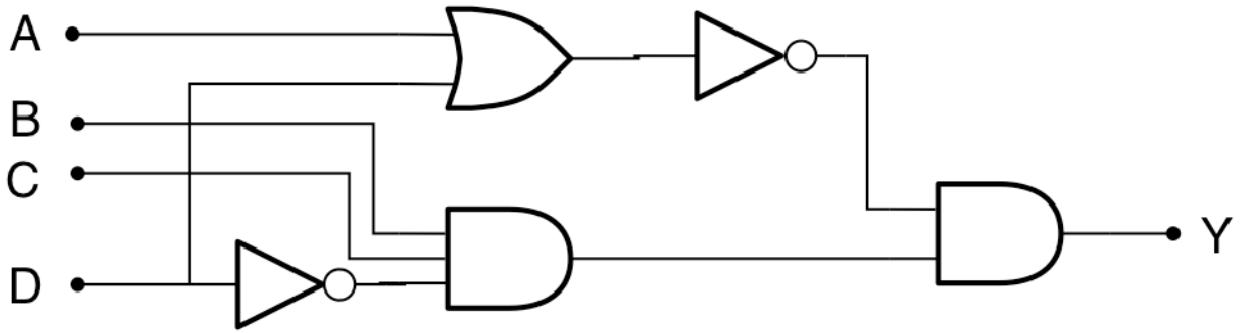
NANYANG TECHNOLOGICAL UNIVERSITY

CX1005
Digital Logic

Sequential Circuits

Review of Combinational Circuits

- So far, we have investigated ***combinational*** circuits
- The output of a combinational circuit is purely a function of the present inputs to it
  - *output = f(input)*
- When the inputs are changed, the output will change to that determined by those inputs (after a short propagation delay)
- Knowing only the ***current*** inputs, we can always determine the output



Combinational and Sequential

NANYANG TECHNOLOGICAL UNIVERSITY

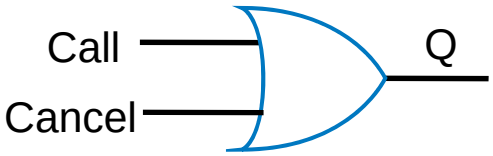
- ***Sequential*** circuits introduce the idea of ***state***
  - The state of a circuit encodes sufficient information about the past to determine how it will react with inputs to produce an output

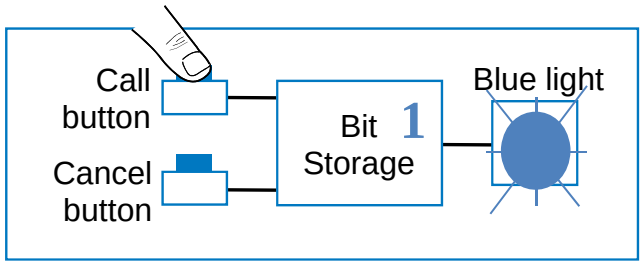
- The output of a sequential circuit depends on **both** the ***current*** inputs and ***previous*** inputs

- In this sense, sequential circuits have ***memory***, whereas combinational circuits have no memory.

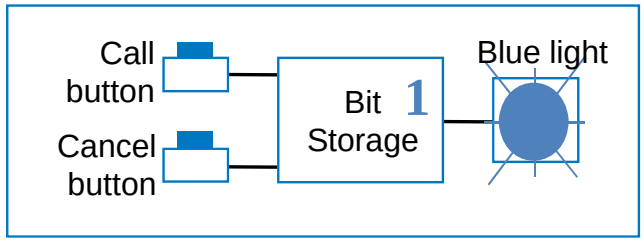
Sequential Logic

- We'd like to store a single bit
- Flight attendant call button
  - Press "Call": light on
    - Stays on after button released
  - Press "Cancel": light off
    - Stays off after button released
  - Can we use a gate?

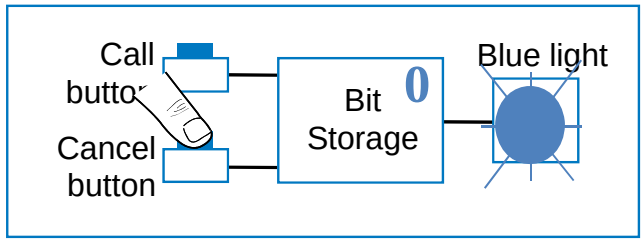




1. Call button pressed – light turns on



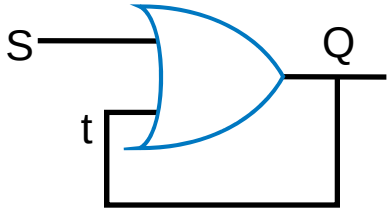
*2. Call button released – light **stays on***

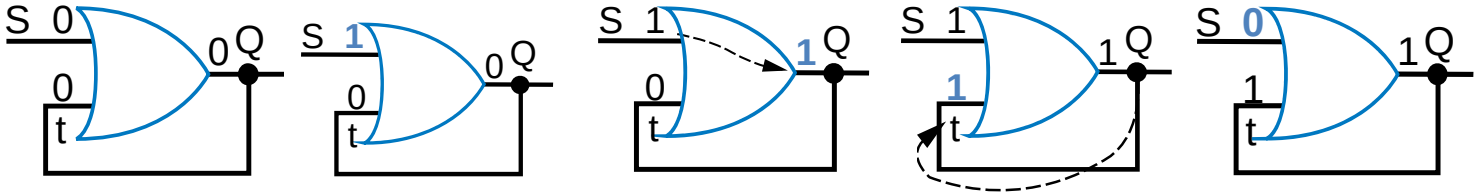


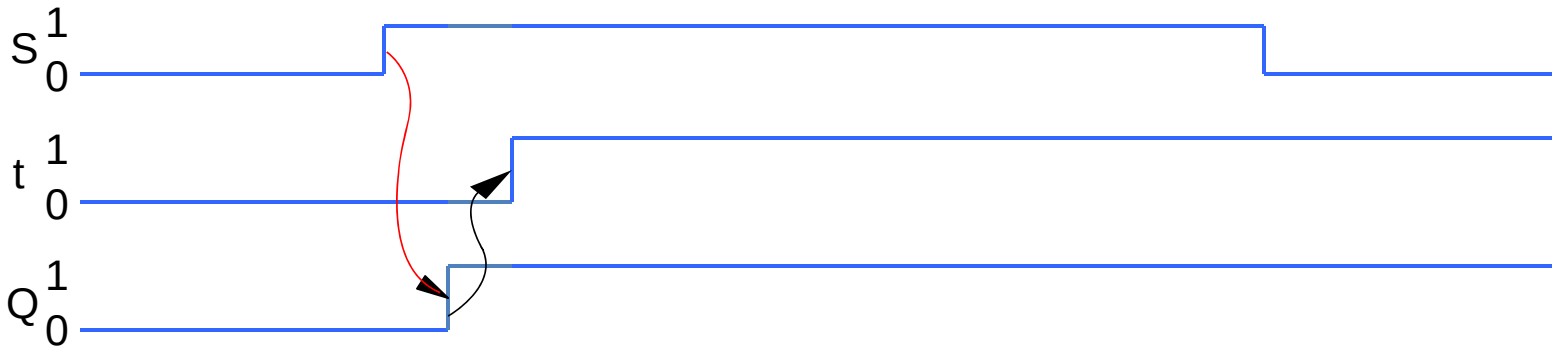
3. Cancel button pressed – light turns off

