

CSEE 3827: Fundamentals of Computer Systems, Spring 2022

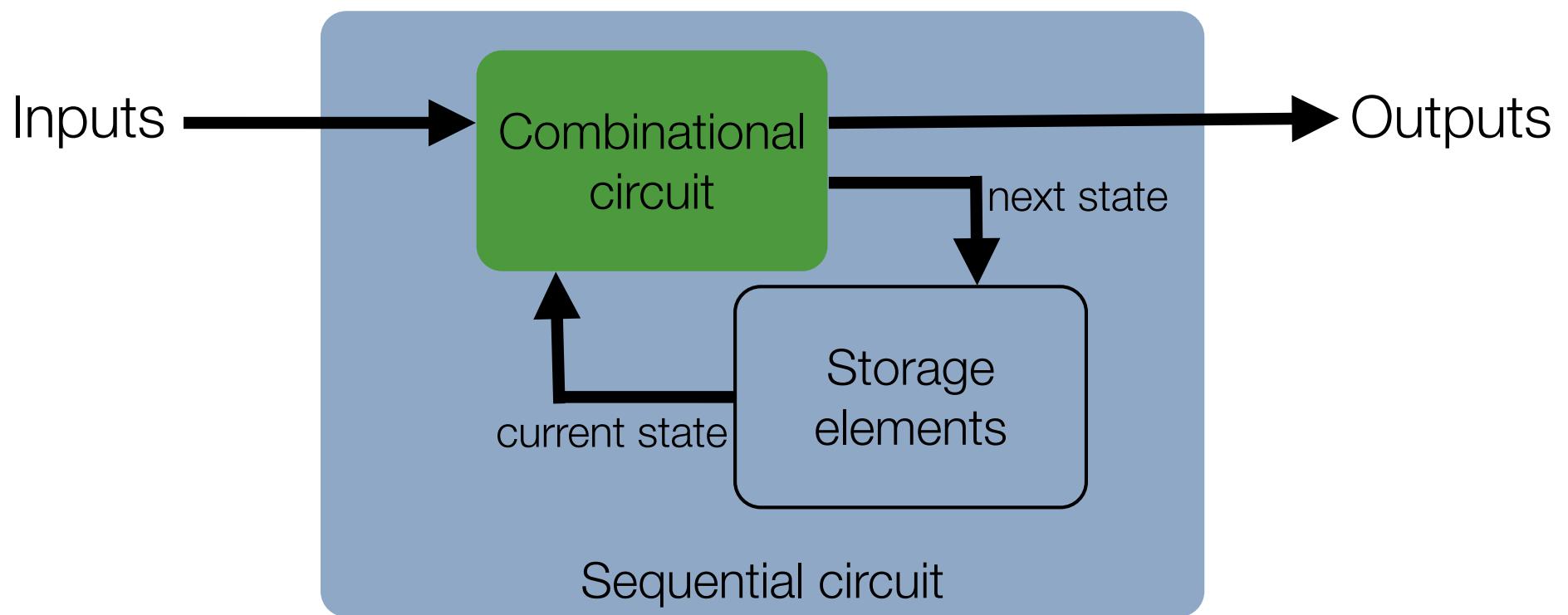
Lecture 5

Prof. Dan Rubenstein (danr@cs.columbia.edu)

Agenda (M&K 5.1-5.3, 5.6)

- Sequential Circuit Design
 - Latches
 - Flip-Flops
 - Timing Issues

Combinational v. sequential logic

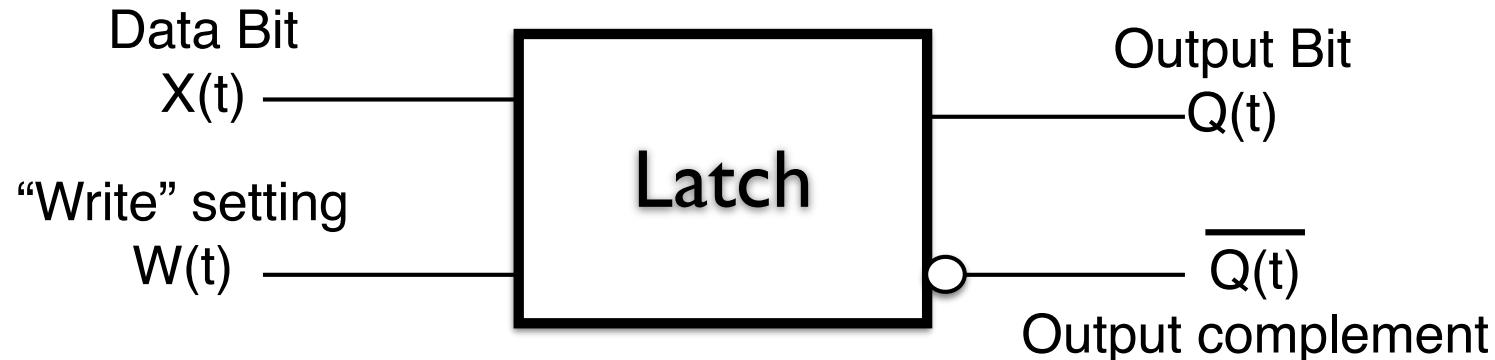


Latches

State of a circuit

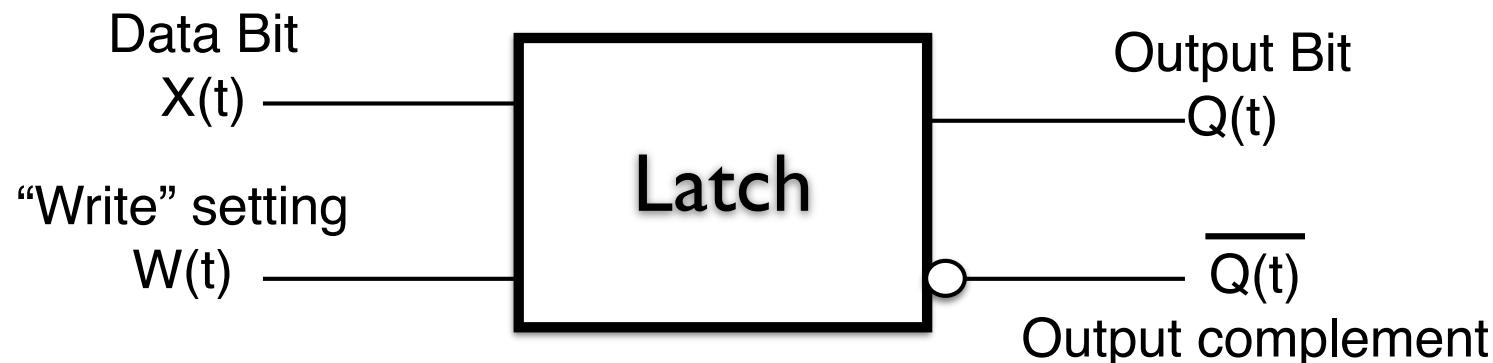
- Q: What is the simplest circuit with state?
- A: Hold a single bit of state where
 - can set the state to a new value (0 or 1)
 - hold the state to the current value

Intuitive Latch Design

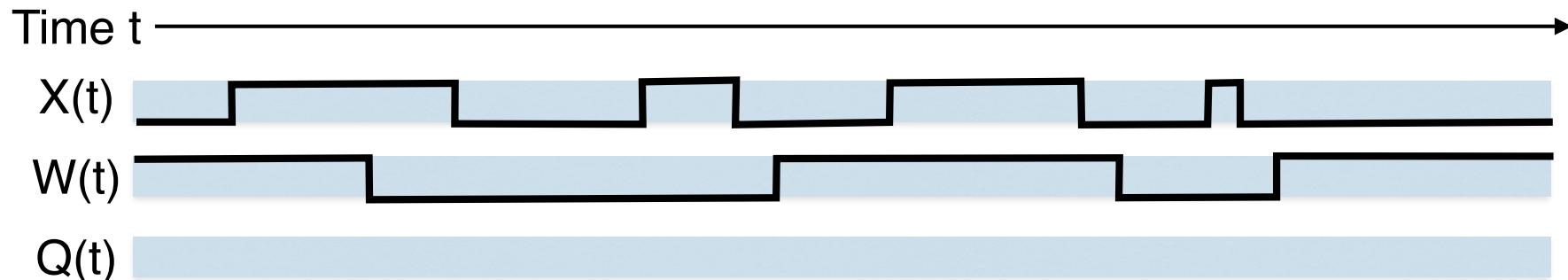


- $W(t) = 1$, write $X(t)$ onto the (1-bit) memory of the latch
- $W(t) = 0$, ignore $X(t)$, hold the latch memory fixed
- $Q(t) = X(s)$ where s is the last time (before t) where $W(s)=0$
 - Latch is **holding** its value when $W=0$

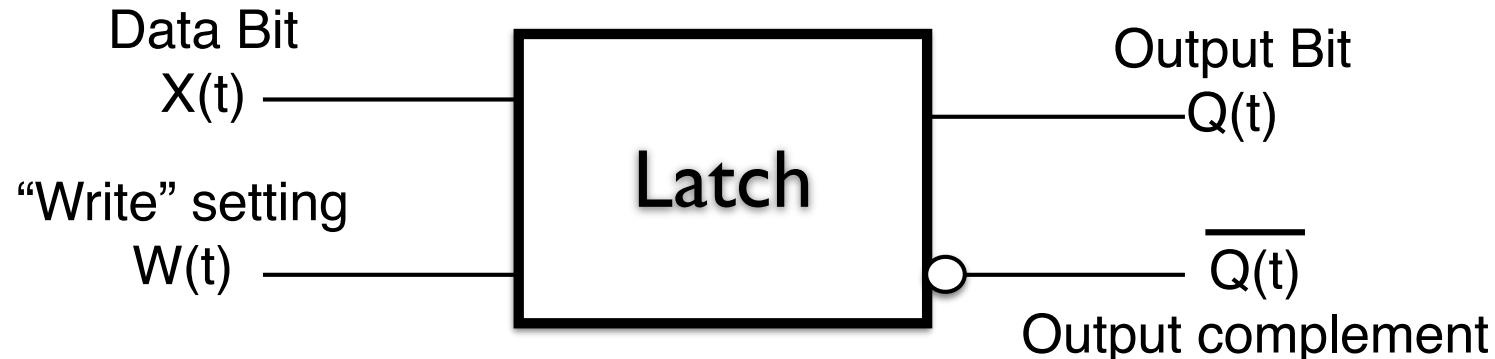
Intuitive Latch Design



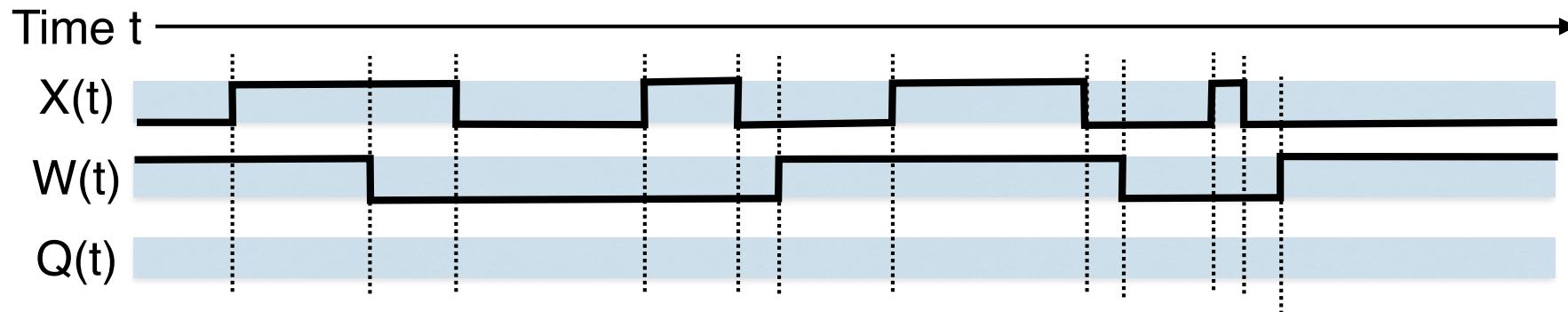
- $W(t) = 1$, write $X(t)$ onto the (1-bit) memory of the latch
- $W(t) = 0$, ignore $X(t)$, hold the latch memory fixed
- $Q(t) = X(s)$ where s is the last time (before t) where $W(s)=0$
 - Latch is **holding** its value when $W=0$



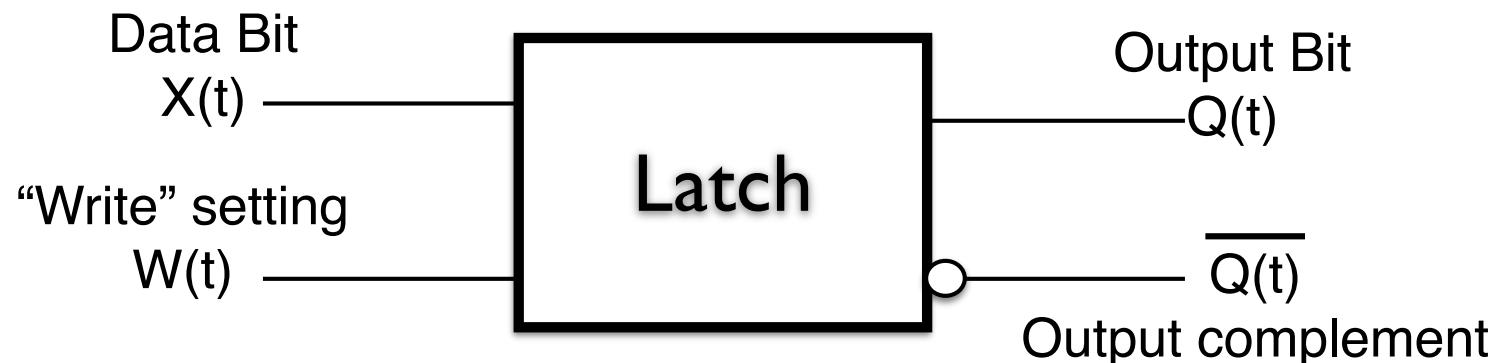
Intuitive Latch Design



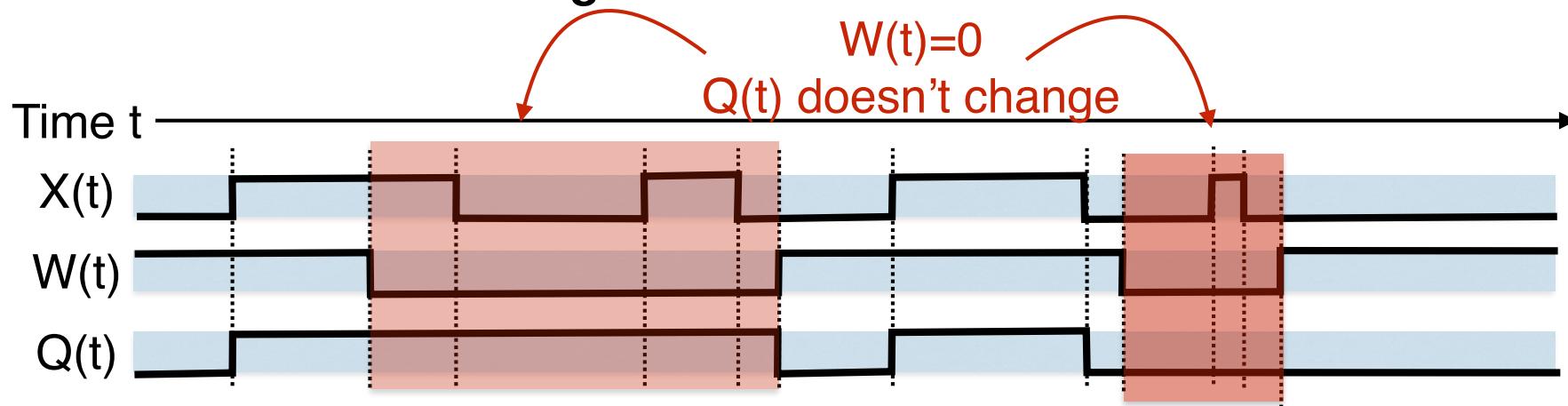
- $W(t) = 1$, write $X(t)$ onto the (1-bit) memory of the latch
- $W(t) = 0$, ignore $X(t)$, hold the latch memory fixed
- $Q(t) = X(s)$ where s is the last time (before t) where $W(s)=0$
 - Latch is **holding** its value when $W=0$



Intuitive Latch Design



- $W(t) = 1$, write $X(t)$ onto the (1-bit) memory of the latch
- $W(t) = 0$, ignore $X(t)$, hold the latch memory fixed
- $Q(t) = X(s)$ where s is the last time (before t) where $W(s)=0$
 - Latch is **holding** its value when $W=0$

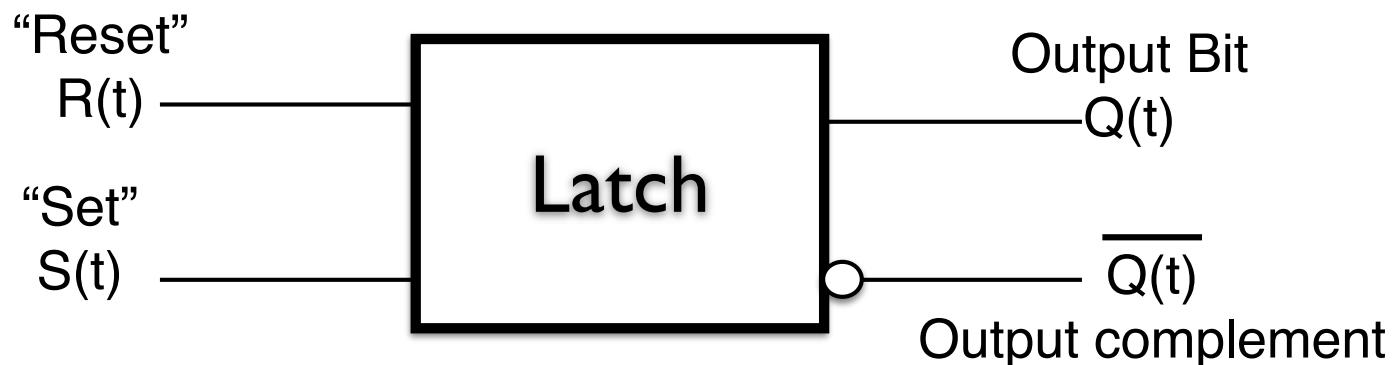


Constructing Latches

- Important points:
 - Can store and hold a bit
 - Can be set to a new value or told to hold current value
 - Inputs to latch might be formulated differently, but same concept

—

1st Latch We build...



- $R=0, S=1$, write 1 into the latch
- $R=1, S=0$, write 0 into the latch
- $R=S=0$, latch “holds” its value
- $R=S=1$: do not use!!!

RECALL: Universal gates: NAND, NOR

x	y	$z = \overline{xy}$
0	0	1
0	1	1
1	0	1
1	1	0



Note: the “o” in a circuit represents a NOT (inverter)

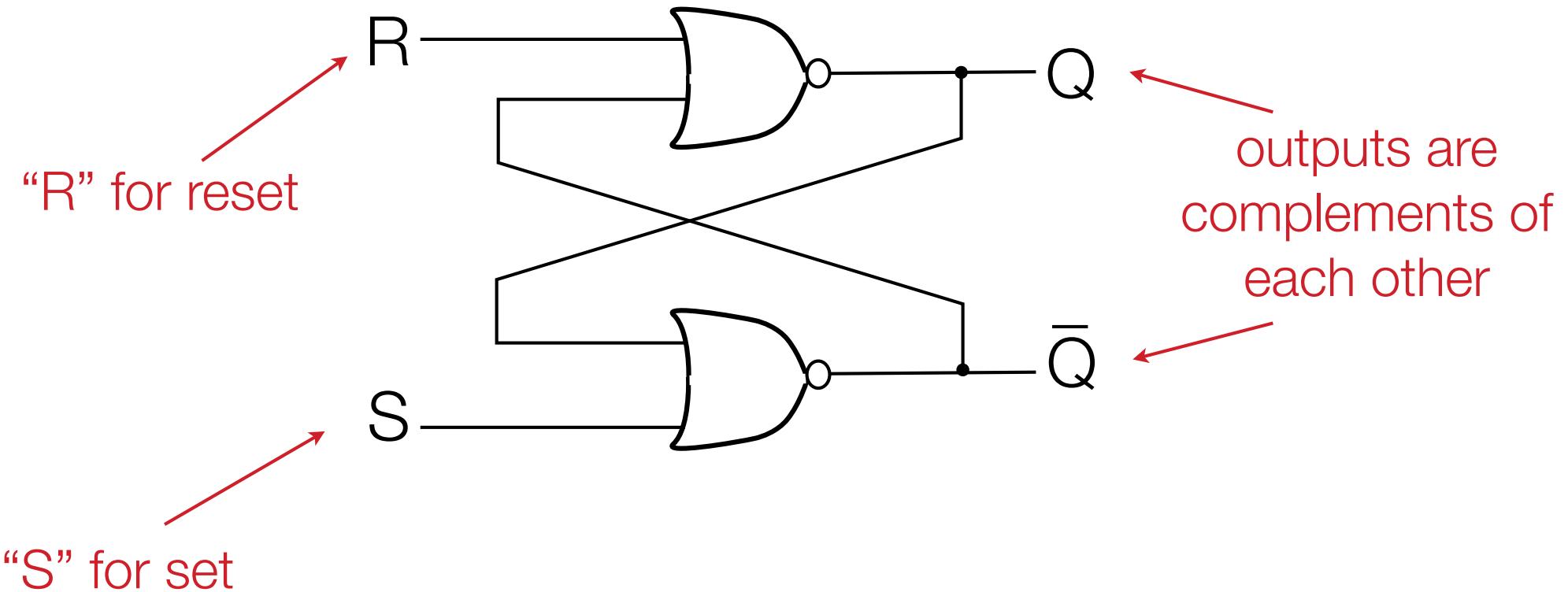
x	y	$z = \overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

Different from “●” which represents wire connector



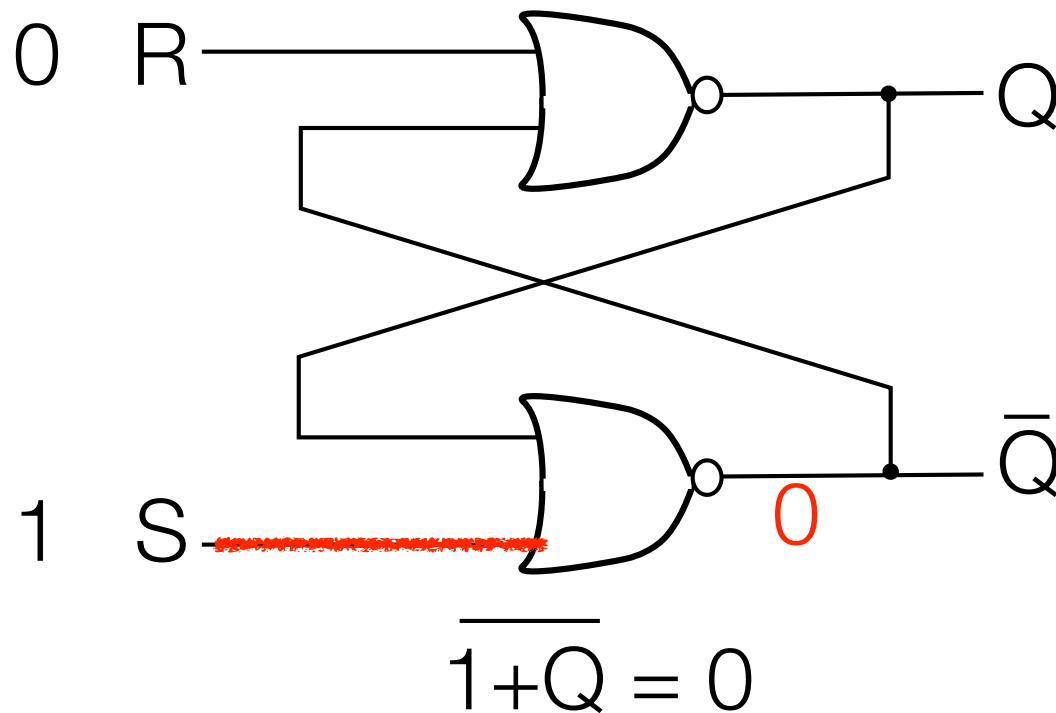
SR latch

- Latch constructed of cross-coupled NOR gates



- What's so new? The wires “loop back” (output feeding back into circuit)
- Note: symmetrical around X axis (swap values for R & S and Q & \bar{Q} swap values)

