

Lecture 8

Sequential Circuits



Outline

- Sequential circuit operation (***)
- Setup and hold timing (***)
- Textbook: 10.1, 10.2



Combinational Logic

1. Output is a **pure function of the CURRENT inputs**
 - Described as a logic equation
2. Memory-less
 - Following from (1), output is independent of past values/history
3. Timing
 - Computation starts as soon as
 - any input switches to a different value
 - There is **no explicit synchronization mechanism to tell each logic OPERATOR when to start processing** a new set of inputs or when a new output from the network is ready



Combinational Circuits

- ***No memory***
- No control over ***when*** the inputs should be ***“read”***
 - The timing of the outcome of computation is decided purely by the individual propagation delays of logic gates
 - As soon as an input changes value computation restarts

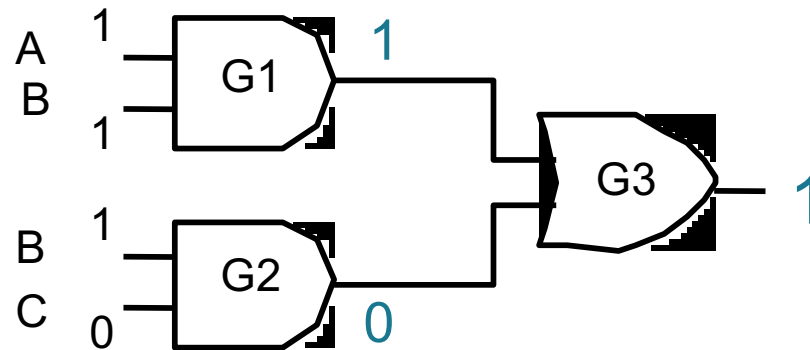


Asynchronous Behavior of Combinational Logic

- Fundamentally combinational logic actually behaves asynchronously



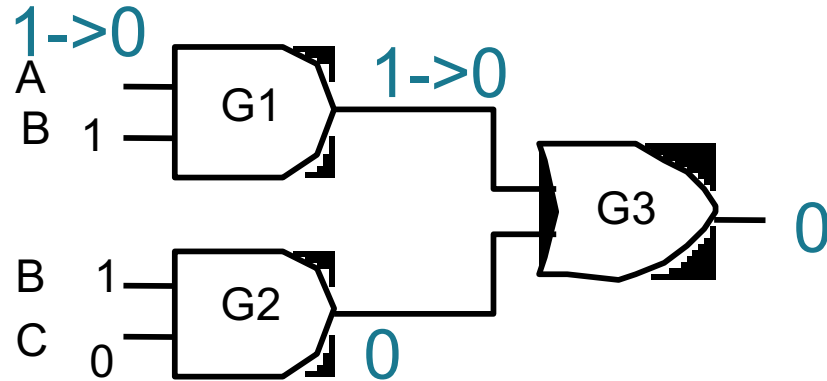
A Combinational Circuit



Consider the input values $ABC = 110$
Corresponding output = 1



A Combinational Circuit



Consider input A switching 0 to 1

$$ABC = 010$$

If all gates compute outputs instantly

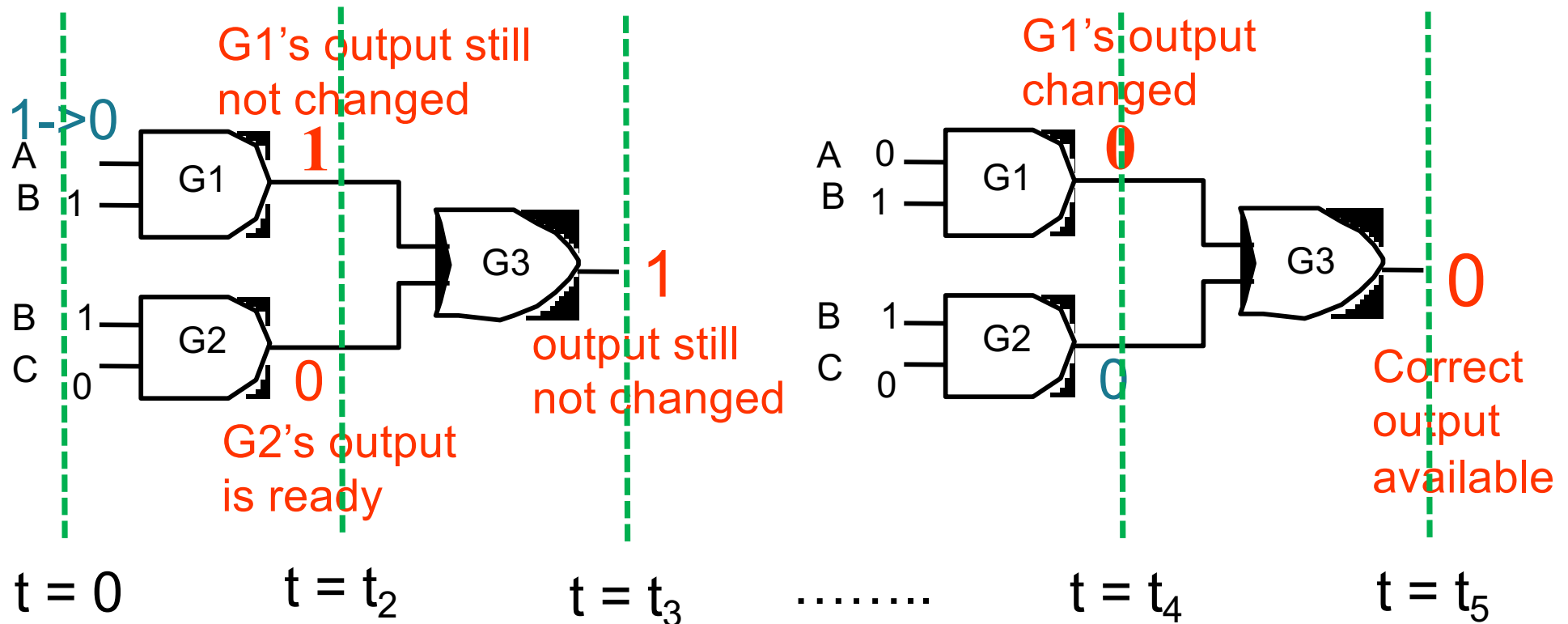
The new value would be $F = 0$



A Combinational Circuit

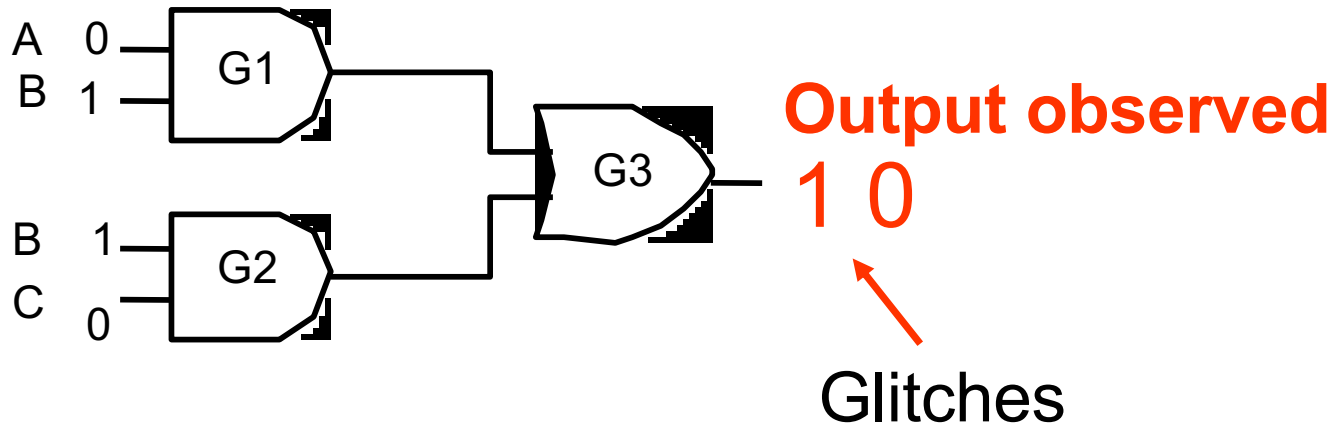
Now, assume:

- Gates G2 and G3 are very fast
- G1 is much slower than G2 and G3





A Combinational Circuit

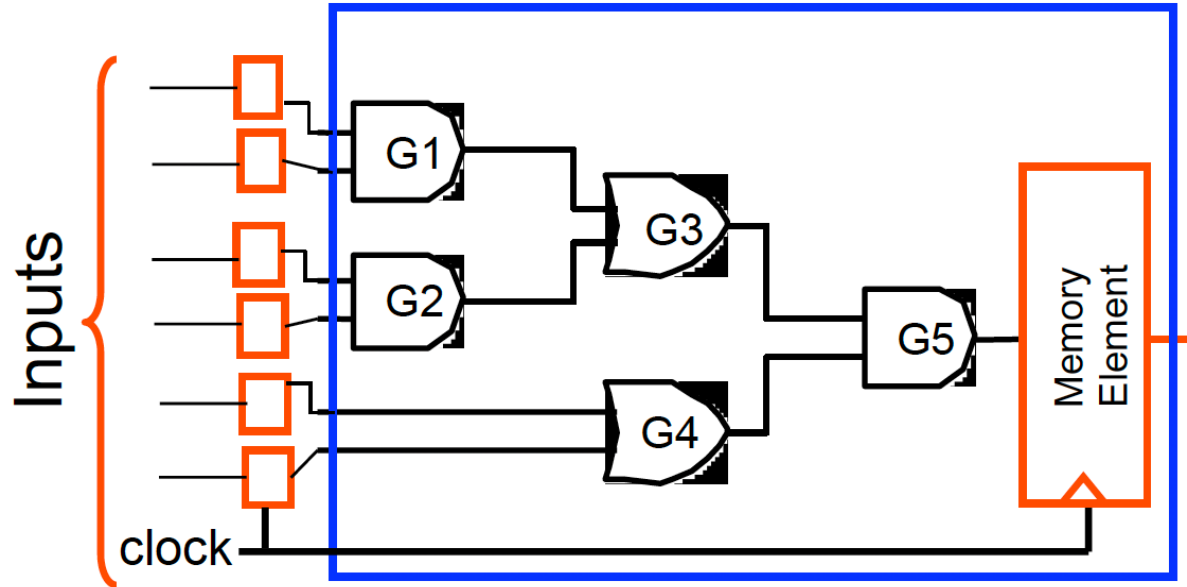


Putting all events together we see that due to the differences in propagation delays along different paths within the circuit the output has fluctuated between different values depending on the ordering of input switching and relative timing.

Glitches are inevitable in the combinational circuits which are behaving "asynchronous" in essence



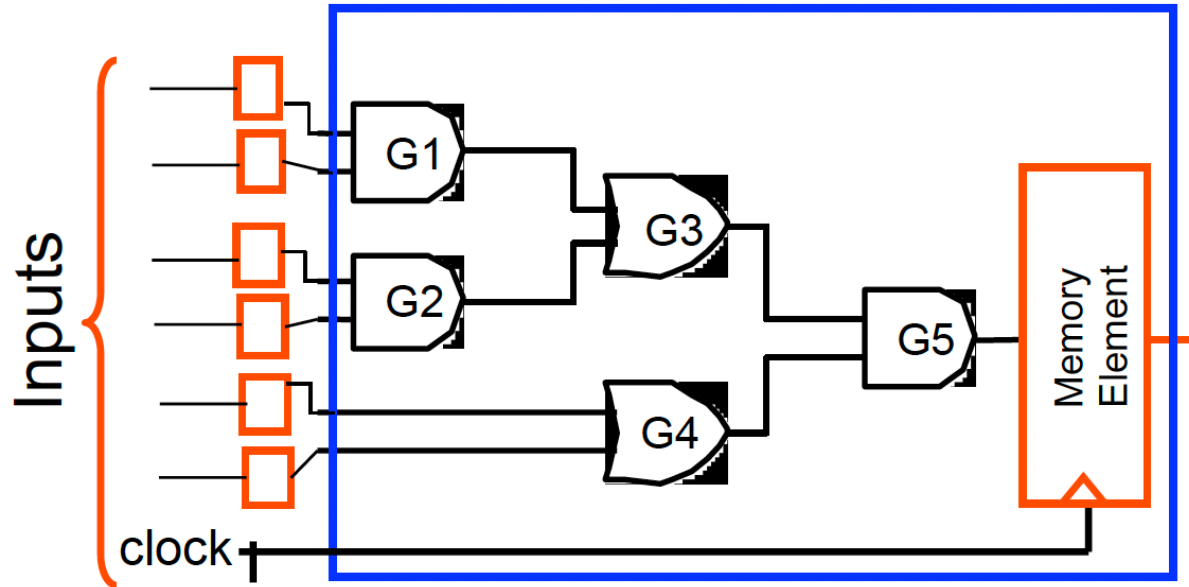
How to synchronize a circuit?



How do we know when is the RIGHT time? How frequently can we apply new inputs and expect the correct stable output?



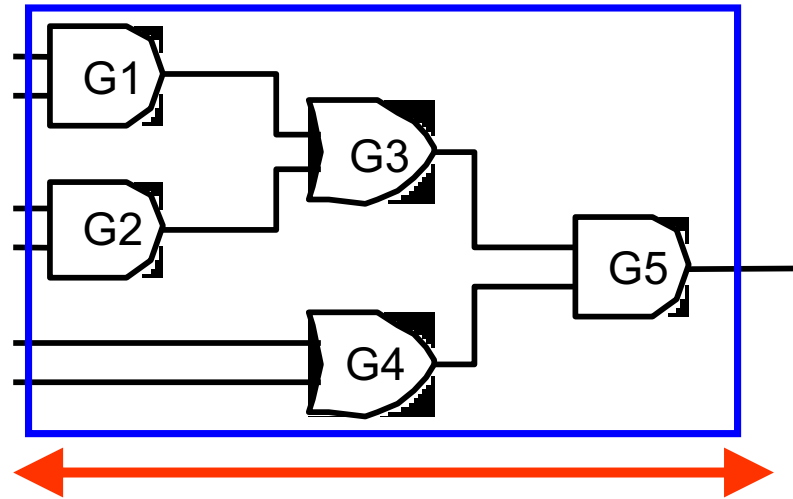
How to synchronize a circuit?