Sequential Logic

- What if we add feedback?
  - S will now cause 1 to be stored

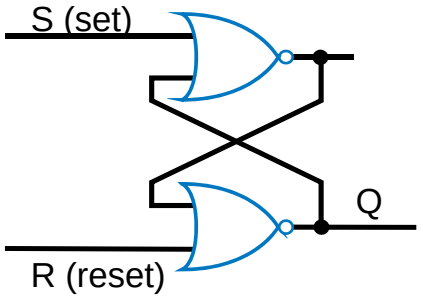


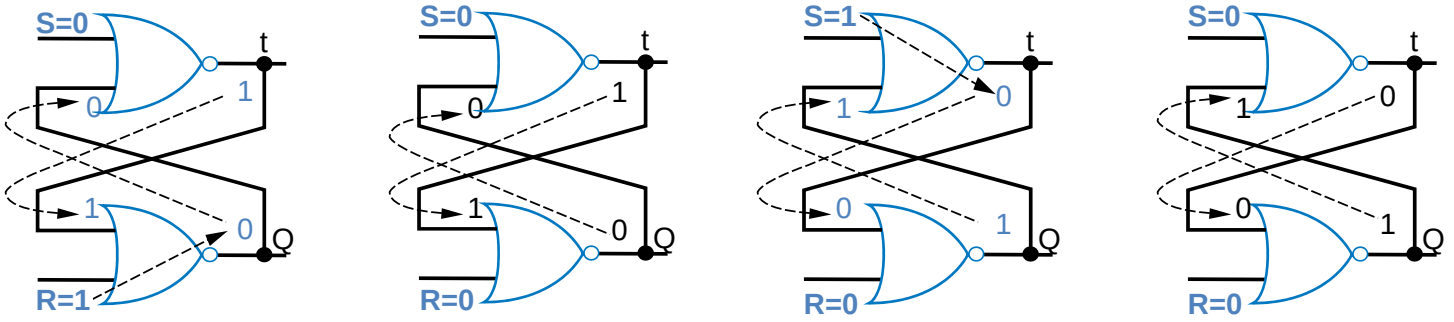


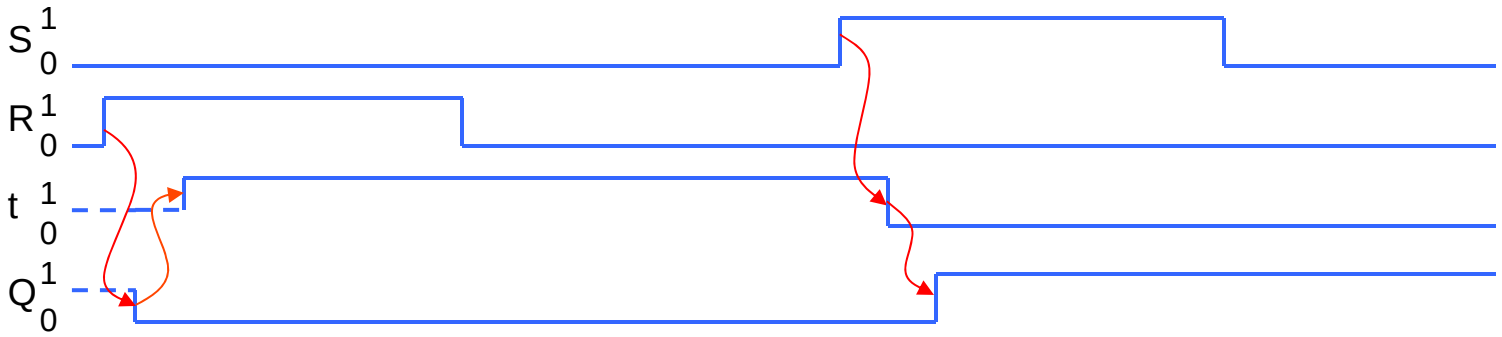


Set-Reset (SR) Latch

- The most basic circuit for storing a bit is the *SR latch*:

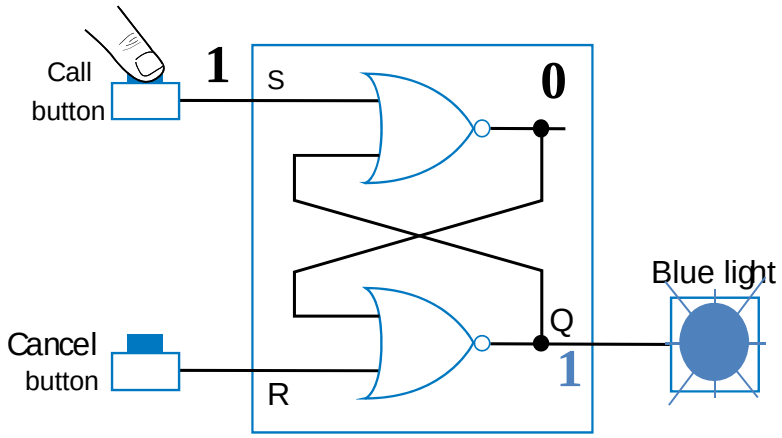






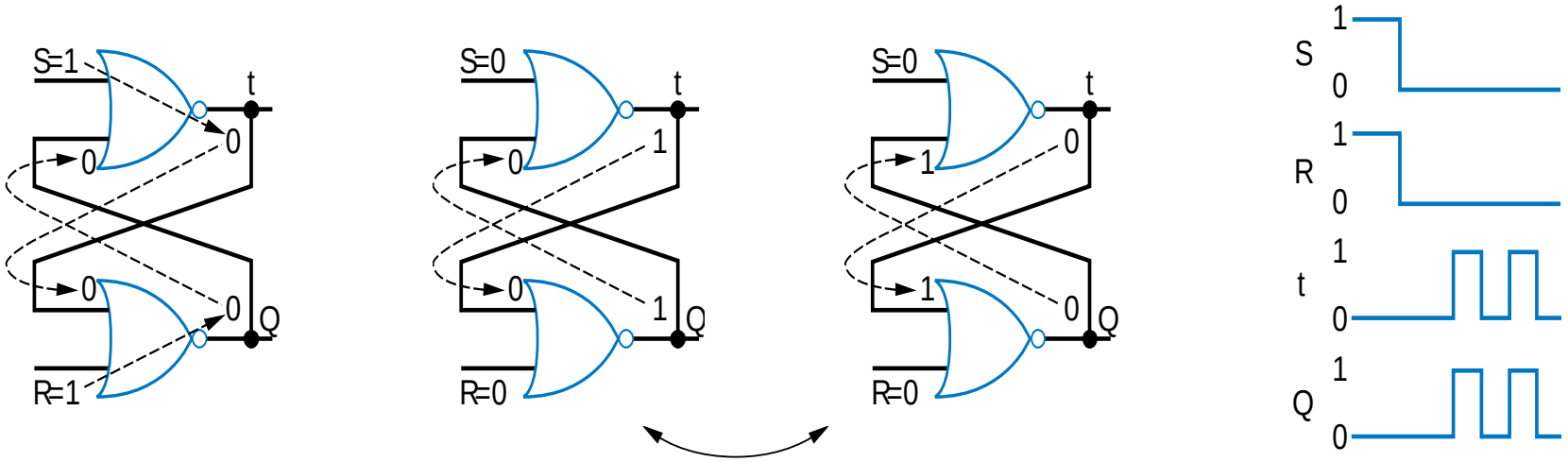
Set-Reset (SR) Latch

- The SR latch allows us to implement the flight attendant call system easily
  - Connect “call” to S, and “cancel” to R
  - Q stays at 1 even after “call is released”



Set-Reset (SR) Latch

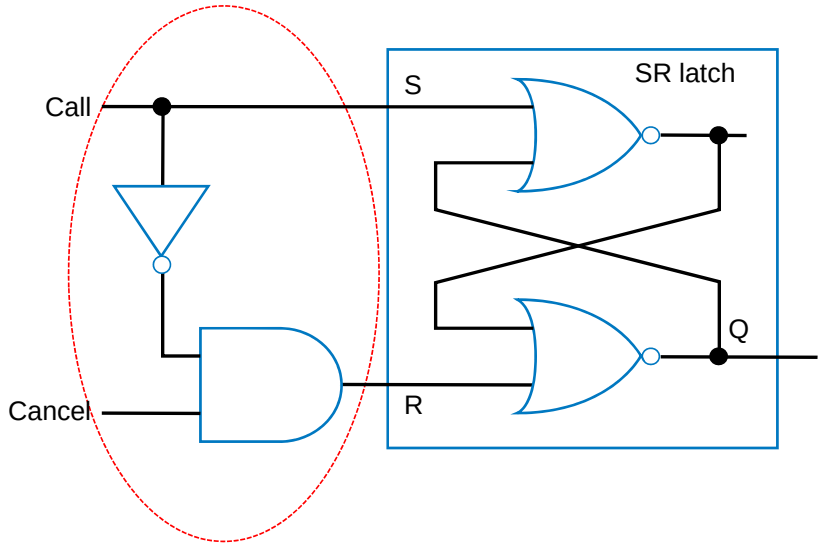
- There is, though, a problem with the SR Latch
- What happens when both S and R are asserted?

