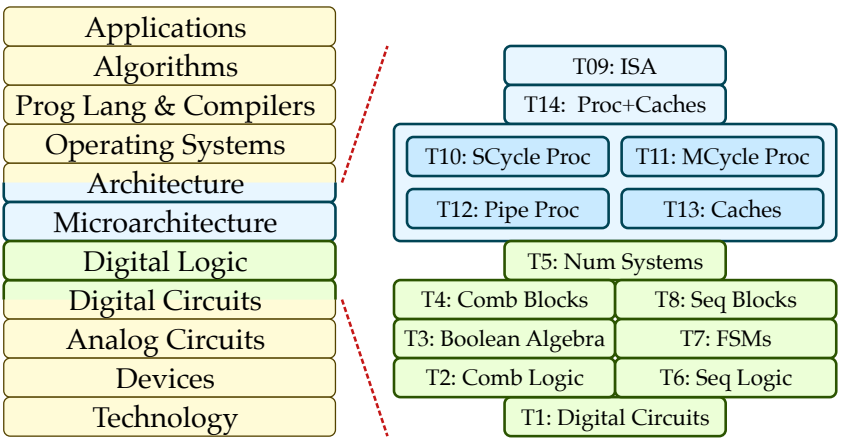
ECE 2300 Digital Logic and Computer Organization Fall 2025

Topic 6: Sequential Logic

School of Electrical and Computer Engineering
Cornell University

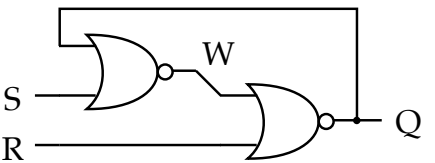
revision: 2025-10-29-22-38



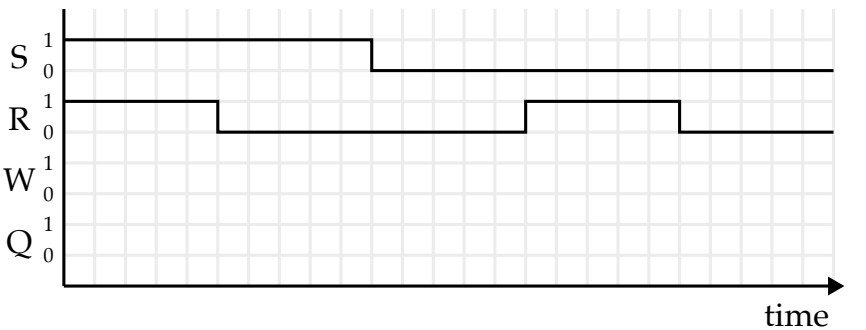
1. Latches, Flip-Flops, and Registers

- *Combinational Logic:* outputs only depend on current inputs
- *Sequential Logic:* outputs depend on current inputs *and previous inputs*

1.1. SR Latch



S	*R*	*W*	*Q*
0	0		
0	1		
1	0		
1	1		



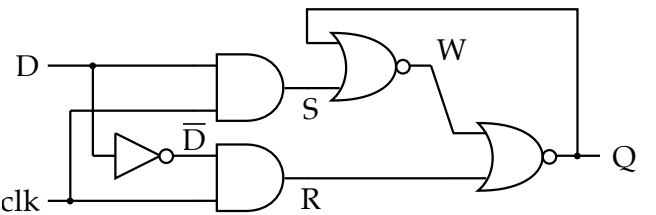
- *SR Latch:* bi-stable state element with set and reset inputs

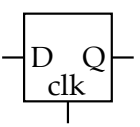
- Let's create a new kind of truth table
- Treat Q_{prev} as an input and Q_{next} as an output

S	R	Q_{prev}	W	Q_{next}	$Q_{next,next}$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

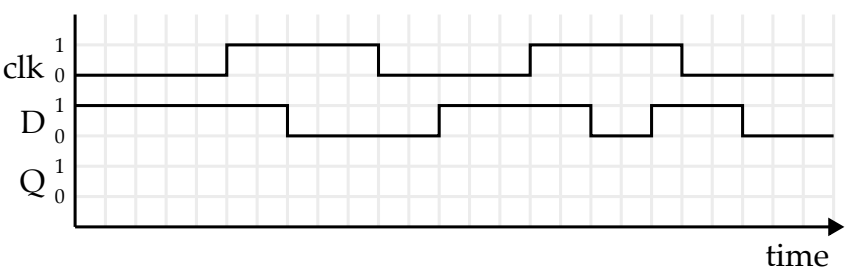
1.2. D Latch

- *SR Latch:* Asserting one input determines not only *what* the new state should be but also *when* it should change
- *D Latch:* One input controls *what* the next state should be, while second input controls *when* the state should change



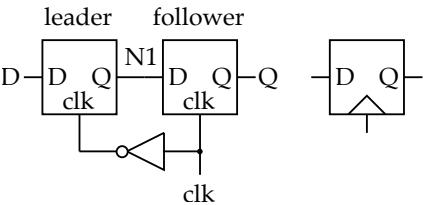


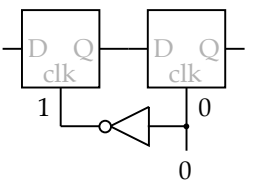
clk	D	$\overline{D}$	S	R	W	Q
0	0					
0	1					
1	0					
1	1					



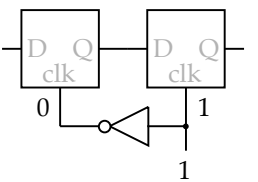
1.3. D Flip-Flop

- *D Latch:* Input is *sampled* continuously when clock is high
- *D Flip-Flop:* Input is only *sampled* at a specific instance in time (on the rising edge of the clock)

