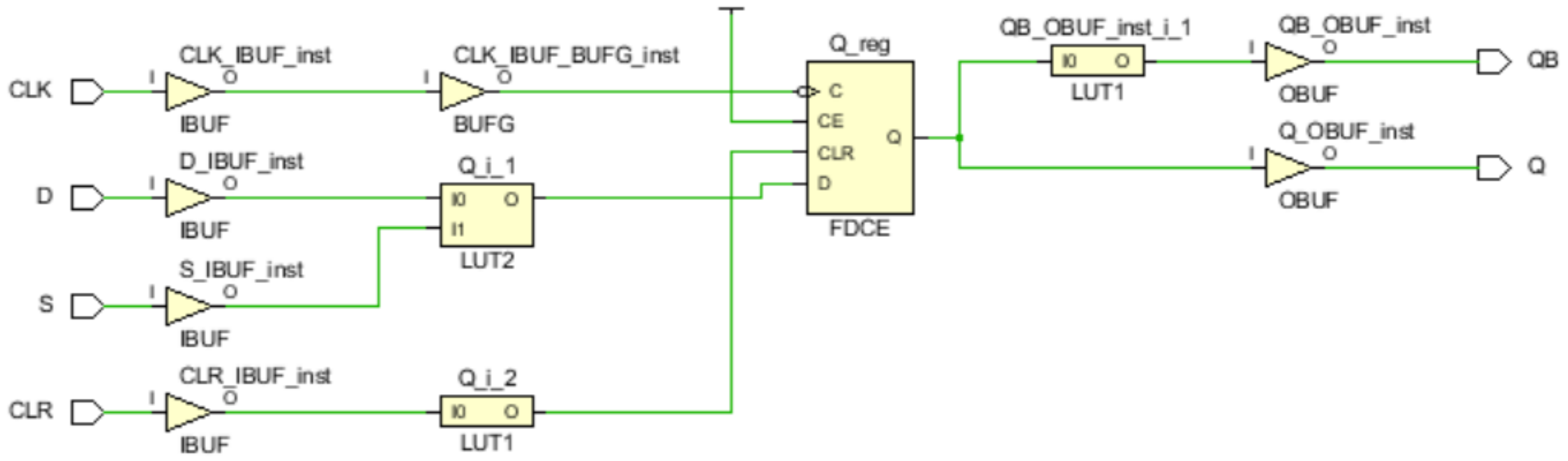
CLK	CLR	S	D	Q^+
X	0	X	X	0
↓	1	0	X	1
↓	1	1	0	0
↓	1	1	1	1

D Flip-Flop… in Vivado?

Vivado RTL Schematic



Vivado Synthesized Schematic



Summary

- Operators (`~`, `*`, `/`, `+`, `-`, `&`, `^`, `|`)
- Continuous assignments (`assign`)
- Procedural assignments (`always`)
  - Blocking assignment (`=`)
  - Non-blocking assignment (`<=`)
- Modeling of multiple D Flip-flops
- Modeling of synchronous and asynchronous inputs