- Synchronize exchange/update of output signals
- Sample the output of the combinational circuit at certain time points within a well defined time interval; generally referred to as **clock cycle**
- A gatekeeper element - MEMORY ELEMENT – keeps track of the heartbeat of the system and relays inputs and outputs ONLY at the allowed time instances to the next consumer.



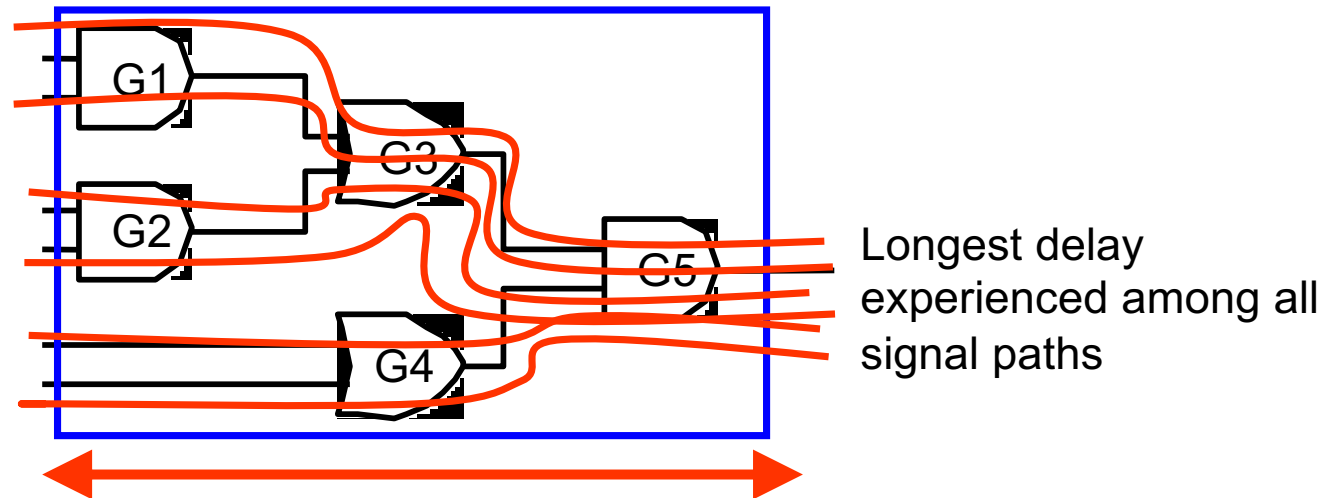
How to synchronize?



D_{comb} = Total delay of combinational logic in the worst case



How to synchronize?



D_{comb} = Total delay of combinational logic in the worst case

How frequently can we sample the output of the circuit?

At MOST at a rate equal to $\frac{1}{D_{\text{comb}}}$

Speed of circuit is statically determined by its slowest path
Worst case performance!



BIG PICTURE

- The existence of memory is essential to building a complete computing system
- ONE reason is the stability issue of combinational circuits discussed until now
- There are other reasons



Some computation cannot simply be just combinational

- *Case A: A computer is tasked with adding two numbers each time a pair of numbers are presented to it*


$$\text{OUTPUT} = \text{input1 PLUS input2}$$

- *Can represent input1 and input2 in binary format*
- *Can push the binary bits representing input1 and input2 through a network of logic gates (which evaluate the logic statements that create the behavior of addition among these bits)*
- *The answer that comes at the other end of the network will be the binary representation of the sum value*
- *Case A represents a Combinational Computing effort*
- *You can still consider “safeguarding” it through the memory gatekeepers we discussed in the previous slides*



*Some computation cannot simply be **just** combinational*

- Case B: A computer is tasked with generating a single bit (0/1) output while continuously observing a stream of bits at its input
 - Output should become '1' if the stream observed thus far contains an odd number of '1's
 - Output should become '0' if the stream observed thus far contains an even number of '1's

Time												
Input	1	0	0	1	1	0	1	0	1	1	1	0
Output	1	1	1	0	1	1	0	0	1	0	1	1



*Some computation cannot simply be **just** combinational*

- Case B: This computer cannot “compute”, make the right decision unless it “remembers” past events!
- Whenever a digital system needs to factor in “history/past events” into its current computation step, it needs ability to REMEMBER (MEMORY)
- Digital circuits constructed to address this paradigm are called SEQUENTIAL DIGITAL CIRCUITS
- We take one step further to classify any and all digital circuits that contain MEMORY as sequential circuits
 - So, combinational computation that include memory elements for safeguarding are also categorized as sequential



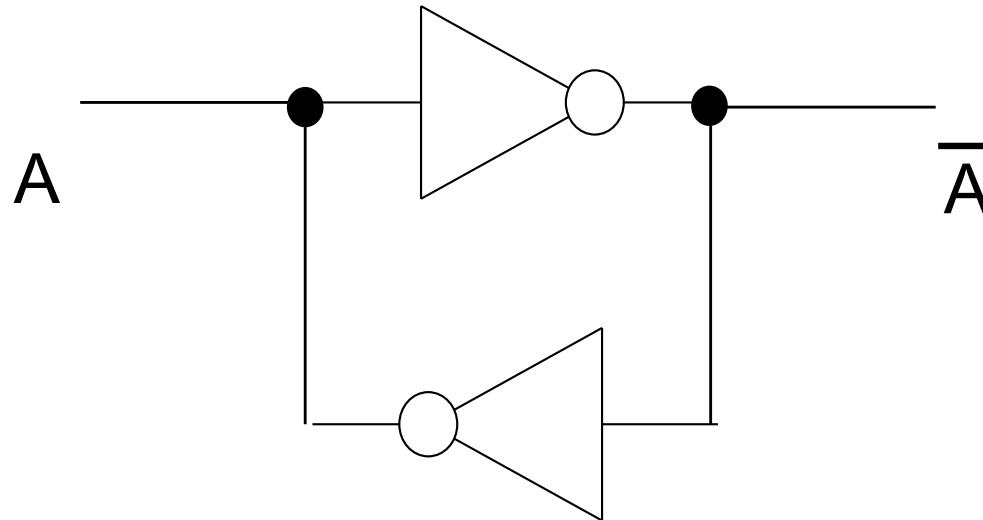
Memory Elements

- We need memory to (1) synchronize operations, (2) remember history
- Two basic types
 - Latch
 - Flip Flop



Feedback and Memory

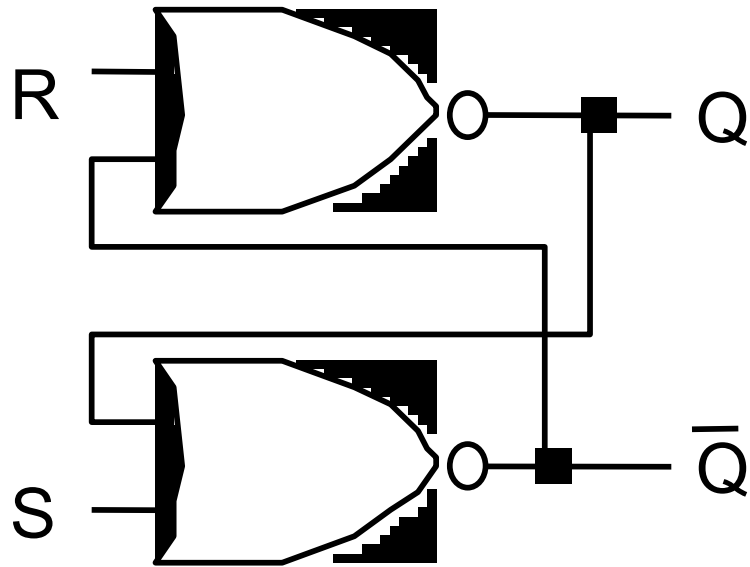
- Feedback is the root of memory
 - Can compose a simple loop from inverters



- However, there is no way to switch new value



Reset/Set Latch



S	R	Q	Q'	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	Maintain Q
1	1	0	0	undefined

Has to operate the device without setting $S = 1, R = 1$

Less evolved version

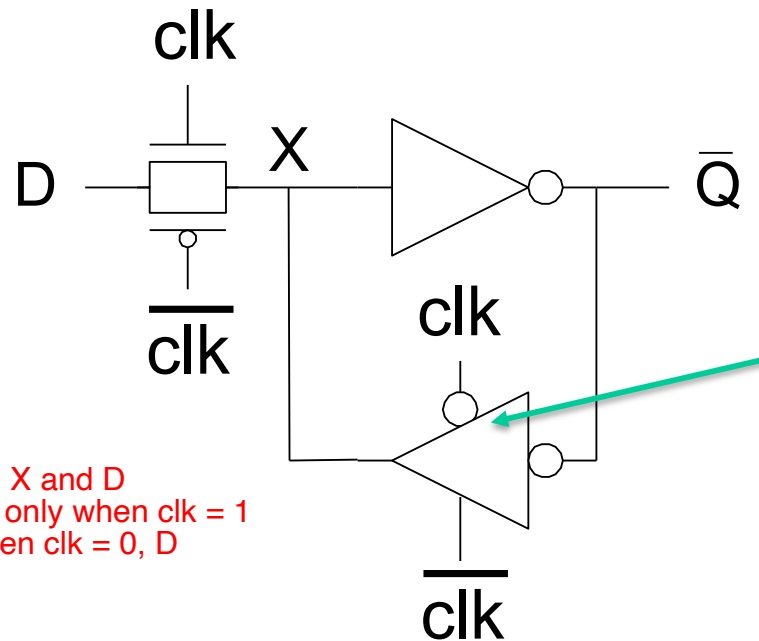
Do not use clock, energy efficient, micro-level control



Better, with only needing one signal - that's why clock is helpful

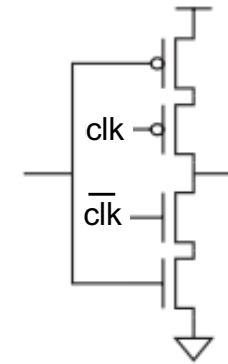
Clock =

Clocked Latch Design



The structure between X and D allows D to be passed only when $\text{clk} = 1$
The other half time when $\text{clk} = 0$, D

Tristate buffer



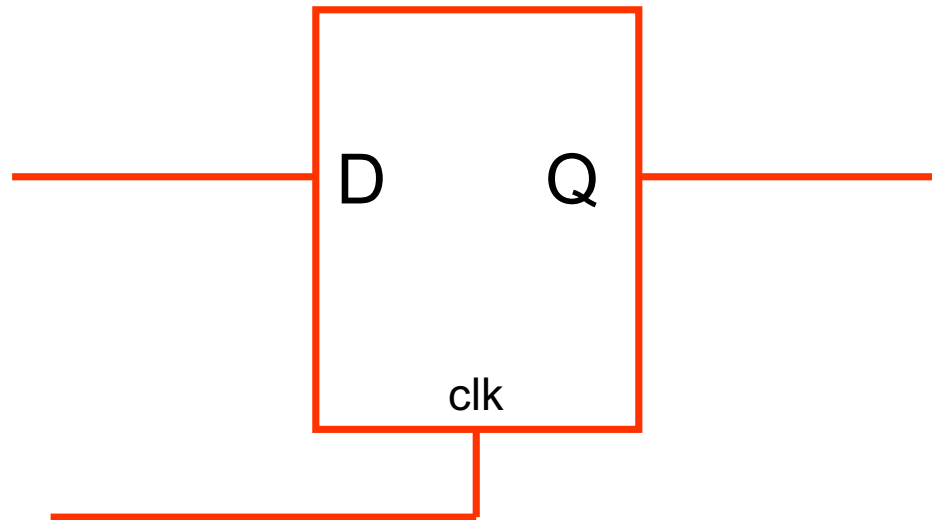
High Z
state when
 $\text{clk}=1$

- Add tristate buffer as feedback
- Passing data: clock is high; disable feedback
- Retention: clock is low; enable feedback
- Inverting output; Add another inverter to create Q



Latch

- Inputs: Data and Clock signals
- Sets output Q to value of input D when clock (Clk) is **high**; otherwise last Q value is retained