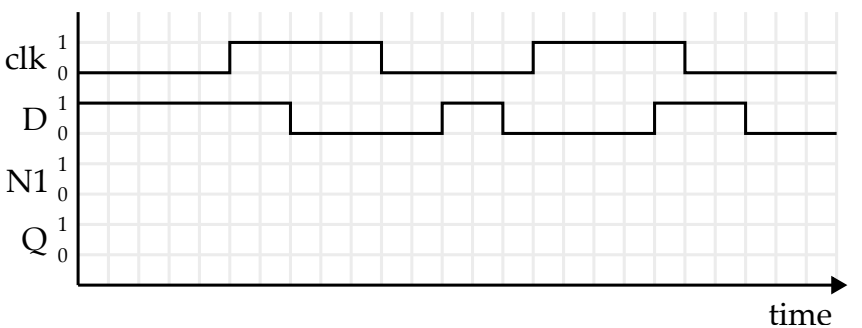




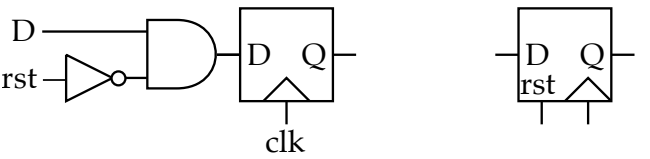
Before Rising Clock Edge

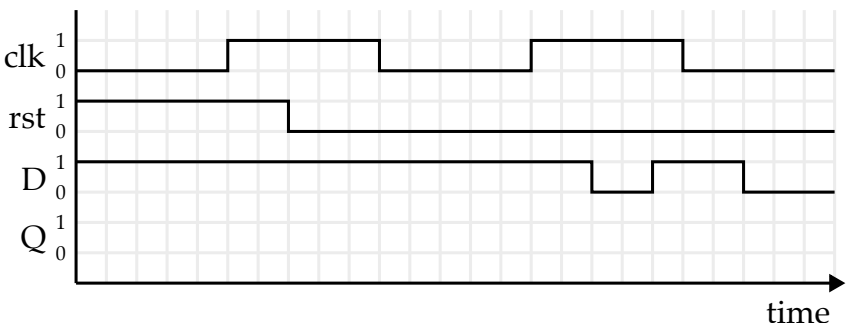


After Rising Clock Edge

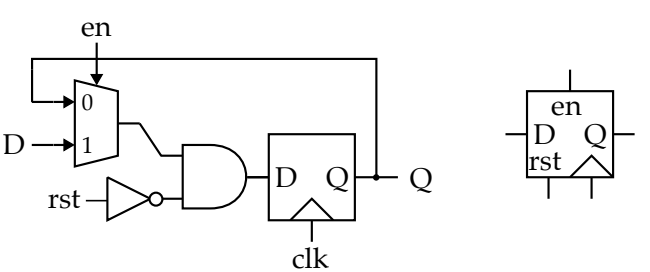


1.4. D Flip-Flop with Synchronous Reset

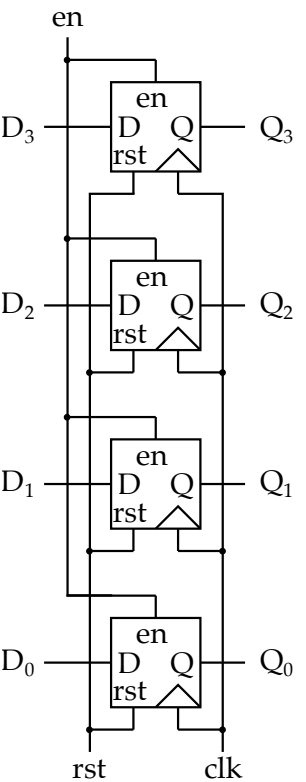


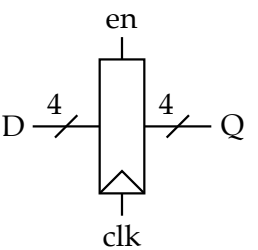


1.5. D Flip-Flop with Synchronous Reset and Enable



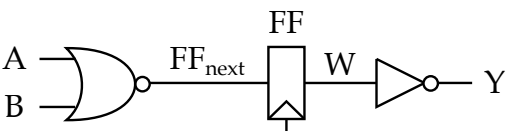
1.6. Multi-Bit Register





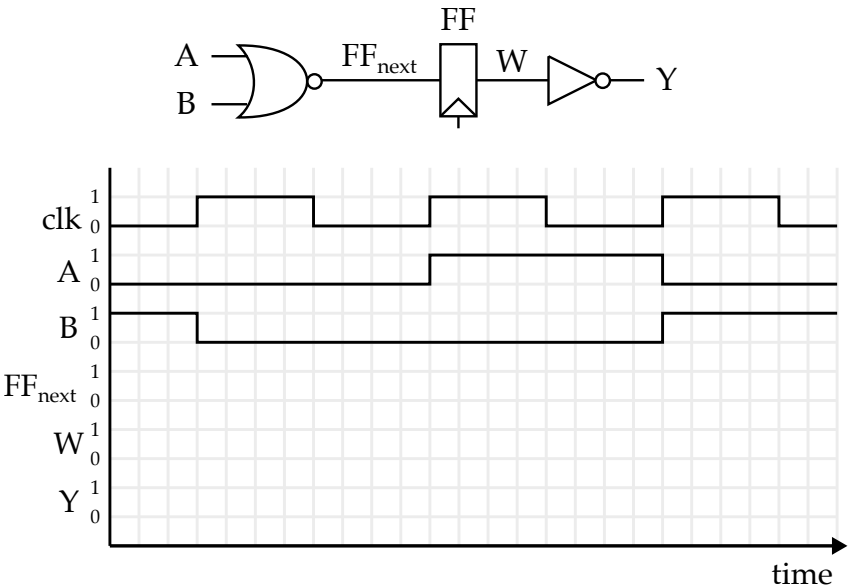
2. Sequential Gate-Level Networks

- *Combinational truth tables* have one column for every input, intermediate signal, and output; one row for all possible input values
- *Sequential truth tables* have one column for every input, *flip-flop*, intermediate signal, and output; one row for all possible input values *and possible flip-flop values*