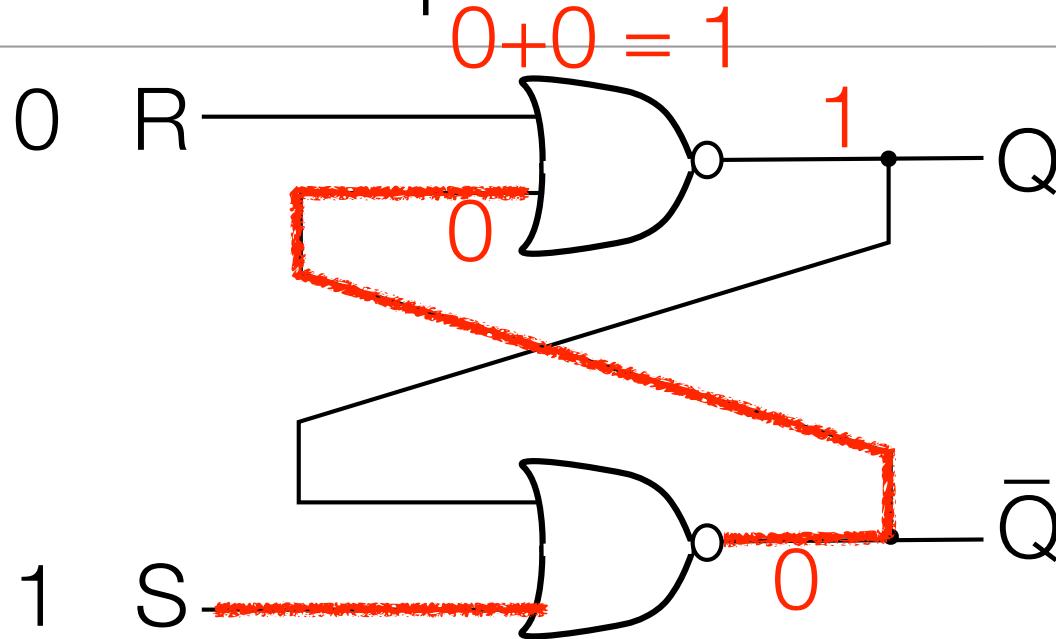
SR latch: set example



R	S	Q	\bar{Q}
0	0		
0	1		
1	0		
1	1		

NOR		
x	y	$z = \overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

SR latch: set example



$$\overline{1+Q} = 0$$

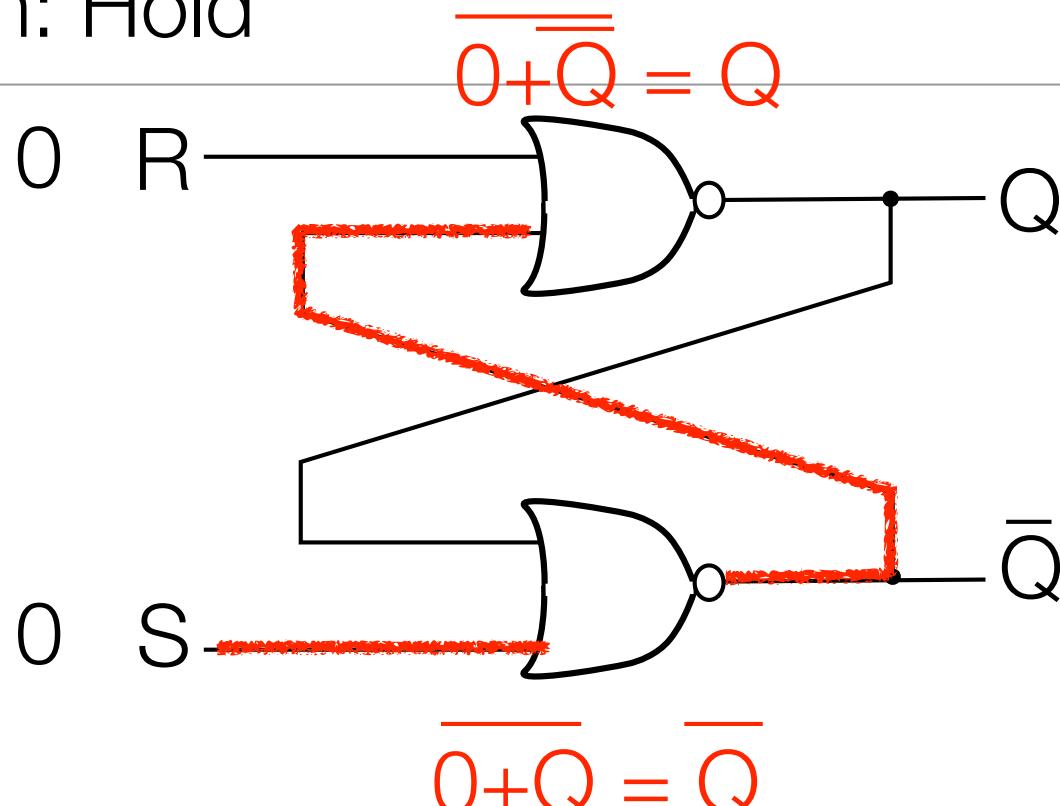
R	S	Q	\bar{Q}
0	0		
0	1	1	0
1	0	0	1
1	1		

By symmetry along
horizontal cut of latch

NOR

x	y	$z = \overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

SR latch: Hold



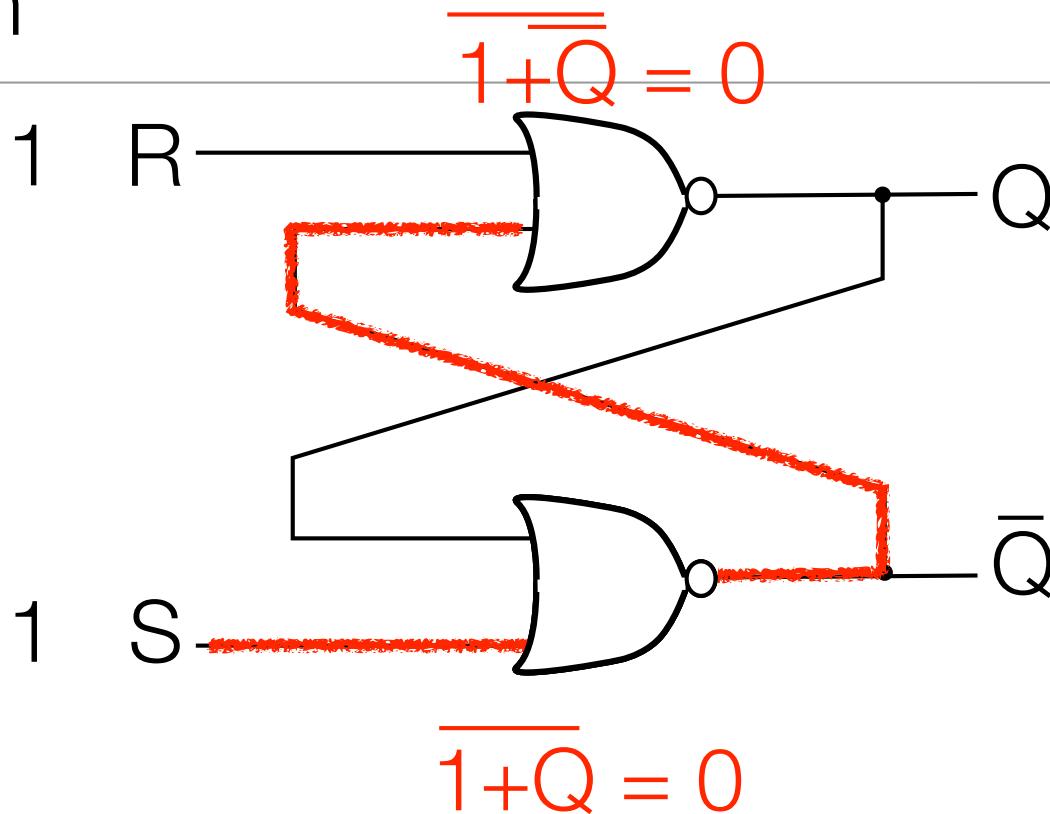
R	S	Q	\bar{Q}
0	0	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1		

Hold previous value

NOR

x	y	$z = \overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

SR latch



R	S	Q	\overline{Q}
0	0	Q	\overline{Q}
0	1	1	0
1	0	0	1
1	1	0	0

No change

Set ($Q=1$)

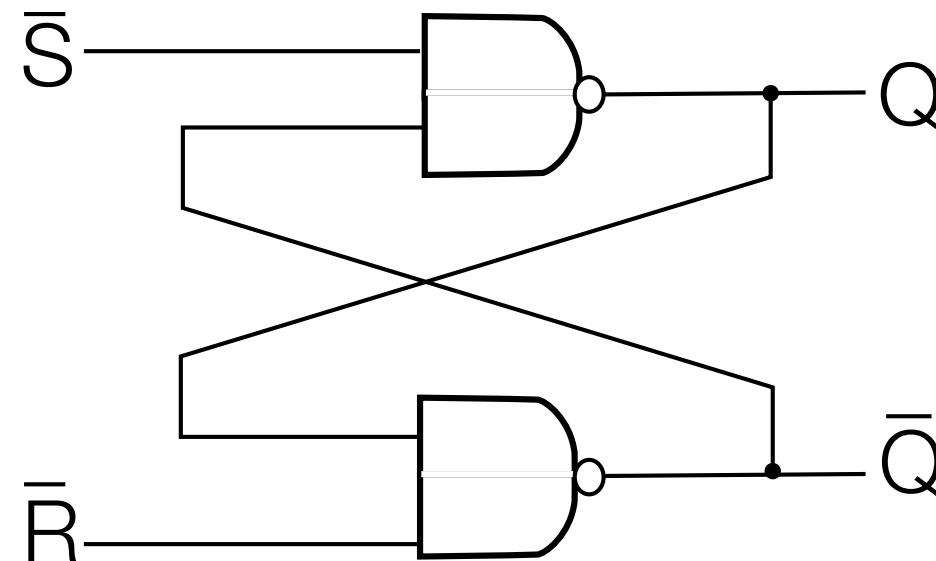
Reset ($Q=0$)

Bad - do not use!

Other Types of Latches

$\bar{S}\bar{R}$ latch

- Latch constructed of cross-coupled **NAND** gates



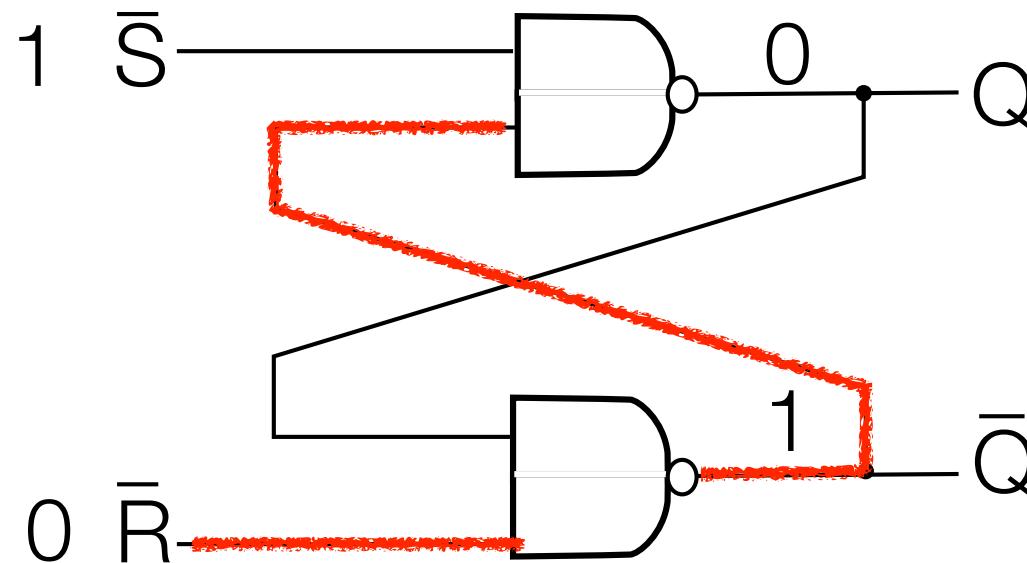
\bar{R}	\bar{S}	Q	\bar{Q}
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	\bar{Q}

\bar{R} and \bar{S} have reverse behavior of SR latch,
so R and S have same behaviors

Bad - do not use!

$\bar{S}\bar{R}$ latch

- Latch constructed of cross-coupled **NAND** gates



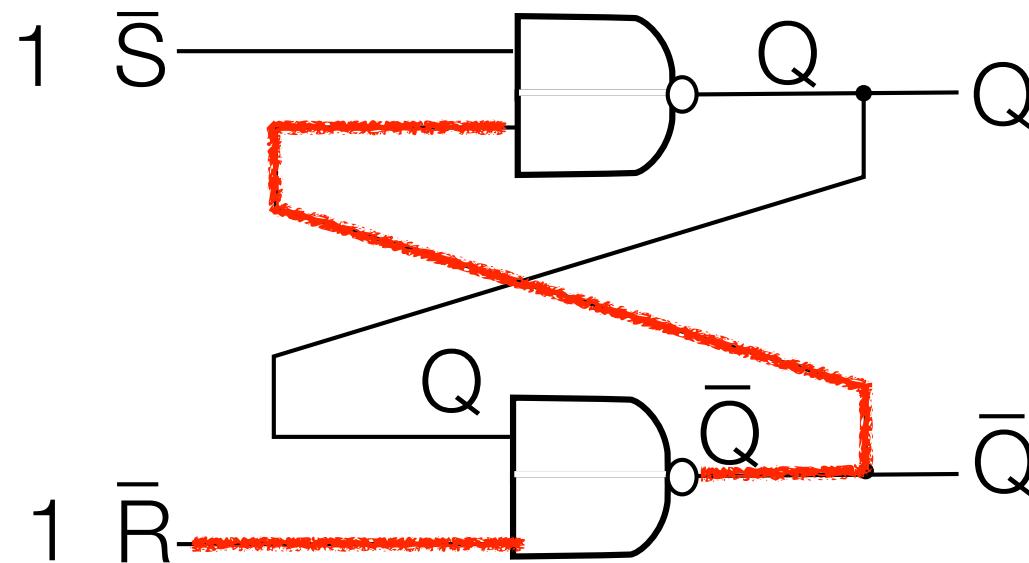
\bar{R}	\bar{S}	Q	\bar{Q}
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	\bar{Q}

\bar{R} and \bar{S} have reverse behavior of SR latch,
so R and S have same behaviors

Bad - do not use!

$\bar{S}\bar{R}$ latch

- Latch constructed of cross-coupled **NAND** gates

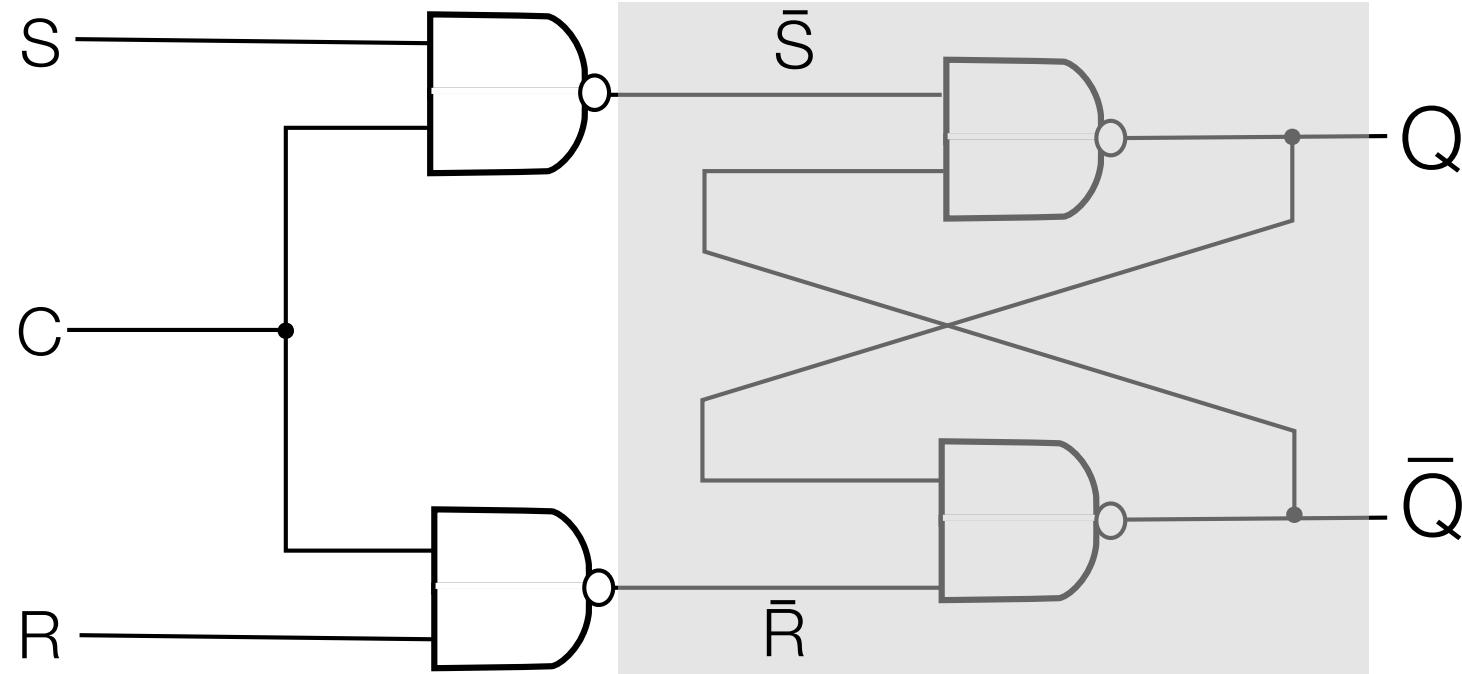


\bar{R}	\bar{S}	Q	\bar{Q}
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	\bar{Q}

\bar{R} and \bar{S} have reverse behavior of SR latch,
so R and S have same behaviors

Bad - do not use!

SR Latch with control

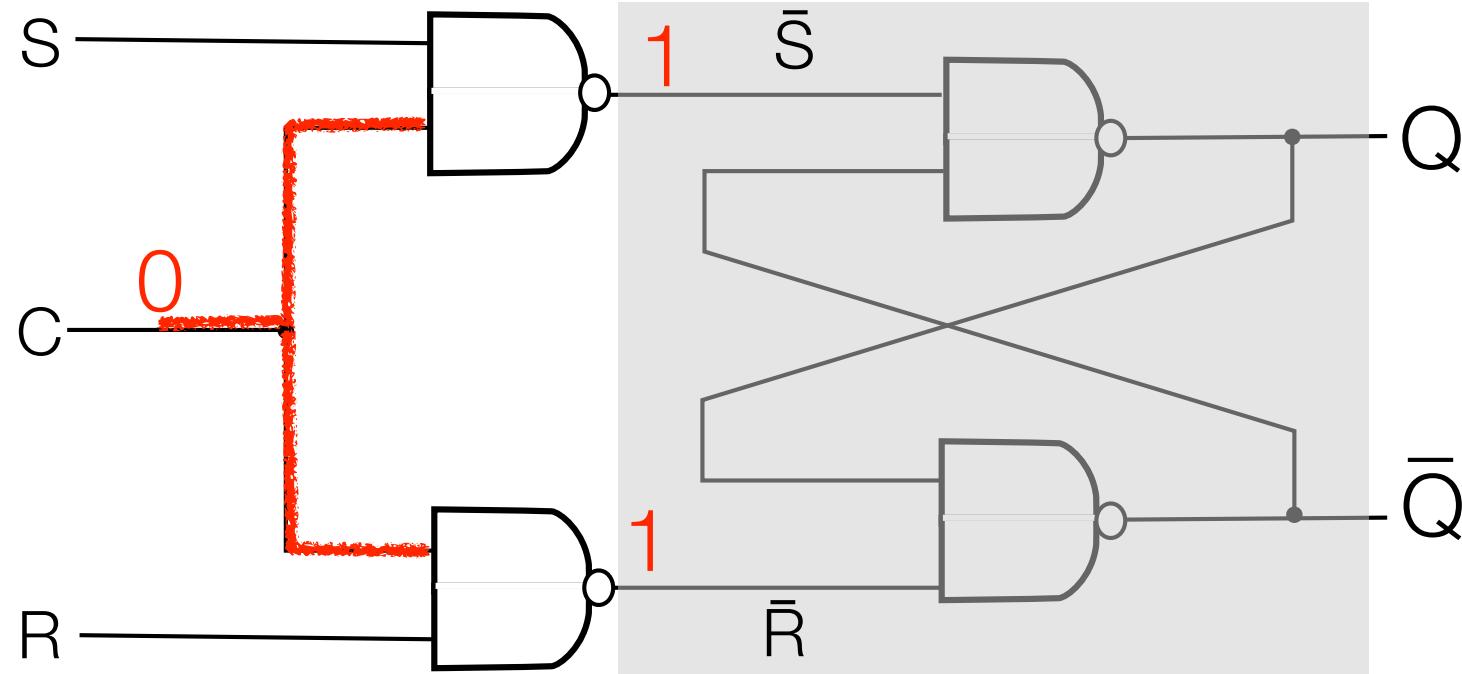


$\bar{S}\bar{R}$ latch

- $C=0$, $\bar{S}\bar{R}$ latch receives $\bar{S}=1$, $\bar{R}=1$, values “hold”
- $C=1$, first set of NAND gates invert S & R inputs to \bar{S} & \bar{R}

C	R	S	Q	\bar{Q}
0	X	X	Q	\bar{Q}
1	0	0	Q	\bar{Q}
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

SR Latch with control

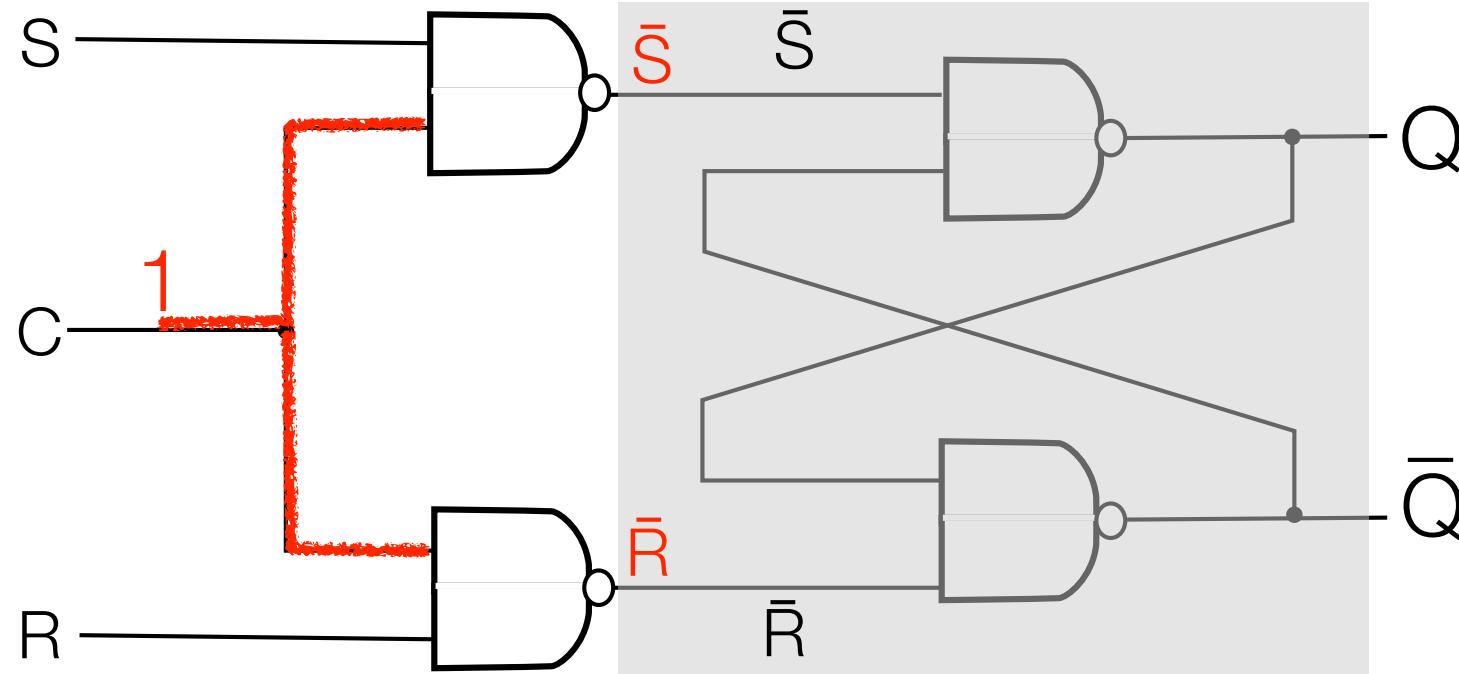


$\bar{S}\bar{R}$ latch

- $C=0$, $\bar{S}\bar{R}$ latch receives $\bar{S}=1$, $\bar{R}=1$, values “hold”
- $C=1$, first set of NAND gates invert S & R inputs to \bar{S} & \bar{R}