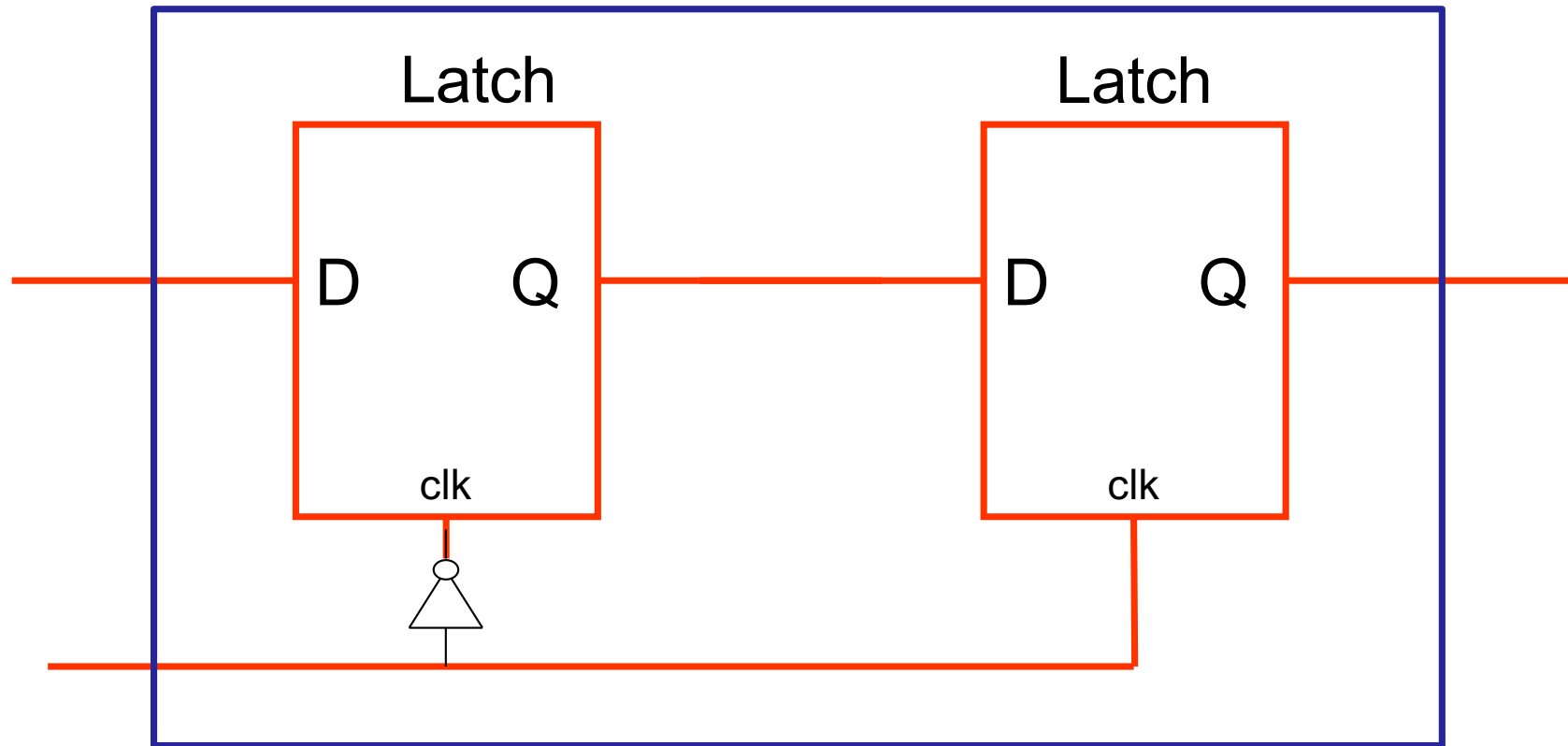


Clk	Q
1	Q=D
0	Q retains



Rising Edge Triggered Flip-Flop

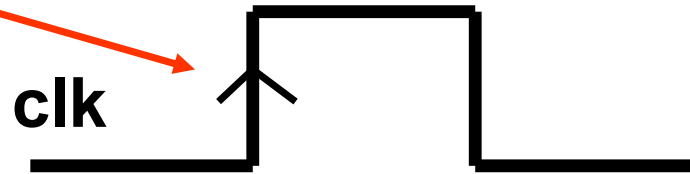
- Flip-flop is two latches in series





Rising Edge Triggered Flip-Flop

Only sample and update data at this moment



- Uses two stages of latches
- Receives a clock signal input and data input
- At the specific time instance, when the clock signal makes a transition from “0” to “1”, then it sets its output equal to the input supplied at that instance
- Retains the current value at all other times



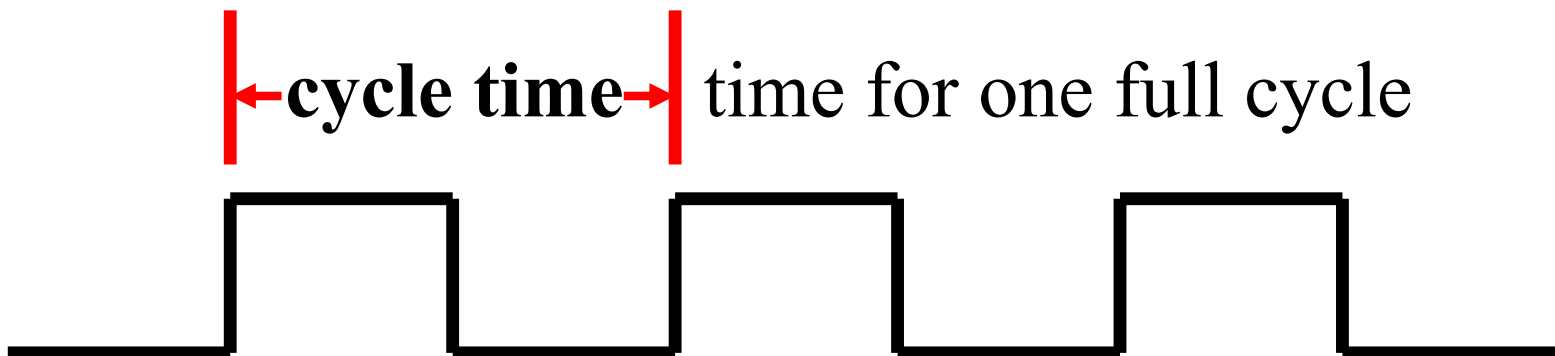
Falling Edge Triggered Flip-Flop

- Works with the same principle, only update at falling edge of clock



Computer's Internal Clock

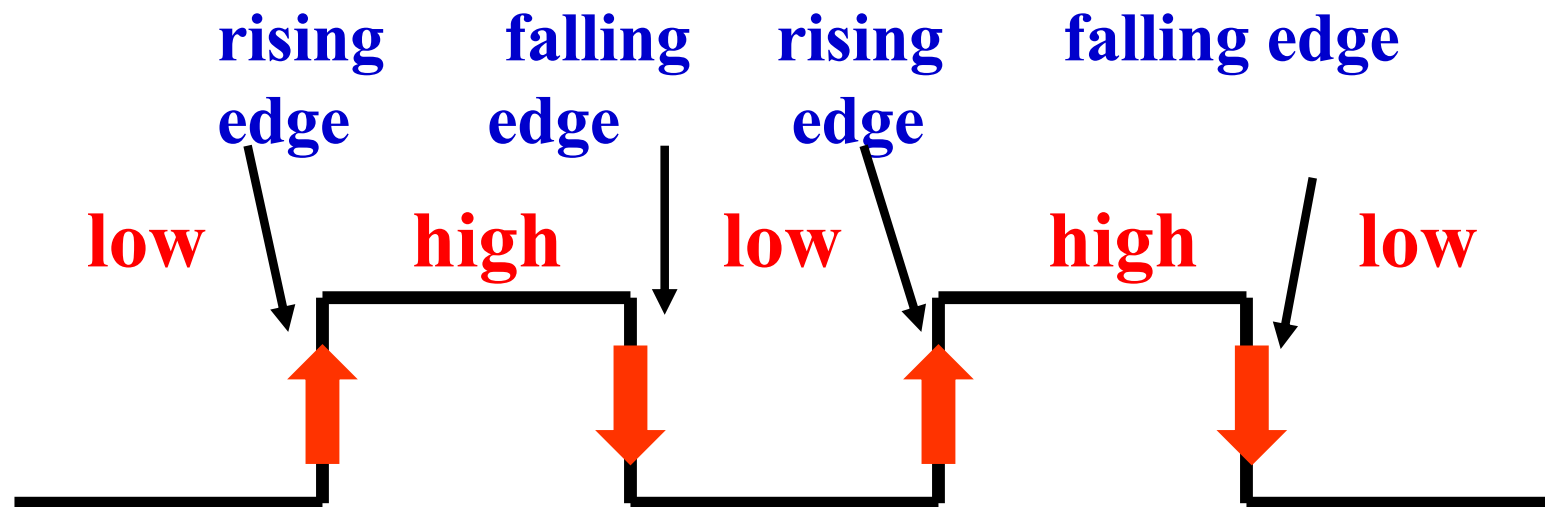
- Series of electrical pulses with a fixed interval between pulses (**cycle time**)
- Clock Cycle Time = clock period (sec/cycle)
= 1/clock frequency (cycles/sec, Hz)
- **Example: 800 MHZ machine = 800×10^6 cyc/sec**
= 1.25×10^{-9} sec/cyc = 1.25 ns/cyc





Computer's Internal Clock

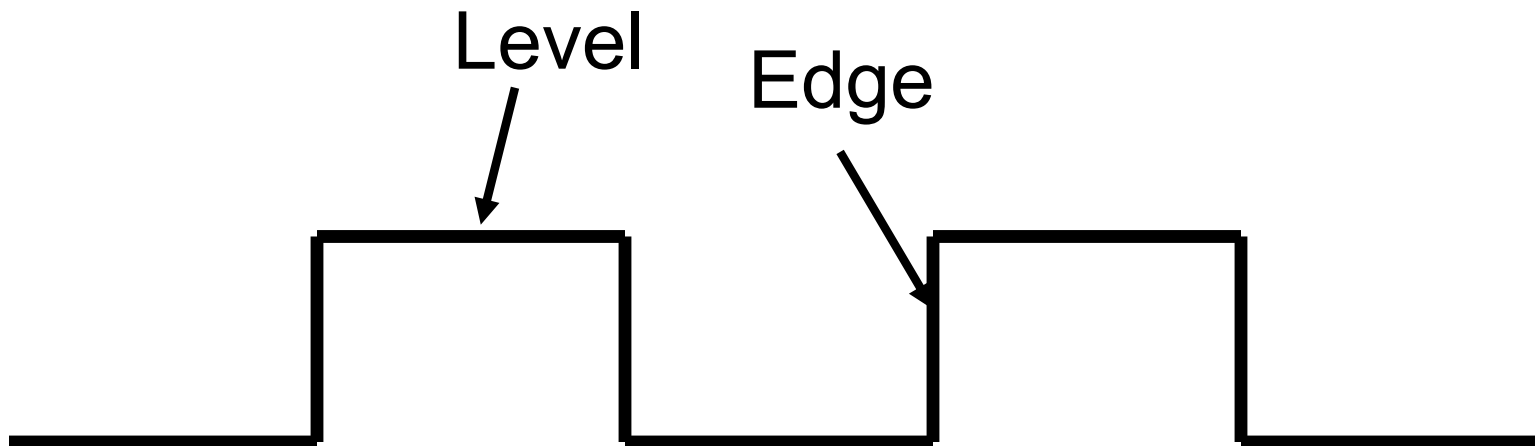
- Clock signal consists of periodic alternating high/low voltages
- **“clock is high”** or **“clock is low”**
- **“clock is falling”** or **“clock is rising”**





Clocking Methodology

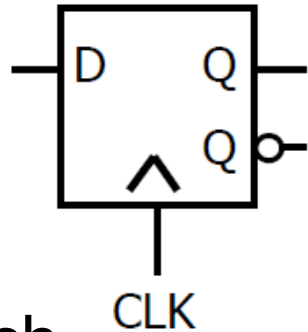
- How is clock used to regulate state changes
- **Edge-triggered:** *(specifies a specific instant)*
 - Falling edge: when clock falls, update state
 - Rising edge: when clock rises, update state
- **Level-triggered:** *(specifies a time window)*



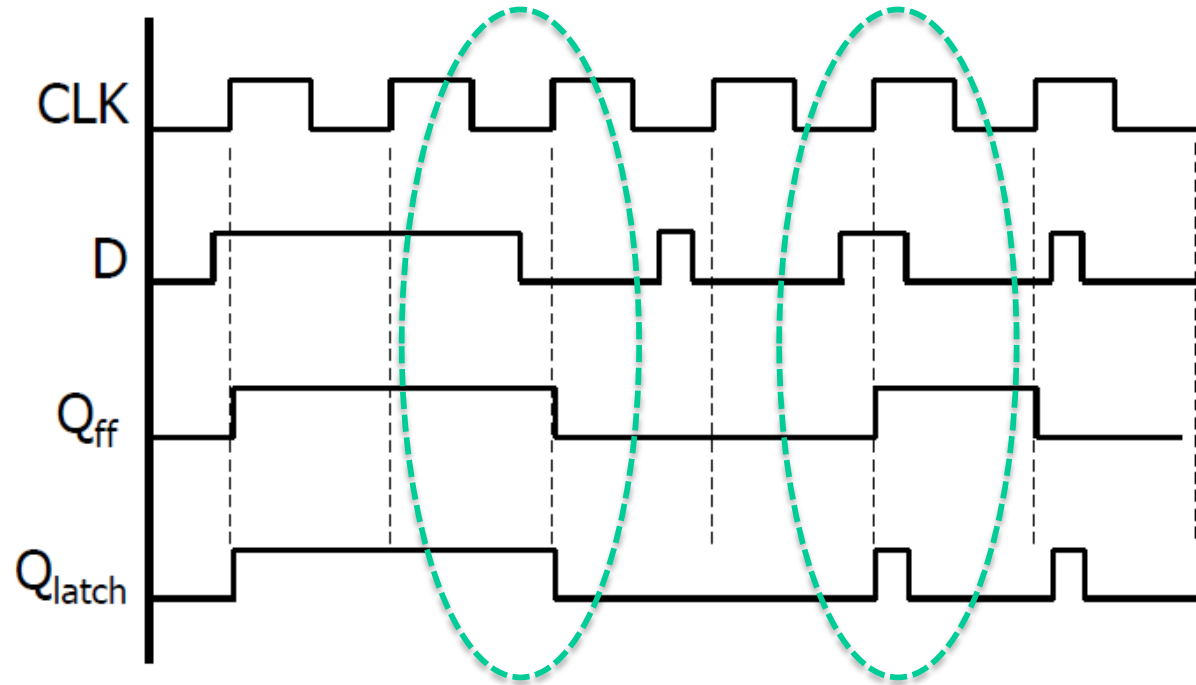
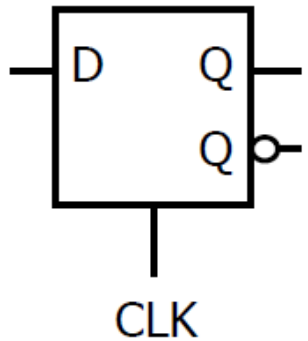


Flip-flop and Latch

Flip-flop



Latch



- Flip-flop: Edge triggered
- Latch: Level triggered

BC operation of latch based on level only
FF orchestration is set by the state changes in clk