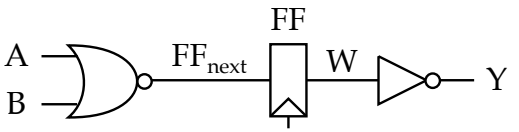


Truth Table

A	B	FF	FF_{next}	W	Y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			



- Drawing waveforms for sequential logic over many cycles can become quite tedious
- *Simulation tables* are a more compact way to illustrate waveforms for sequential logic assuming a zero delay model



Explicit-Clock Simulation Table

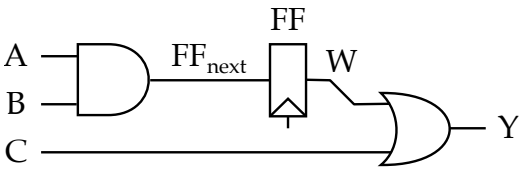
- Clock signal is shown as a column
- One row per *clock phase*, row shows values during that clock phase

clk	A	B	FF	FF_{next}	W	Y
1	0	1				
0	0	1				
1	0	0				
0	0	0				
1	1	0				
0	1	0				
1	0	1				
1	0	1				

Implicit-Clock Simulation Table

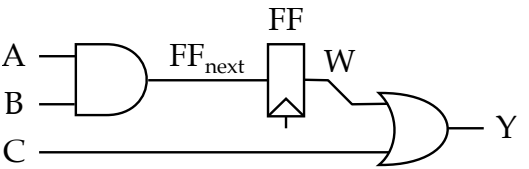
- Clock signal is *not* shown as a column
- One row per *clock cycle*, row shows values when clock is high phase

A	B	FF	FF_{next}	W	Y
0	1				
0	0				
1	0				
0	1				



Truth Table

A	B	C	FF	FF_{next}	W	Y
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

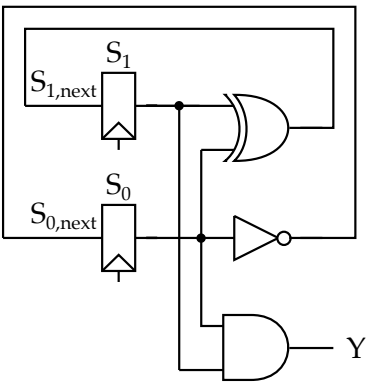


Implicit-Clock Simulation Table

cycle	A	B	C	FF	FF_{next}	W	Y
0	1	1	0				
1	1	1	0				
2	0	1	0				
3	0	1	1				
4	0	1	1				

- Use arrows to show dependencies between values
  - *Horizontal dependency arrows* correspond to combinational logic
  - *Vertical dependency arrows* correspond to sequential logic

Complete the truth table and implicit-clock simulation table for the given sequential gate-level network.



Truth Table

S_1	S_0	$S_{1,next}$	$S_{0,next}$	Y
0	0			
0	1			
1	0			
1	1			

Implicit-Clock Simulation Table

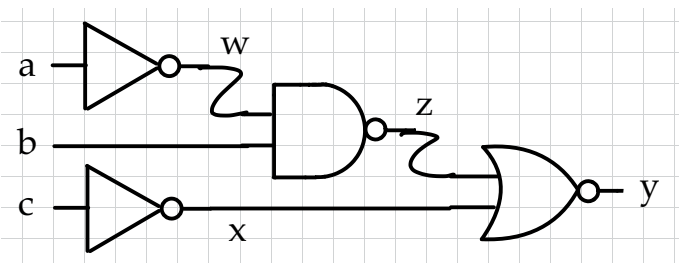
Assume all flip-flops are reset to zero.

cycle	S_1	S_0	$S_{1,next}$	$S_{0,next}$	Y
0					
1					
2					
3					
4					
5					
6					
7					

3. Sequential Gate-Level Timing

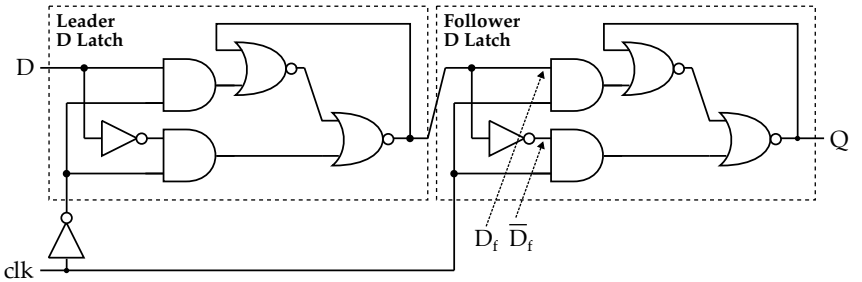
- **Critical Path for Combinational Gate-Level Networks:** longest propagation path delay from any input to any output

Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ
AND2	3τ	1τ
OR2	4τ	1τ
XOR2	7τ	1τ
XNOR2	7τ	1τ



Path	Propagation Delay	Critical Path?
A → NOT → NAND2 → NOR2 → Y		
B → NAND2 → NOR2 → Y		
C → NOT → NOR2 → Y		