C	R	S	Q	\bar{Q}
0	X	X	Q	\bar{Q}
1	0	0	Q	\bar{Q}
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

SR Latch with control

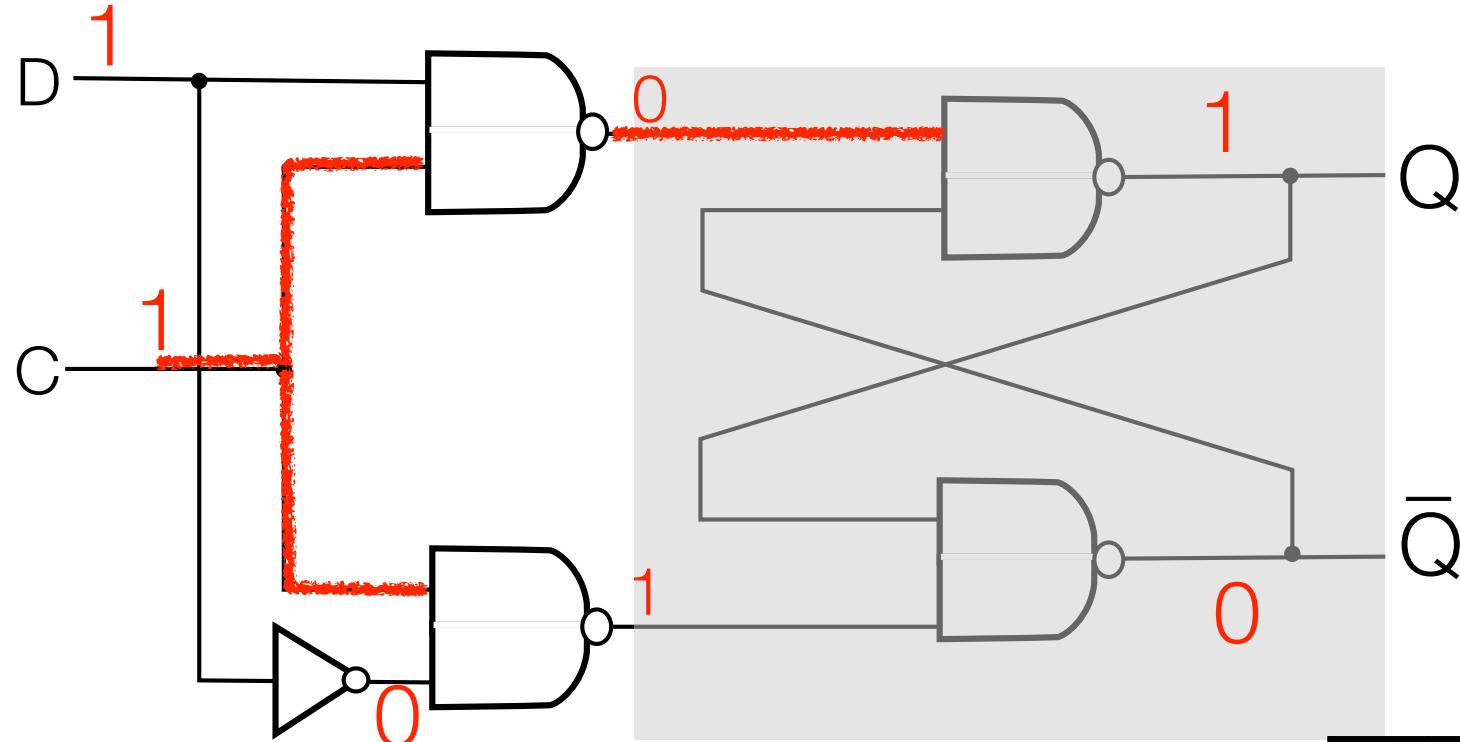


$\bar{S}\bar{R}$ latch

- $C=0$, $\bar{S}\bar{R}$ latch receives $\bar{S}=1$, $\bar{R}=1$, values “hold”
- $C=1$, first set of NAND gates invert S & R inputs to \bar{S} & \bar{R}

C	R	S	Q	\bar{Q}
0	X	X	Q	\bar{Q}
1	0	0	Q	\bar{Q}
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

D Latch with control (our “intuitive latch” with data & write)

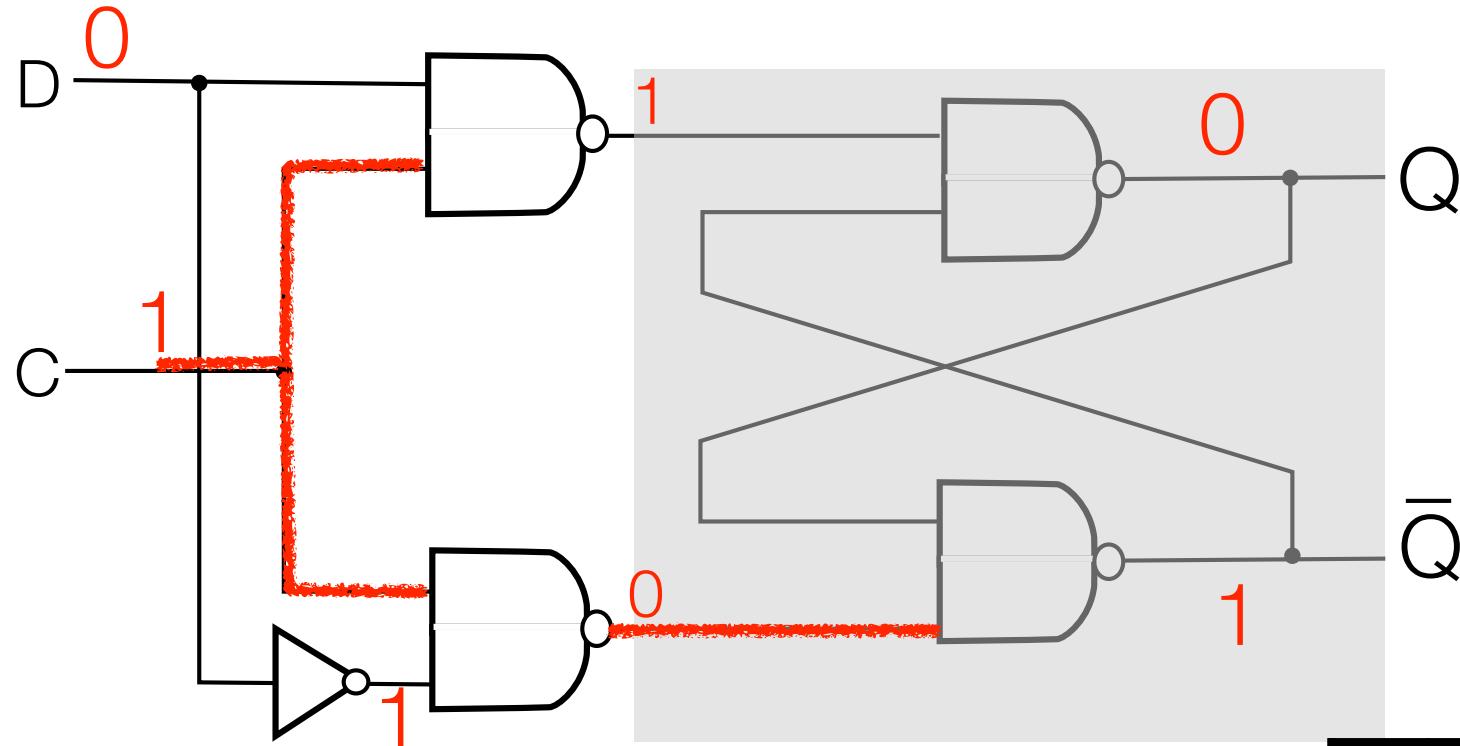


$\bar{S}\bar{R}$ latch

- With the control (C), no reason to ever have $S=R$
- $C=0$, latch holds value, $C=1$, $Q=D$

C	D	Q	\bar{Q}
0	X	Q	\bar{Q}
1	0	0	1
1	1	1	0

D Latch with control (our “intuitive latch” with data & write)

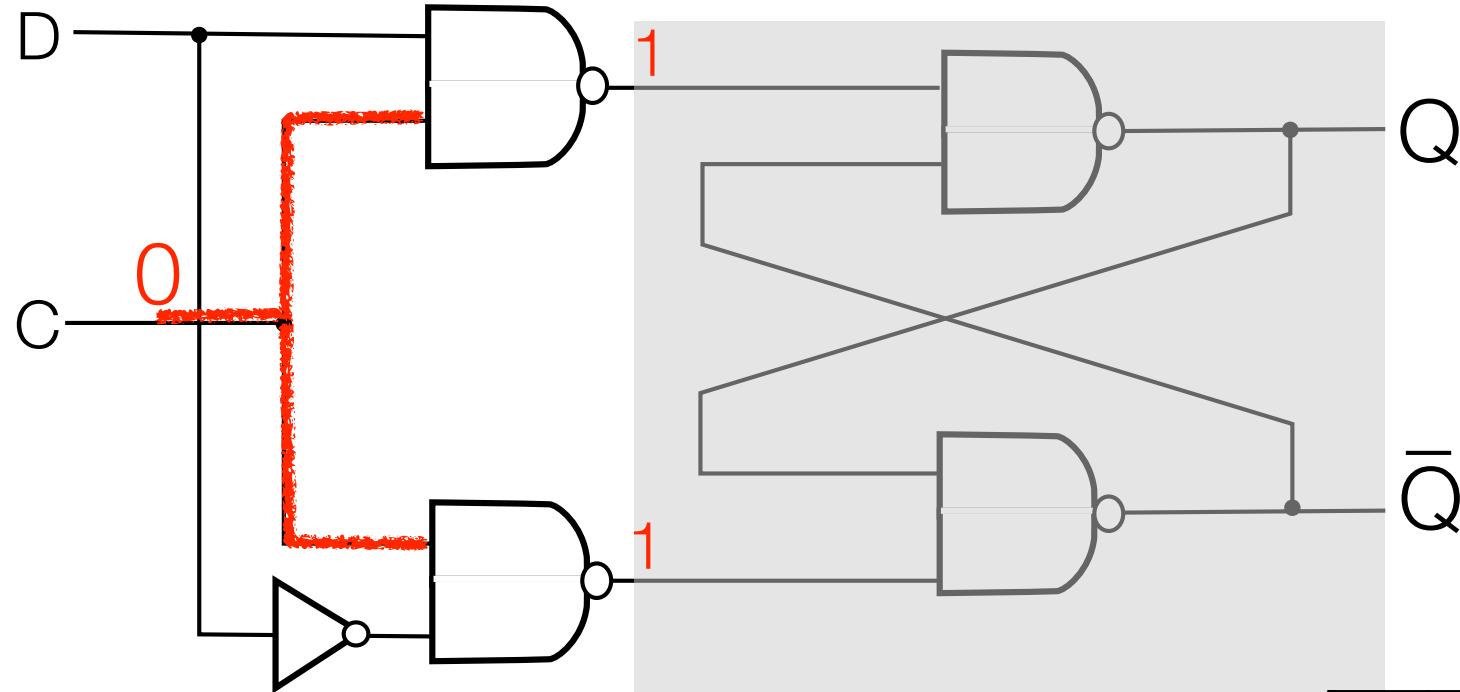


$\bar{S}\bar{R}$ latch

- With the control (C), no reason to ever have S=R
- C=0, latch holds value, C=1, Q=D

C	D	Q	\bar{Q}
0	X	Q	\bar{Q}
1	0	0	1
1	1	1	0

D Latch with control (our “intuitive latch” with data & write)



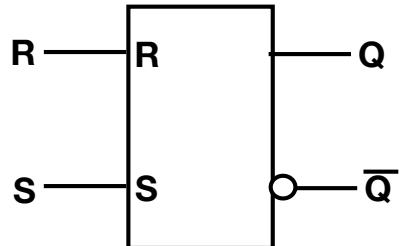
$\bar{S}\bar{R}$ latch

- With the control (C), no reason to ever have S=R
- C=0, latch holds value, C=1, Q=D

C	D	Q	\bar{Q}
0	X	Q	\bar{Q}
1	0	0	1
1	1	1	0

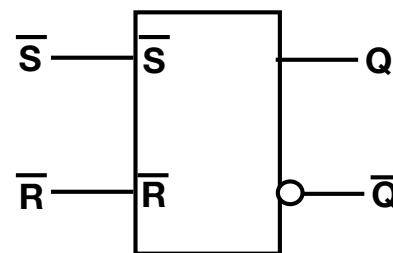
Circuit diagrams for latches

SR Latch



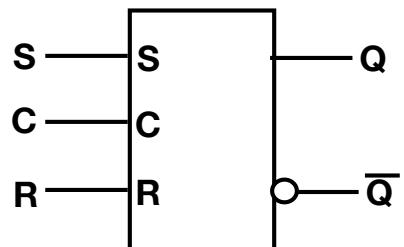
R	S	Q	\bar{Q}
0	0	Q	\bar{Q}
0	1	1	0
1	0	0	1
1	1	0	0

$\bar{S}\bar{R}$ Latch



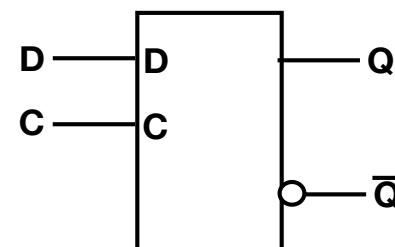
\bar{S}	\bar{R}	Q	\bar{Q}
0	0	0	0
0	1	1	0
1	0	0	1
1	1	Q	\bar{Q}

SR Latch with control



C	R	S	Q	\bar{Q}
0	X	X	Q	\bar{Q}
1	0	0	Q	\bar{Q}
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

D Latch with control



C	D	Q	\bar{Q}
0	X	Q	\bar{Q}
1	0	0	1
1	1	1	0

**Problem with Latches:
No clocking**

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0				
1				
2				
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1				
2				
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1	X			
2				
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1	x	$y =$ $f(x,0)$		
2				
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1	x	$y = f(x,0)$	$z = g(x,0)$	
2				
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1	x	$y = f(x,0)$	$z = g(x,0)$	$h(y,z)$
2				
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

```
x=0; y=0; z=0;
```

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1	x	$y = f(x,0)$	$z = g(x,0)$	$h(y,z)$
2	x	$y = f(x,y)$	$z = g(x,z)$	$h(y,z)$
3				

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

```
while (){  
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)  
}
```

iter #	x	y	z	output
0	0	0	0	?
1	x	$y = f(x,0)$	$z = g(x,0)$	$h(y,z)$
2	x	$y = f(x,y)$	$z = g(x,z)$	$h(y,z)$
3	x	$y = f(x,y)$	$z = g(x,z)$	$h(y,z)$

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

while (){

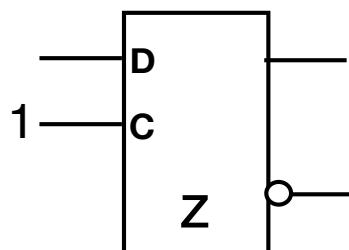
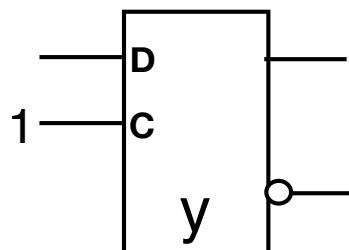
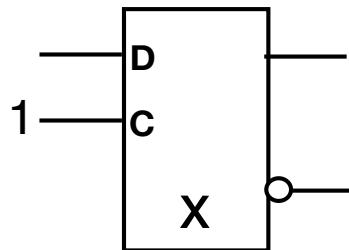
 input x;

 y = f(x,y)

 z = g(x,z)

 output h(y,z)

}



Use 3 D-latches to store the 3 bits of state (x,y,z)

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

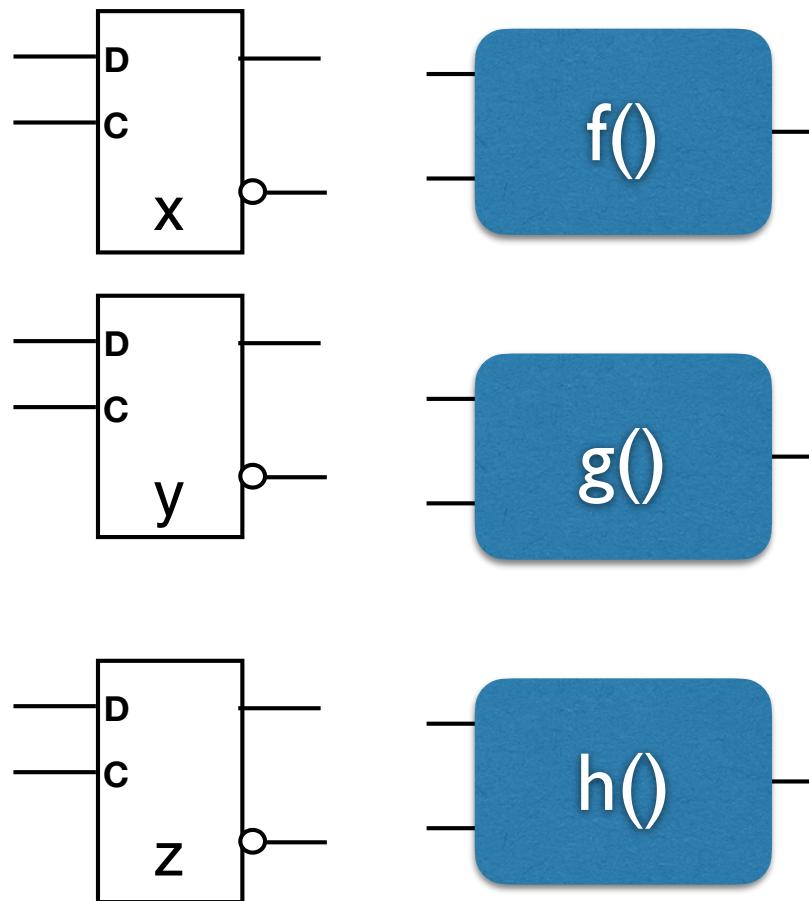
$x=0; y=0; z=0;$

while (){

```
    input x;  
    y = f(x,y)  
    z = g(x,z)  
    output h(y,z)
```