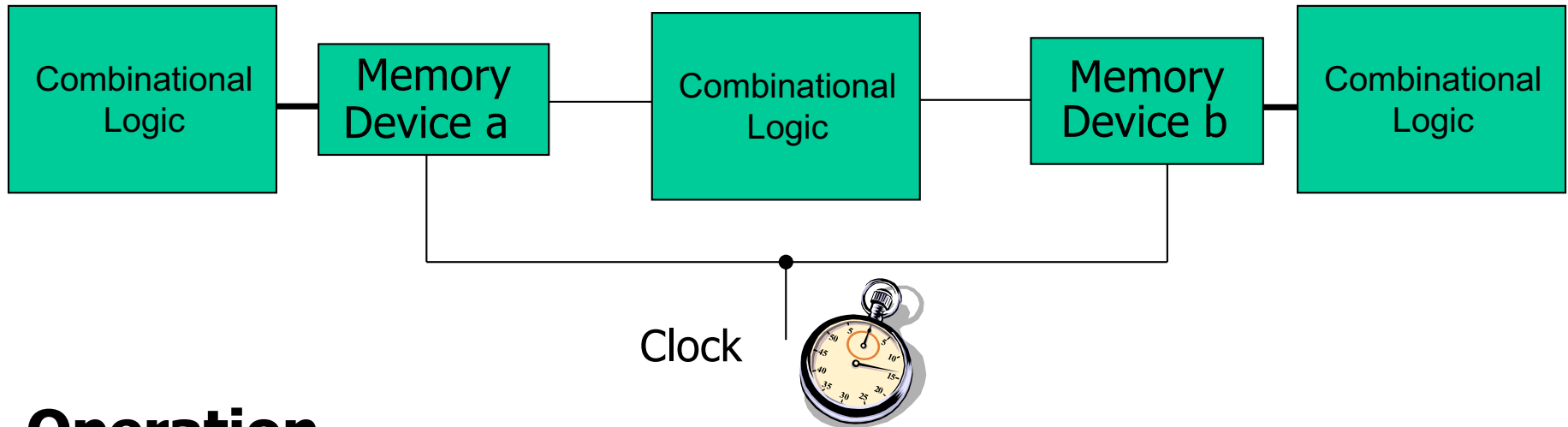


Level Triggered vs Edge Triggered

- Most synchronous sequential systems employ edge triggered devices
 - Safer operation compared to level triggered
- We will be focused on flip-flop based edge triggered design in this class



Synchronous Memory Elements - Operation

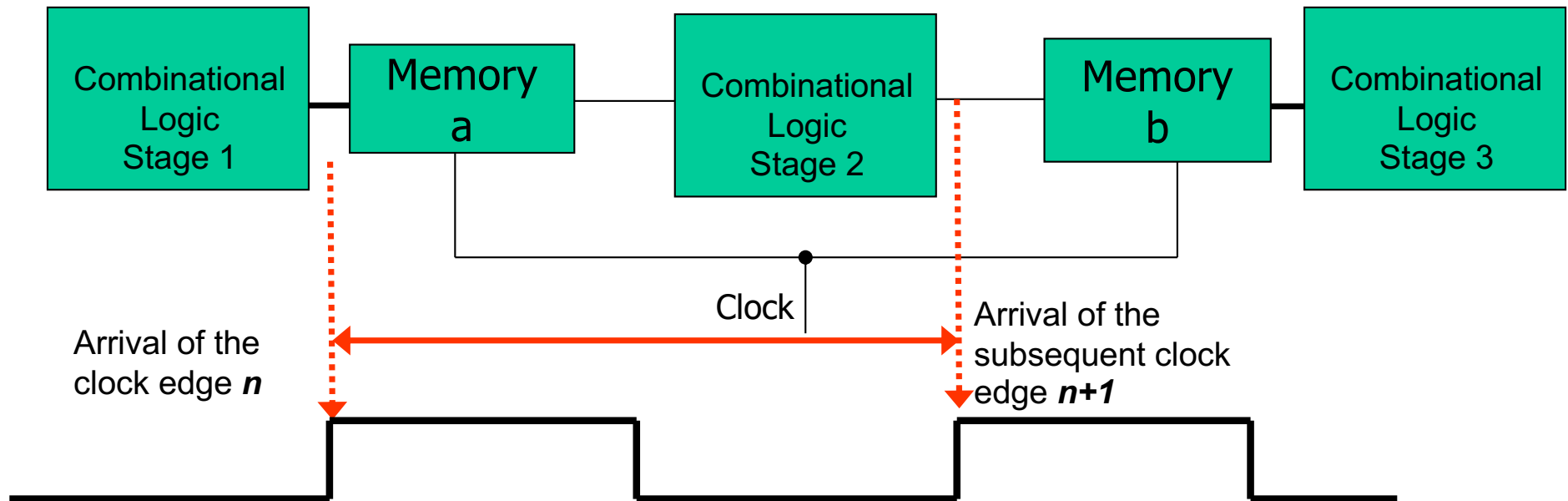


Operation

- These memory devices are used to synchronize the communication between combinational logic blocks
- Also referred as “pipelined” design
 - Data transfer between stages is controlled by clocks
- Use flip-flops for the memory devices



Synchronous Memory Elements - Operation



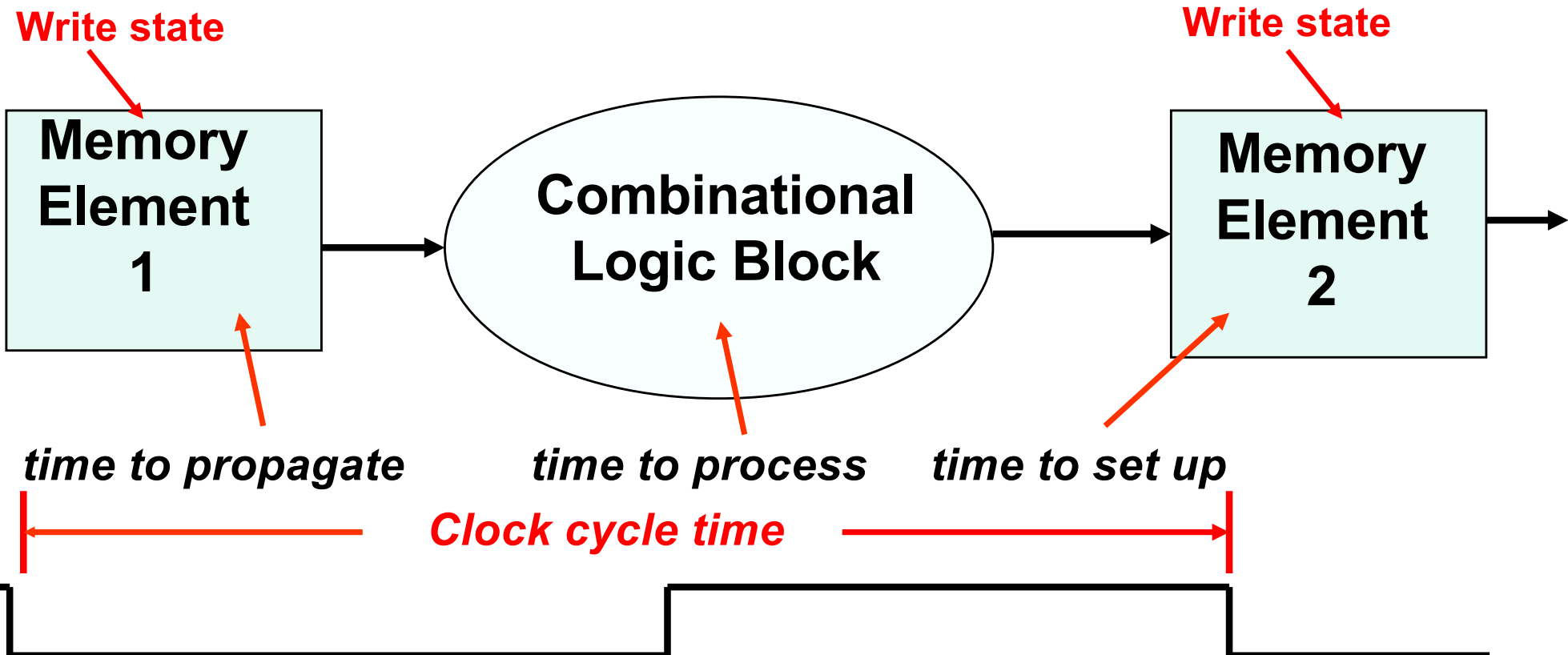
From the instant when the clock edge n arrives until clock edge $n+1$ arrives:

- 1. Propagate:** Memory device **a** must complete updating its content
- 2. Process:** CombL Stage 2 must complete processing the newly updated value supplied by device **a**
- 3. Setup:** CombL Stage 2 must have its stable final output present and ready for device **b** to be accepted upon arrival of clock edge $n+1$



Constraints on Clock Cycle Time

- Clock cycle time must be long enough for state updates to become valid (unchanging) to prevent a **timing violation** (computation not completed when clock arrives)



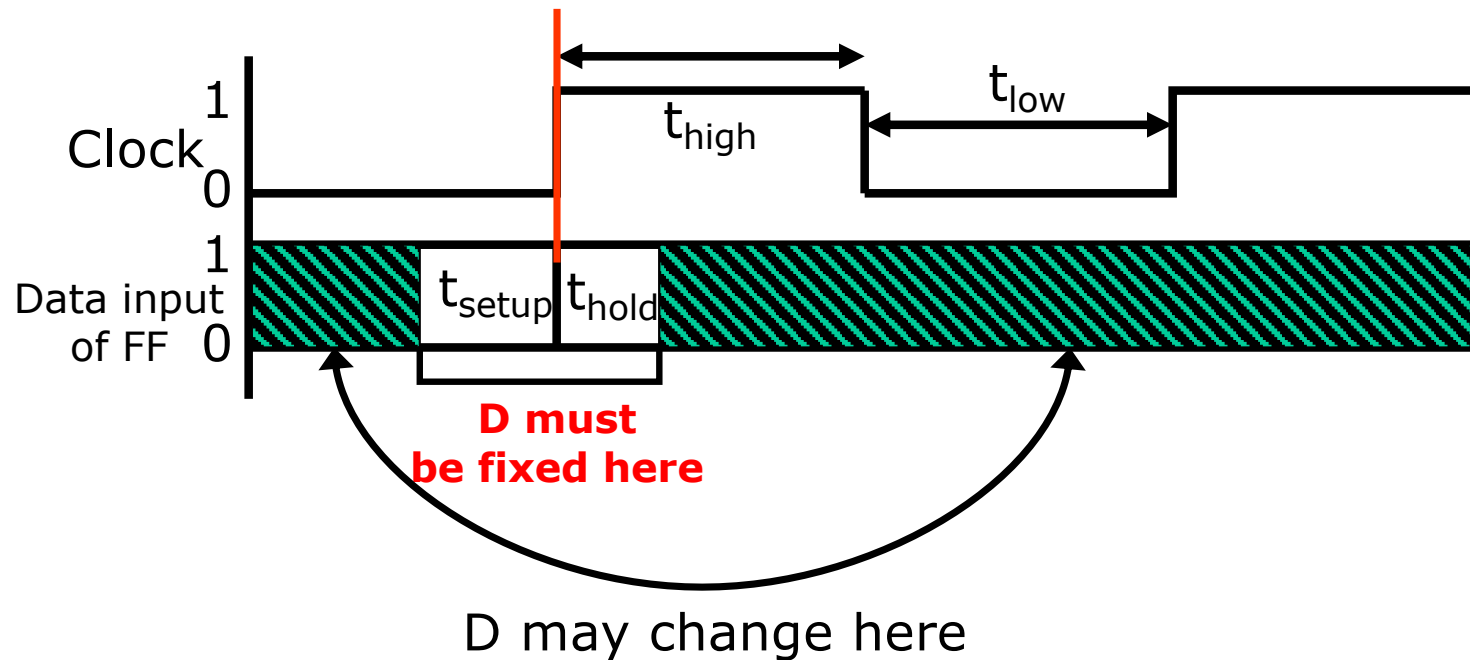


Timing Constraints

- Therefore, there must be a set of safety rules and margins imposed on the memory elements, the combinational logic blocks, and the choice of the clock cycle time to ensure correct operation in a synchronous system



Definition of Terms in Clocking: Edge Triggered Flip Flop



There is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be correctly stored
Setup Time + Hold Time

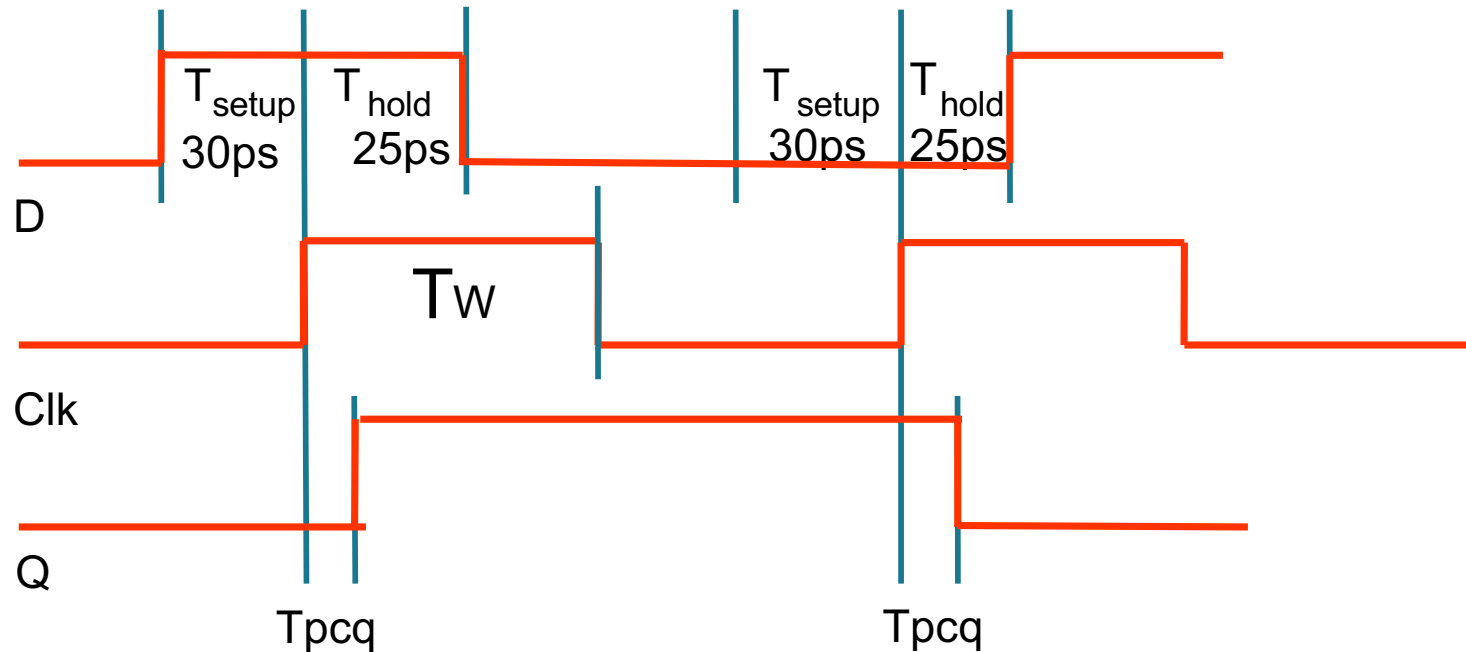


FF Timing

- T_{setup} , T_{hold}
- Minimum clock width, T_w
 - Usually period / 2
- Propagate delay: delay of flip-flop from clock pin to Q pin
 - T_{pcq} (clock to q delay)