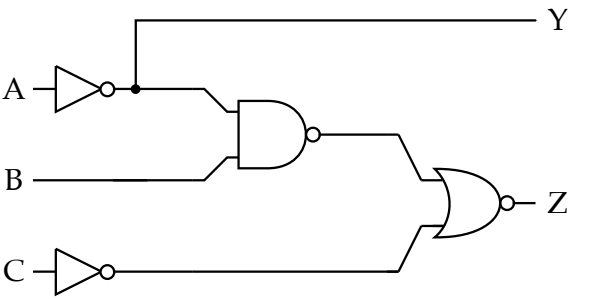
- Identify all paths from any input to any output



Path
D → AND2 → NOR2 → NOR2 → AND2 → NOR2 → NOR2 → Q

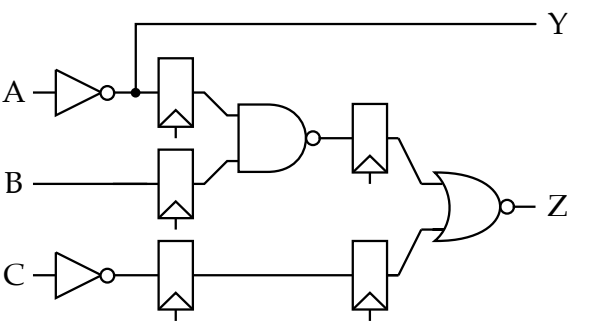
Critical Path for Combinational Gate-Level Networks

- Longest propagation path delay from:
  - any input to any output

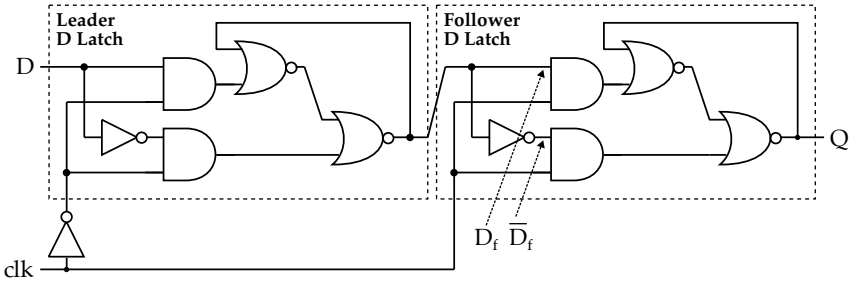


Critical Path for Sequential Gate-Level Networks

- Longest propagation path delay from:
  - any input to any output
  - any input to any flip-flop
  - any flip-flop to any output
  - any flip-flop to any flip-flop



3.1. Setup Time

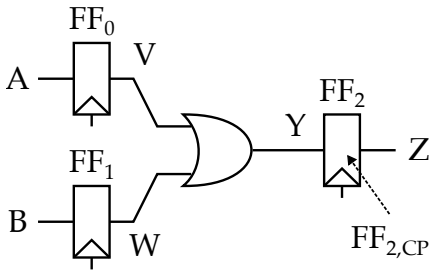


Clock-to-Q Propagation Delay ($t_{pd,cq}$)

Path	Propagation Delay	Critical Path?

Setup Time (t_{setup})

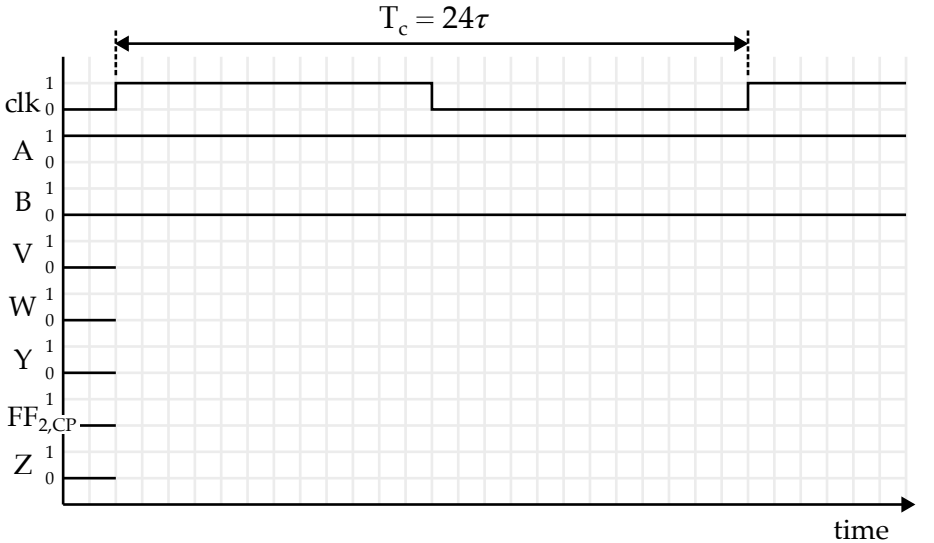
Path	Propagation Delay	Critical Path?

