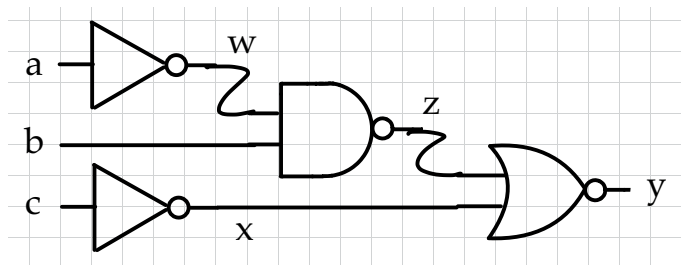


3. Sequential Gate-Level Timing

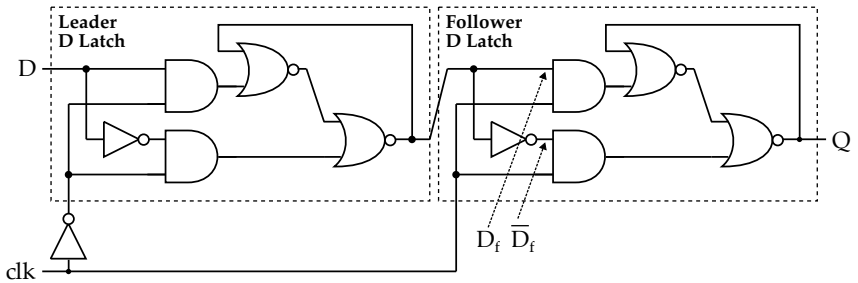
- Critical Path for Combinational Gate-Level Networks:** longest propagation path delay from any input to any output

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

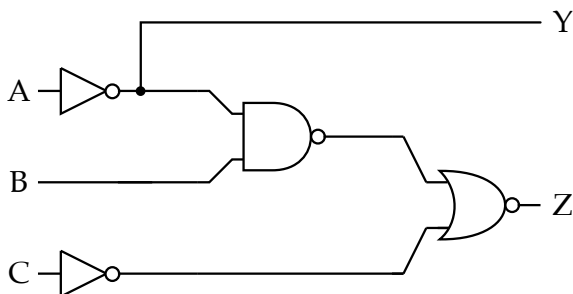


Path

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

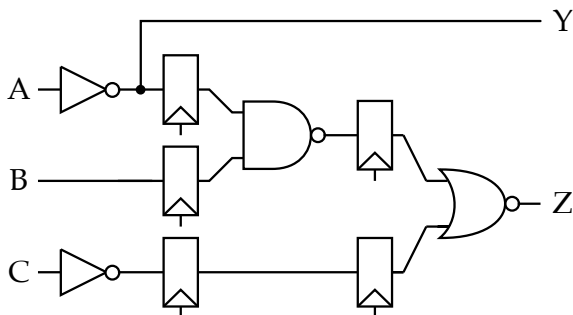
Critical Path for Combinational Gate-Level Networks

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

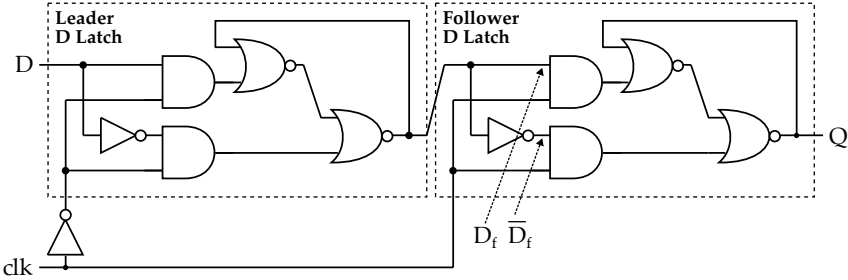


Critical Path for Sequential Gate-Level Networks

- 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

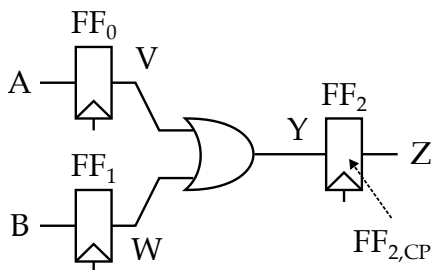


Clock-to-Q Propagation Delay ($t_{pd,cq}$)

Path	Propagation Delay	Critical Path?