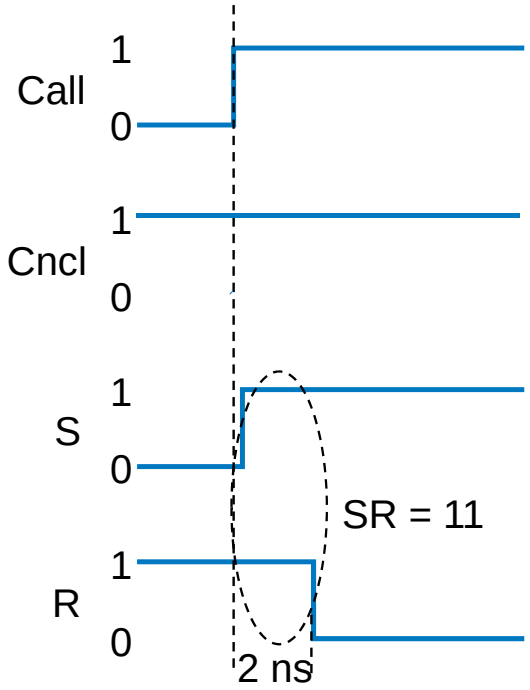


- When both S and R is then de-asserted at the same time, we may get oscillation at the output:
  - Constant swapping from 0 to 1 to 0 …
- Normally, one gate is a little faster so output will eventually stabilize, but which?
  - Impossible to know

Set-Reset (SR) Latch

- Maybe the designer can stop the problem of both S and R being 1

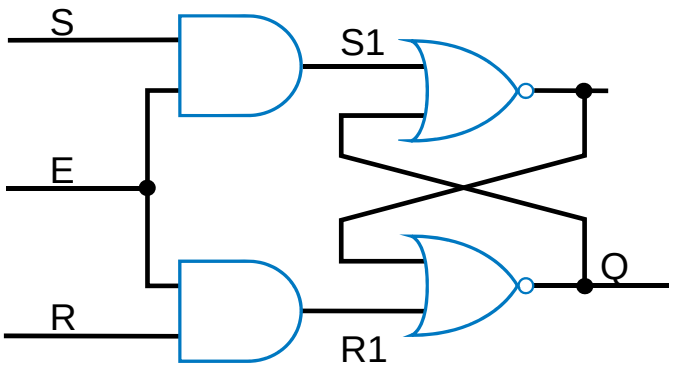


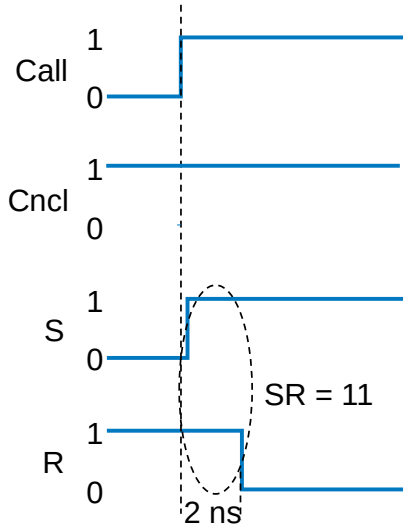


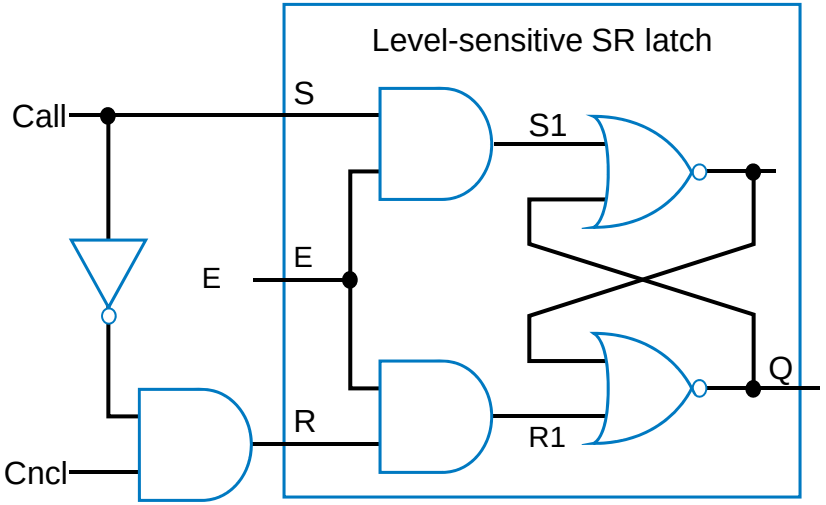
- The problem is, signals take time to travel through gates, so call to R is longer than call to S

Gated/Enabled SR Latch

- The *enabled SR latch* will only respond to inputs when E is high – *level-sensitive*
- Can now ensure S and R never both high when E high – glitches unimportant

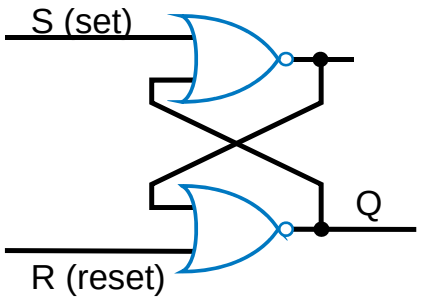






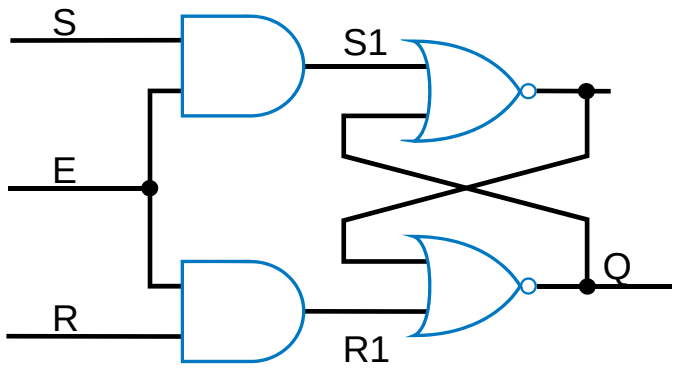
SR Latches

- We can capture the behavior of the above latches using a truth table:
  - Q means the current state
  - Q+ means the next state



SR Latch

S	R	Q+	Function
0	0	Q	Store
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined



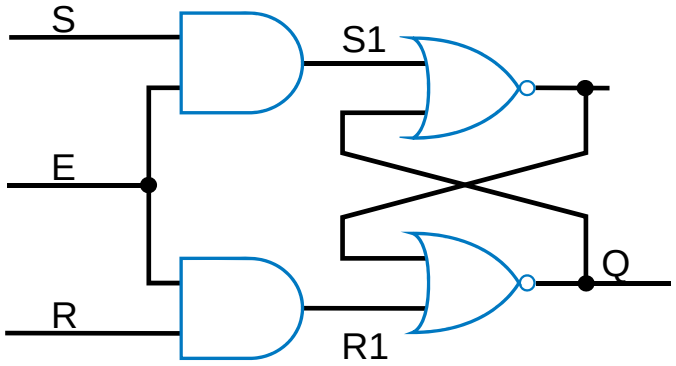
Enabled SR Latch

E	S	R	Q+	Function
0	X	X	Q	Store
1	0	0	Q	Store
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	?	Undefined