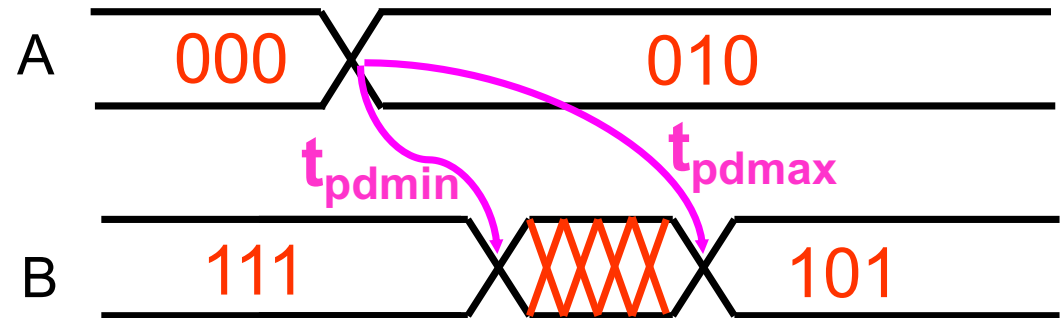
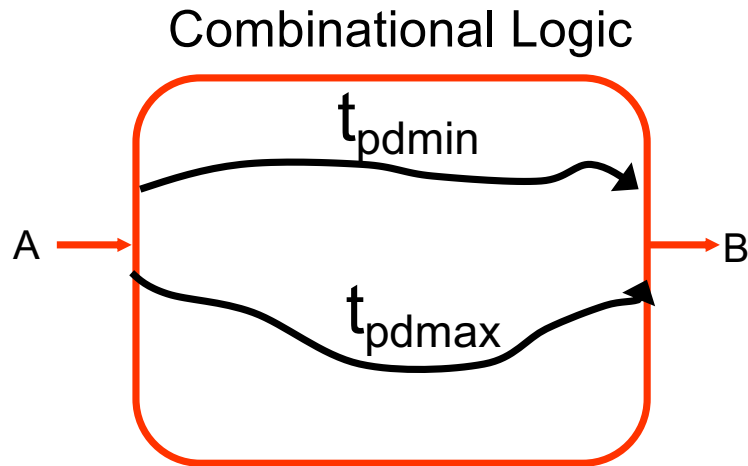
Timing Specifications of FFs



- Setup time T_{setup}
- Hold time T_{hold}
- Minimum clock width
- Propagation delays T_{pcq}



Generalized Timing Diagram for signals consisting of multiple bits



- Combinational logic will also exhibit a certain timing behavior with bounds of delay:
minimum delay and maximum delay
- A generalized format for timing diagram shown above
- Each signal can be multi-bit value carried through a bus

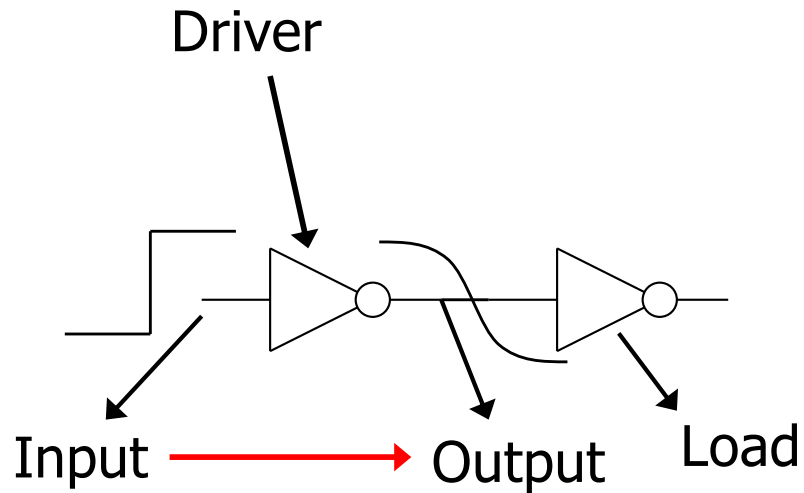


Combinational Logic Delay

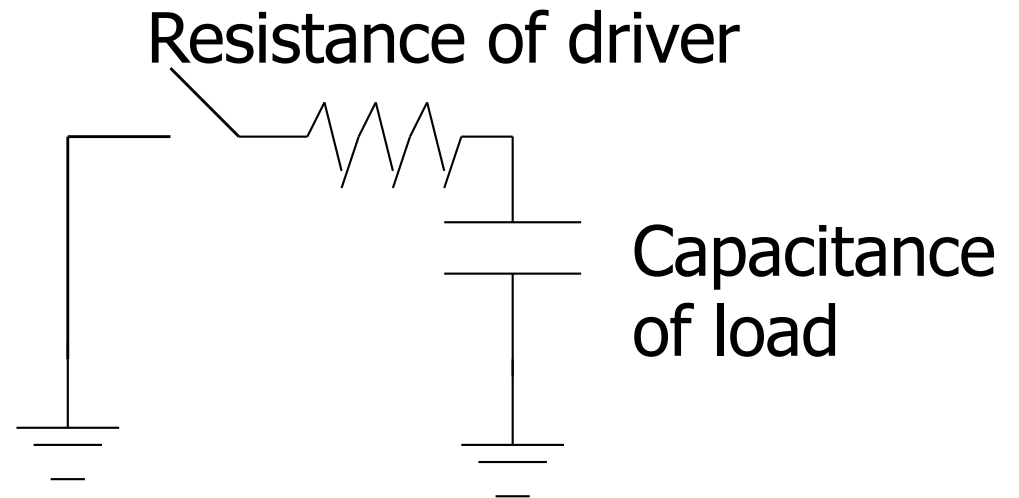
- Let's open a small window here to discuss how we can compute the delay of a combinational logic block



Cell or Gate Delays



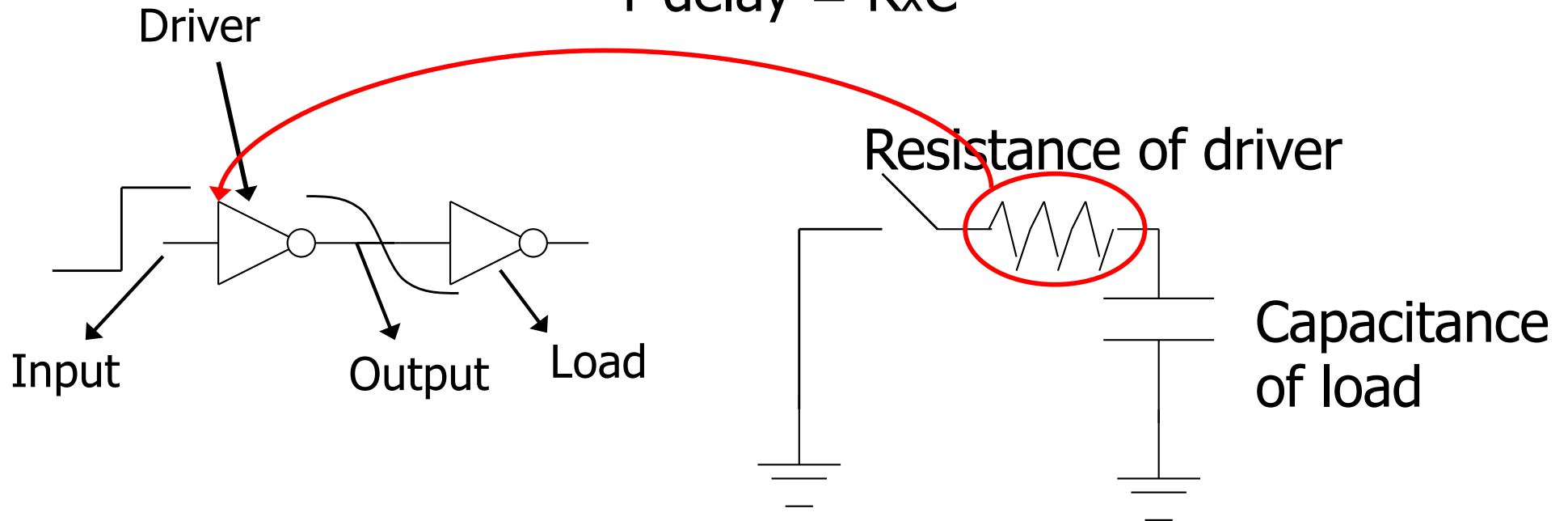
$$T_{\text{delay}} = R \times C$$





Cell Delays

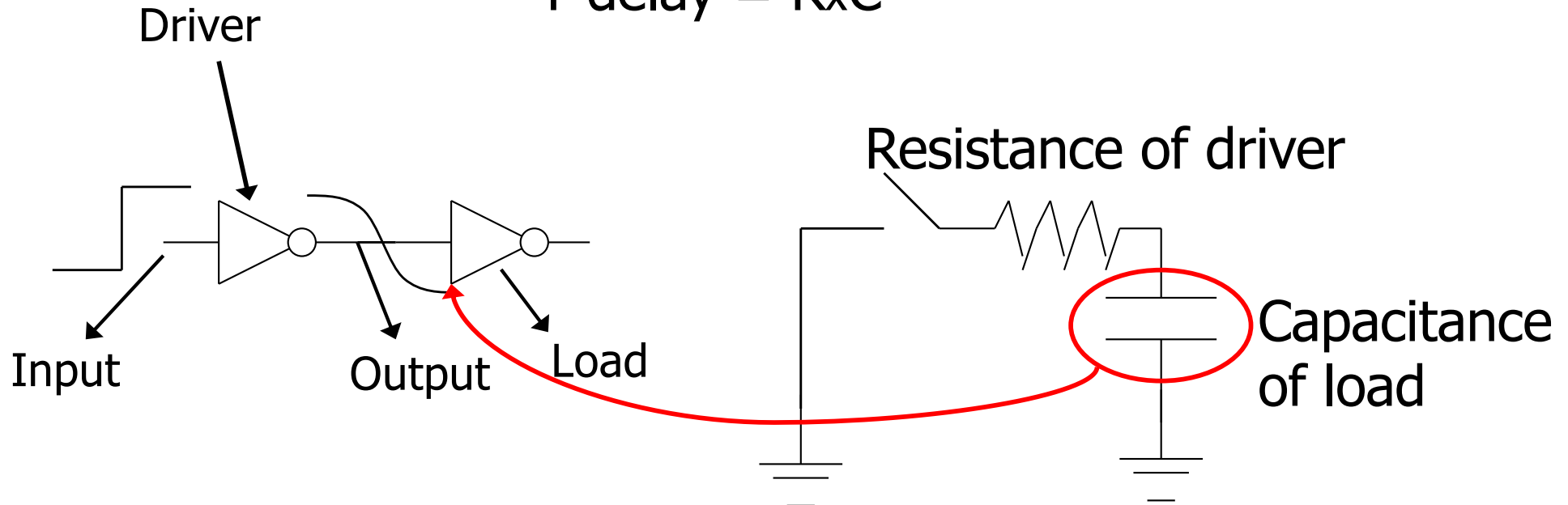
$$T_{\text{delay}} = R \times C$$





Cell Delays

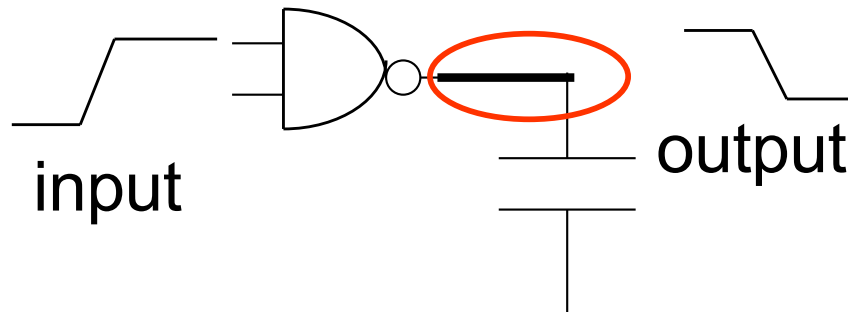
$$T \text{ delay} = R \times C$$



- **Also gates may exhibit different speeds for H-to-L versus L-to-H output transitions**