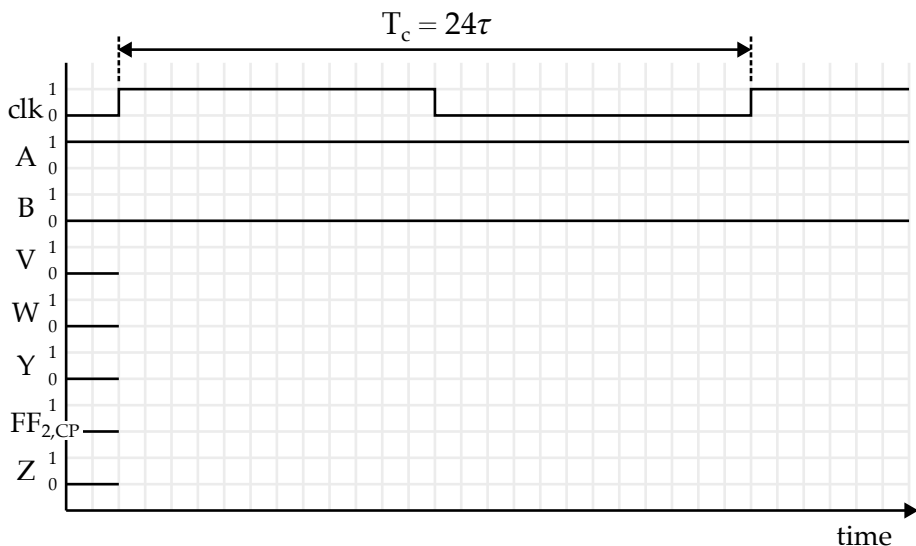
Setup Time (t_{setup})

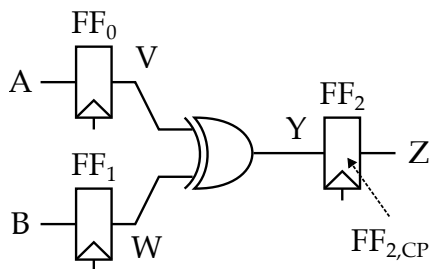
Path	Propagation Delay	Critical Path?



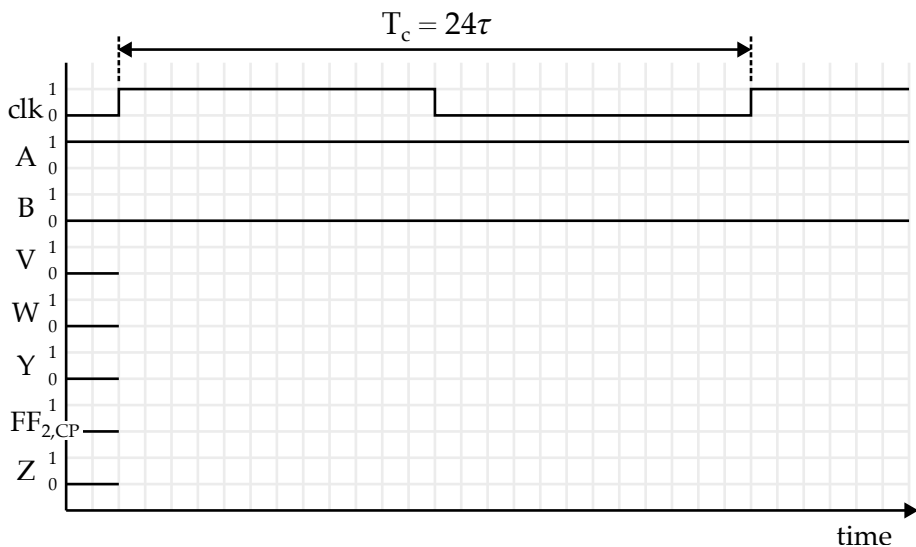
	t_{pd}	t_{cd}
OR2	4τ	1τ
FF (t_{cq})	9τ	
FF (t_{setup})	10τ	
T_C	24τ	

- *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

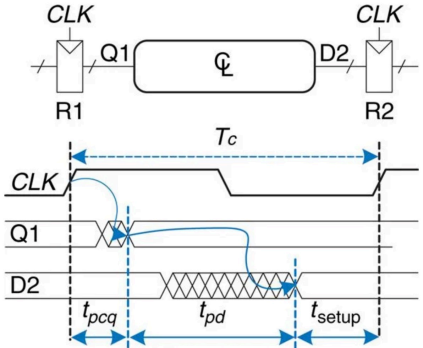




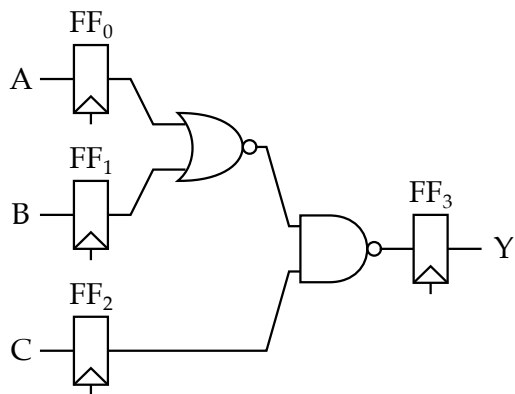
	t_{pd}	t_{cd}
XOR2	7τ	1τ
FF (t_{cq})	9τ	
FF (t_{setup})	10τ	
T_C	24τ	