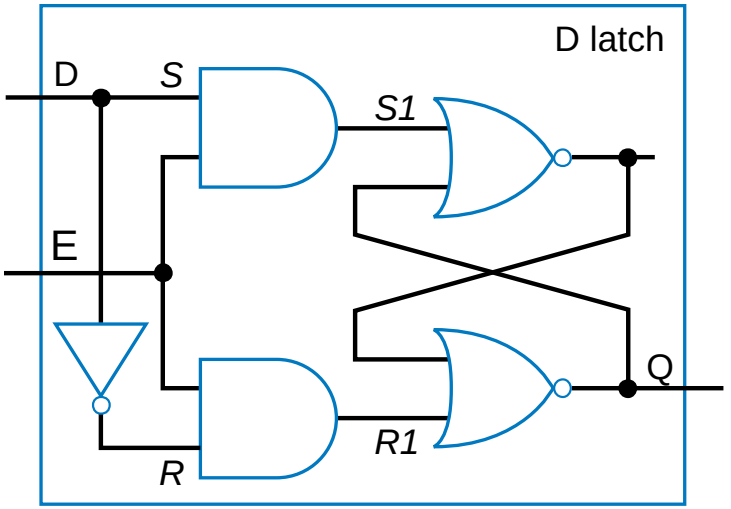
Transparent D Latch

- The D latch, also called the transparent latch
- Removes the possibility of invalid inputs
  - Inverter ensures R is always opposite of S
  - A single input controls both states of the latch

Enabled SR Latch



E	S	R	Q+	Function
0	X	X	Q	Store
1	0	0	Q	Store
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	?	Undefined

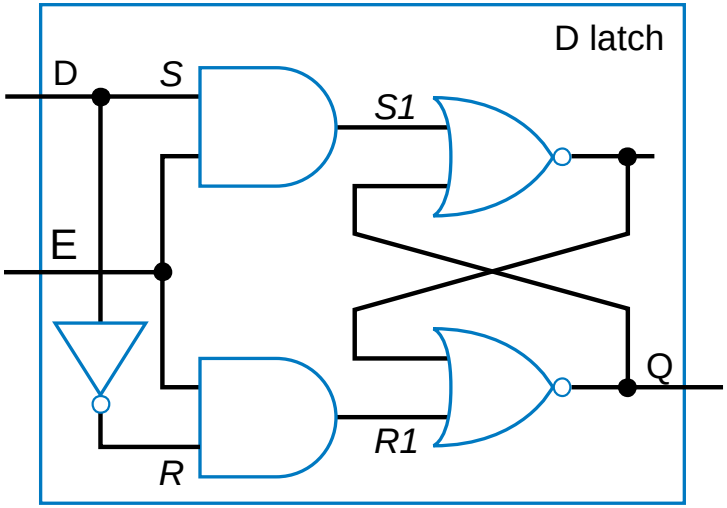


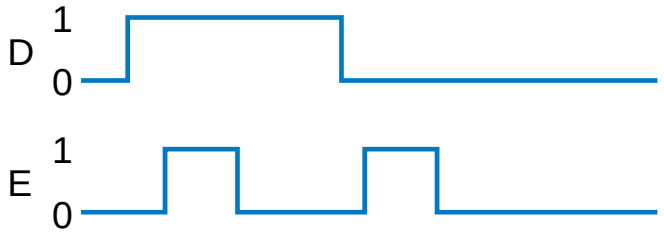
E	D	Q+
0	X	Q
1	0	0
1	1	1

Transparent D Latch

- The D latch effectively passes its input through to its output whenever the enable input is high
- Hence, “transparent”
- If enable is low, the output is held
- Truth table for a D latch:

E	D	Q+	Function
0	X	Q	Store
1	0	0	Transparent
1	1	1	Transparent

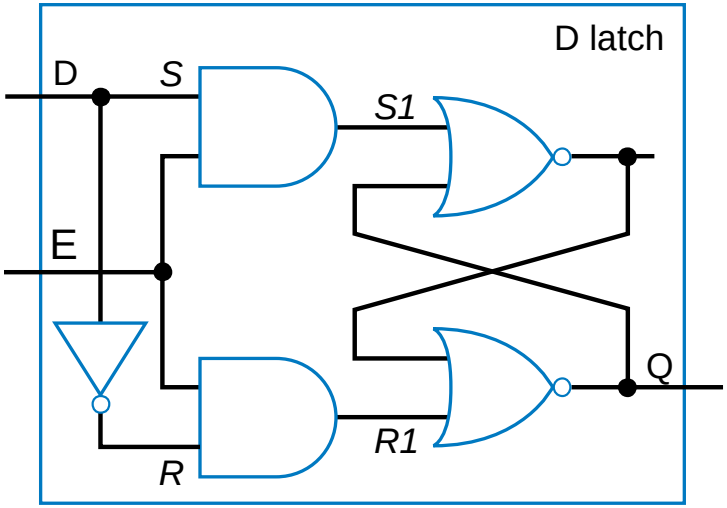


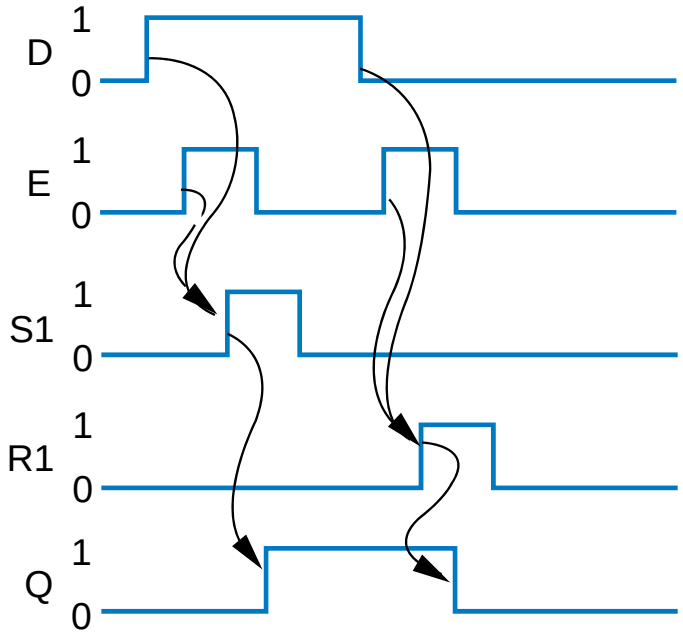


Transparent D Latch

- The D latch effectively passes its input through to its output whenever the enable input is high
- Hence, “transparent”
- If enable is low, the output is held
- Truth table for a D latch:

E	D	Q+	Function
0	X	Q	Store
1	0	0	Transparent
1	1	1	Transparent

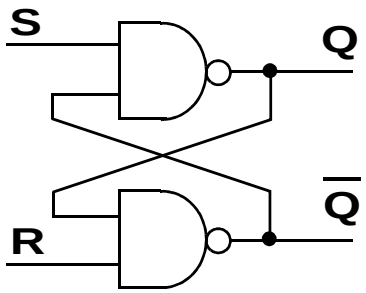




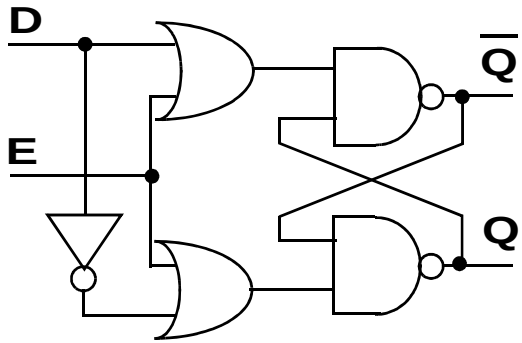
Active-Low Latches

- Note that it's possible to create active-low versions of the latches previously shown
- Swap the **nor** for **nand** and the **or** for **and**:

Active-Low SR Latch



Active-Low Transparent Latch



E	D	Q+	Function
1	X	Q	Store
0	0	0	Transparent
0	1	1	Transparent

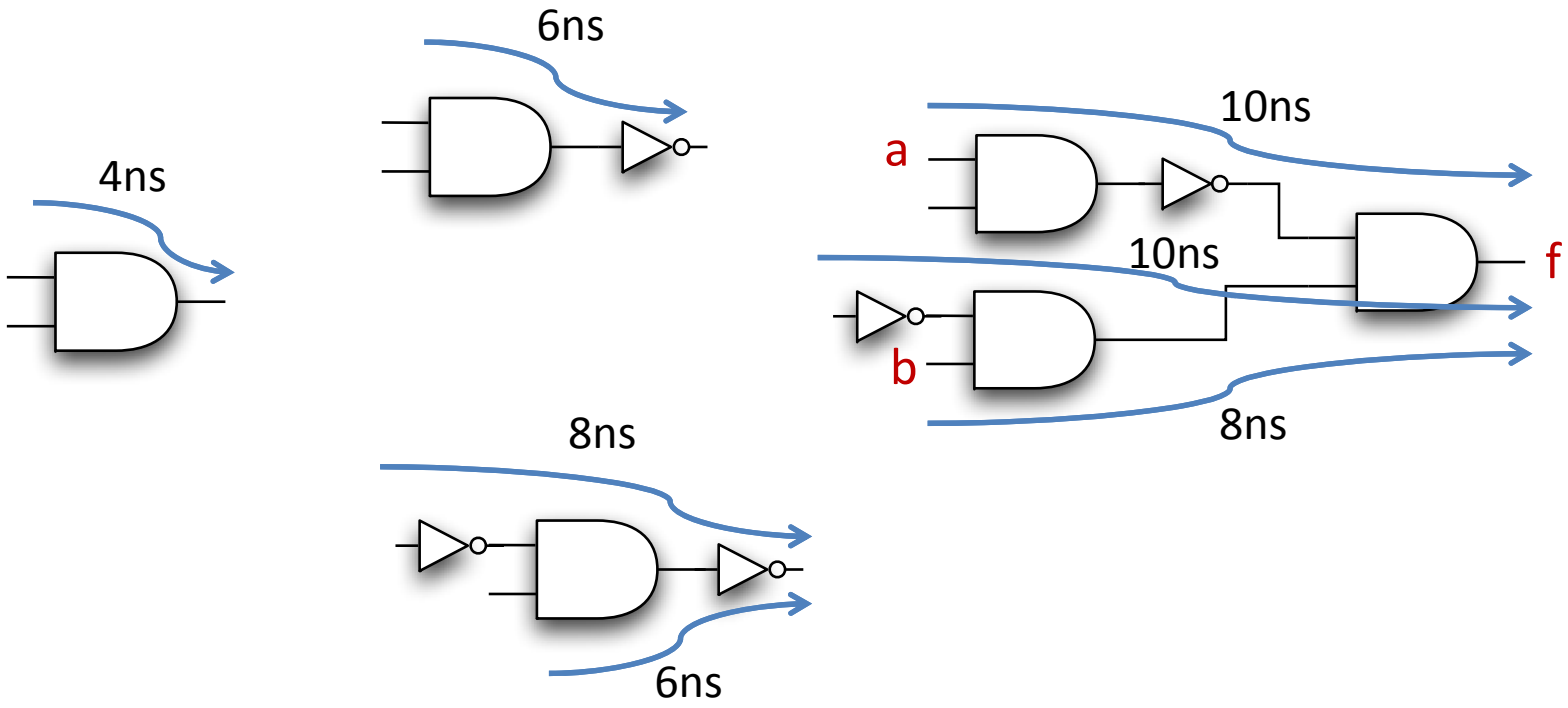
Timing in Circuits

NANYANG TECHNOLOGICAL UNIVERSITY

- So far, we have assumed that inputs cause a change in the output of a gate immediately
- The reality is it takes a short amount of time
- We call this the propagation delay
- This is the time it takes for a change at the input to create the requisite change at the output
- We can work out the propagation delay through a circuit by summing the propagation delays of the individual gates

Timing in Circuits

- Assume an **and** gate has a delay of 4ns and an **inverter** has a delay of 2ns:

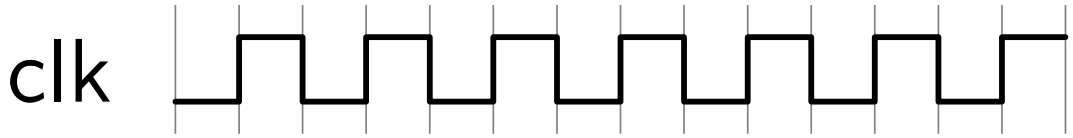


Timing in Circuits

- The propagation delay, t_p, may differ between different inputs and outputs in the same circuit
- Hence we can give the propagation delay for each input-output pair:
  - a input to f output t_p = 10ns
  - b input to f output t_p = 8ns
- If we report a single number, we always use the worst (i.e. longest path)

The Clock

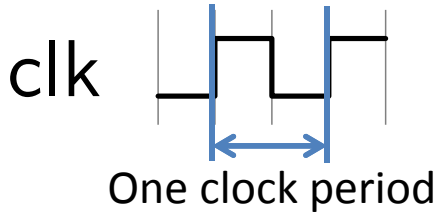
- In digital systems, sequential components are connected to what we call a "clock"
- The clock is a signal that continuously toggles between 0 and 1 at a fixed rate:



- It is called a clock because it oscillates much like the "tick-tock" of a clock
- This is where the MHz and GHz numbers for chips come from – how fast does the clock oscillate?

The Clock

- The period of the clock is the time it takes to complete one complete cycle – we call this a cycle



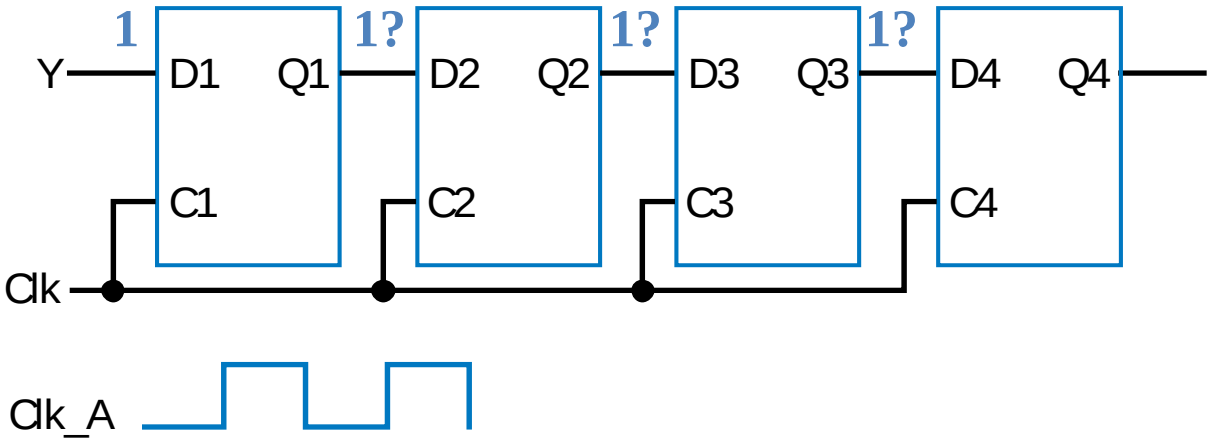
- The frequency is the inverse of the period:
  - 10ns period = 10×10^{-9}s, $1/10 \times 10^{-9} = 10^{8}$Hz = 100MHz
  - 1ns period = 1×10^{-9}s, $1/1 \times 10^{-9} = 10^{9}$Hz = 1GHz
  - 4ns period = 4×10^{-9}s, $1/4 \times 10^{-9} = 2.5 \times 10^{8}$Hz = 250MHz