}

Combinatorial circuits compute f , g , and h



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

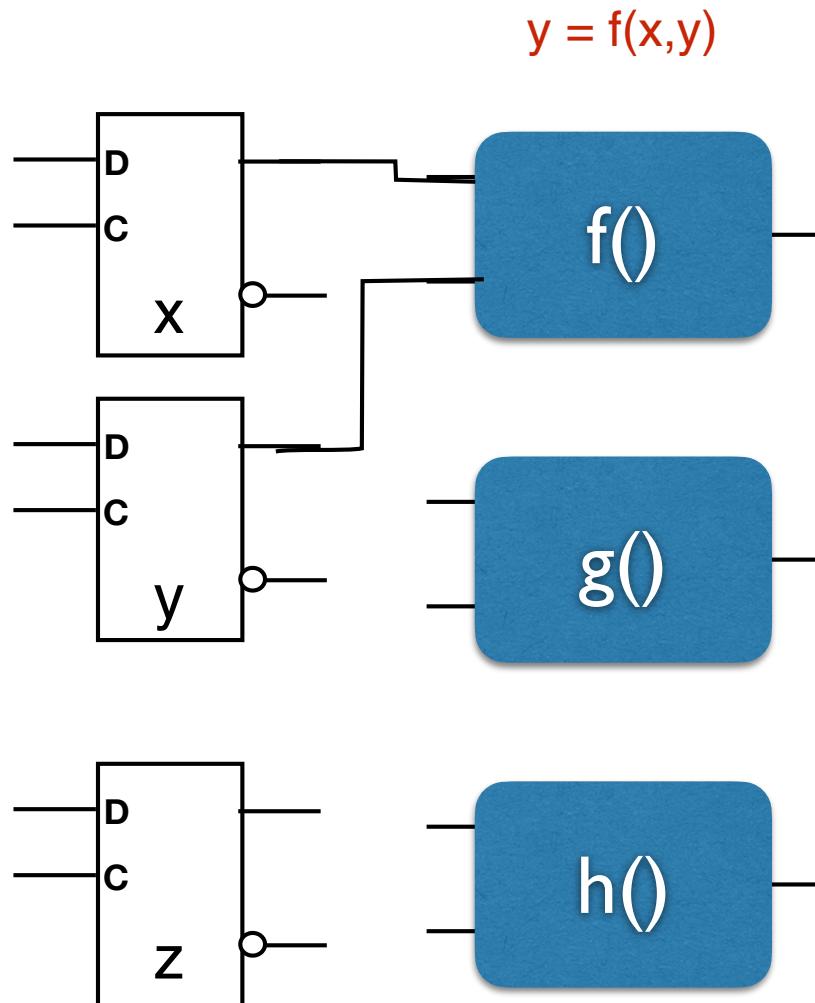
while (){

$y = f(x,y)$

$z = g(x,z)$

 output $h(y,z)$

}



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

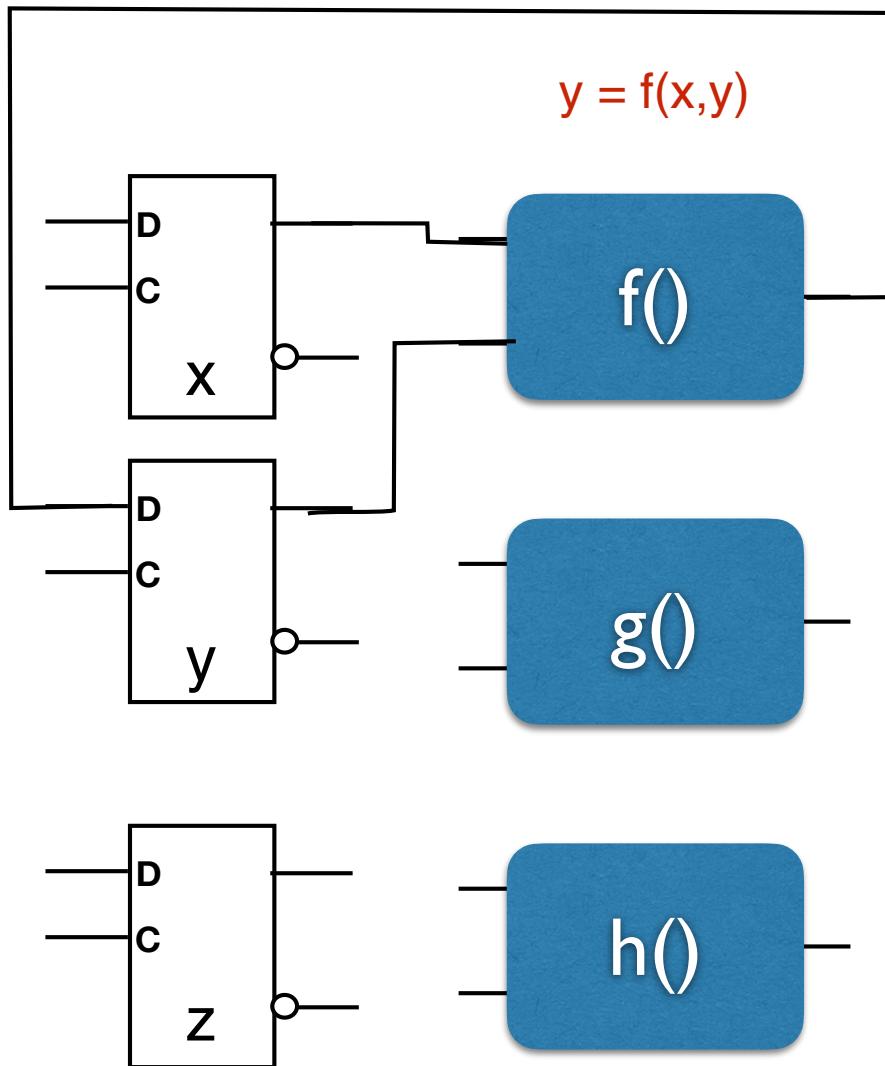
while (){

$y = f(x,y)$

$z = g(x,z)$

 output $h(y,z)$

}



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

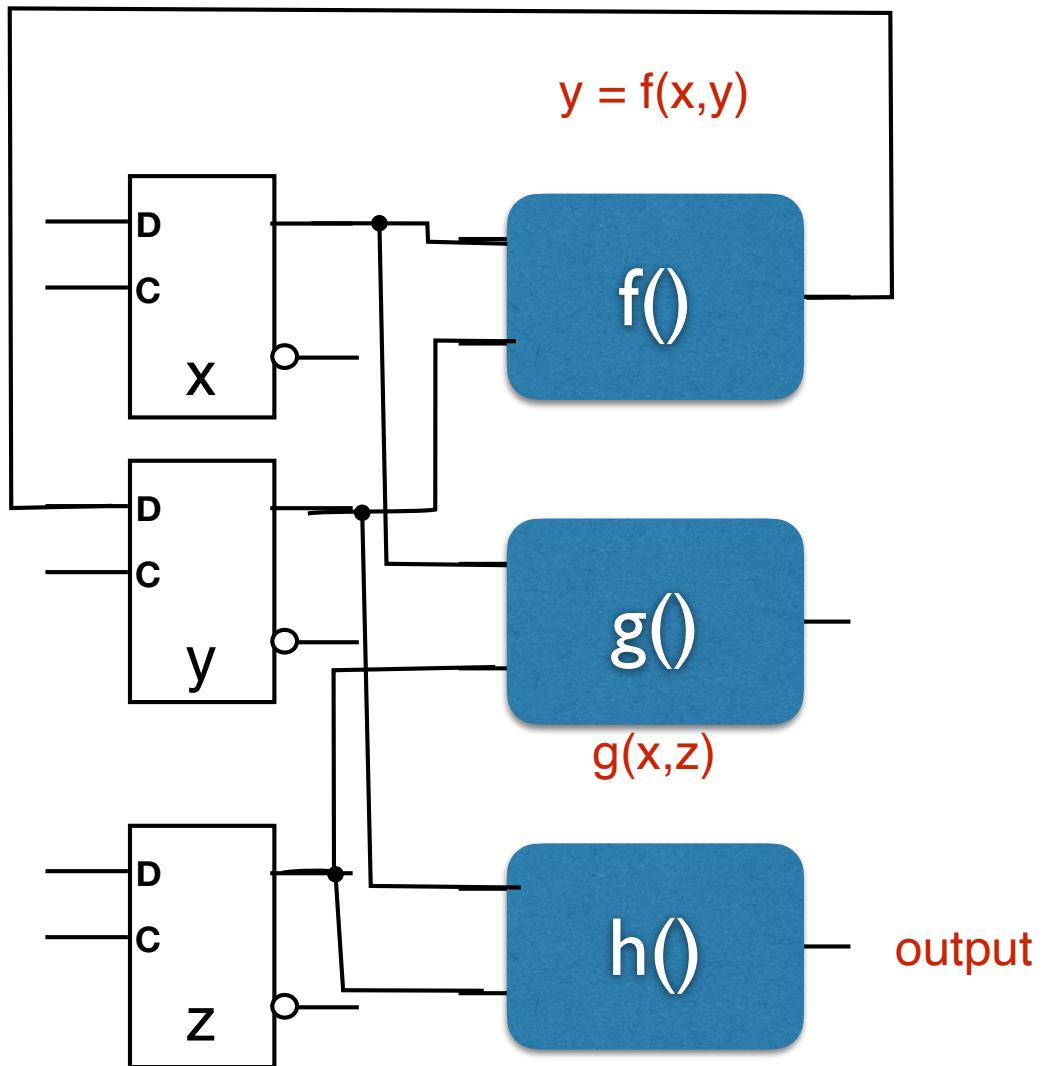
while (){

$y = f(x,y)$

$z = g(x,z)$

 output $h(y,z)$

}



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

while (){

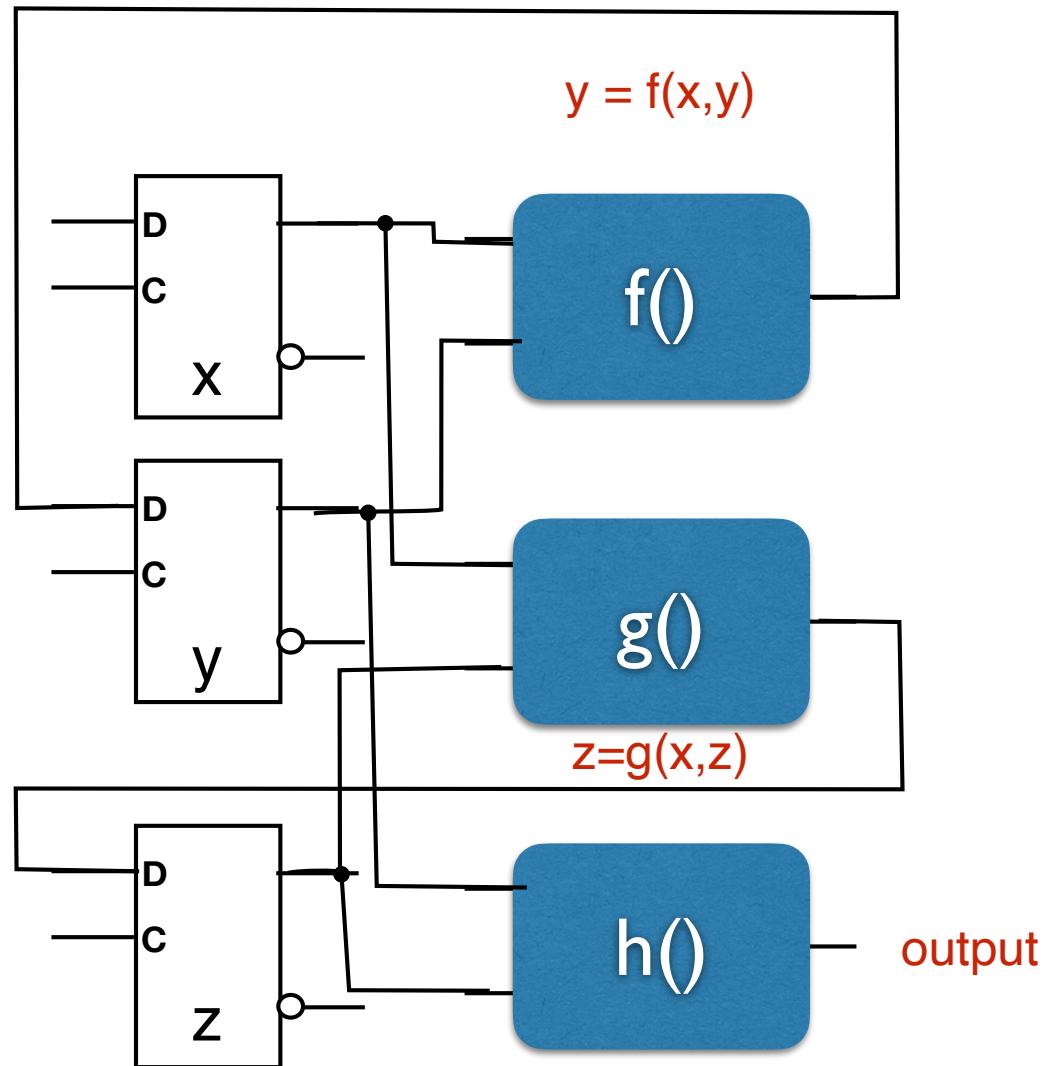
 input x;

$y = f(x,y)$

$z = g(x,z)$

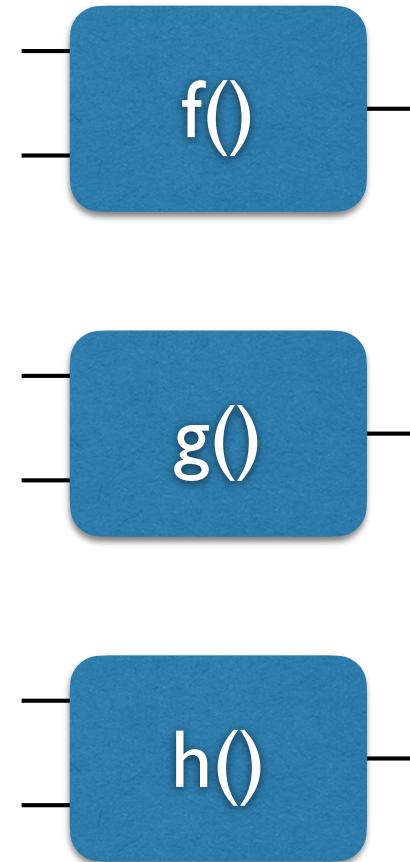
 output h(y,z)

}



Suppose f , g , h have differing timing requirements...

- Suppose:
 - $f()$ computed by circuit with very large depth: very very slow
 - $g()$ and $h()$ low-depth, much much faster...



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

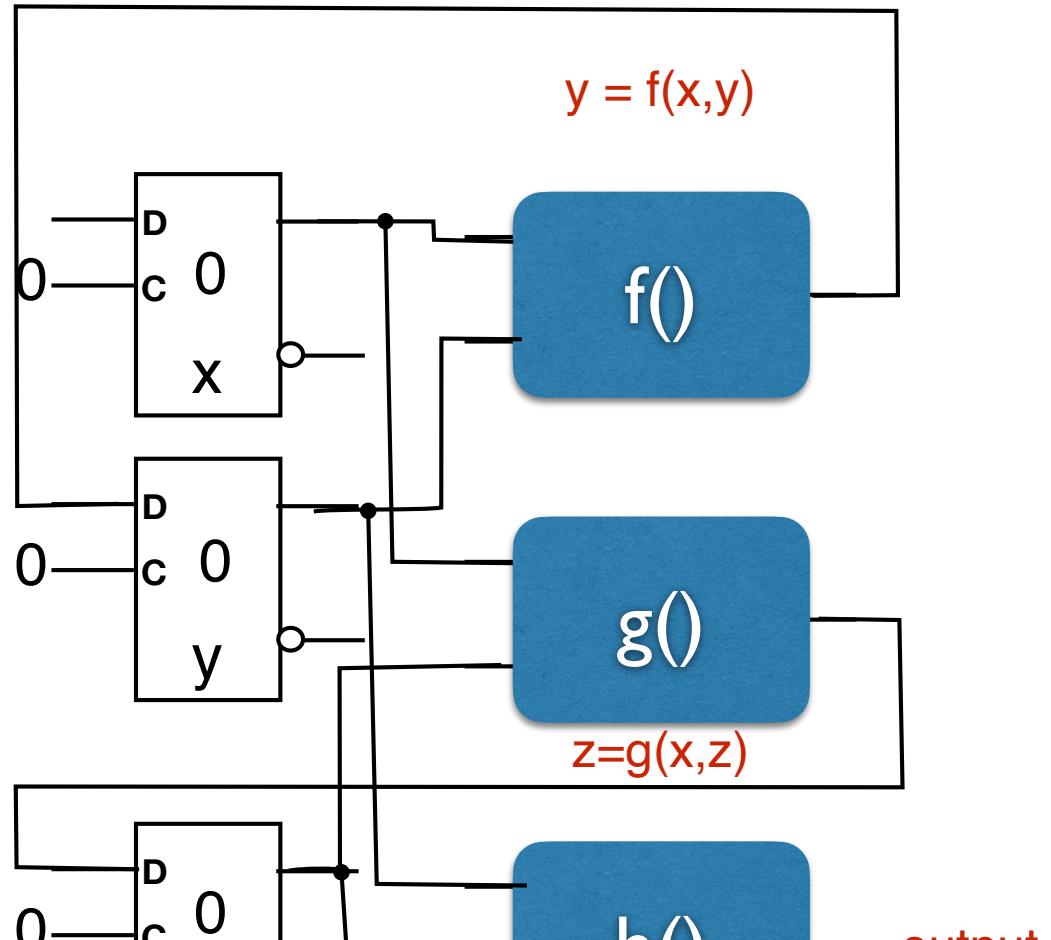
$x=0; y=0; z=0;$

while (){

 input x;
 $y = f(x,y)$
 $z = g(x,z)$
 output h(y,z)

}

Latches “on hold”: holding at iteration 0



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

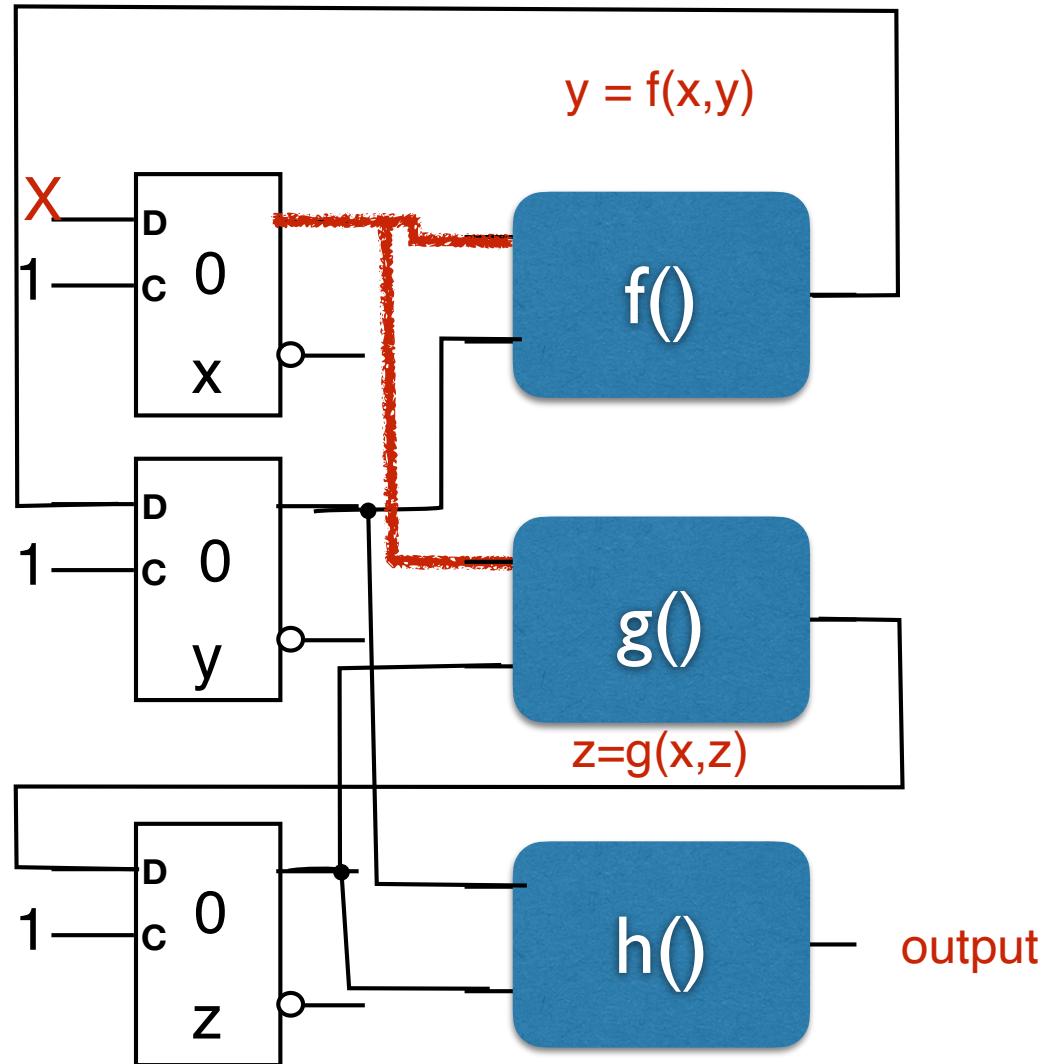
$x=0; y=0; z=0;$

while (){

 input x;
 $y = f(x,y)$
 $z = g(x,z)$
 output h(y,z)

}

Go!!!



Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

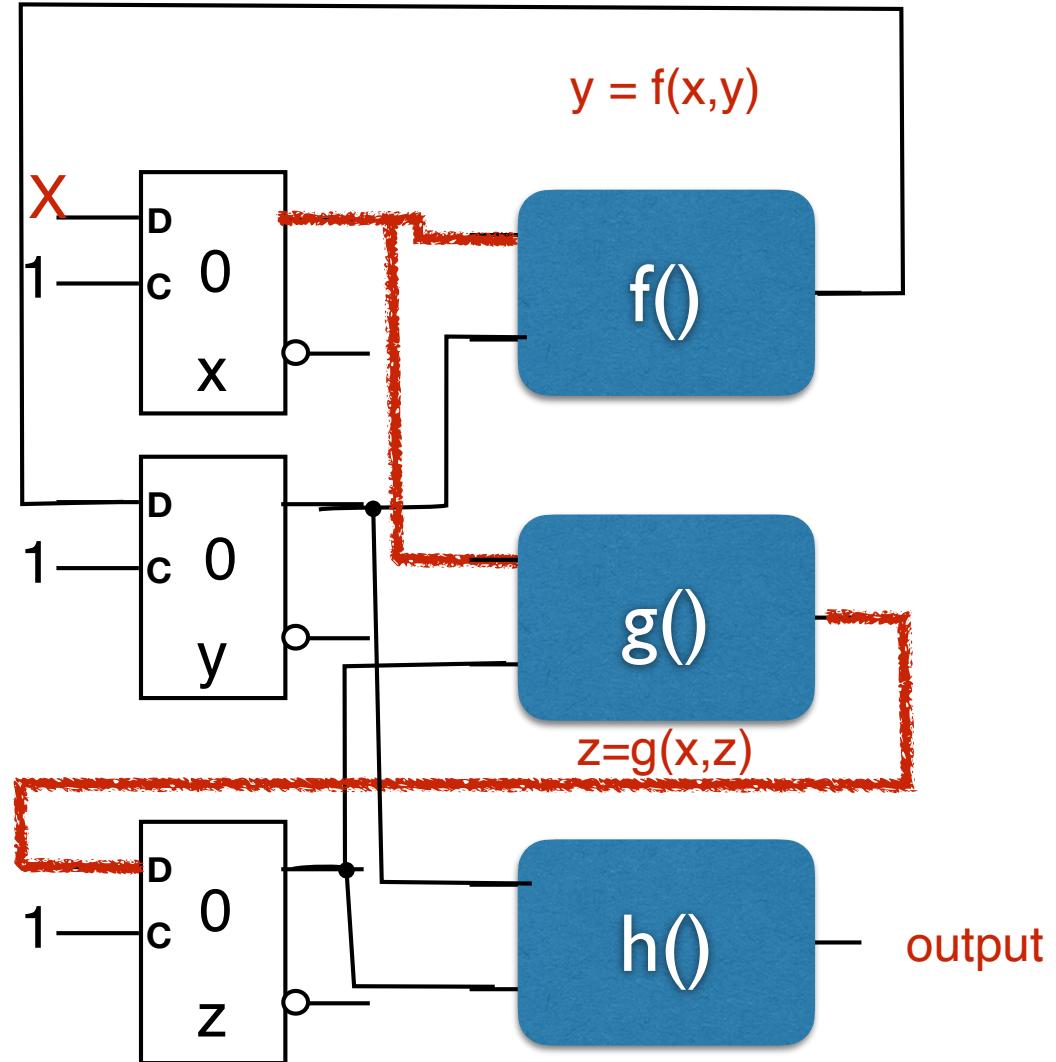
$x=0; y=0; z=0;$

while (){

 input x;
 $y = f(x,y)$
 $z = g(x,z)$
 output h(y,z)

}

moments later... $g()$ ready, $f()$ not done, $h()$ awaiting “red” z



Where we are, where we are headed

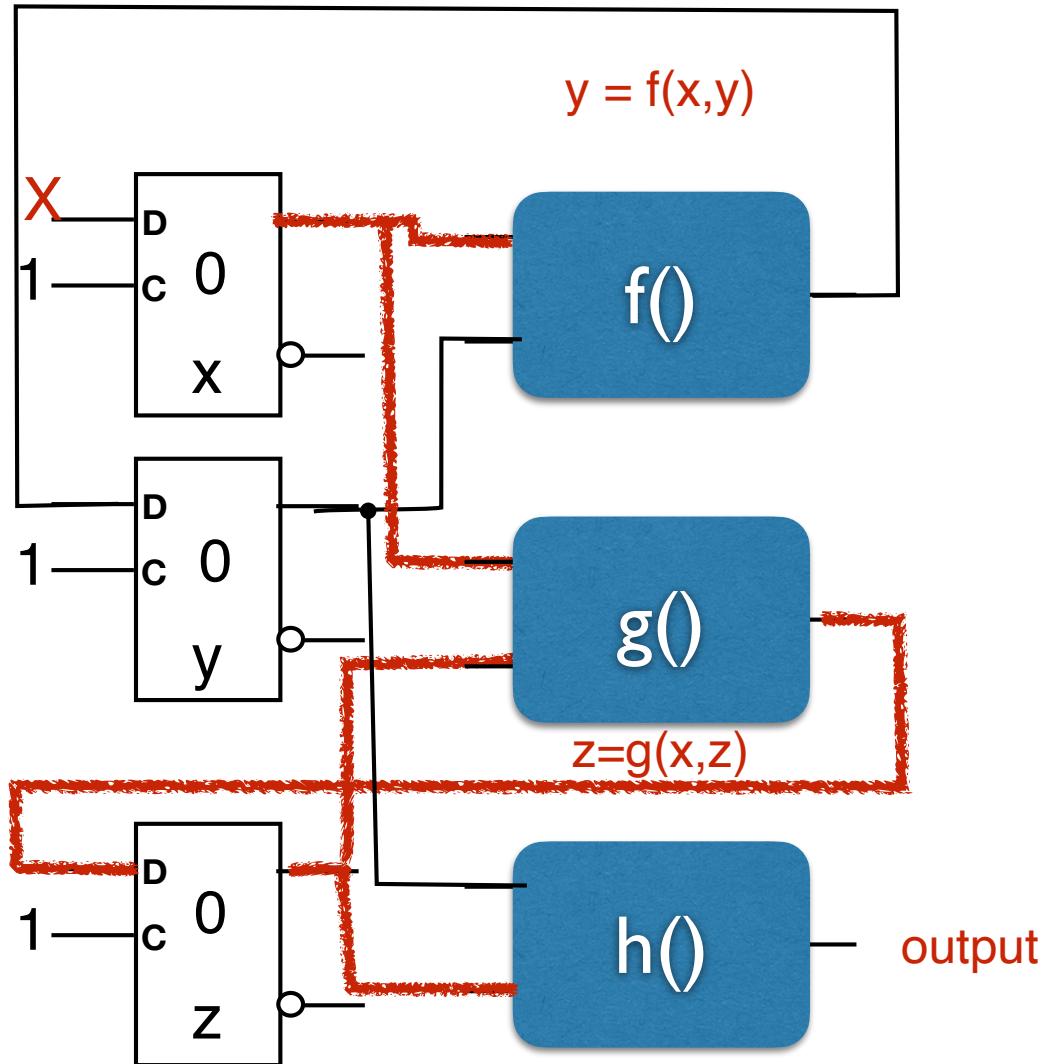
- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

while (){

 input x;
 $y = f(x,y)$
 $z = g(x,z)$
 output h(y,z)

}



z now “red”, but output still not correct:
needs “red” y : $f()$ still computing

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

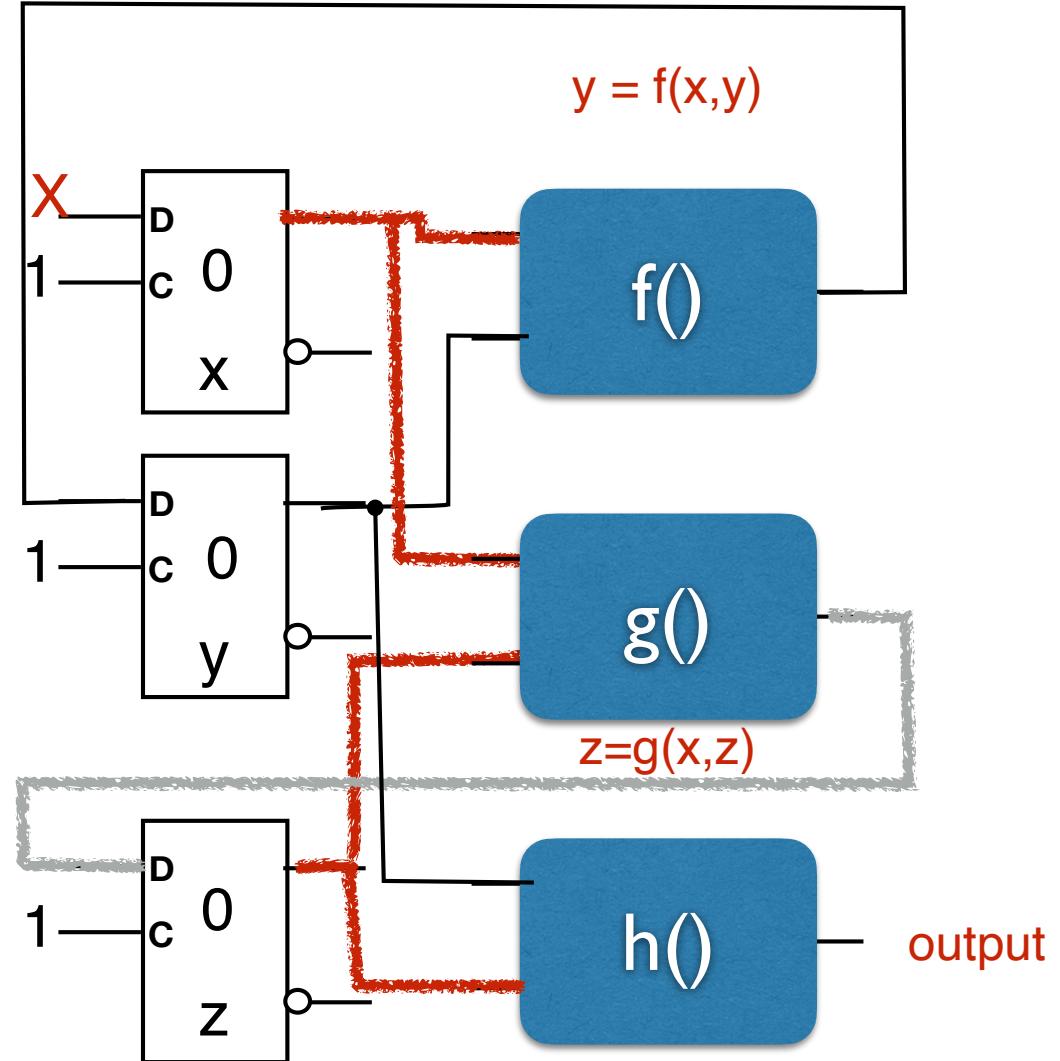
$x=0; y=0; z=0;$

while (){

 input x;
 $y = f(x,y)$
 $z = g(x,z)$
 output h(y,z)

}

Output still not
correct: needs (red) y



g circuit being fed red x, red
z: not supposed to happen!