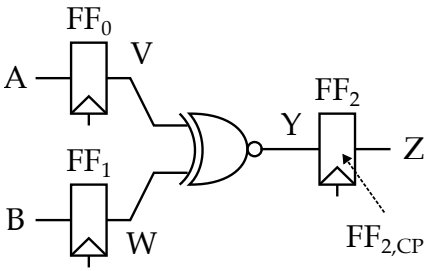


	t_{pd}	t_{cd}
OR2	4τ	1τ
FF (t_{cq})	9τ	

FF (t_{setup})	10τ
T_C	24τ

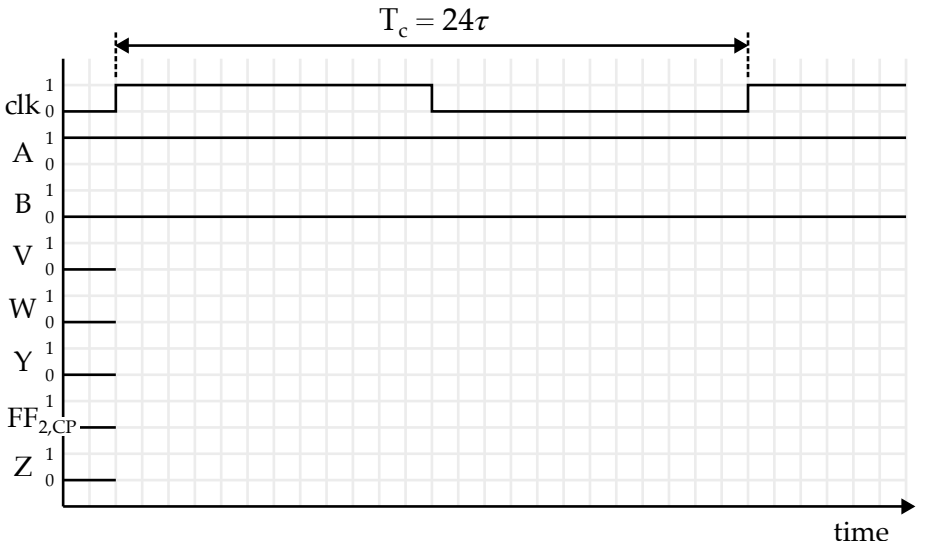
- *clock period* or *cycle time* (T_C): time between rising edges of clock
- *clock frequency* ($f_C = 1/T_C$): rate of rising edges of the clock



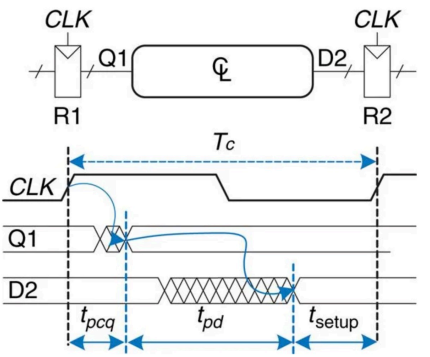


	t_{pd}	t_{cd}
XNOR2	7τ	1τ
FF (t_{cq})	9τ	

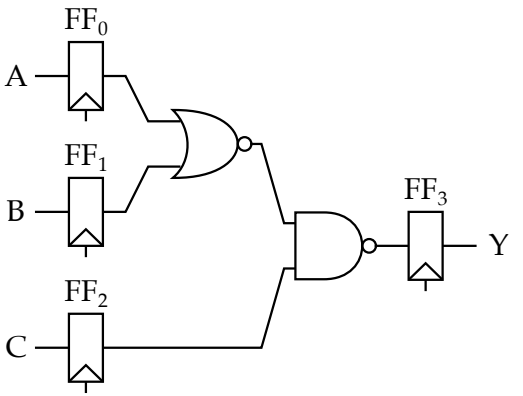
FF (t_{setup})	10τ
T_C	24τ



Setup Time Constraint



- Setup time constraint is a *race between the clock and data*
  - We want the data to win, we want the data to be sampled by the leader latch before the clock causes the leader latch to become opaque
- How can we fix a setup time violation?
- *Static timing analysis table* for sequential logic shows the *setup time constraints* on every path through the gate-level network that are required to ensure correct operation
- In this course we assume the following are unconstrained
  - paths from input to any output
  - paths from input to any flip-flop
  - paths from any flip-flop to any output
- Each constraint is an inequality which is either:
  - $\geq$ or $\leq$ (satistified, no timing violation)
  - $\ngeq$ or $\nleq$ (not satistified, timing violation)



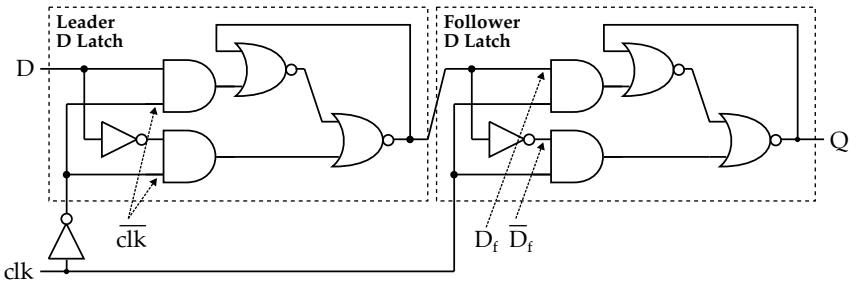
	t_{pd}	t_{cd}
NAND2	2τ	1τ
NOR2	3τ	1τ
FF (t_{cq})	9τ	

FF (t_{setup})	10τ
T_C	21τ

Path Start Point	**Path End Point**	_____ **Constraint**

- Are there any unsatisfied constraints (i.e., timing violations)?
- What is the critical path of this gate-level network?
- What is the minimum clock period (T_C) that would still ensure correct operation?
- What is the maximum clock frequency that would still ensure correct operation?

3.2. Hold Time

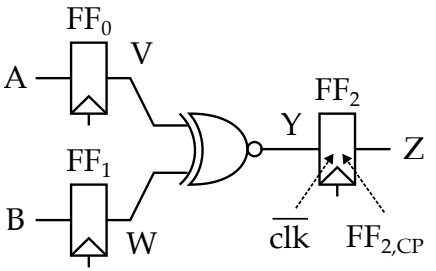


Clock-to-Q Contamination Delay ($t_{cd,cq}$)

Path	Contamination Delay	Shortest Path?