Timing of Latches

Truth table of a D Latch

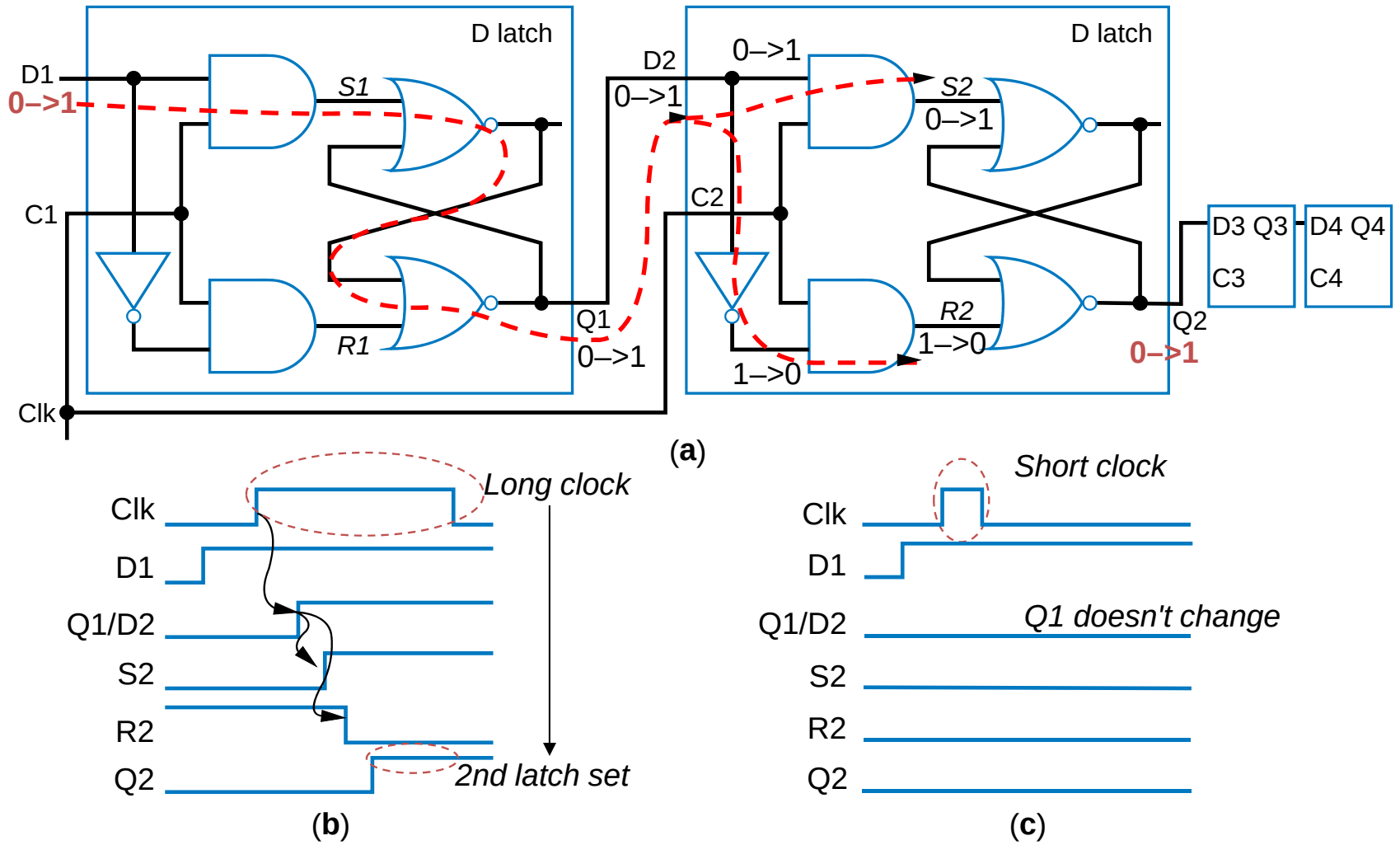
E	D	Q+	Function
0	X	Q	Store
1	0	0	Transparent
1	1	1	Transparent

- Consider a chain of latches:



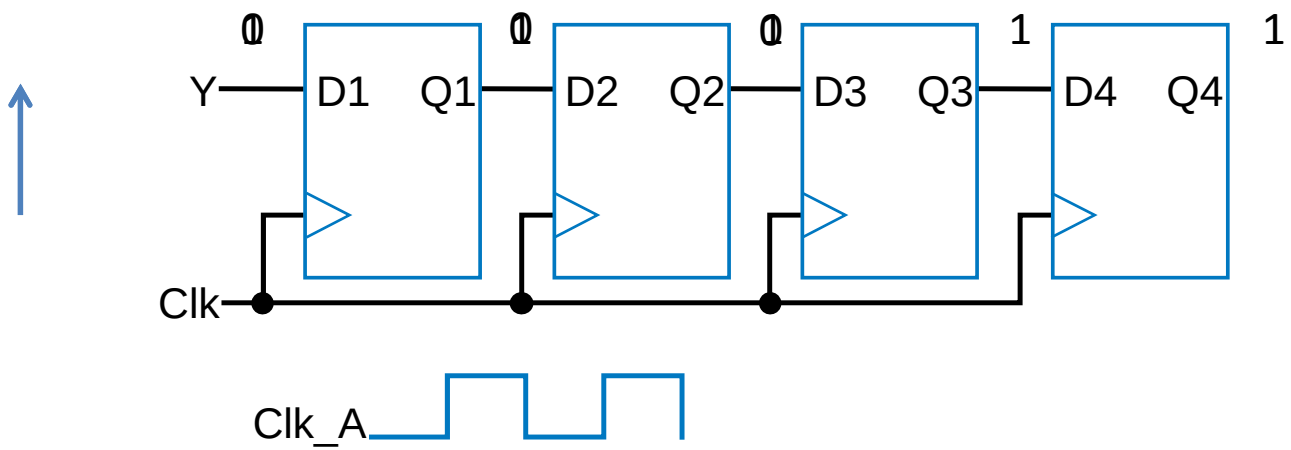
- When Clk_A is 1, how many latches will the input pass through?
- With Clk_A, it may be more, Clk_B, fewer

Timing of Latches



Edge Triggered Flip-Flops

- We can use an edge triggered flip-flop to overcome the limitation of D latches
- At each clock rising edge each flip flop will take its D input and produce that value on its Q output
- How do we implement an edge triggered flip flop?



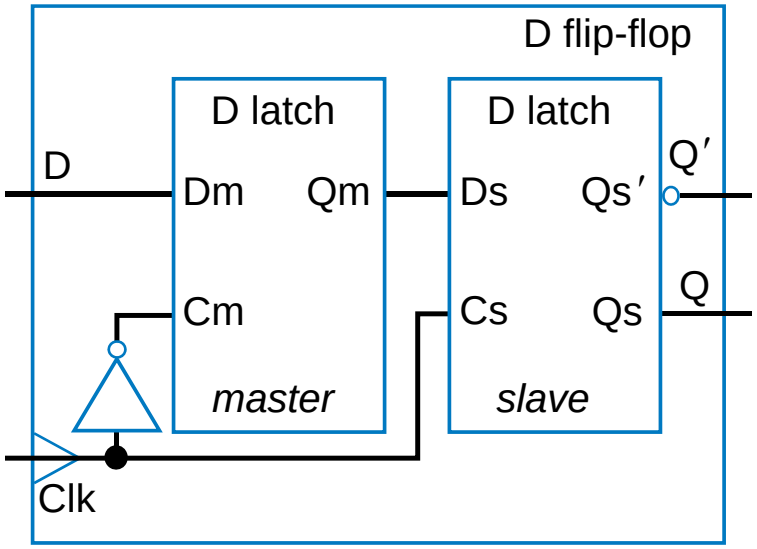
Edge-Triggered Flip-Flops

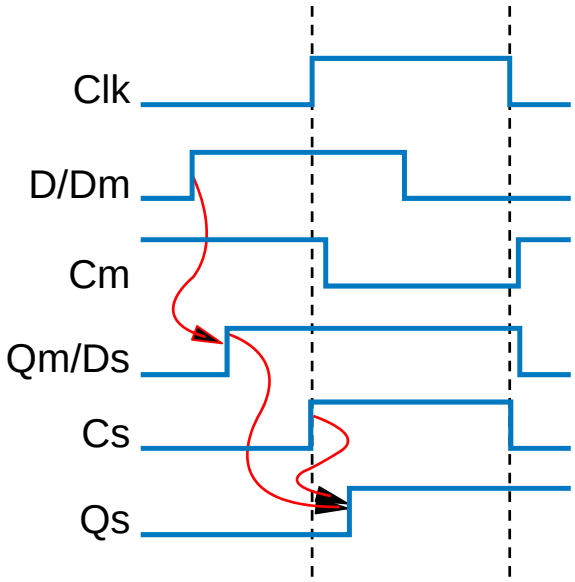
NANYANG TECHNOLOGICAL UNIVERSITY

- Master-Slave implies two latch stages:
  - Master stage is enabled when control is low, and samples the current input, while slave is disabled, locking it in the middle
  - When control goes high, the master is disabled, and the slave uses the value from the middle to determine its output
  - Hence, the only time an output will change is at the rising edge
  - For a negative-edge triggered setup, the master responds to high enable, and the slave to low enable