Delays in Combinational Logic



Wires contribute to the load of each gate also

Wire load
Capacitance C



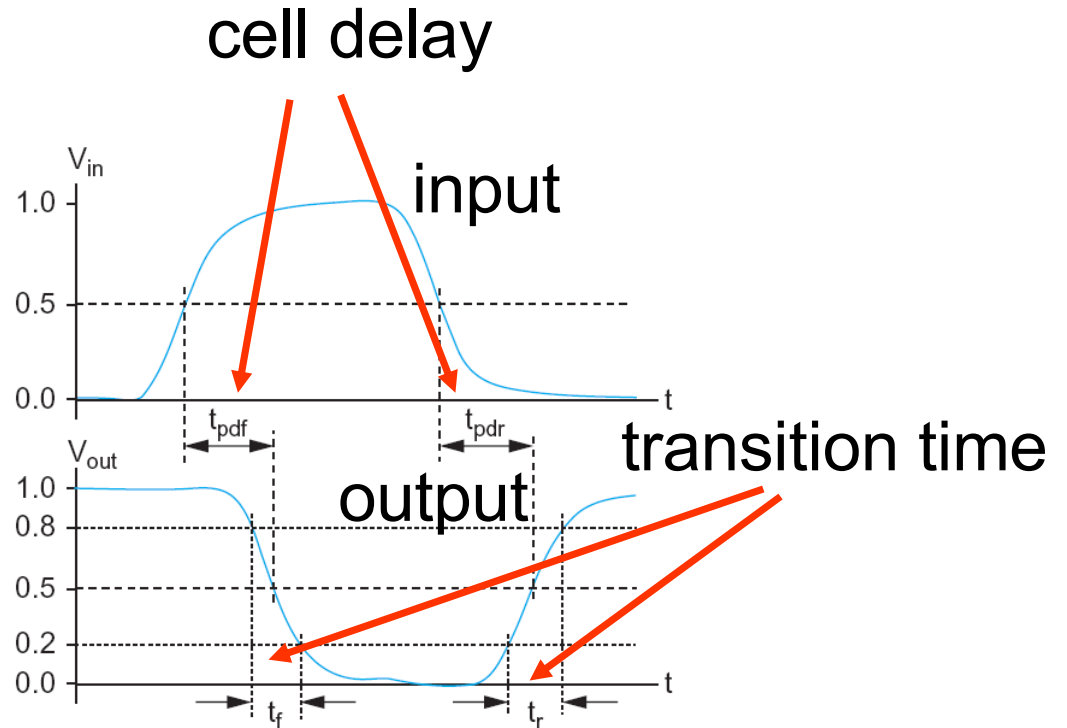
Improving Cell Delay

- Decrease load
 - Use smaller transistors
 - Use shorter wire
- Increase drive strength by increasing width of driving transistors
 - Causes problems
 - Larger transistors provide larger loads to their own respective inputs
 - In general, keep transistors small unless they absolutely need to drive large loads



Handling Cell Delay and Transition Time in EDA Tools

Waveforms from spice simulation of circuits



- Cell delay: measured from 50% of input to 50% of output
- Transition time: how fast a signal changes from 0 to 1 or 1 to 0, e.g. from 20% of Vdd to 80% of Vdd



Handling Cell Delay and Transition Time in EDA Tools

Cell Delay

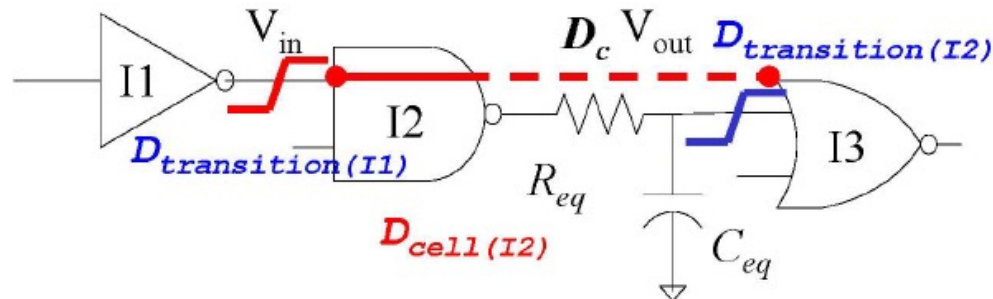
$$D_{\text{cell}}(I2) = f(D_{\text{transition}}(I1), C_{\text{eq}})$$

Transition Time

$$D_{\text{transition}}(I2) = g(D_{\text{transition}}(I1), C_{\text{eq}})$$

Output Capacitance	Input Transition		
	0	0.5	1
0.1	0.123	0.234	0.456
0.2	0.222	0.432	0.801

index1: input transition
Index2: output capacitance

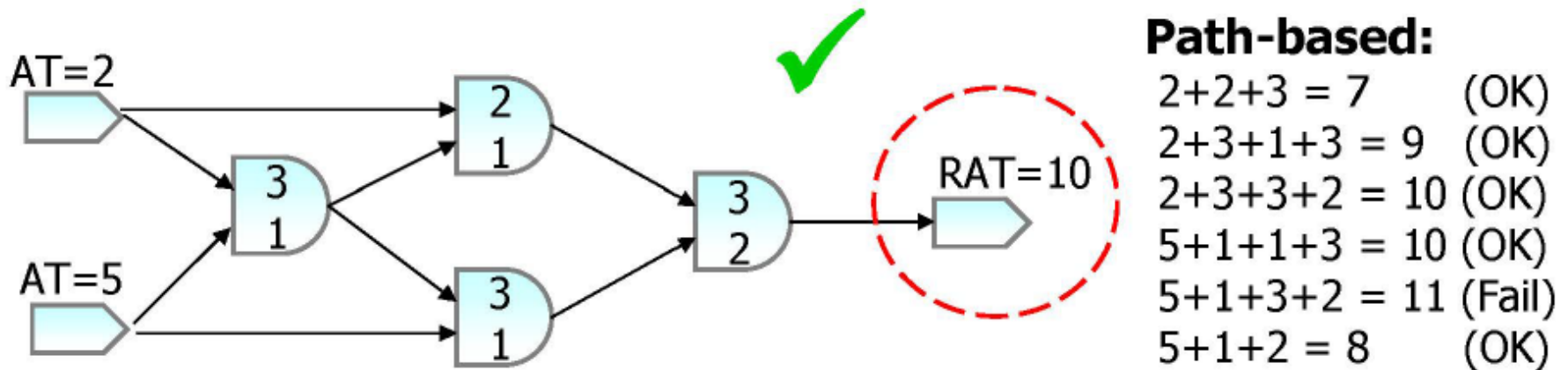


- Cell delay and output transition time are functions of input transition time and output capacitance



Static Timing Analysis

- Calculate the Arrival Time (AT) by adding cell delay in timing paths
- Check all path delays to see if the given Required Arrival Time (RAT) is met





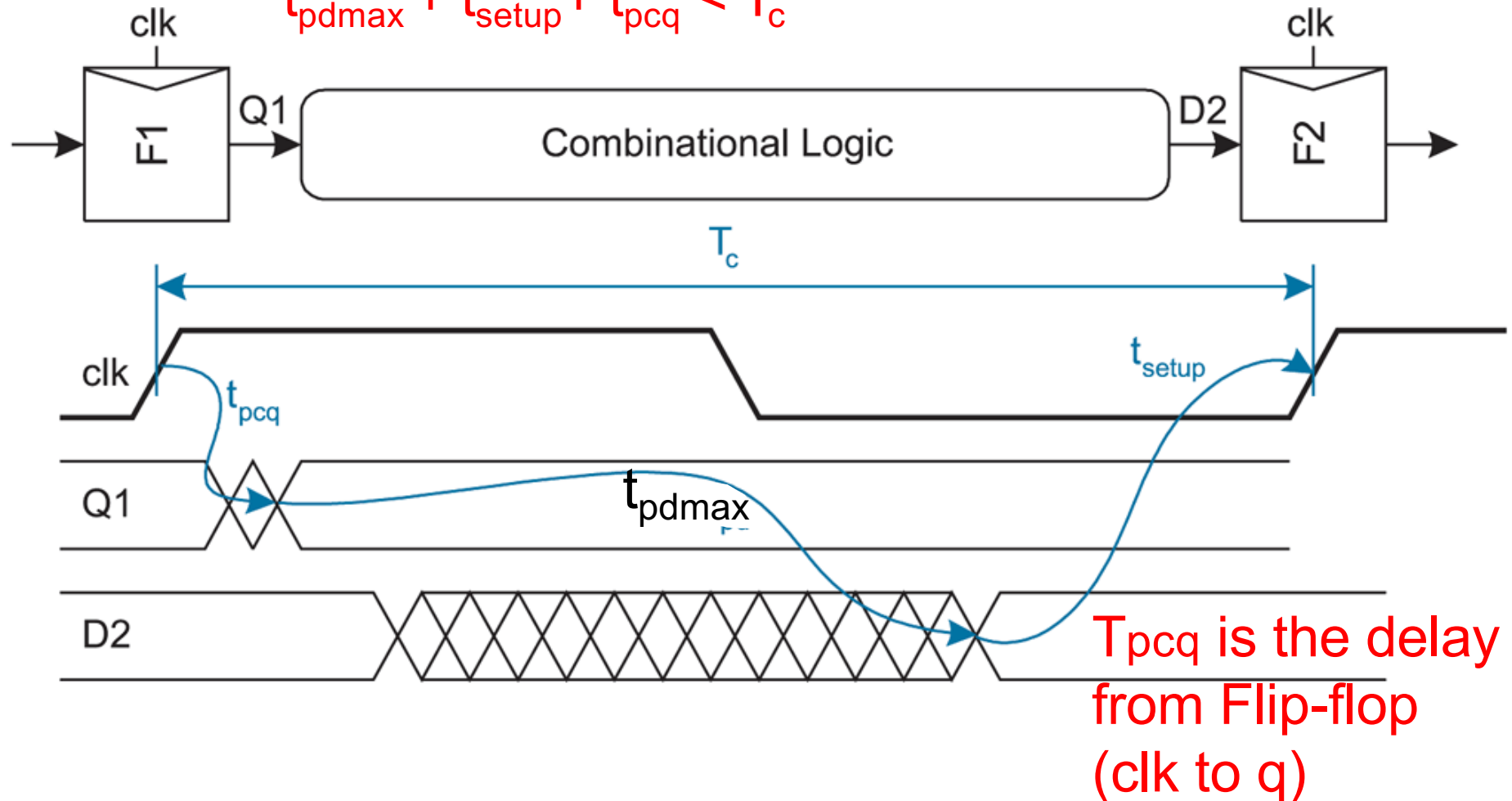
Timing Constraints

- Having the timing parameters for the memory elements and combinational logic available, now let's go back to the constraints on the overall system



Setup constraint: defines the relationship between clock period and the worst case delay of combinational logic

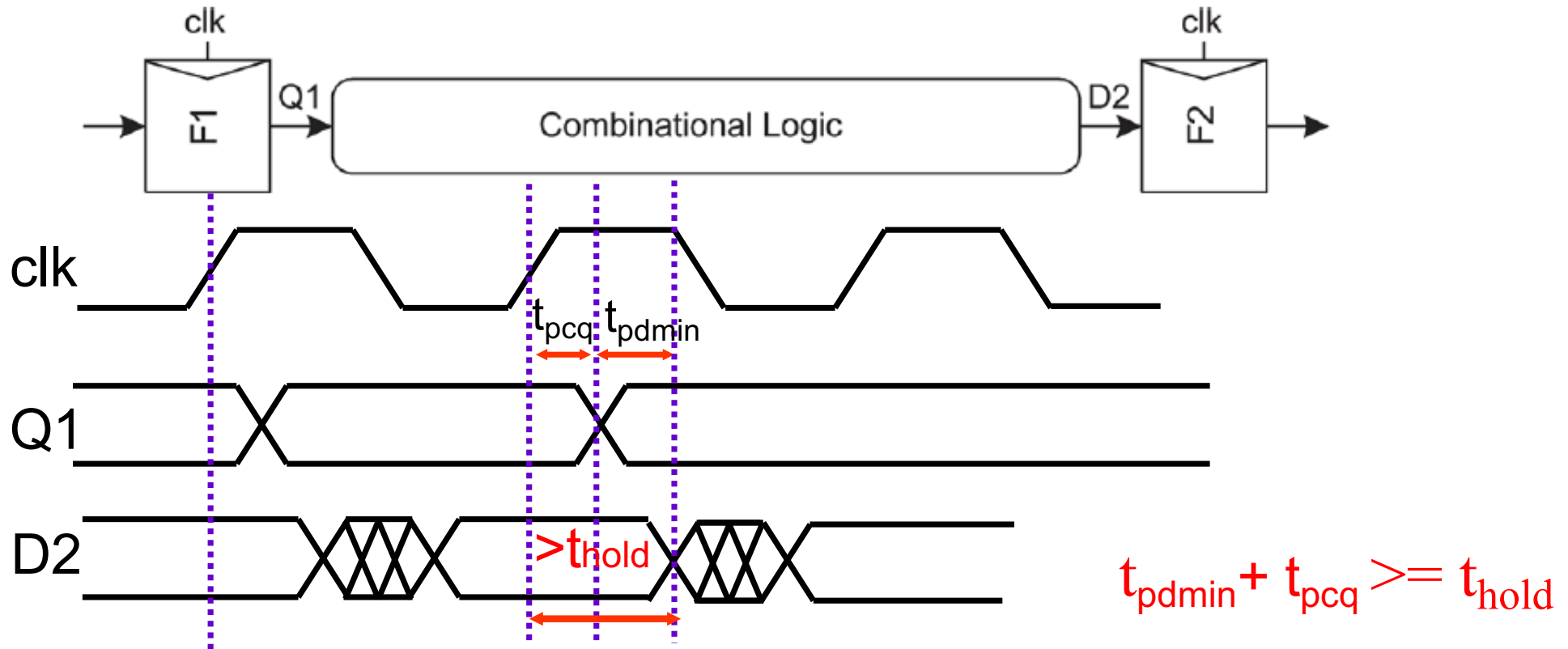
$$t_{pdmax} + t_{setup} + t_{pcq} < T_c$$





Hold constraint: Defines the relationship between the fastest propagation delay of combinational logic and the hold time of the flip flop

t_{pdmin} is also called contamination delay



The stable time period of D2 after the clock edge cannot be shorter than the hold time of F2
If the subsequent computation of the combinational logic is very quick and it catches up with the previous D2 value prematurely, F2 will not be able to store the previous D2 value in a stable manner