Where we are, where we are headed

iter #	x	y	z	output
0	0	0	0	?
1	x	$y = f(x, 0)$	$z = g(x, 0)$	$h(y, z)$
2	x	$y = f(x, y)$	$z = g(x, z)$	$h(y, z)$
3				

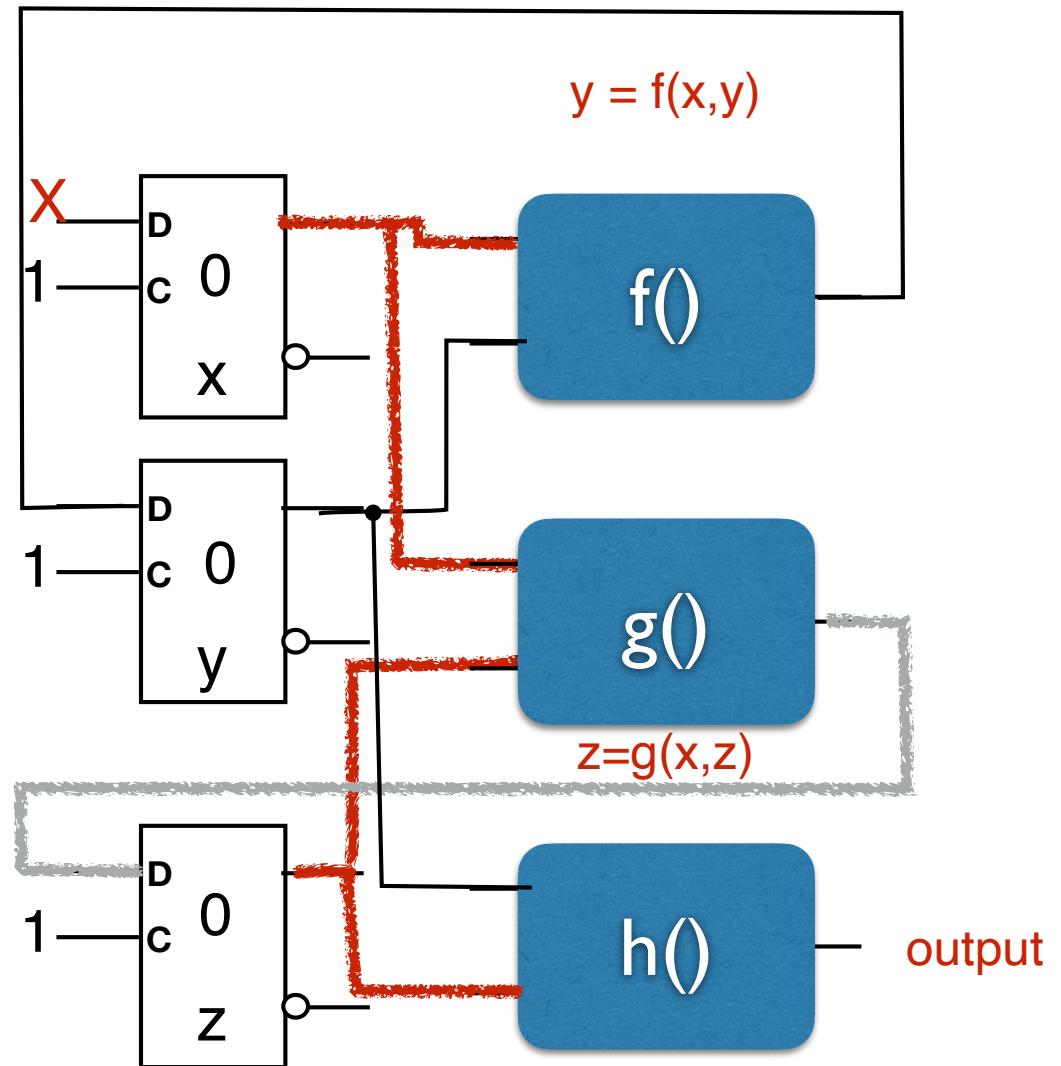
$x=0; y=0; z=0;$

while (){

```
    input x;
    y = f(x,y)
    z = g(x,z)
    output h(y,z)
```

}

Output still not
correct: needs (red) y



g circuit being fed red x, red
z: not supposed to happen!

Where we are, where we are headed

- latches are circuits that can store “state”
 - set the latch to a value (0 or 1)
 - put the latch in a “same value” mode to hold the value
- Some computations use previously computed state
 - e.g., consider following pseudocode

$x=0; y=0; z=0;$

while (){

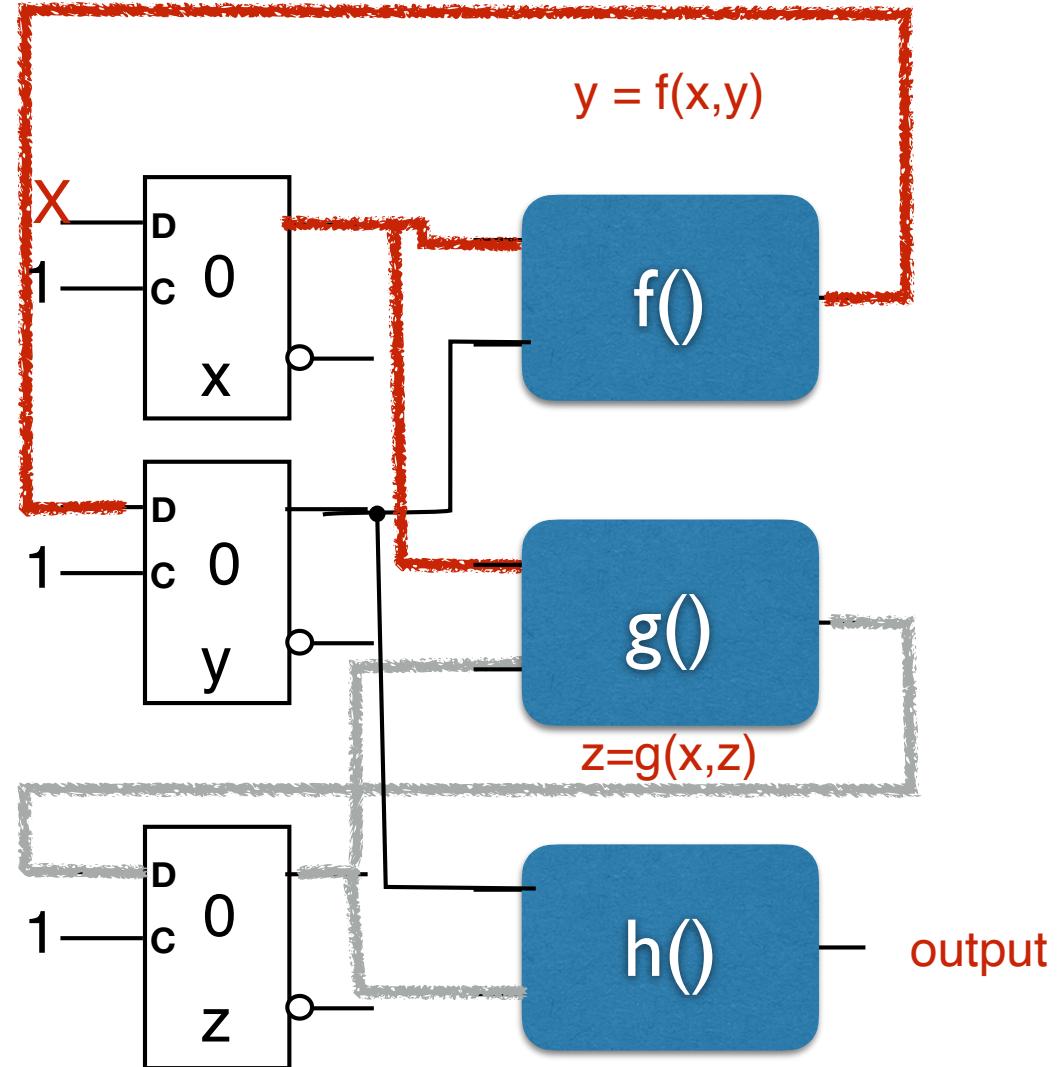
 input x;
 $y = f(x,y)$
 $z = g(x,z)$
 output h(y,z)

}

Y value ready, but...

Output messed up!

Value of z messed up!



Problem with Latches

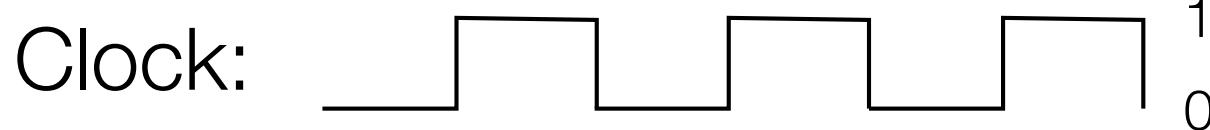
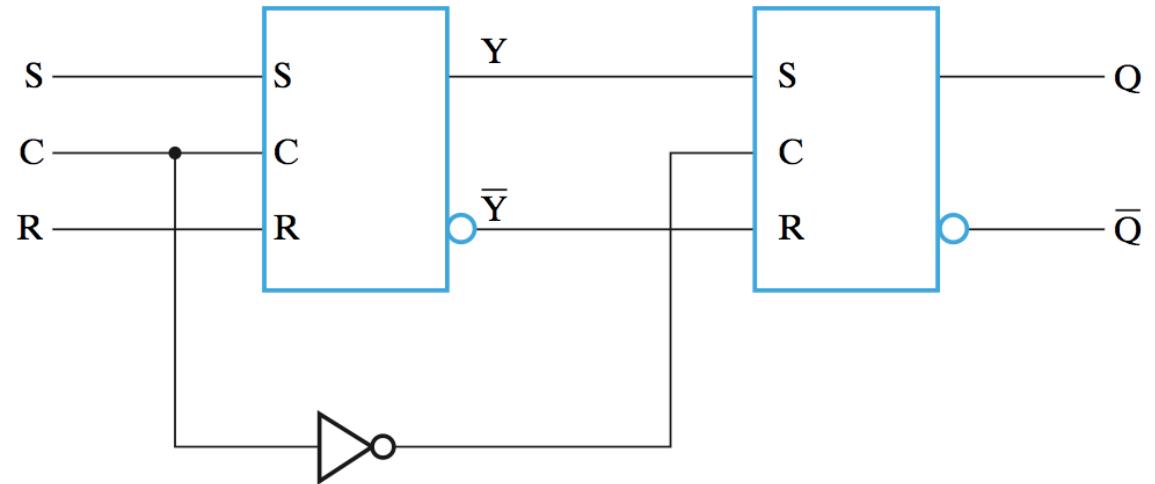
- Their values will change as soon as the input to them change.
- Need some way to synchronize them

—

Flip Flops

SR Flip-Flop

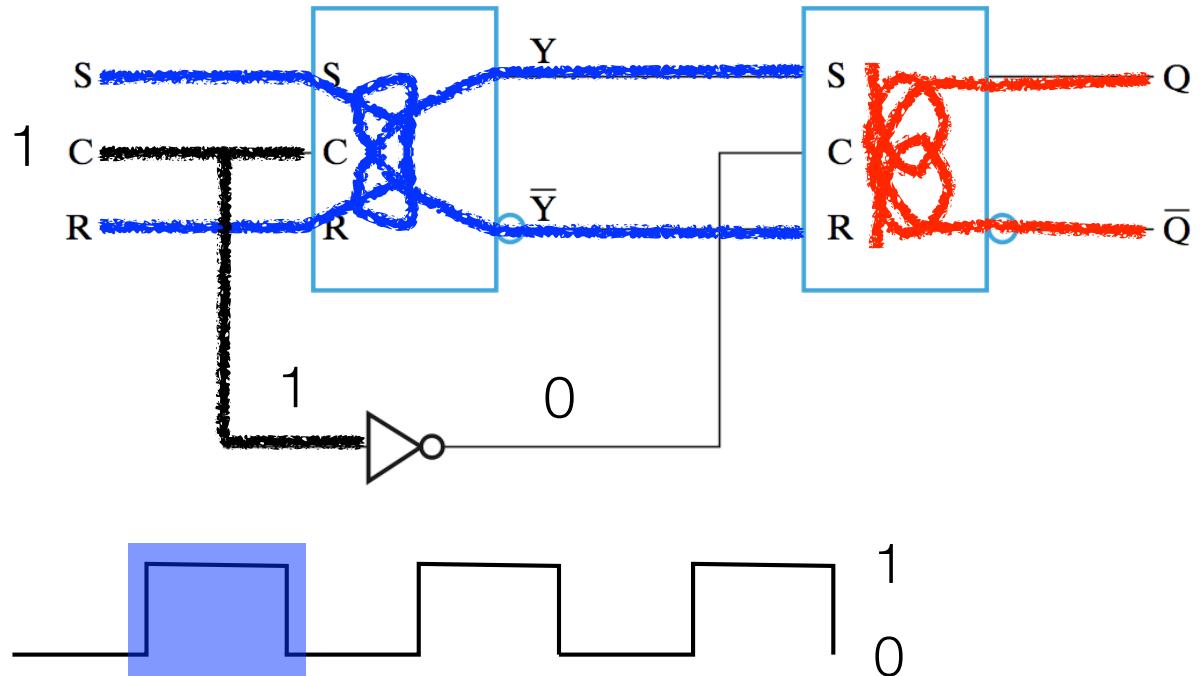
2 SR Latches
with control



- C (Control) is fed a clock pulse (alternates between 0 and 1 with fixed period)
 - C=1: Leader latch “on”, Follower latch “off”
 - New S & R inputs read into leader
 - Previous Q values still emitted (not affected by new S&R inputs)
 - C=0: Leader latch “off”, Follower latch “on”
 - Changing S & R inputs has no effect on Lead (or Follower) latch
 - S&R inputs from last time C=1 stored safely in Leader and transferred into Follower

SR Flip-Flop

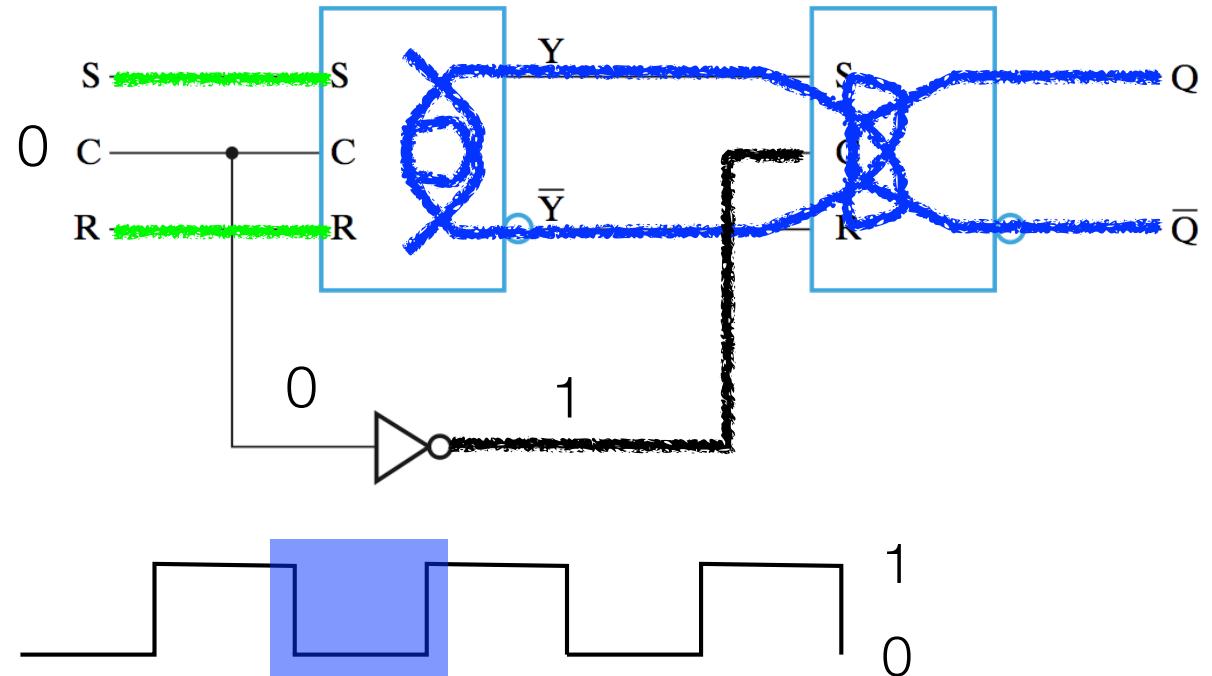
2 SR Latches
with control



- C (Control) is fed a clock pulse (alternates between 0 and 1 with fixed period)
 - C=1: Leader latch “on”, Follower latch “off”
 - New S & R inputs read into leader
 - Previous Q values still emitted (not affected by new S&R inputs)
 - C=0: Leader latch “off”, Follower latch “on”
 - Changing S & R inputs has no effect on Lead (or Follower) latch
 - S&R inputs from last time C=1 stored safely in Leader and transferred into Follower

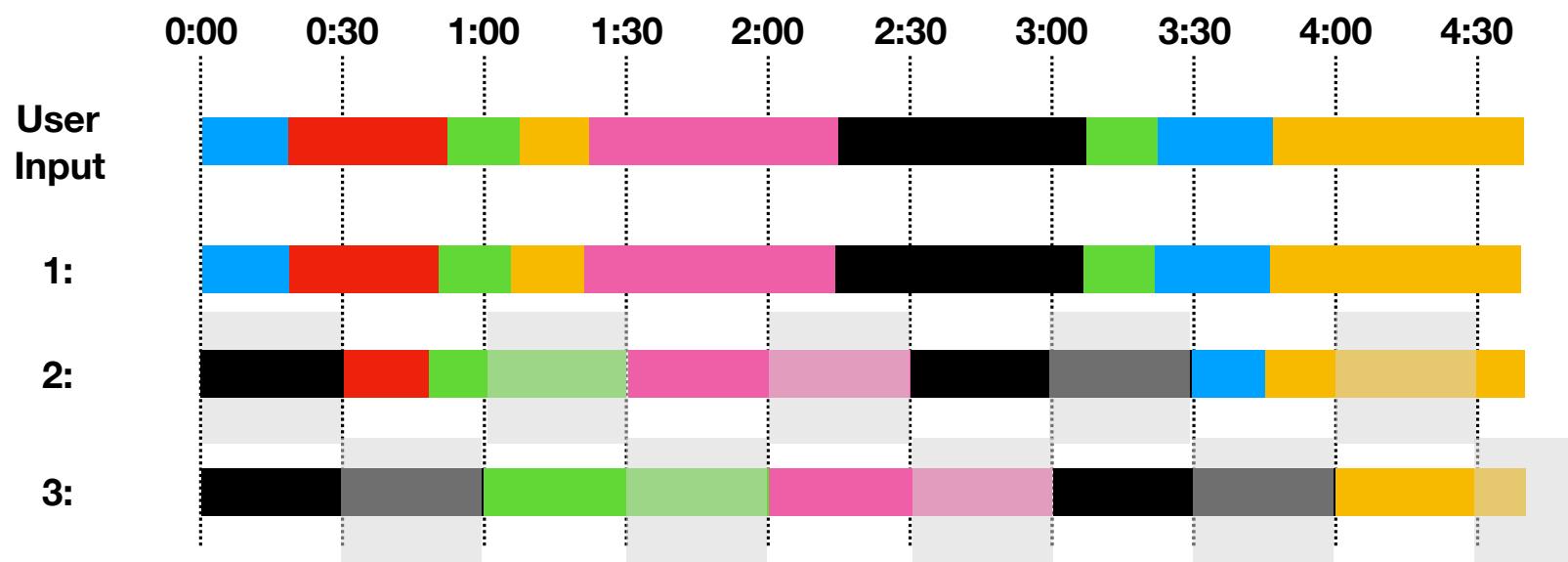
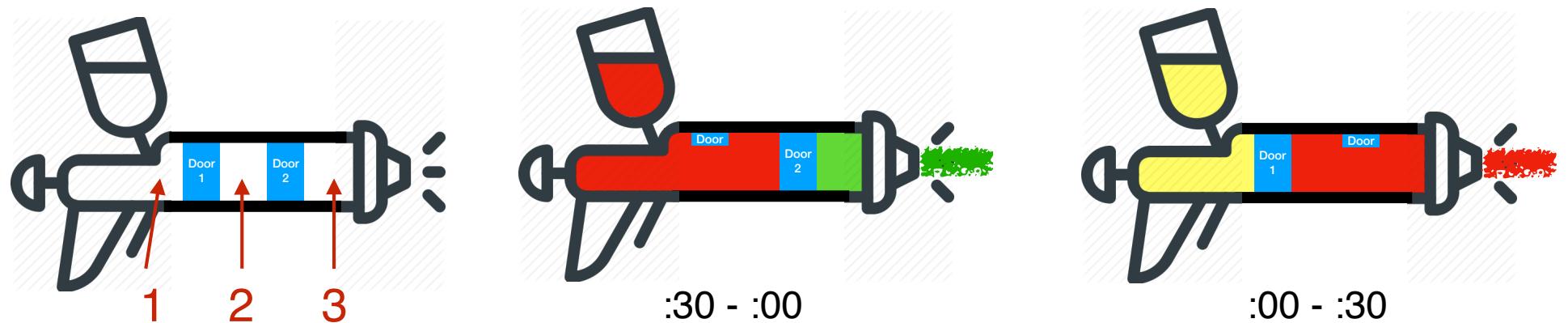
SR Flip-Flop

2 SR Latches
with control



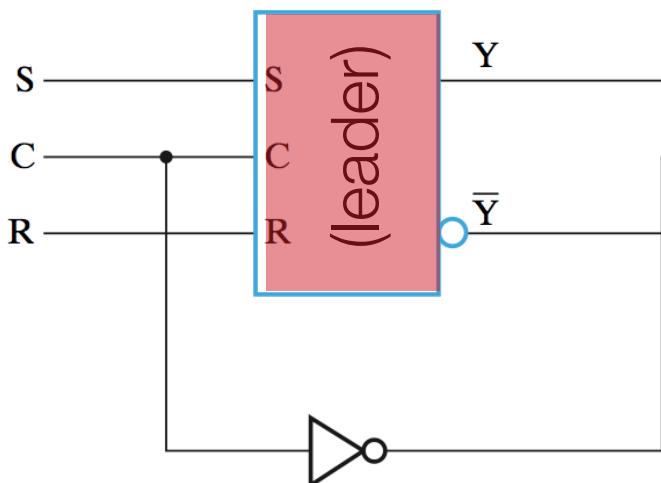
- C (Control) is fed a clock pulse (alternates between 0 and 1 with fixed period)
 - C=1: Leader latch “on”, Follower latch “off”
 - New S & R inputs read into leader
 - Previous Q values still emitted (not affected by new S&R inputs)
 - C=0: Leader latch “off”, Follower latch “on”
 - Changing S & R inputs has no effect on Lead (or Follower) latch
 - S&R inputs from last time C=1 stored safely in Leader and transferred into Follower

Like the Paint Gun from HW#0 (problem 10)