Edge-Triggered Flip-Flops

- The master and slave are each just a D latch:



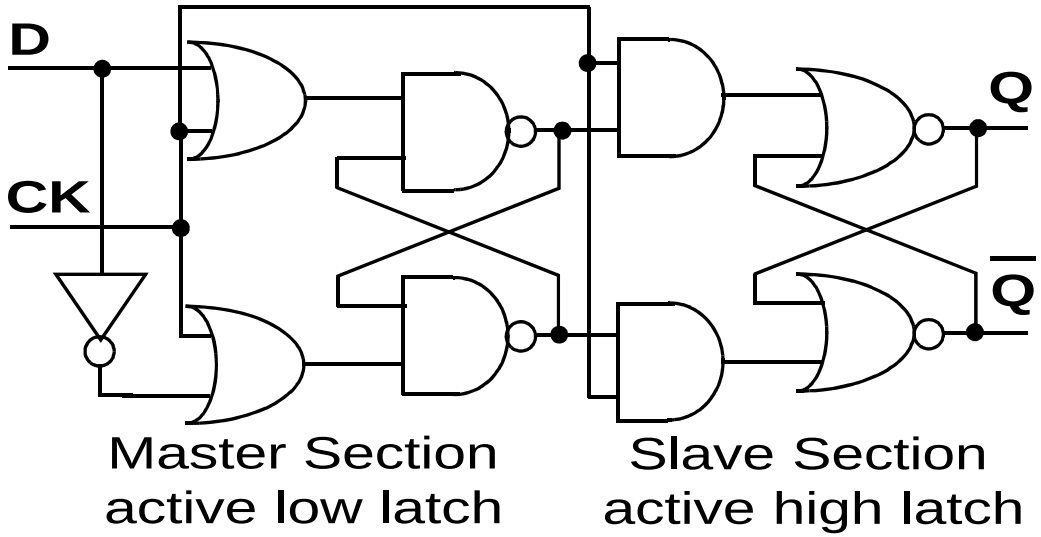


Truth table of a D Latch

E	D	Q+	Function
0	X	Q	Store
1	0	0	Transparent
1	1	1	Transparent

Edge-Triggered Flip-Flops

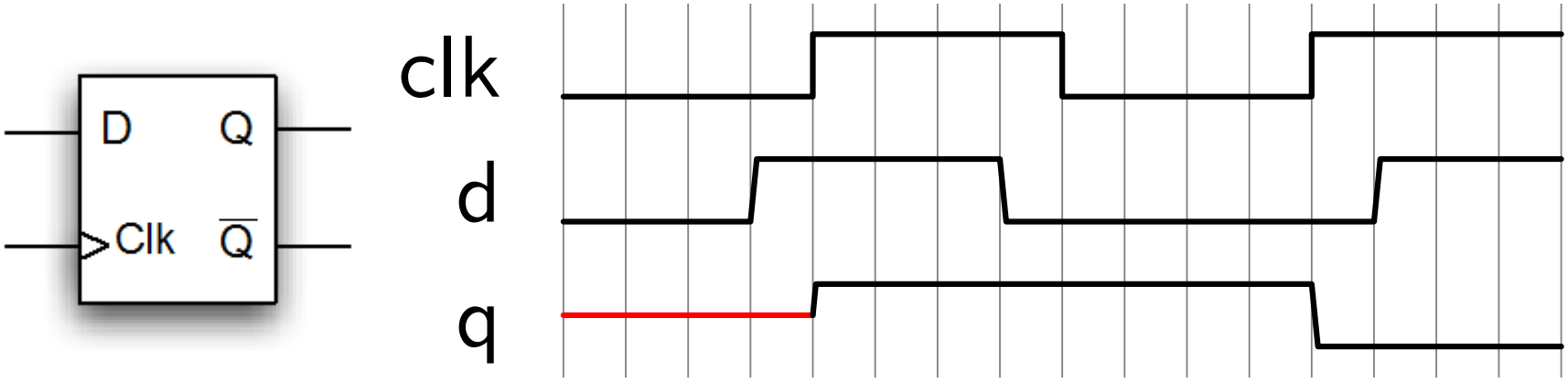
- Rather than invert the clock as in the last slide, we can use an active low version (more correct)
- Example design of D flip-flop:



- Note that there are many possible designs, and many are more compact than above

Edge-Triggered Flip-Flops

- The timing diagram for for a D-type flip-flop is as follows:



- Note transitions on the output only occur at the (rising) clock edge

Edge-Triggered Flip-Flops

NANYANG TECHNOLOGICAL UNIVERSITY

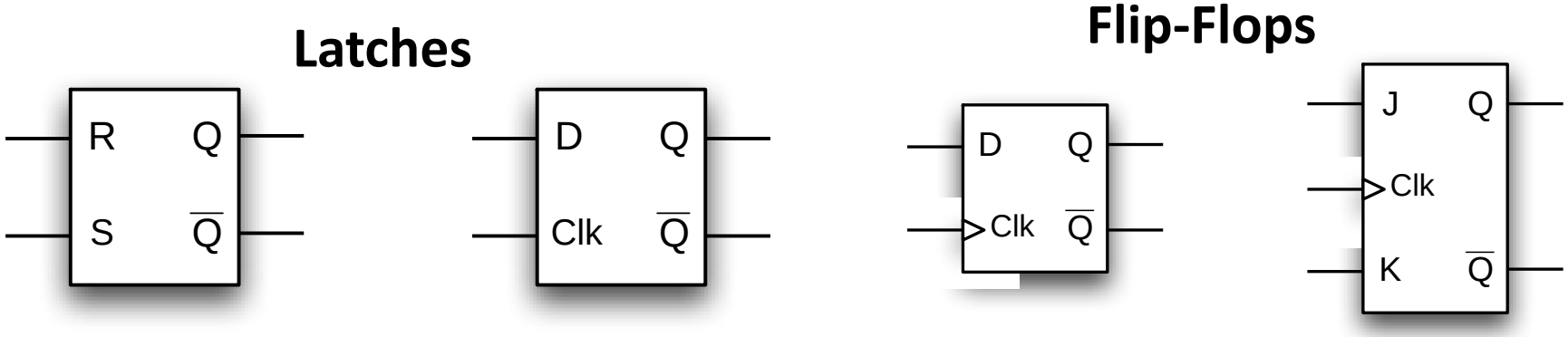
- **Latches** can be level-sensitive, i.e., their outputs change when the control (enable) input is high

- A **flip-flop** is a circuit that changes its outputs only at the **control signal's edges**
  - **Positive edge-triggered**: output changes when control goes from 0 to 1
  - **Negative edge-triggered**: output changes when control goes from 1 to 0