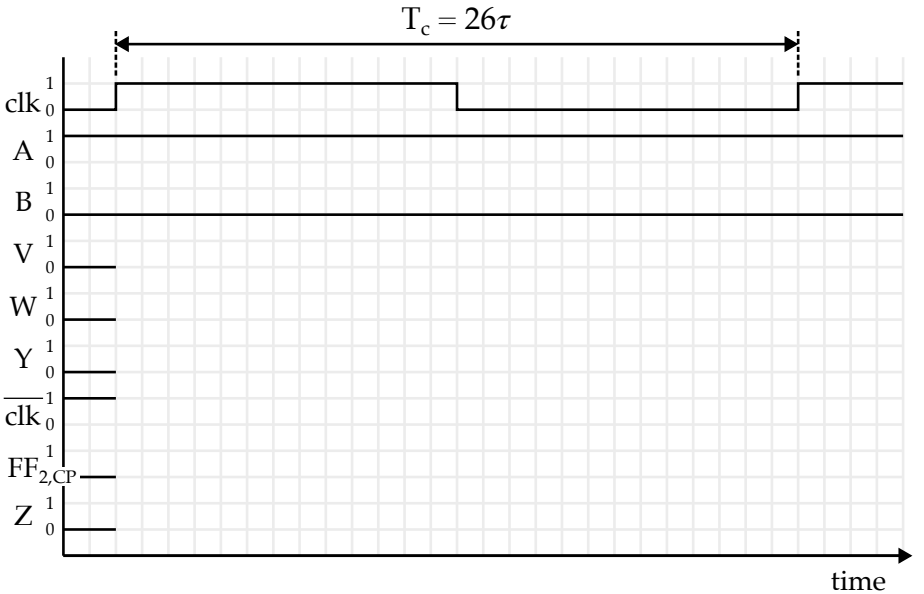
Hold Time (t_{hold})

Path	Contamination Delay	Shortest Path?

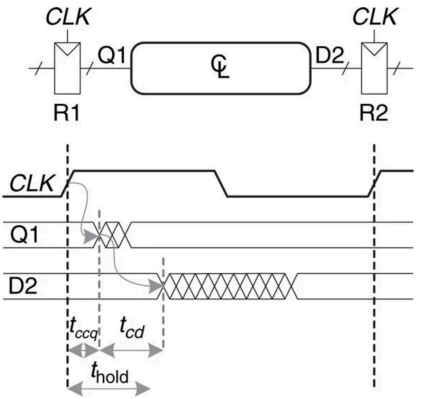


	t_{pd}	t_{cd}
XNOR2	7τ	1τ
FF (t_{cq})	9τ	2τ

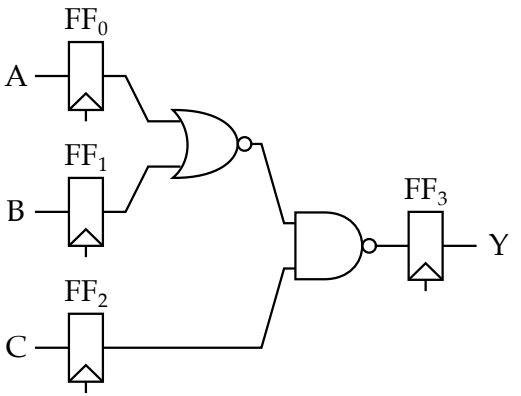
FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	26τ



Hold Time Constraint



- Hold time constraint is a *race between the clock and data*
  - We want the clock to win, we want the clock to cause the leader latch to become opaque before the data is sampled
- How can we fix a hold time violation?
- *Static timing analysis table* for sequential logic shows *the setup and hold time constraints* on every path through the gate-level network that are required to ensure correct operation
- In this course we assume the following are unconstrained
  - paths from input to any output
  - paths from input to any flip-flop
  - paths from any flip-flop to any output
- Each constraint is an inequality which is either:
  - $\geq$ or $\leq$ (satistified, no timing violation)
  - $\ngeq$ or $\nleq$ (not satistified, timing violation)

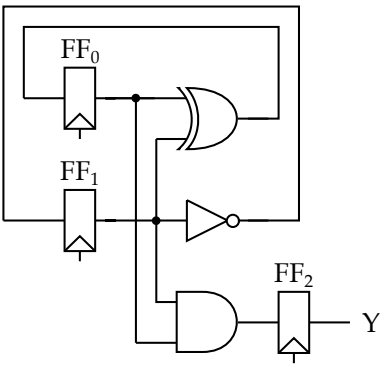


	t_{pd}	t_{cd}
NAND2	2τ	1τ
NOR2	3τ	1τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	30τ

Path Start Point	Path End Point	_____ Constraint	_____ Constraint

- Are there any unsatisfied constraints (i.e., timing violations)?
- What is the critical path and short path of this gate-level network?
- What is the minimum clock period (T_C) that would still ensure correct operation?
- What is the maximum clock frequency that would still ensure correct operation?



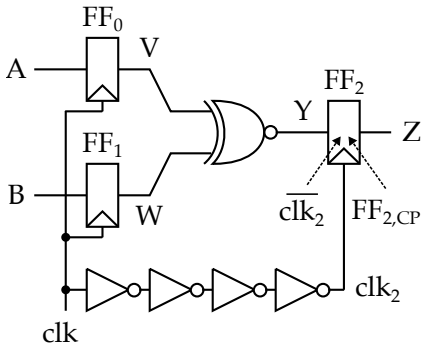
	t_{pd}	t_{cd}
NOT	1τ	1τ
AND2	3τ	1τ
XOR2	7τ	1τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	23τ