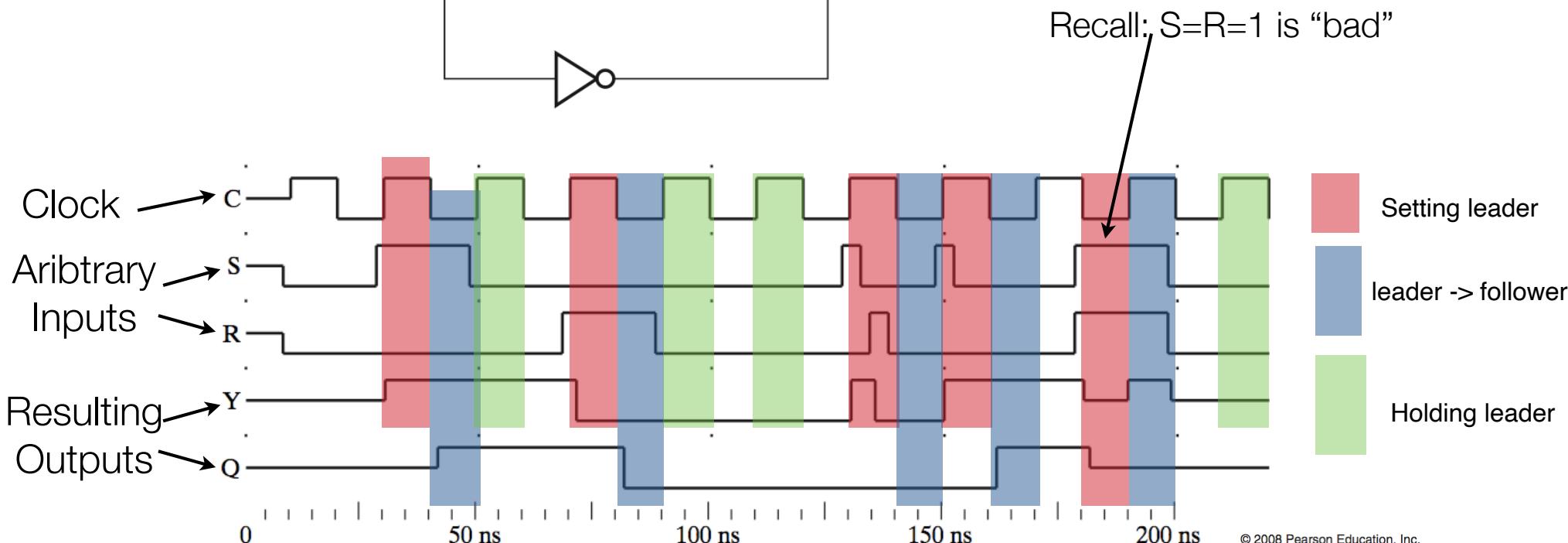


SR Leader-Follower flip-flop

*Internal state (Y) updated
when $CLK=1$*

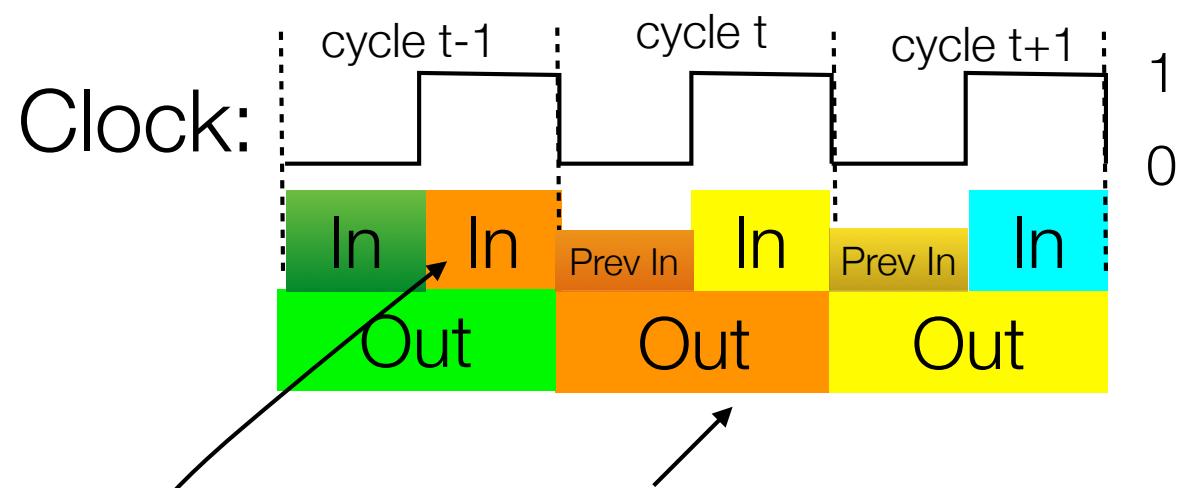
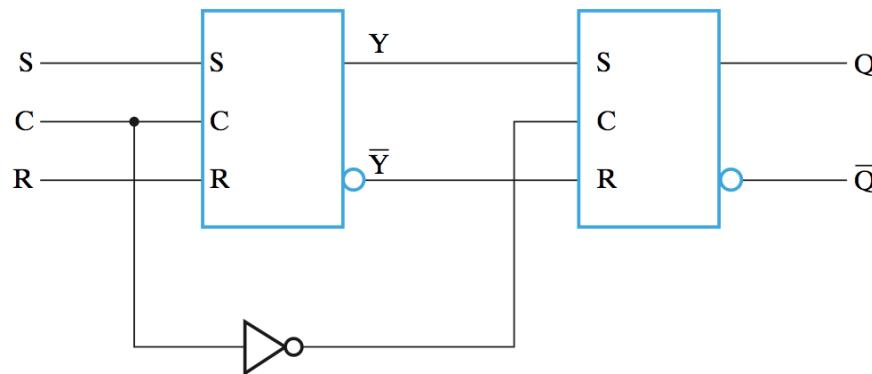


*External state (Q)
updated when $CLK=0$*



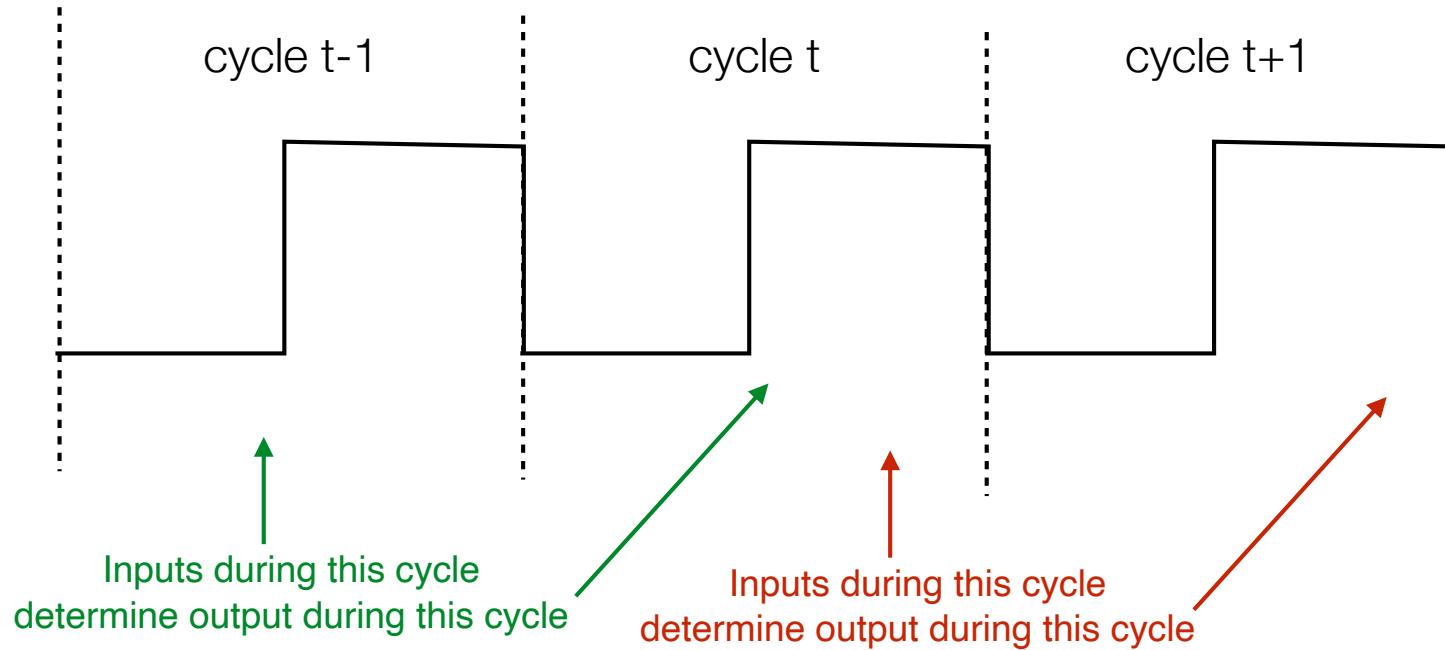
Using a Flip-Flop

SR Flip-Flop: Input, Output, Clock cycle



- $Q(t)$: value output by Flip-Flop during the t^{th} clock cycle (clock =0, then 1 during a full cycle)
- Depends on input during end of $t-1^{\text{st}}$ cycle

So to use a Flip-Flop



Excitation Table

S(t)	R(t)	Q(t+1)
0	0	$Q(t)$
0	1	0
1	0	1
1	1	Don't Use

Example

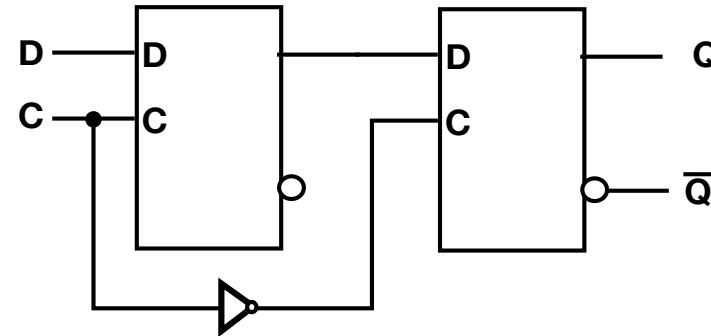
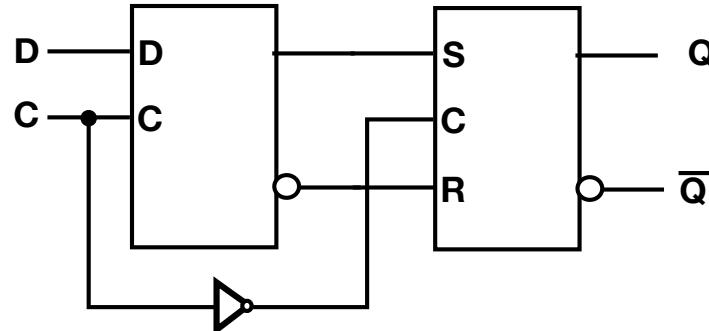
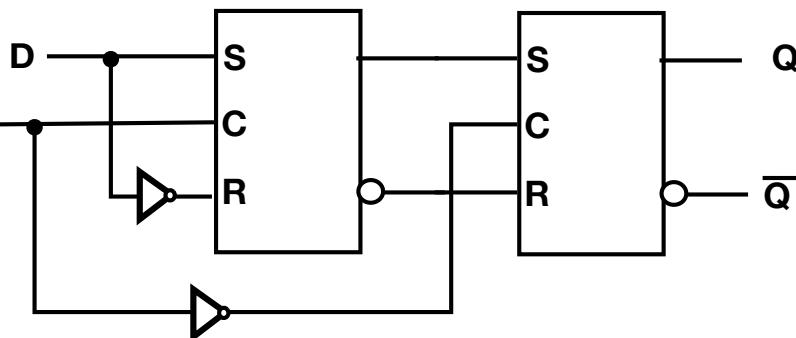
t	0	1	2	3	4	5	6	7	8
S(t)	0	0	1	1	0	1	0	0	0
R(t)	1	0	0	0	0	0	0	1	0
Q(t)	X	0 → 0	1	1 → 1	1	1 → 1	1	1	0

Squares of same color show dependencies between input and output

Other Types of Flip Flops (SR, D, T, JK)

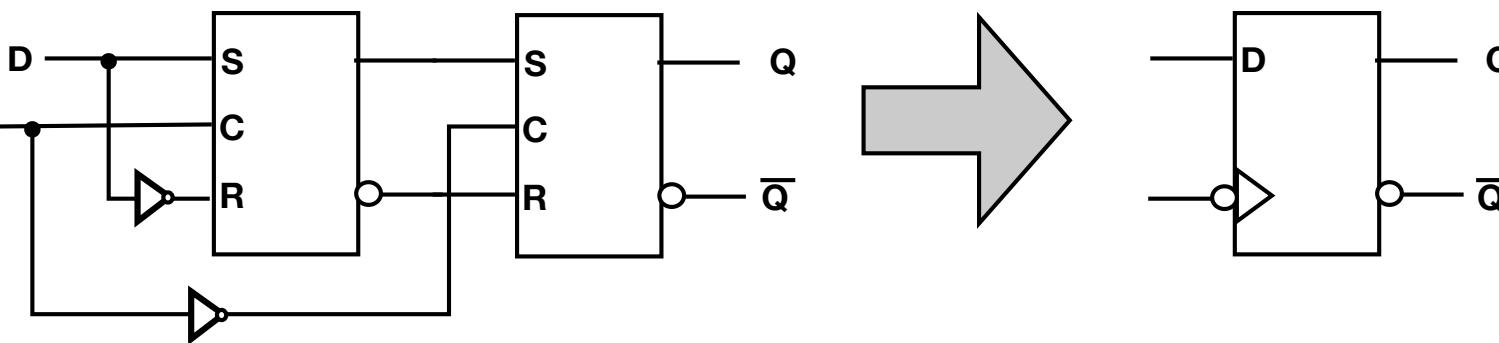
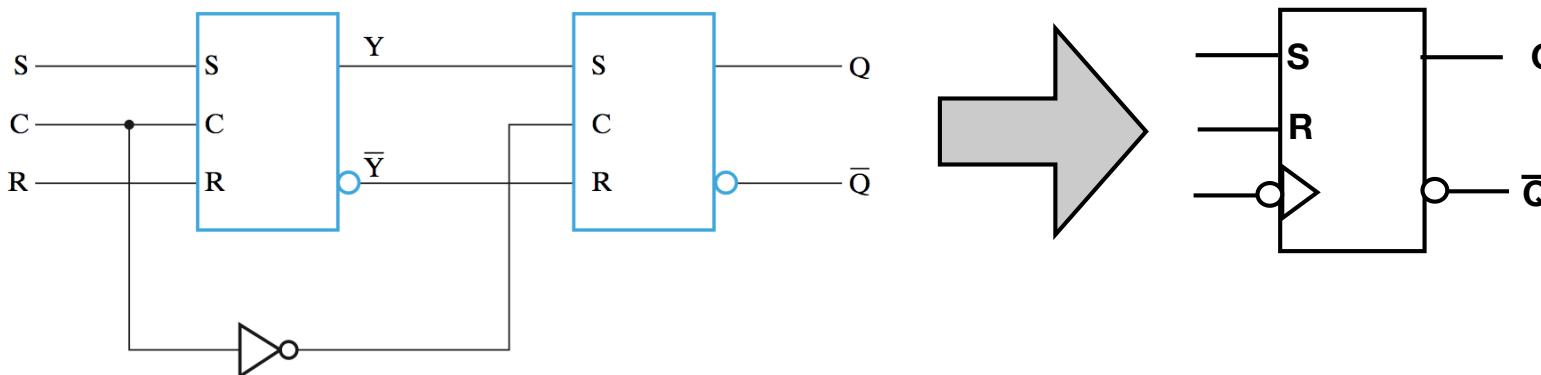
D Flip-Flop

- Can build lots of ways - here are three

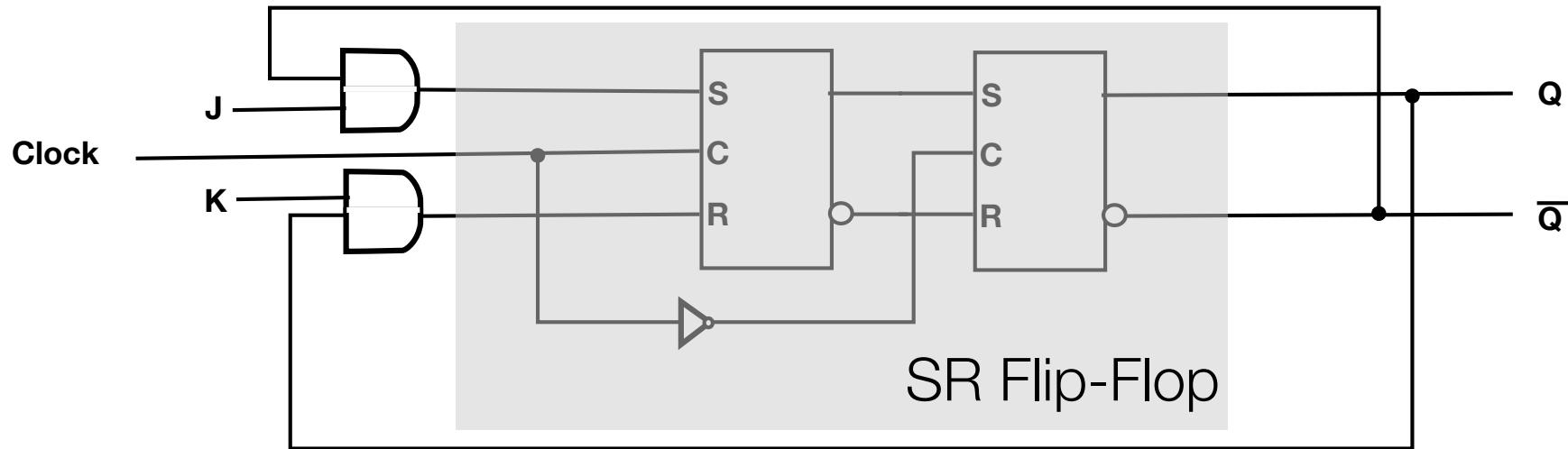


D(t)	Q(t+1)
0	0
1	1

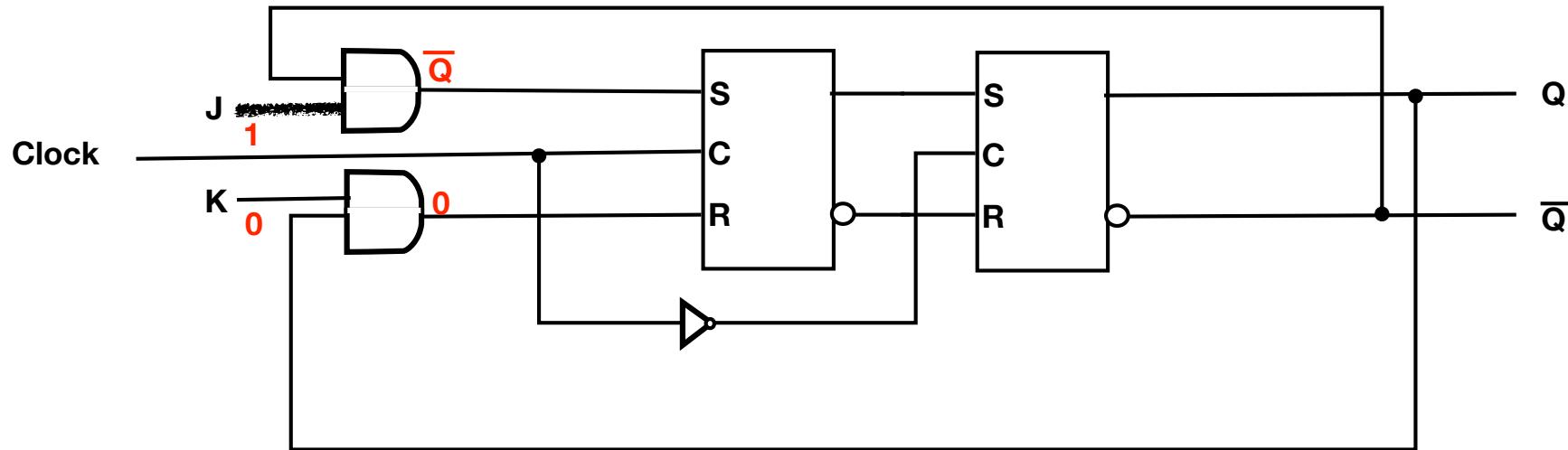
Circuit Diagram for Flip-Flops

- D
 - 
 - 

JK Flip Flop from SR F.F.

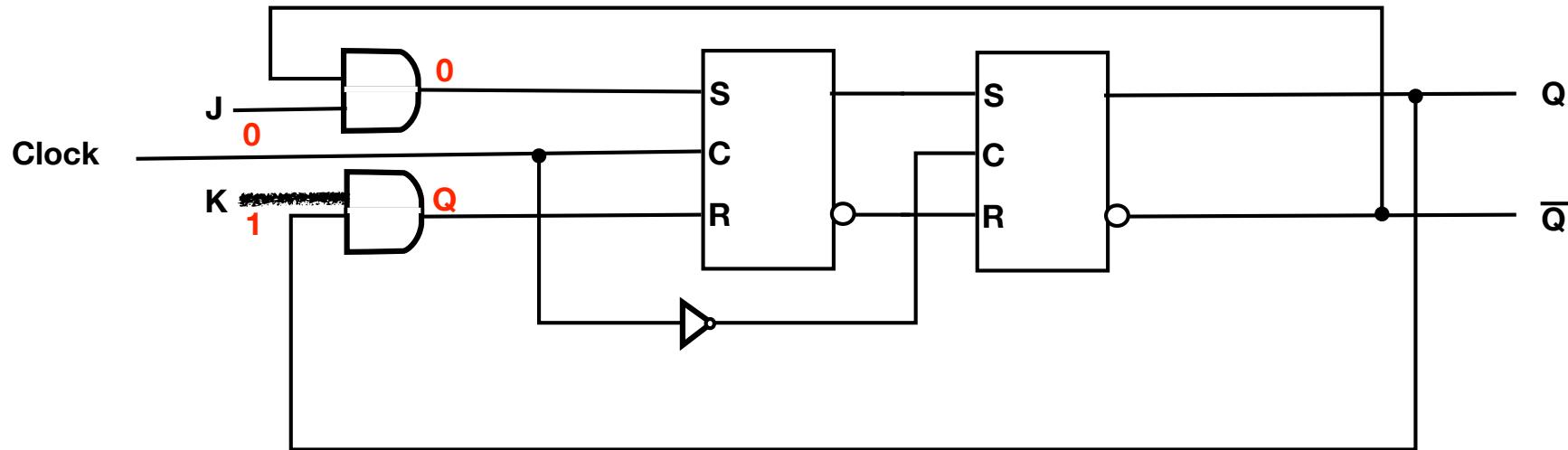


JK Flip Flop from SR F.F.



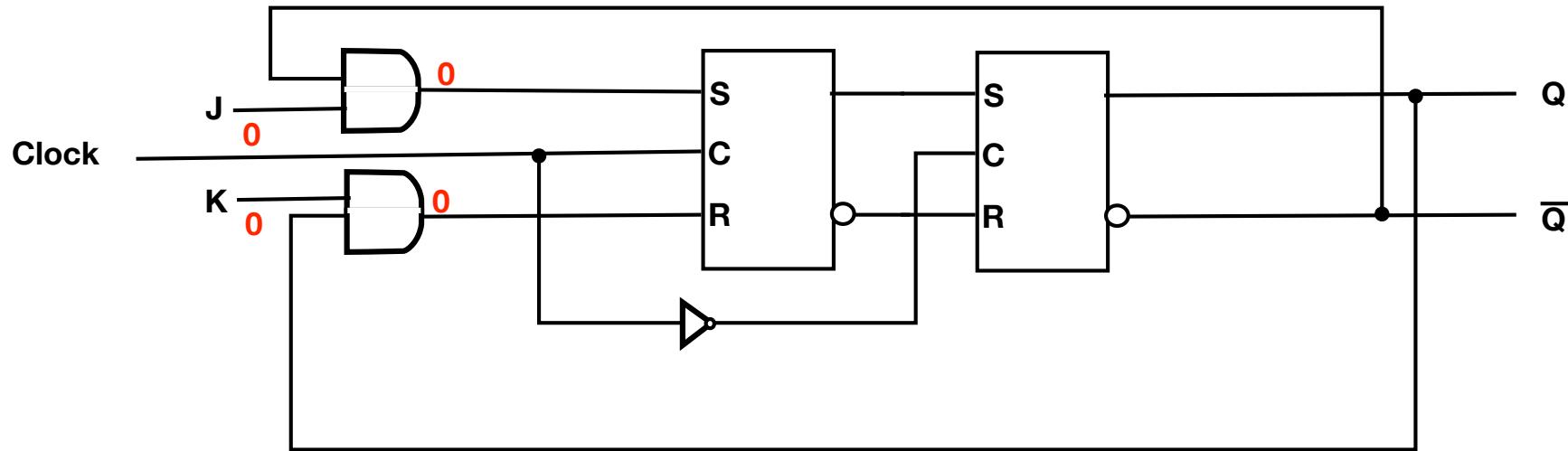
- $J=1, K=0$
 - if $Q(t)=1$, then $\bar{Q}(t)=0$ SR F.F. fed $S=0, R=0$, stays the same: $Q(t+1)=1$
 - if $Q(t)=0$, then $\bar{Q}(t)=1$, SR F.F. fed $S=1, R=0$, set: $Q(t+1)=1$
 - So regardless of $Q(t)$ value, $J=1, K=0$ **sets** the JK F.F.: $Q(t+1) = 1$

JK Flip Flop from SR F.F.