- We can build such circuits using a master-slave setup

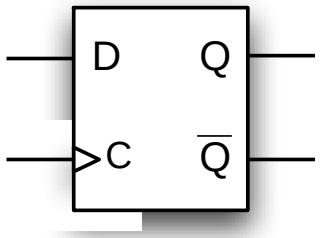
Edge-Triggered Flip-Flops

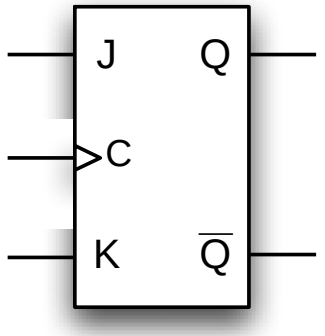
- A few notes:
  - In this course, *latch* means **level-sensitive**, *flip-flop* means **edge-triggered** – correct modern convention
    - Flip-flops can be **positive edge-triggered** or **negative edge-triggered**
    - In our lectures, we will assume that flip-flops are positive edge-triggered unless we specify otherwise
  - Some older references may not follow this convention, so **be careful** when using other resources
  - Good news: the symbol for an edge triggered component always has a small triangle on the control input:

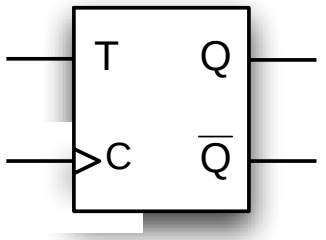


Edge-Triggered Flip-Flops

- We've seen the D flip-flop:
  - Q takes the value on D at the rising edge of C
- There's also a J-K flip-flop:
  - Q becomes 1 if J is asserted at the rising edge and 0 if K is asserted
  - If both J and K are asserted, the output toggles at the rising edge
- And a T flip-flop:
  - Q toggles at the rising edge if T=1
- (Also a lesser seen SR flip-flop)







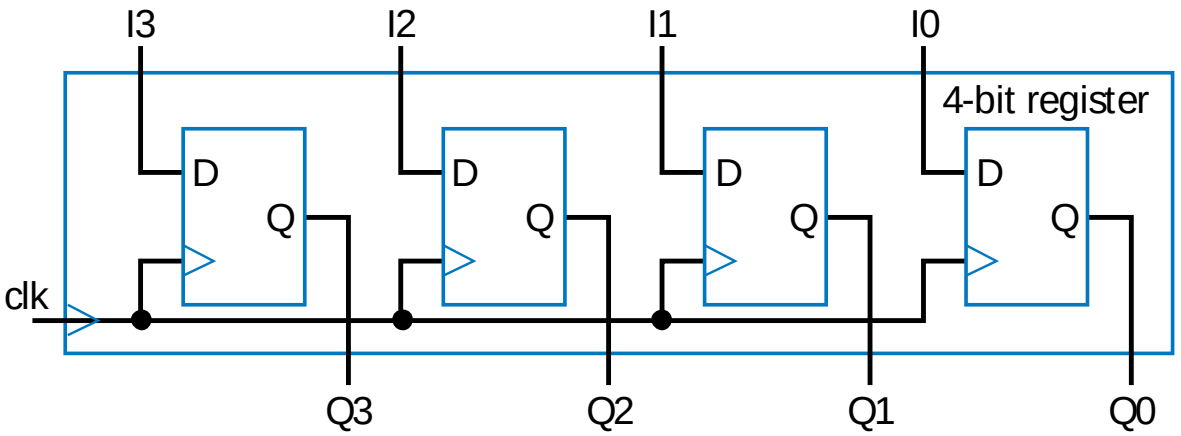
The Clock

NANYANG TECHNOLOGICAL UNIVERSITY

- In synchronous design, we generally use one clock for all the sequential components in our circuit
- Hence, the outputs of all those blocks will only update at the clock (rising) edge
- That's why prefer (edge-triggered) flip-flops
- That way, we can more easily analyze our circuit, and model its behavior

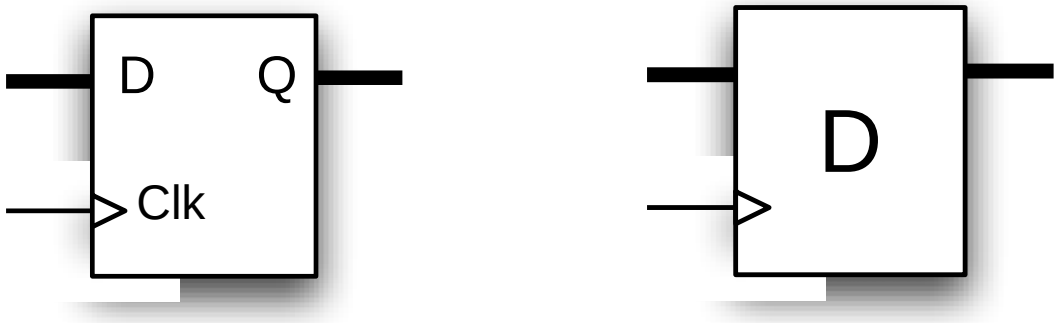
Registers

- The D-type flip-flop is the fundamental building block in synchronous design
- But we often deal with multi-bit signals
- When we combine multiple D flip-flops together to store multiple bits, we call this a **register**:



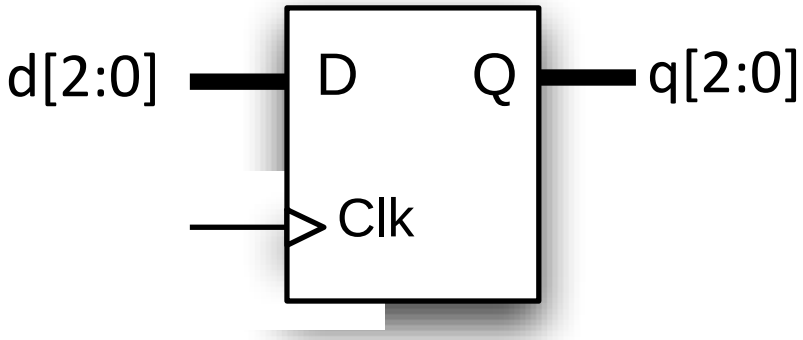
Registers

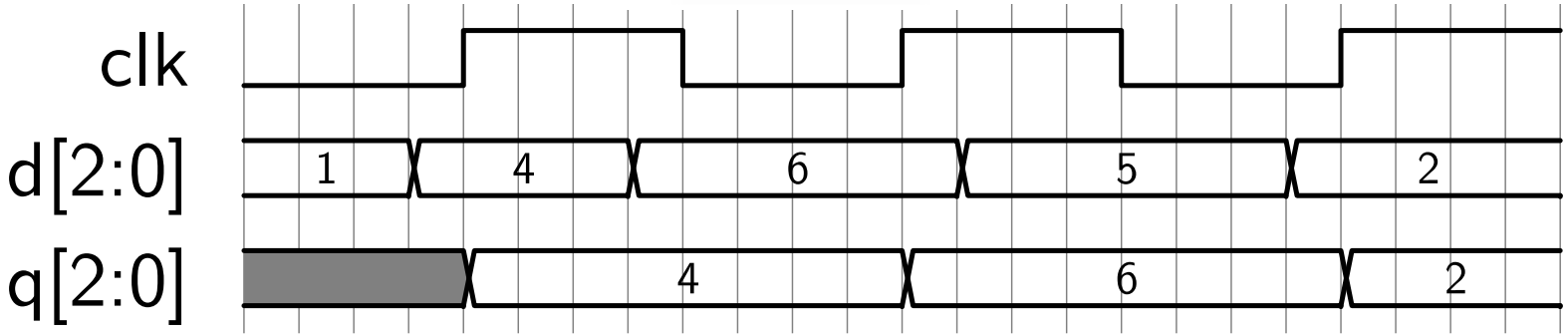
- Henceforth, we can use only registers as our fundamental sequential component
- We now know how they are constructed internally, but we no longer need to worry about latches and flip-flops
- Even a 1-bit signal is stored in a “register”
- We can use a similar symbol to that used for a D flip-flop:



Registers

- When dealing with multi-bit registers, we can show these values in a timing diagram without splitting up the bits:





- We can use decimal or hex, but make it clear
- Again, transitions only happen at the rising edge

Registers

- What about this arrangement?