Path Start Point	**Path End Point**	_____ **Constraint**	_____ **Constraint**

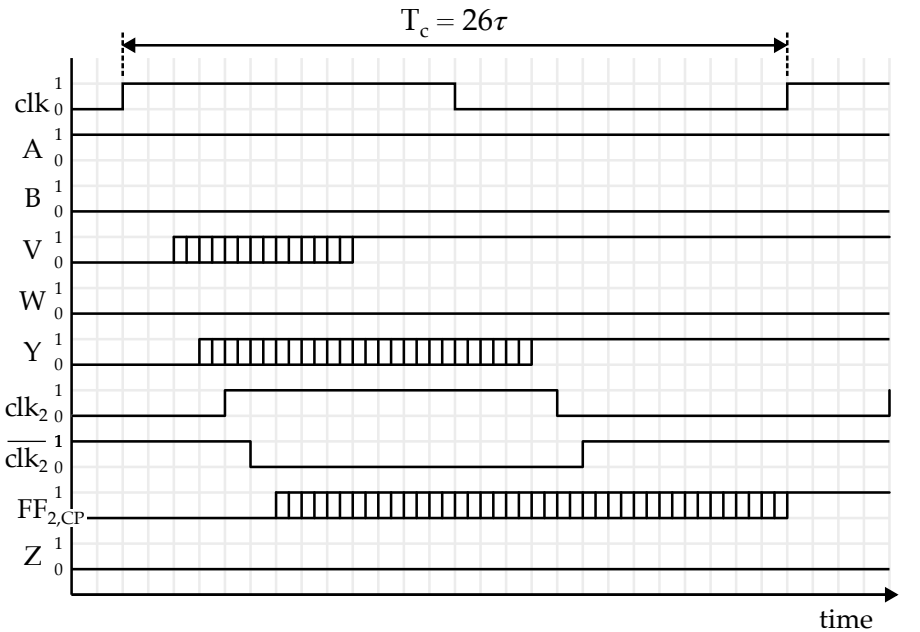
- Are there any unsatisfied constraints (i.e., timing violations)?
- What is the critical path and short path of this gate-level network?
- What is the minimum clock period (T_C) that would still ensure correct operation?

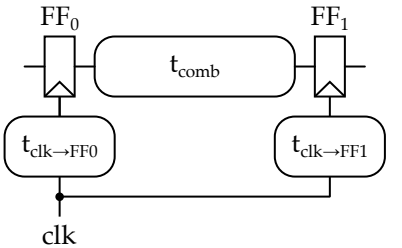
3.3. Clock Skew



	t_{pd}	t_{cd}
NOT	1τ	1τ
XNOR2	7τ	1τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
T_C	26τ

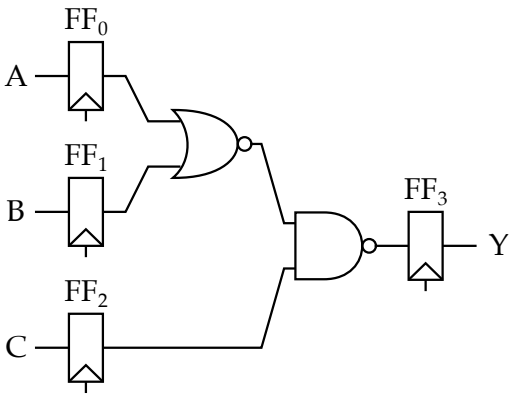




Setup Time Constraint with Clock Skew

Hold Time Constraint with Clock Skew

- Timing constraints are a *race between the clock and data*
  - Setup time constraint → we want the data to win
  - Hold time constraint → we want the clock to win
- Positive clock skew
  - Slows down the clock, easier for the data to win the race
  - Decreases *effective* setup time, easier to satisfy setup time contraint
  - Increases *effective* hold time, harder to satisfy hold time constraint
- Negative clock skew
  - Effectively slows down the data, easier for clock to win the race
  - Increases *effective* setup time, harder to satisfy setup time contraint
  - Decreases *effective* hold time, easier to satisfy hold time constraint



	t_{pd}	t_{cd}
NAND2	2τ	1τ
NOR2	3τ	1τ
FF (t_{cq})	9τ	2τ

FF (t_{setup})	10τ
FF (t_{hold})	1τ
FF (t_{skew})	4τ
T_C	30τ

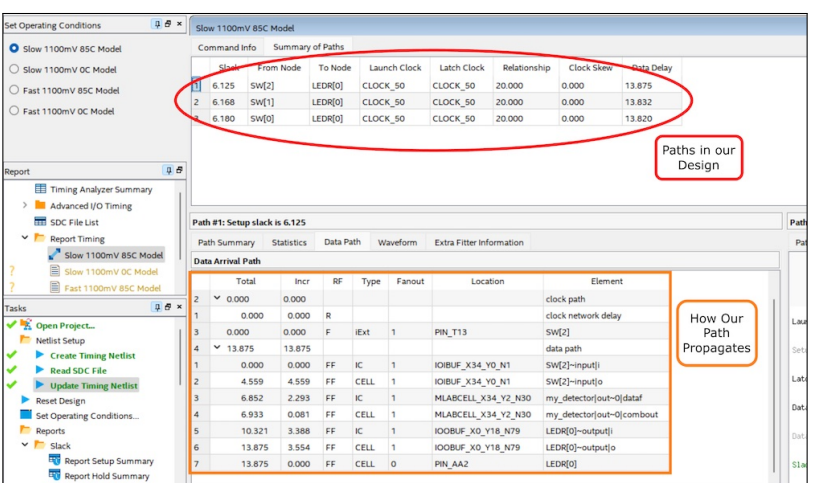
Path Start Point	Path End Point	_____ Constraint	_____ Constraint

- Are there any unsatisfied constraints (i.e., timing violations)?
- What is the critical path and short path of this gate-level network?
- What is the minimum clock period (T_C) that would still ensure correct operation?
- What is the maximum clock frequency that would still ensure correct operation?