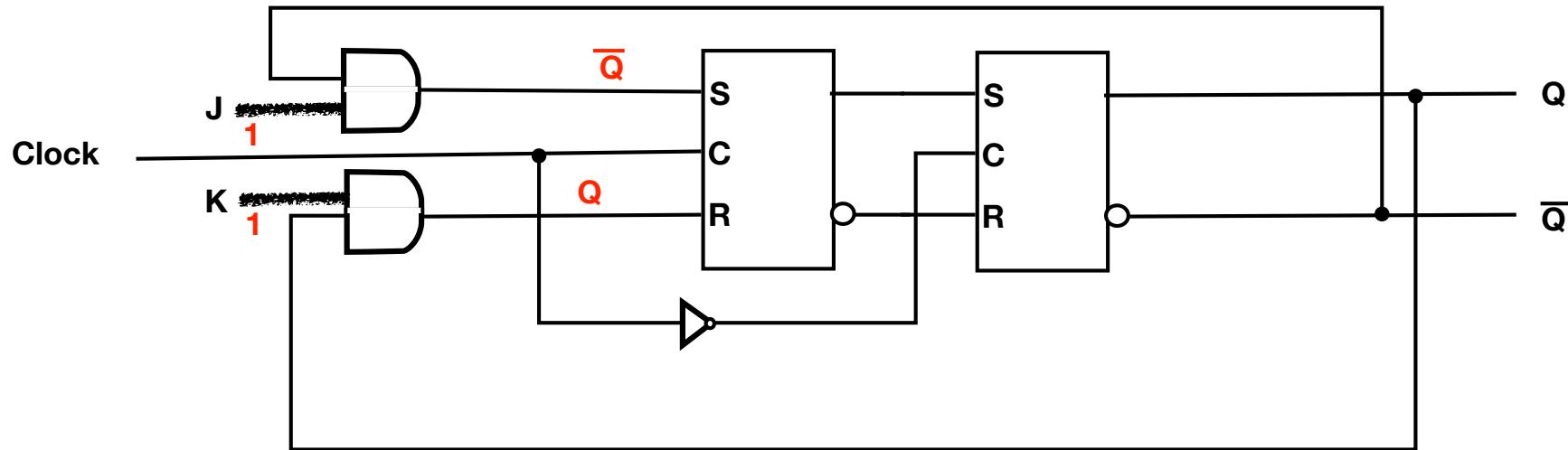
- $J=0, K=1$
 - if $Q(t)=1$, then SR F.F. fed $S=0, R=1$, reset: $Q(t+1)=0$
 - if $Q(t)=0$, then SR F.F. fed $S=0, R=0$, stay same: $Q(t+1) = 0$
 - So regardless of $Q(t)$ value, $J=0, K=1$ **resets** the JK F.F.: $Q(t+1) = 0$

JK Flip Flop from SR F.F.



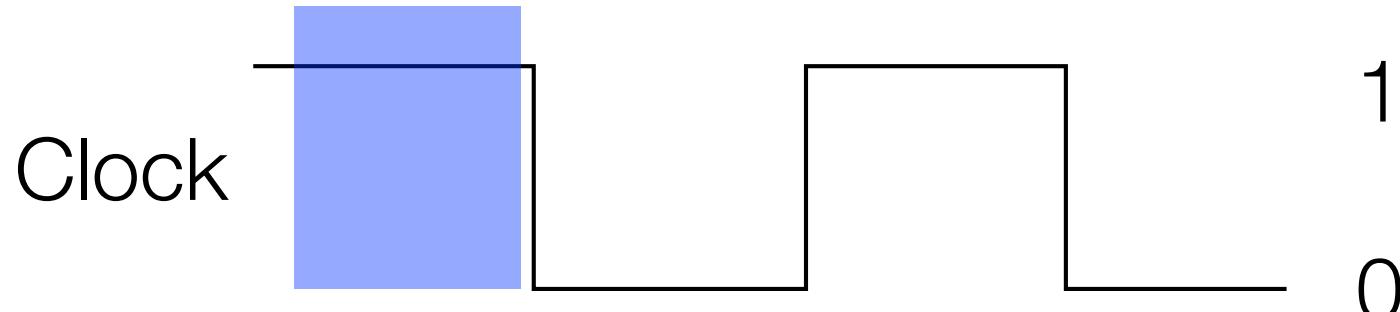
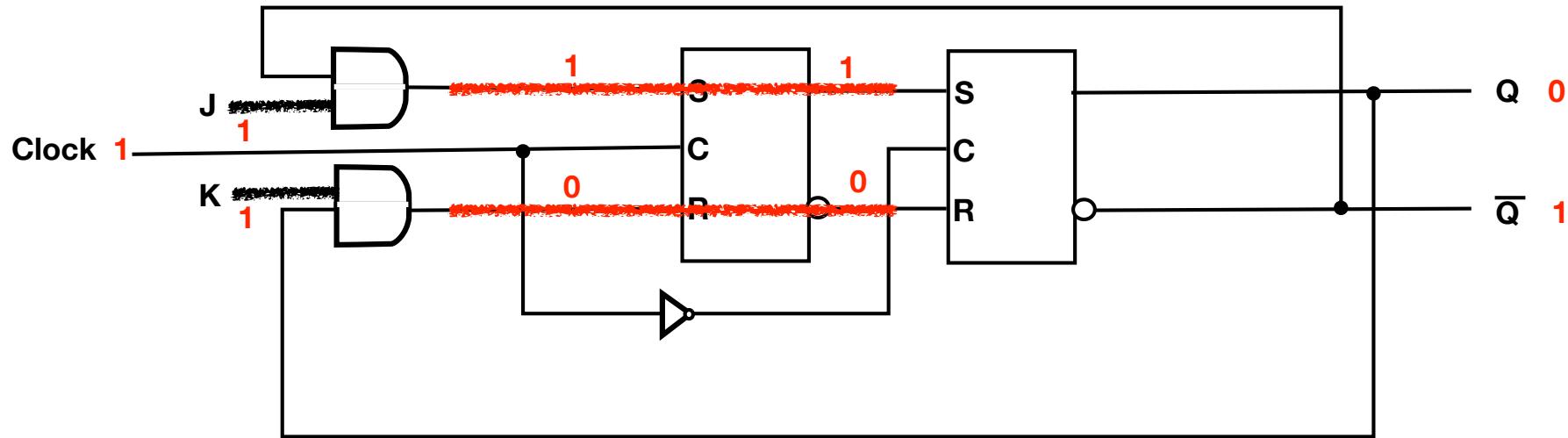
- $J=0, K=0$
 - $S=0, R=0$, hold: $Q(t+1)=Q(t)$
 - so $J=0, K=0$, F.F. **stays same**

JK Flip Flop from SR F.F.



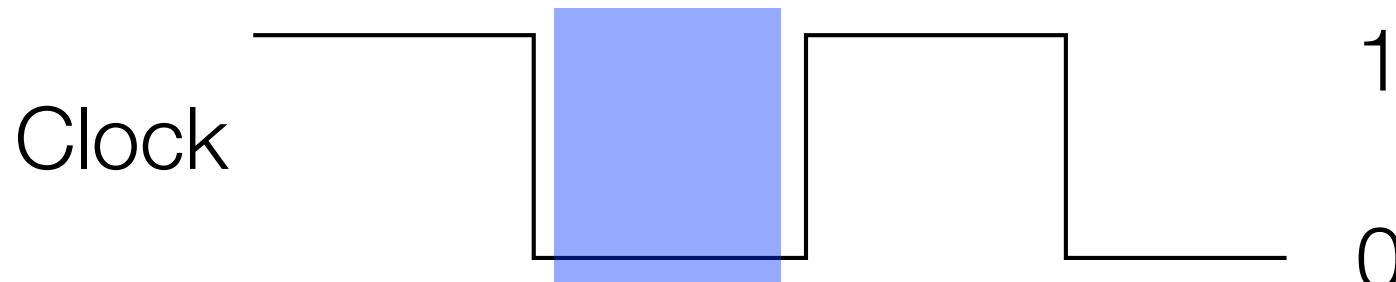
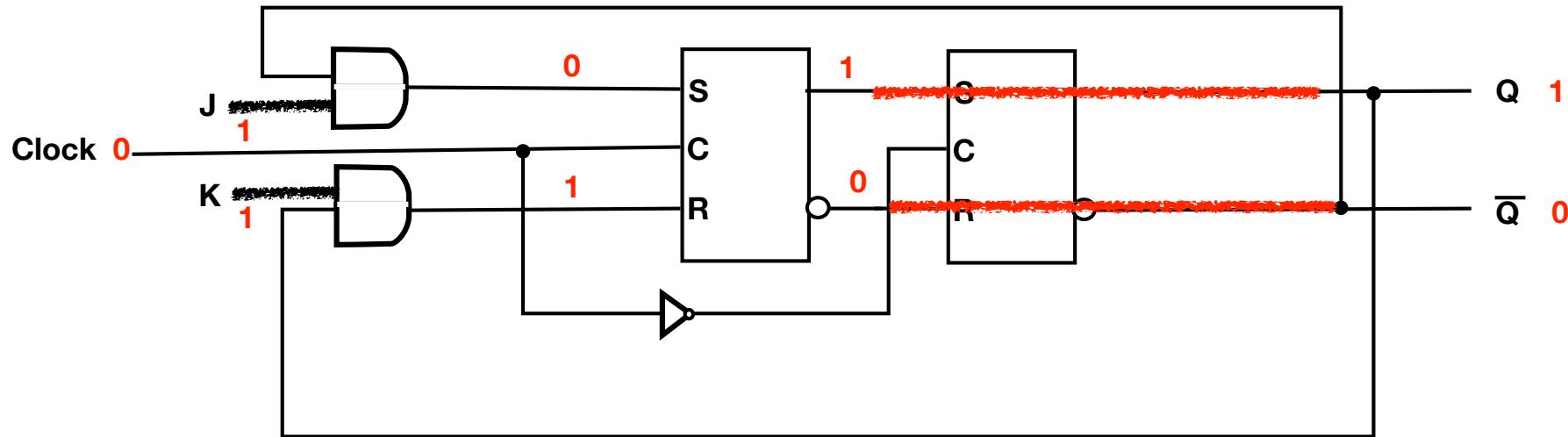
- $J=1, K=1$
 - if $Q(t)=1$, then $\bar{Q}(t)=0$ SR F.F. fed $S=0, R=1$, reset: $Q(t+1)=0$
 - if $Q(t)=0$, then $\bar{Q}(t)=1$, SR F.F. fed $S=1, R=0$, set: $Q(t+1)=1$
 - So $J=1, K=1$ **compliments** the JK F.F.: $Q(t+1) = \bar{Q}(t)$

JK Flip Flop with 1-1 inputs (starting with Q=0)



- C=1:
 - Q' value sent to J, Q value sent to K
 - Leader latch set to compliment values
 - Cannot yet flow through to follower latch

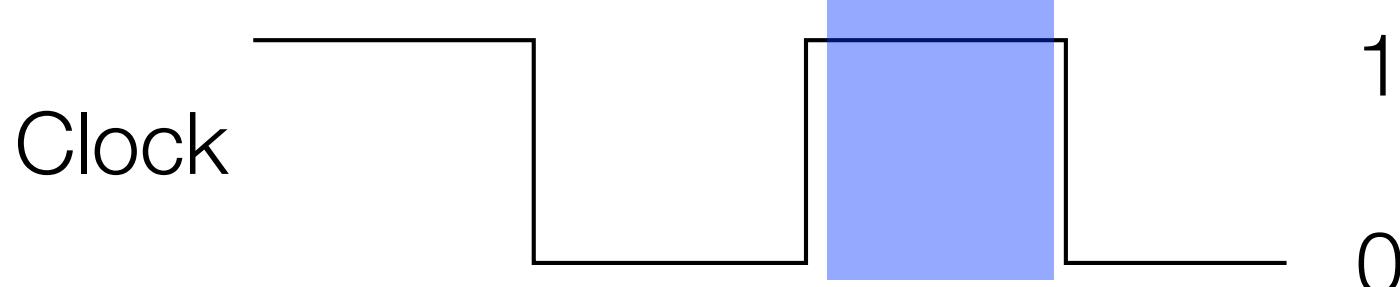
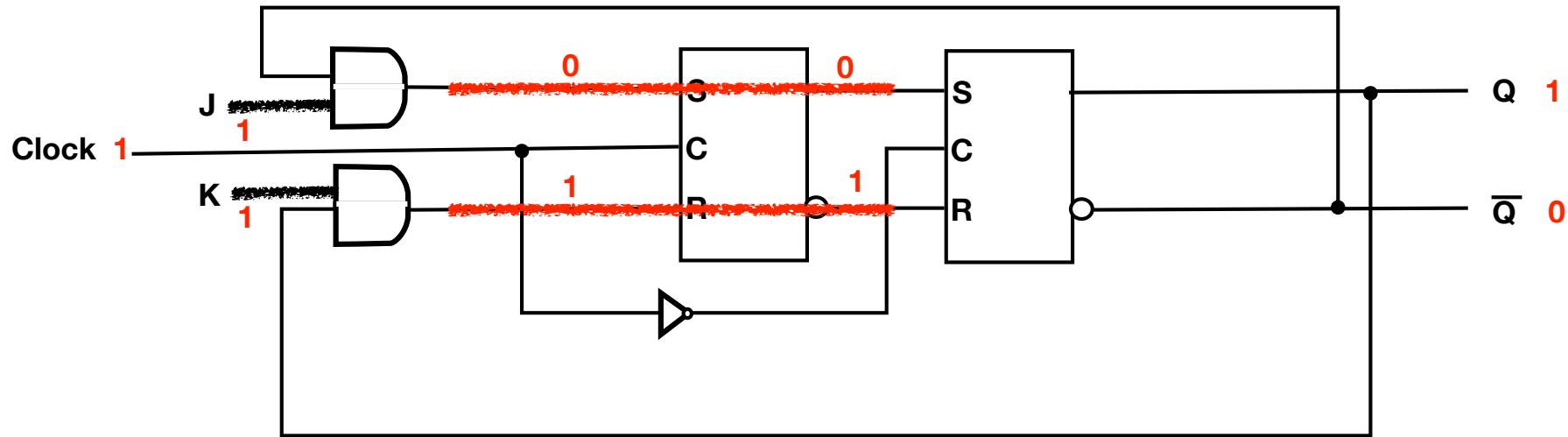
JK Flip Flop with 1-1 inputs (starting with Q=0)



- C=0:

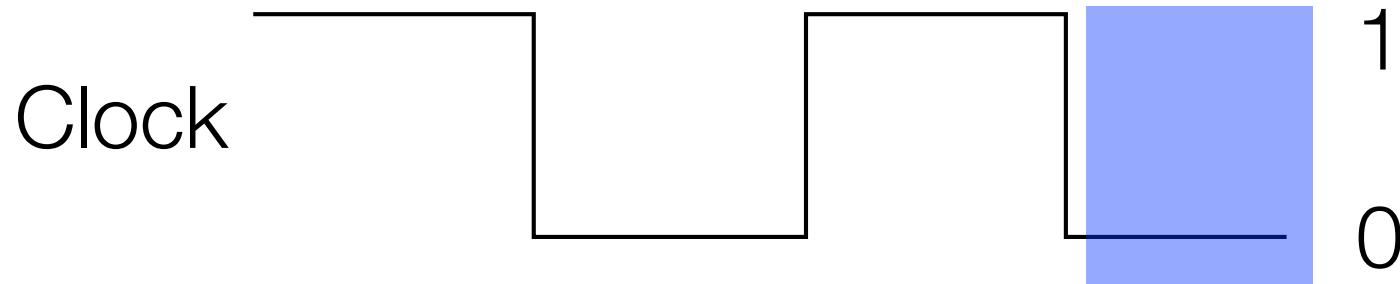
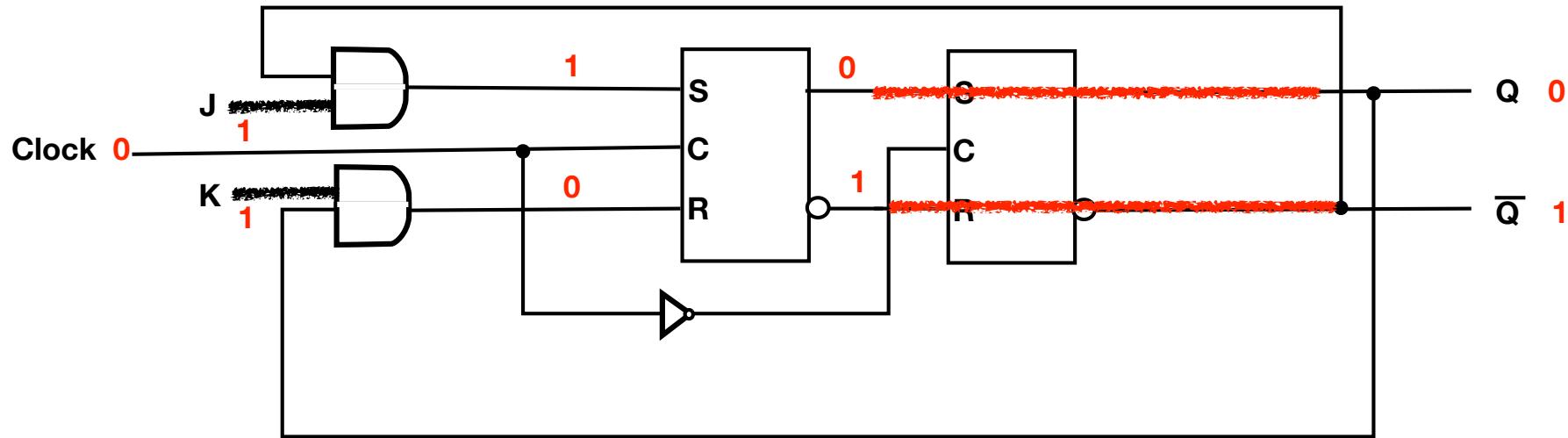
- Compliment values flow from Leader to Follower latches
- Output values have been complimented (start of new cycle)
- Leader inputs complimented, but latch is not enabled yet

JK Flip Flop with 1-1 inputs (starting with Q=0)



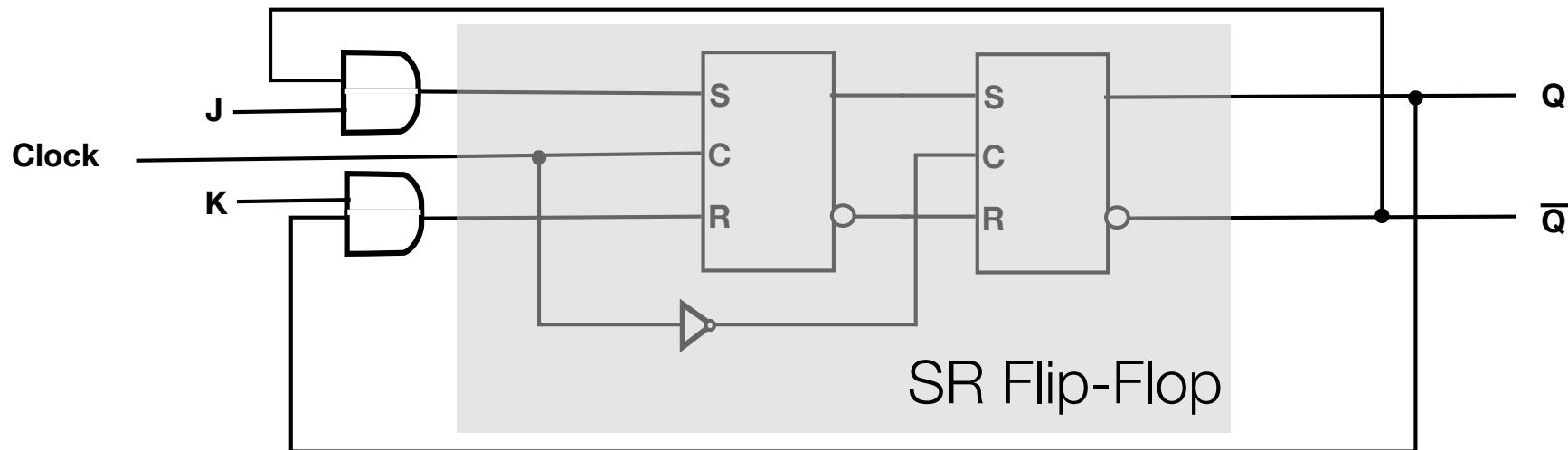
- C=1:
 - Q' value sent to J, Q value sent to K
 - Leader latch set to compliment values
 - Cannot yet flow through to follower latch

JK Flip Flop with 1-1 inputs (starting with Q=0)



- C=0:
 - Compliment values flow from Leader to Follower
 - Output values have been complimented (start of new cycle)
 - Leader inputs complimented, but latch is not enabled yet

JK Flip Flop from SR F.F.