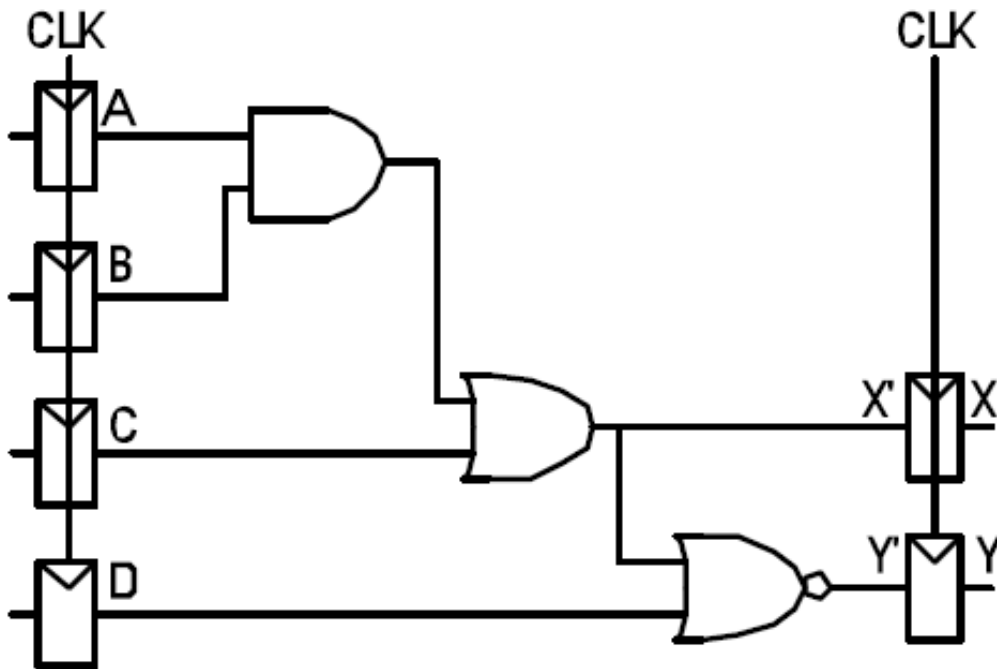


Putting it all together

- We can analyze whether a given sequential circuit will operate correctly by checking these two timing constraints



Example



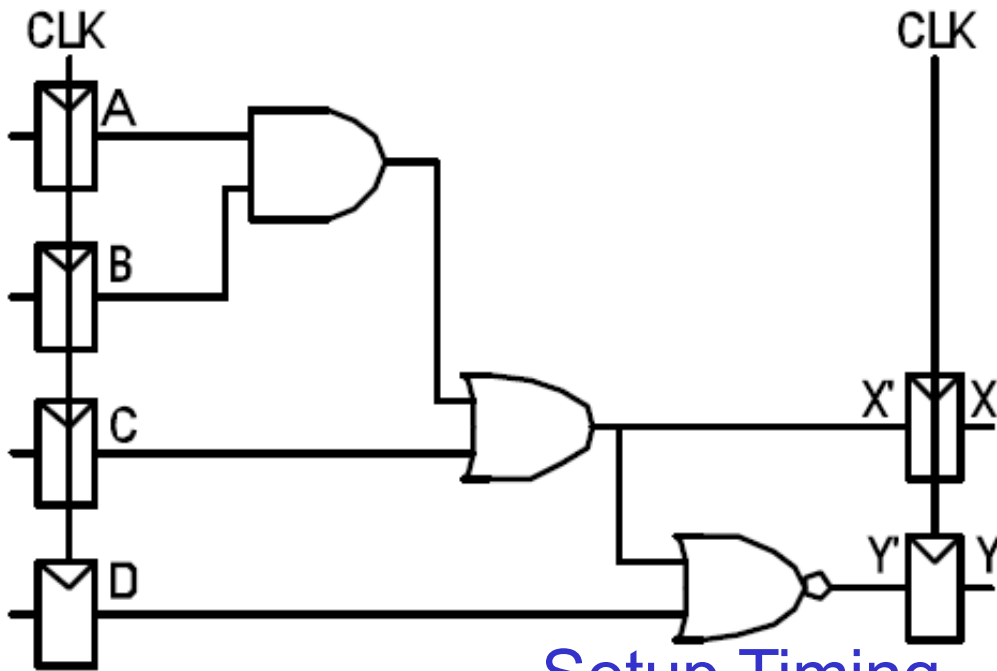
- $t_{pcq} = 40\text{ps}$
- $t_{\text{setup}} = 60\text{ps}$
- $t_{\text{hold}} = 80\text{ps}$
- $t_{pd\text{max}} = 55\text{ps}$ (each gate)
- $t_{pd\text{min}} = 35\text{ps}$ (each gate)

$$t_{pd\text{max}} + t_{\text{setup}} + t_{pcq} < T_c$$

$$t_{pd\text{min}} + t_{pcq} \geq t_{\text{hold}}$$



Example



- $t_{pcq} = 40\text{ps}$
- $t_{setup} = 60\text{ps}$
- $t_{hold} = 80\text{ps}$
- $t_{pdmax} = 55\text{ps}$ (each gate)
- $t_{pdmin} = 35\text{ps}$ (each gate)

Setup Timing

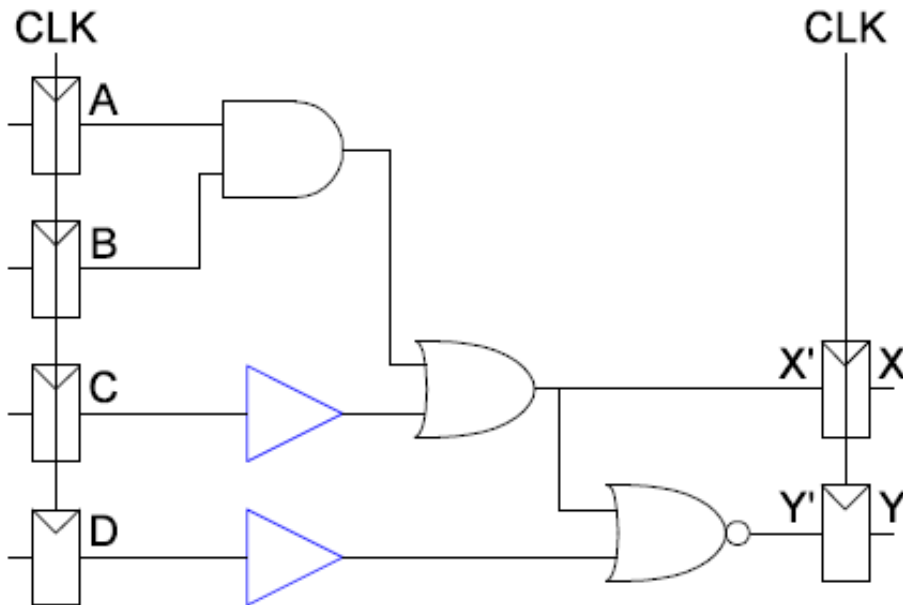
$$t_{pdmax} + t_{setup} + t_{pcq} < T_c \quad \Rightarrow \quad 165 + 60 + 40 = 265\text{ps} < T_c$$

Hold Timing

$$t_{pdmin} + t_{pcq} \geq t_{hold} \quad \Rightarrow \quad 35 + 40 \geq 80\text{ps} \quad \text{Violated !}$$



Timing Fix



- Hold violation: add delay cells
- Setup violation: slow down clock or fasten the circuit.

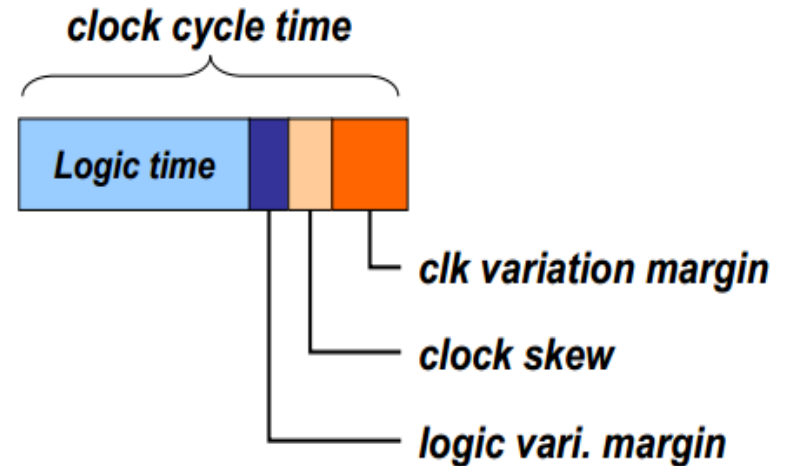
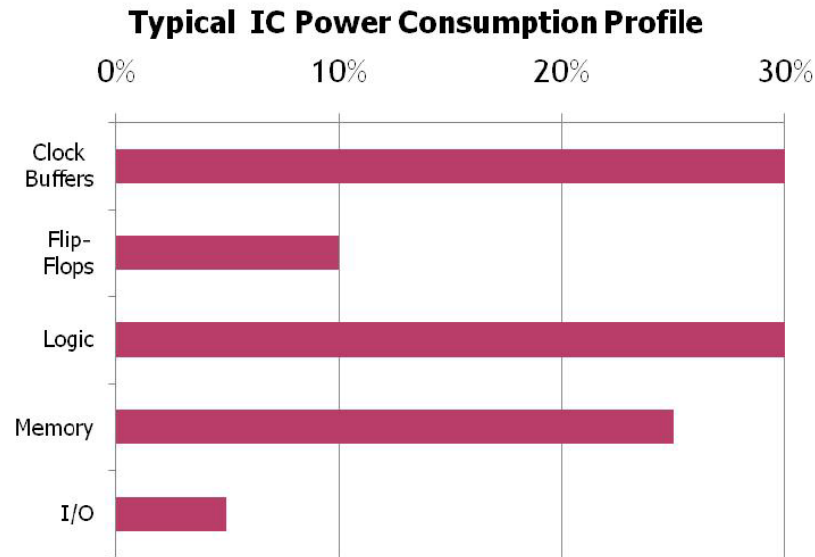


Clock Tree and Skew

- Clock is the most important signal
- We need to achieve:
 - Fast clock transition
 - Balanced clock tree
 - Minimum clock skew
 - Clock skew needs to be added into the timing constraints



Clock Design Basics

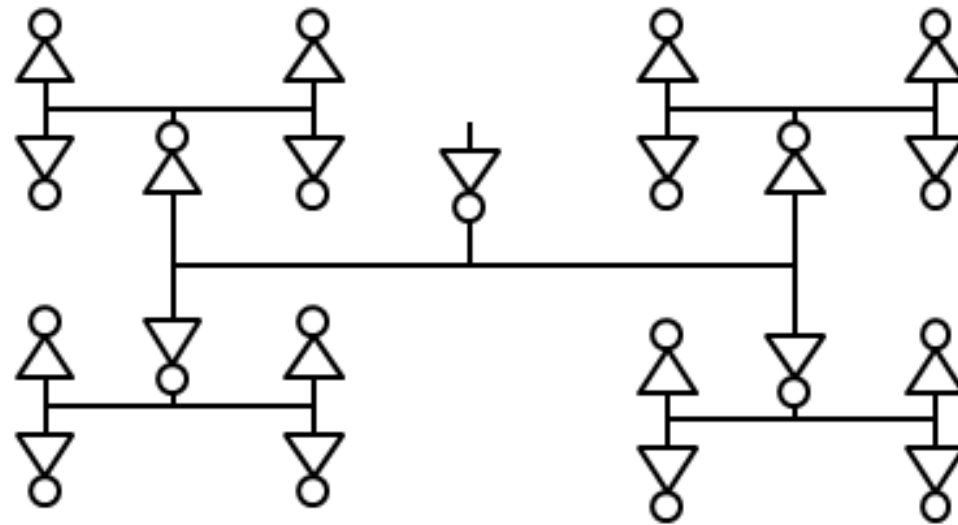


Clock skew = when clock connect to two components, the difference in time

- Clock distribution is a key design challenge in VLSI
- Clock design consideration:
 - Clock skew: one of the main source for timing violation
 - Clock power: 15~40% of chip power
 - Clock jitter: random clock arrival time due to noises Jitter = arrival time fluctuates



Clock Design Basics



Inverter symbols are actually

Can even build tunable buffer

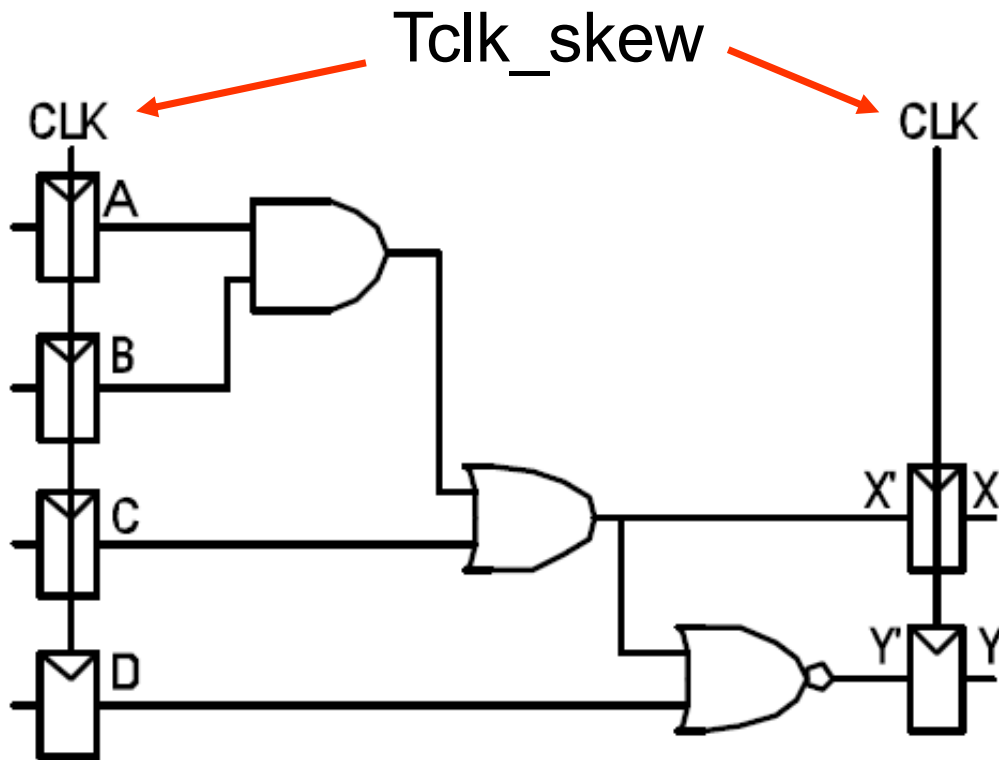
H Tree Structure

- Key clock tree design: Minimize skew
 - Use balanced tree structure, e.g. H Tree
 - Need to balance the wire routing of clock tree
 - But manufacturing variation will worsen the situation

Note: these buffers add additional delays, so need to drive with higher power to compensate for the delay to make the clock as fast.