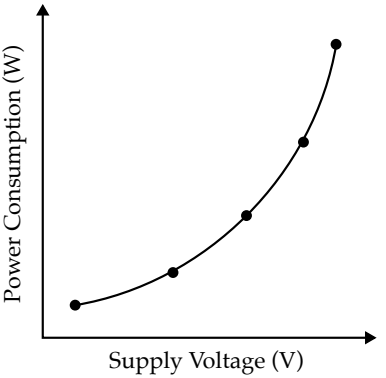
3.4. Timing Constraints in Practice

- *Static timing analysis* involves "statically" analyzing all constraints across all paths in the entire design
- *Timing closure* is the process of guaranteeing that all timing constraints are satisfied in a design (i.e., no setup time constraint violations and no hold time constraint violations)

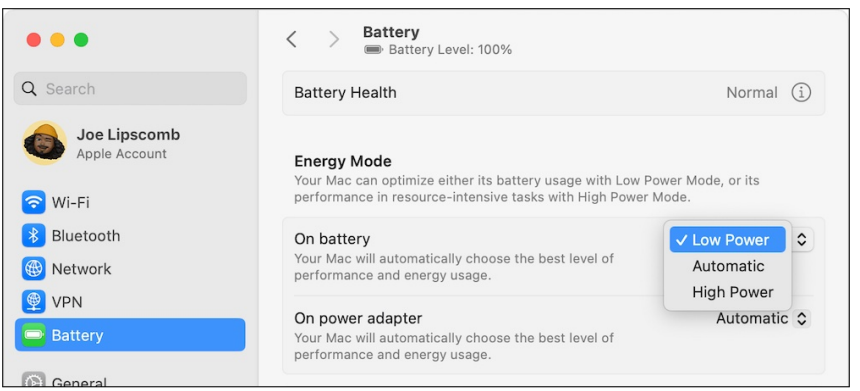
```
set_max_delay -from [all_inputs] -to [all_outputs] 20
set_min_delay -from [all_inputs] -to [all_outputs]  0

create_clock -name clk -period 20 [get_ports {clk}]

set_input_delay  -add_delay -clock clk -max  0 [all_inputs]
set_input_delay  -add_delay -clock clk -min  0 [all_inputs]
set_output_delay -add_delay -clock clk -max  0 [all_outputs]
set_output_delay -add_delay -clock clk -min  0 [all_outputs]
```



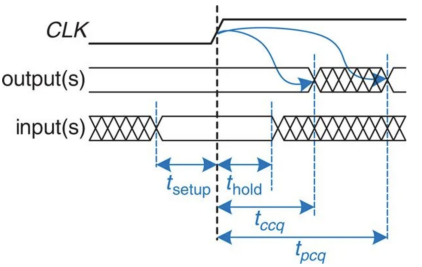
- Increasing the supply voltage makes the transistors faster
  - requires higher power consumption and uses up more energy
  - enables running at a shorter clock period (higher clock frequency)
- Decreasing the supply voltage makes the transistors slower
  - requires lower power consumption and uses up less energy
  - enables running at a longer clock period (lower clock frequency)

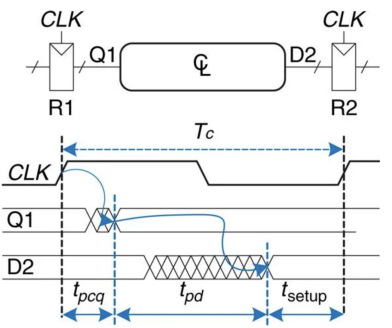


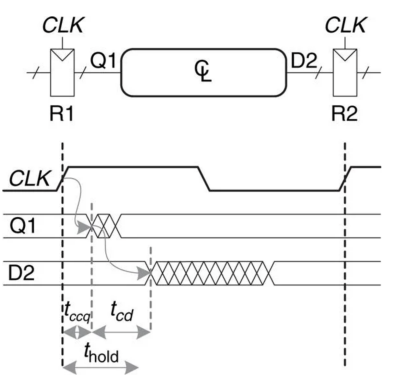
- Laptop power modes enable trade-off performance vs. power
  - When your laptop goes into *sleep mode*, it lowers the supply voltage and uses a longer clock period (lower clock frequency) to reduce power
  - When your laptop goes into *turbo boost mode*, it increases the supply voltage and uses a shorter clock period (higher clock frequency) to improve performance (but at higher power causing the fan to turn on)



3.5. Summary of Sequential Gate-Level Timing

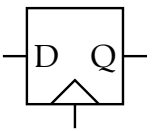




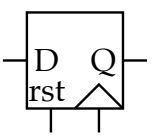


4. Verilog Modeling

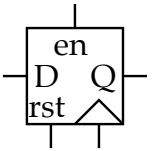
- Although possible to use *gate-level modeling* for sequential logic, far more common to use *register-transfer-level modeling* (RTL)




```
module DFF_RTL
(
  input  logic clk,
  input  logic d,
  output logic q
);

  always_ff @( posedge clk ) begin
    q <= d;
  end

endmodule
```




```
module DFFR_RTL
(
  input  logic clk,
  input  logic rst,
  input  logic d,
  output logic q
);

  always_ff @( posedge clk ) begin

    if ( rst )
      q <= 1'b0;
    else
      q <= d;

    `ECE2300_SEQ_XPROP( q, $isunknown(rst) );
  end

endmodule
```




```
1  module DFFRE_RTL
2  (
3    input  logic clk,
4    input  logic rst,
5    input  logic en,
6    input  logic d,
7    output logic q
8  );
9
10   always_ff @( posedge clk ) begin
11
12     if ( rst )
13       q <= 1'b0;
14     else if ( en )
15       q <= d;
16
17     `ECE2300_SEQ_XPROP( q, $isunknown(rst) );
18     `ECE2300_SEQ_XPROP( q, (rst == 0)
19                               && $isunknown(en) );
20   end
21
22 endmodule
```

- Even though we are raising the level of abstraction, **must always keep in mind the hardware we are modeling!**
  - Critical to keep clean separation between combinational logic and sequential logic
- In this course, `always_ff` blocks are *only* allowed in:
  - `DFF_RTL`, `DFFR_RTL`, `DFFRE_RTL`
  - Registers, Memories