JK Flip-Flop Characteristic Table

J(t)	K(t)	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

Two more flip-flops: JK and T flip-flops

Type	Symbol	Logic Diagrams			Characteristic Table			Characteristic Equation			Excitation Table			
D		See Figure 5-12			D	Q(t+1)	Operation	$Q(t + 1) = D(t)$			Q(t+1)	D	Operation	
					0	0	Reset				0	0	Reset	
SR		See Figure 5-9	S	R	Q(t+1)	Operation	$Q(t + 1) = S(t) + \bar{R}(t) Q(t)$	Q(t)		Q(t+1)	S	R	Operation	
			0	0	$Q(t)$	No change				0	0	0	X	No change
			0	1	0	Reset				0	1	1	0	Set
			1	0	1	Set				1	0	0	1	Reset
JK			J	K	Q(t+1)	Operation	$Q(t + 1) = J(t) \bar{Q}(t) + \bar{K}(t) Q(t)$	Q(t)		Q(t+1)	J	K	Operation	
			0	0	$Q(t)$	No change				0	0	0	X	No change
			0	1	0	Reset				0	1	1	X	Set
			1	0	1	Set				1	0	X	1	Reset
			1	1	$\bar{Q}(t)$	Complement				1	1	X	0	No Change
T			T		Q(t+1)	Operation	$Q(t + 1) = T(t) \oplus Q(t)$	Q(t)		Q(t+1)	T		Operation	
			0		$Q(t)$	No change				$Q(t)$		0		No change
			1		$\bar{Q}(t)$	Complement				$\bar{Q}(t)$		1		Complement

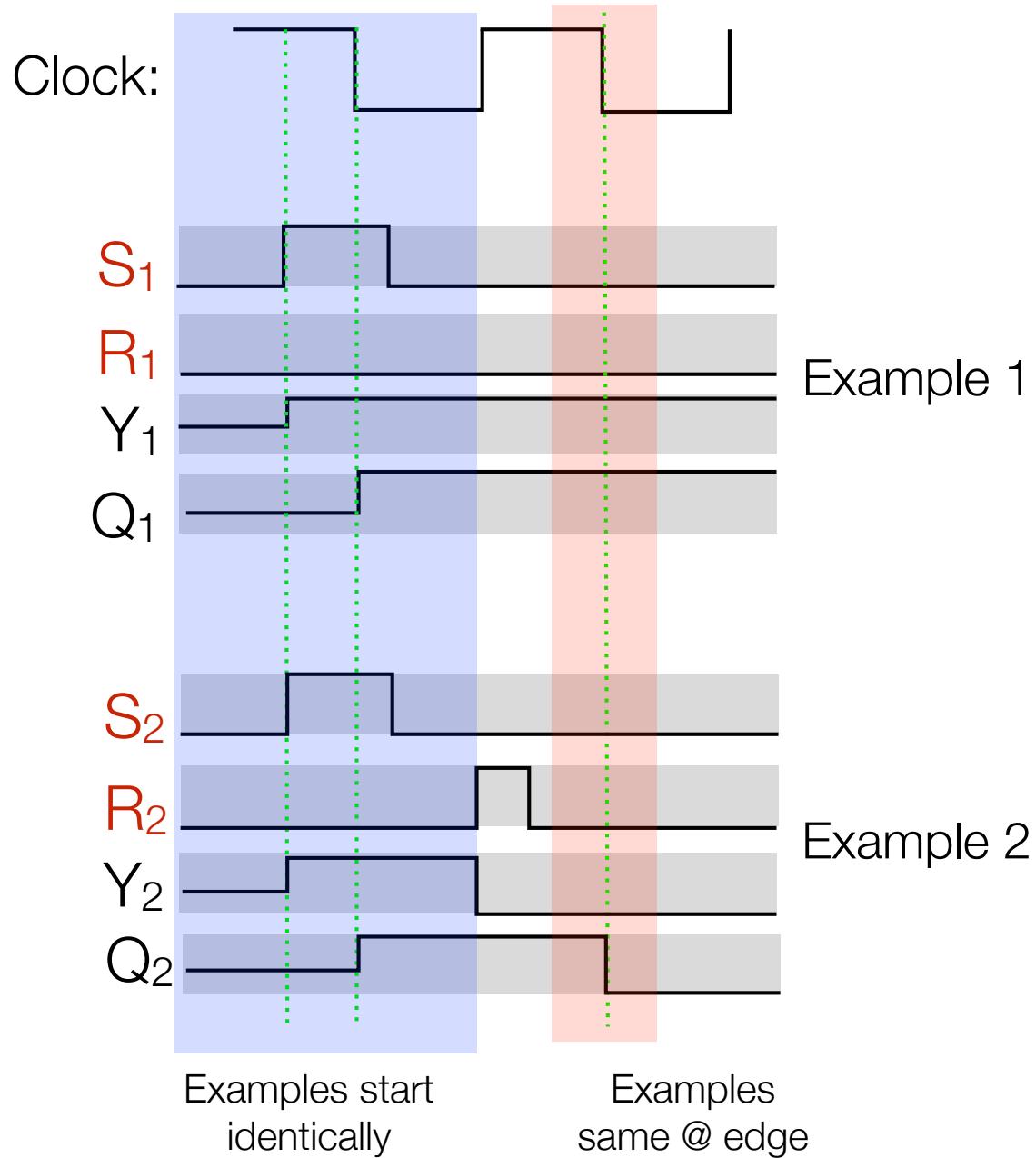
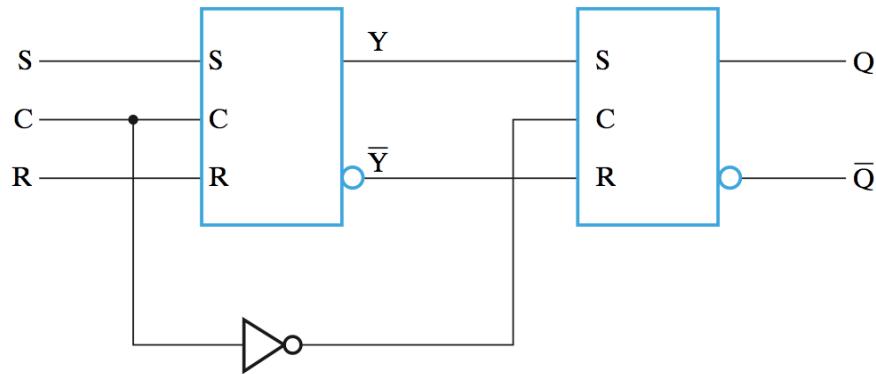
Flip-Flop Trigger Types

Edge v. Pulse triggered FF's

- **Edge triggered**: the output value of the FF depends only on the inputs at the instant in time when the clock transitions in value
- **Pulse triggered**: the output value of the FF can depend on the sequence of input values during the interim of the pulse
- Positive or Negative:
 - **Positive Edge**: output value depends on the input during the 0-to-1 transition
 - **Negative Edge**: output value depends on the input during the 1-to-0 transition
 - **Positive Pulse**: Pulse Triggered and Leader latch active when C=1
 - **Negative Pulse**: Pulse Triggered and Leader latch active when C=0
- D FF's are negative edge triggered (take on whatever value D is set to when clock "flops" from 1 to 0)
- SR FF's are positive pulse triggered (e.g., S=1, R=0 at start of pulse, then switch to S=0, R=0 before end).

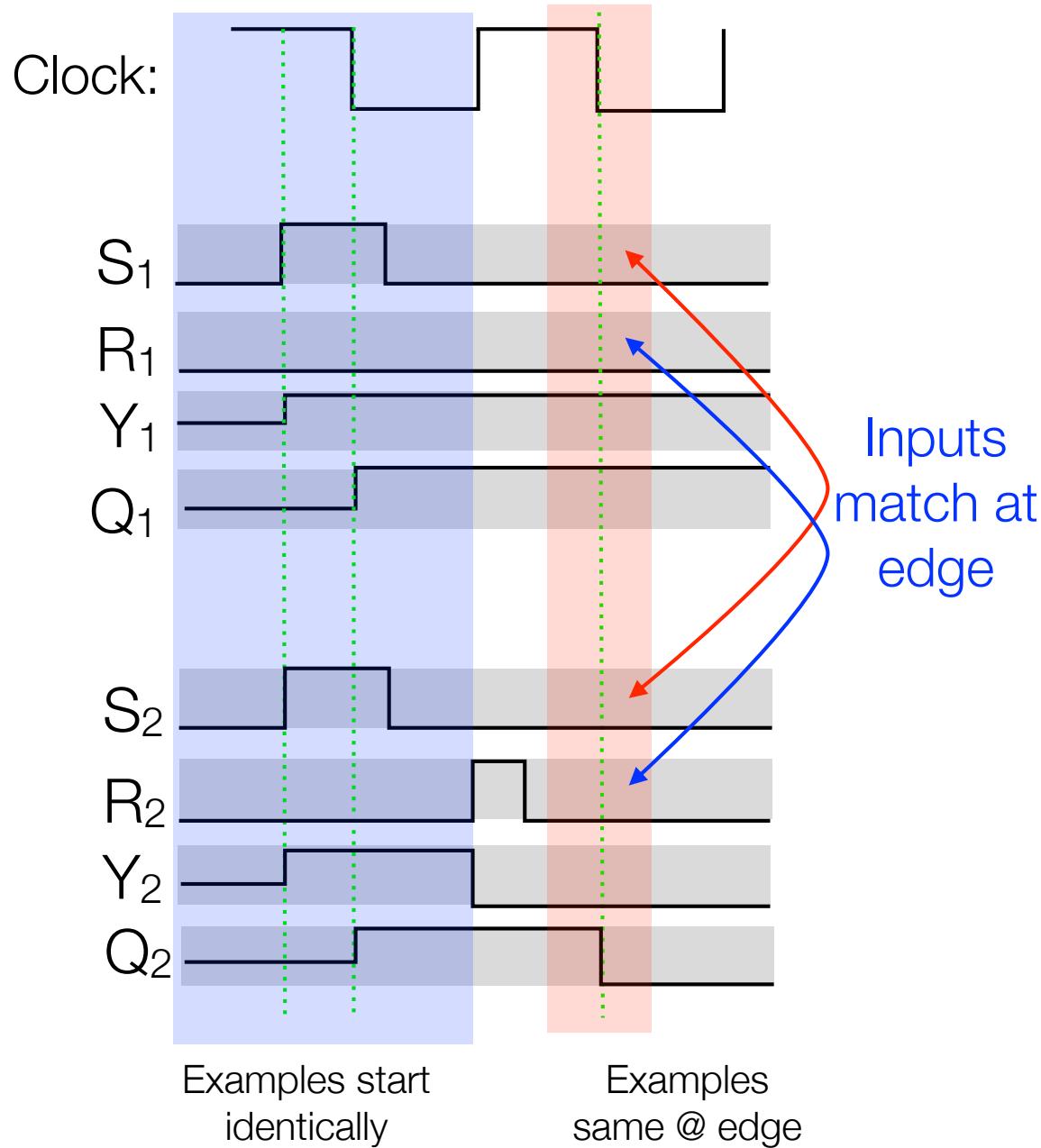
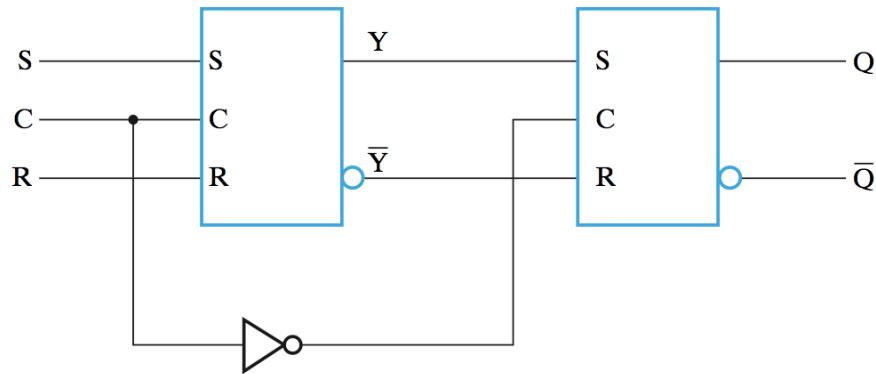
Pulse triggered example

- Pulse triggered: cannot simply look @ inputs at pulse edge to determine value of output
- Example:
 - S_1, R_1 are inputs for 1st example to pulse-triggered FF
 - S_2, R_2 are inputs to 2nd example



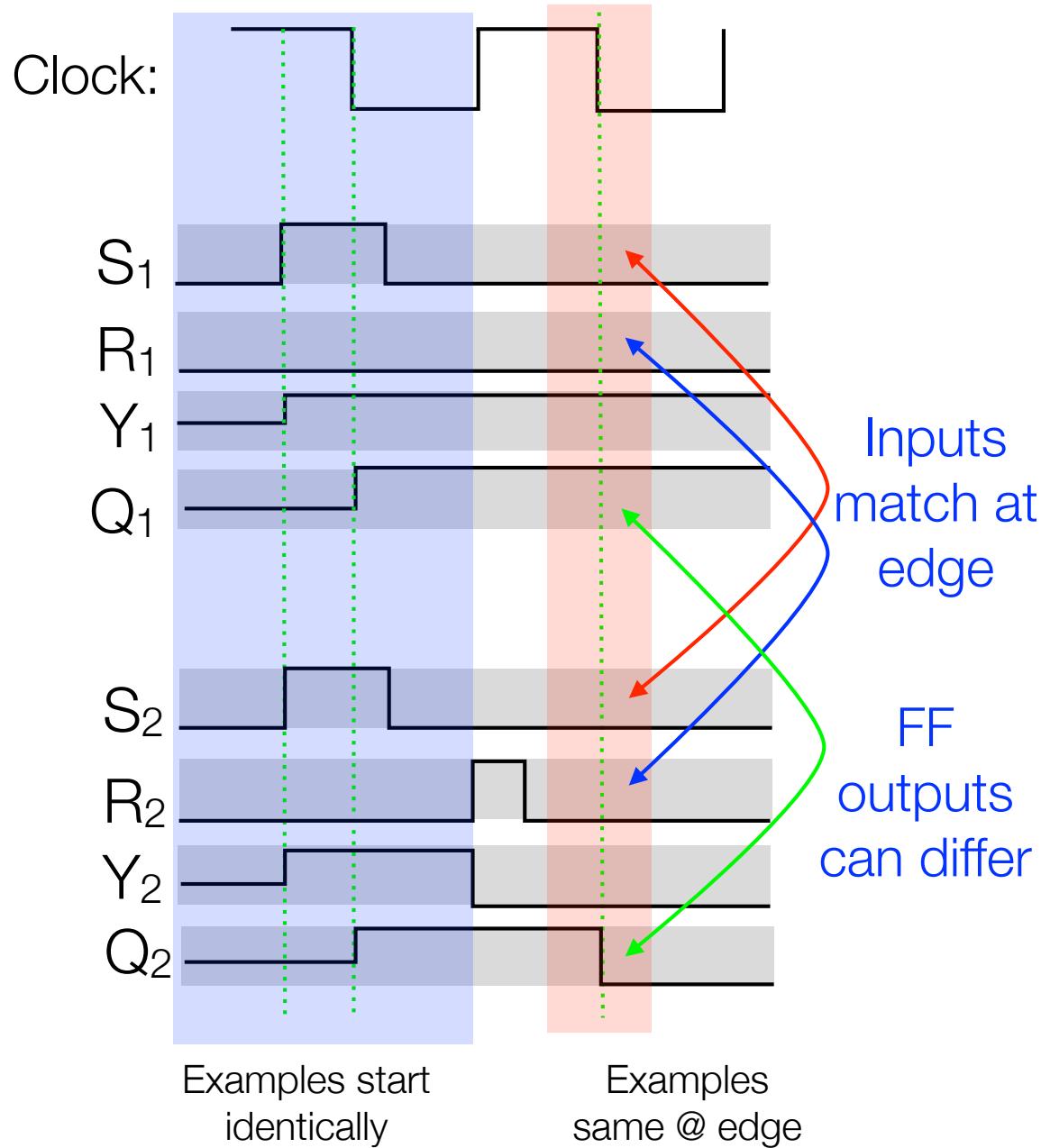
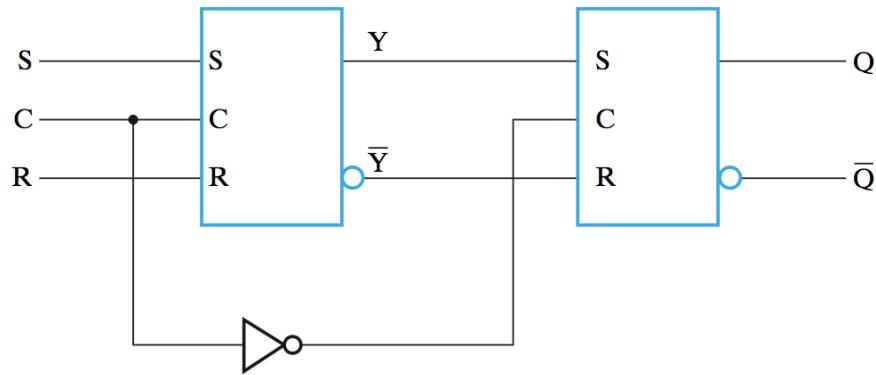
Pulse triggered example

- Pulse triggered: cannot simply look @ inputs at pulse edge to determine value of output
- Example:
 - S_1, R_1 are inputs for 1st example to pulse-triggered FF
 - S_2, R_2 are inputs to 2nd example



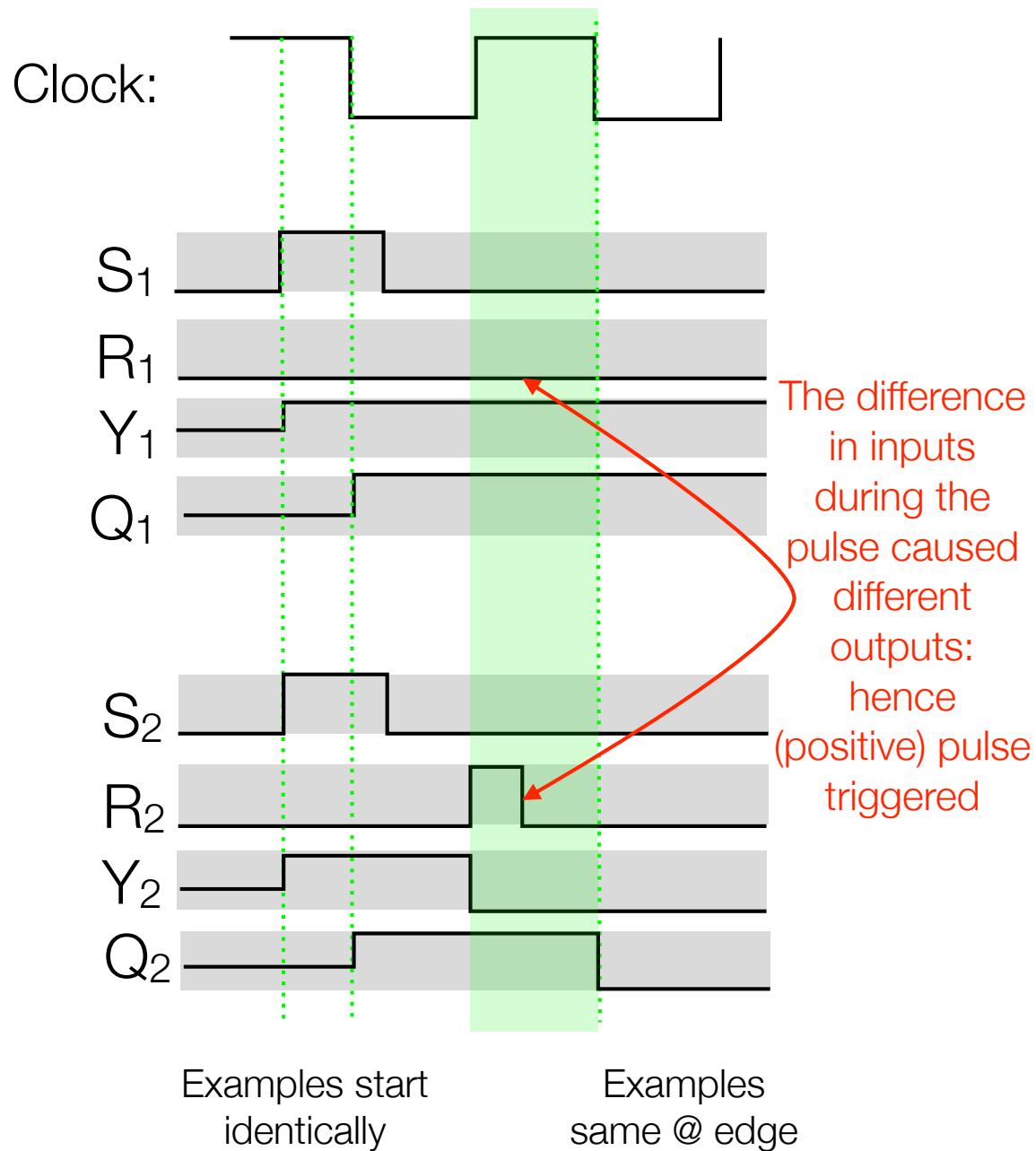
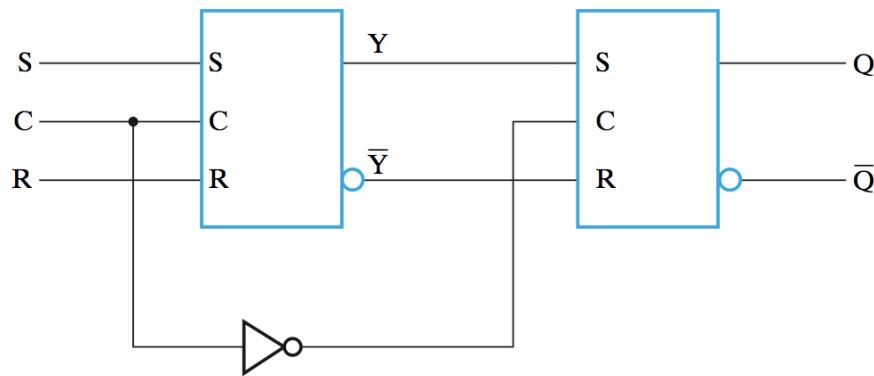
Pulse triggered example

- Pulse triggered: cannot simply look @ inputs at pulse edge to determine value of output
- Example:
 - S_1, R_1 are inputs for 1st example to pulse-triggered FF
 - S_2, R_2 are inputs to 2nd example

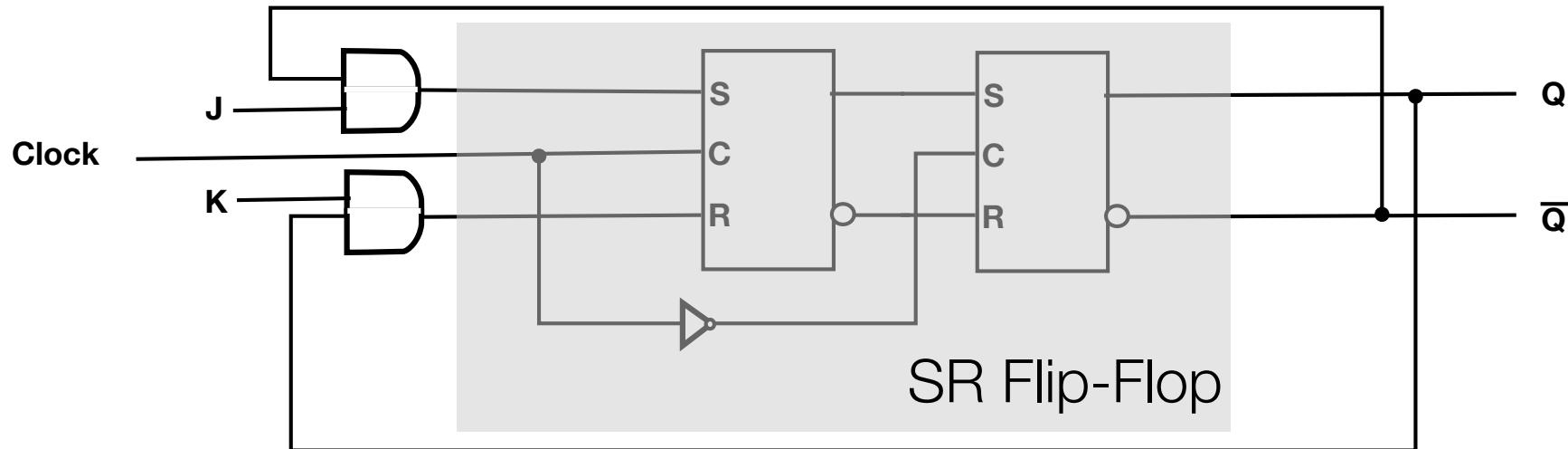


Pulse triggered example

- Pulse triggered: cannot simply look @ inputs at pulse edge to determine value of output
- Example:
 - S_1, R_1 are inputs for 1st example to pulse-triggered FF
 - S_2, R_2 are inputs to 2nd example



JK Flip Flop from SR F.F.

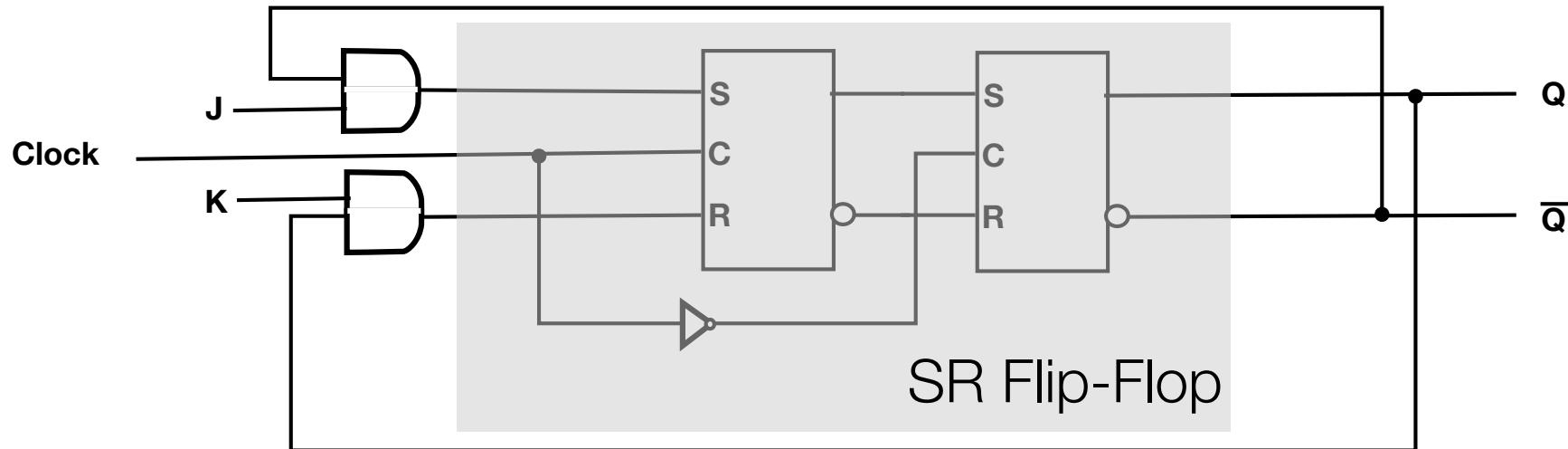


JK Flip-Flop Characteristic Table

J(t)	K(t)	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

Q: Edge or pulse triggered?

JK Flip Flop from SR F.F.



JK Flip-Flop Characteristic Table

J(t)	K(t)	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

Q: Edge or pulse triggered?

A: Pulse: during positive pulse, set JK=11 (leader is opposite follower), then change to JK=00 to hold this opposite value