Example



- $t_{pcq} = 40\text{ps}$
- $t_{setup} = 60\text{ps}$
- $t_{hold} = 80\text{ps}$
- $t_{pdmax} = 55\text{ps}$ (each gate)
- $t_{pdmin} = 35\text{ps}$ (each gate)

If X clock is Delta later than A, it helps the scenario, loosen the requirement

$$t_{pdmax} + t_{setup} + t_{pcq} < T_c$$

$$t_{pdmin} + t_{pcq} \geq t_{hold}$$

Add clock skew into these equations:
More stringent timing



Writing SDC files

- One of the most critical files
- Used for both synthesis and backend place & route
- Set constraint for timing:
 - Flip-flop to Flip-flop
 - set_clock_uncertainty
 - Input to Flip-flop
 - set_input_delay
 - Flip-flop to Output
 - set_output_delay
 - Input to Output
 - set_max_delay/set_min_delay

```
create_clock -name clk -period 1.0 -waveform { 0 0.5 } [get_ports clk]
```

```
# ----- Input constraints -----  
set_input_delay -max 0.5 -clock clk [get_ports {din, start, rstb,  
wr_ctrl_test_ctrl}]  
set_input_delay -min -0.2 -clock clk [get_ports {din, start, rstb,  
wr_ctrl_test_ctrl}]
```

```
# ----- Output constraints -----
```

```
set_output_delay -max 0.5 -clock clk [get_ports {addr_out*}]  
set_output_delay -min -0.2 -clock clk [get_ports {addr_out*}]
```

```
set_max_delay 1 -from [all_inputs] -to [all_outputs]
```

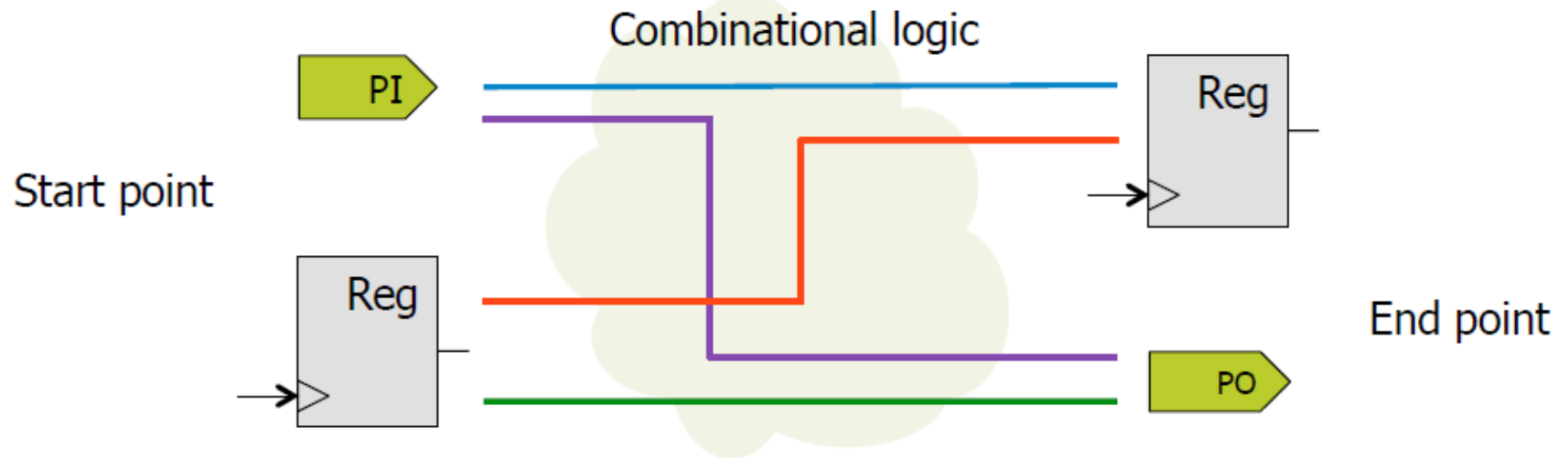
```
# Assume 50fF load capacitances everywhere:  
set_load 0.050 [all_outputs]  
# Set 10fF maximum capacitance on all inputs  
set_max_capacitance 0.010 [all_inputs]
```

```
# set clock uncertainty of the system clock (skew and jitter)  
set_clock_uncertainty -setup 0.03 [get_clocks clk*]  
set_clock_uncertainty -hold 0.03 [get_clocks clk*]
```

```
set_false_path -from [get_ports {start rstb wr_ctrl test_ctrl}]  
# set maximum transition at output ports  
set_max_transition 0.07 [current_design]  
#set_attr use_scan_seqs_for_non_dft false
```



Four types of paths



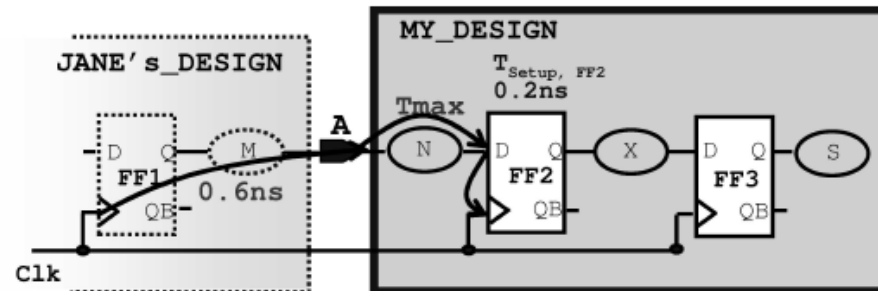
- Set constraint for timing:
 - Flip-flop to Flip-flop (covered already)
 - Input to Flip-flop
 - Flip-flop to Output
 - Input to Output



Set_input_delay

The user must specify the latest arrival time of the data at input A

What is T_{\max} for N ?



```
create_clock -period 2 [get_ports Clk]
```

```
set_input_delay -max 0.6 -clock Clk [get_ports A]
```

- Set "-max" to a number to account for delay from input

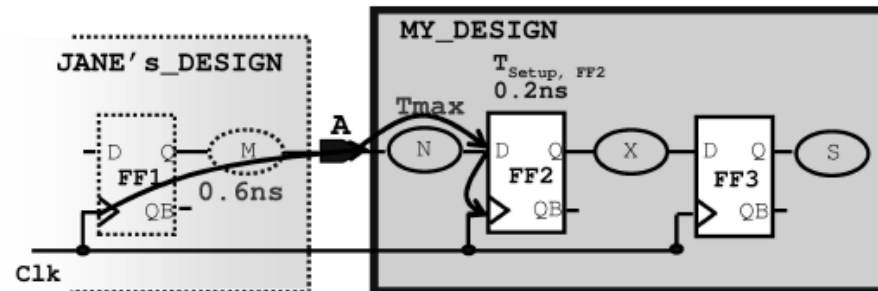
$$T_{\max} = 2\text{ns} - 0.6\text{ns} - 0.2\text{ns}(\text{setup from FF}) = 1.2\text{ns}$$



Set_input_delay

The user must specify the latest arrival time of the data at input A

What is T_{\max} for N ?



```
create_clock -period 2 [get_ports Clk]
```

```
set_input_delay -max 0.6 -clock Clk [get_ports A]
```

```
set_input_delay -min -0.1 -clock clk [get_ports A]
```

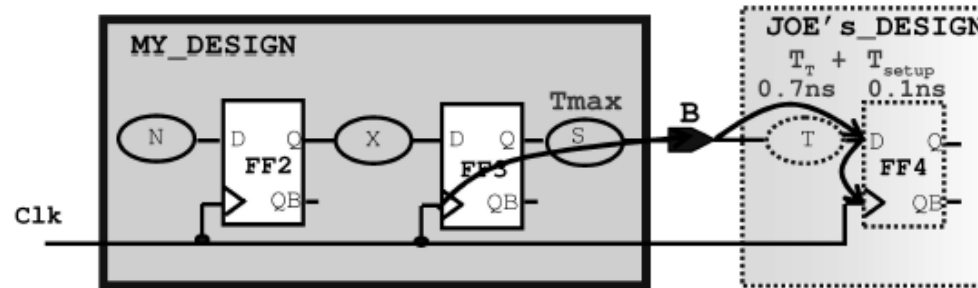
- Set "-min" to negative number to avoid hold violation
 - Does not have to be negative; Needs to be a small value
 - Force tools to insert some delay (equal to the negative value you give)
- Design related; Need to have "top level" understanding of the system across modules



Set_output_delay

The user must specify the latest arrival time of the data at output B

What is T_{\max} through S ?



```
create_clock -period 2 [get_ports Clk]
```

```
set_output_delay -max 0.8 -clock Clk [get_ports B]
```

- Set "-max" to a number to account for delay at output

$$T_{\max} = 2\text{ns} - 0.8\text{ns} = 1.2\text{ns}$$

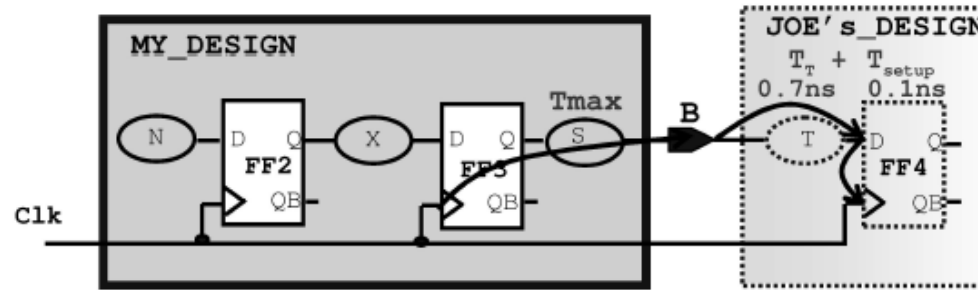
(Note 1.2ns includes $T_{\text{clk_q}}$)



Set_output_delay

The user must specify the latest arrival time of the data at output B

What is T_{\max} through S ?



```
create_clock -period 2 [get_ports Clk]
```

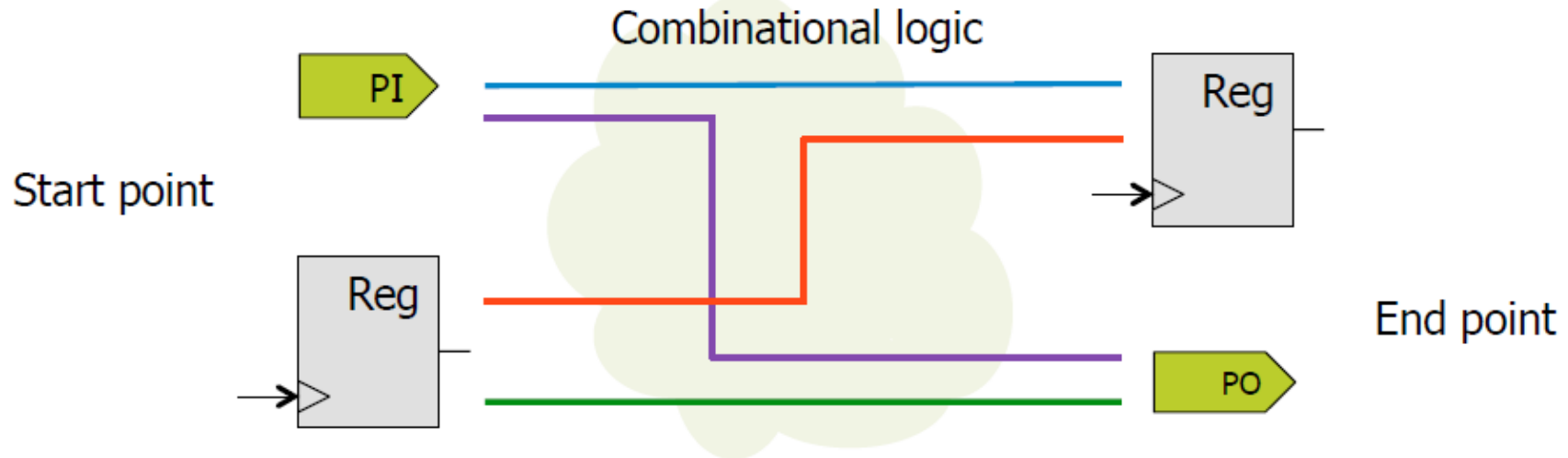
```
set_output_delay -max 0.8 -clock Clk [get_ports B]
```

```
set_output_delay -min -0.1 -clock clk [get_ports B]
```

- Set “-min” to negative number to avoid hold violation
 - Does not have to be negative; Needs to be a small value
 - Force tools to insert some delay (equal to the negative value you give)
- Design related; Need to have “top level” understanding of the system across modules



Set_max_delay & Set_min_delay



set_max_delay 1 -from [all_inputs] -to [all_outputs]

- Constrain combinational path from PI to PO
 - Does not constrain clocked paths

Appendix





Clock Design File

- Innovus file: “Clock.ctstch”
- Use in backend design
 - Whenever you have a clock
- Define:
 - Clock names;
 - Clock delay constraints;
 - Clock skew constraints;
 - Clock level balancing;
 - Clock drivers;
 - Clock transition time;
- Be careful with clock names:
“clk” or “Clk” or “clock”

```
#-- Clock Group --
#ClkGroup
#+ <clockName>
#-----
# Clock Root   : clk
# Clock Name   : clk
# Clock Period : 1.0ns
#-----
AutoCTSRootPin clk
Period        1.0ns
MaxDelay      300ps # sdc driven default
MinDelay      30ps  # sdc driven default
MaxSkew       50ps  # sdc driven default
SinkMaxTran   50ps  # sdc driven default
BufMaxTran    50ps  # sdc driven default
MaxFanout     12
Buffer        CLKBUF_X1 CLKBUF_X2 CLKBUF_X3
LevelBalanced YES
MaxCap
+ CLKBUF_X1 2ff
+ CLKBUF_X2 4ff
+ CLKBUF_X3 6ff
NoGating      NO
DetailReport   YES
#SetDPinAsSync NO
#SetIoPinAsSync NO
#SetASyncSRPinAsSync NO
#SetTriStEnPinAsSync NO
#SetBBoxPinAsSync NO
RouteClkNet    YES
PostOpt        YES
OptAddBuffer    YES
#RouteType     specialRoute
#LeafRouteType regularRoute
END
```