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
- Three ways to fix a setup time violation
 - 1. Increase clock period (T_C)
 - 2. Decrease propagation delay (t_{pd}) by optimizing combinational logic
 - 3. Decrease propagation delay (t_{pd}) by adding registers
- *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:
 - $\geq$ (satisfied, no timing violation)
 - $\nless$ (not satisfied, timing violation)



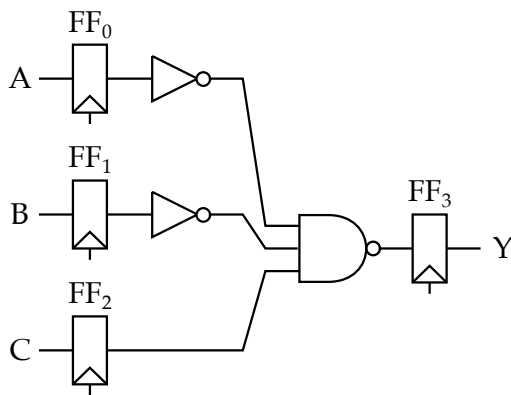
	t_{pd}	t_{cd}
NAND2	2τ	1τ
NOR2	3τ	1τ
FF (t_{cq})	9τ	
FF (t_{setup})	10τ	
T_C	23τ	

Path Start Point	Path End Point	Constraint

- What is the critical path and setup time slack?
- Will this gate-level network *always* operate correctly?

1. Increase clock period (T_C)

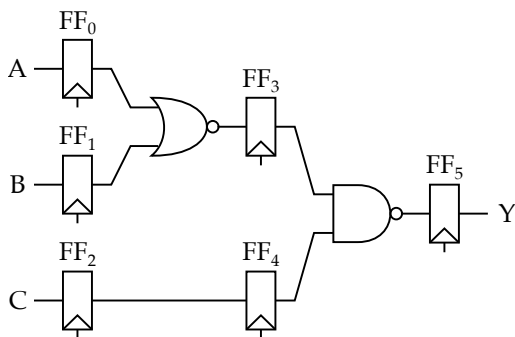
Path Start Point	Path End Point	Setup Constraint
FF0	FF3	$24 \geq 9 + 3 + 2 + 10$
FF1	FF3	$24 \geq 9 + 3 + 2 + 10$
FF2	FF3	$24 \geq 9 + 2 + 10$

2. Decrease propagation delay (t_{pd}) by optimizing combinational logic

	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND3	3τ	1τ
FF (t_{cq})	9τ	
FF (t_{setup})		10τ
T_C		23τ

Path Start Point	Path End Point	Setup Constraint
FF0	FF3	$23 \geq 9 + 1 + 3 + 10$
FF1	FF3	$23 \geq 9 + 1 + 3 + 10$
FF2	FF3	$23 \geq 9 + 3 + 10$

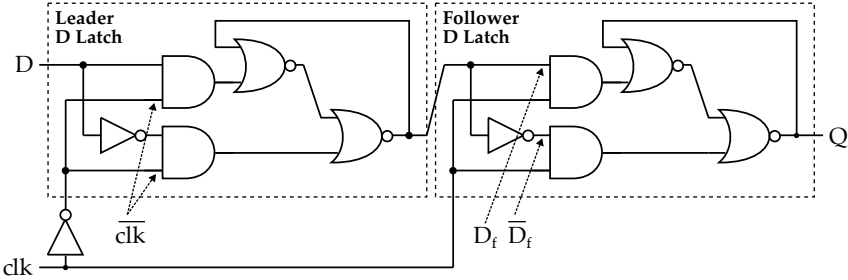
3. Decrease propagation delay (t_{pd}) by adding registers



	t_{pd}	t_{cd}
NAND2	2τ	1τ
NOR2	3τ	1τ
FF (t_{cq})	9τ	
FF (t_{setup})	10τ	
T_C	23τ	

Path Start Point	Path End Point	Setup Constraint

3.2. Hold Time

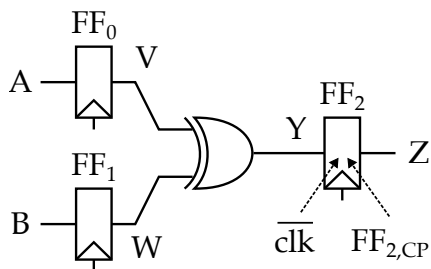


Clock-to-Q Contamination Delay ($t_{cd,cq}$)

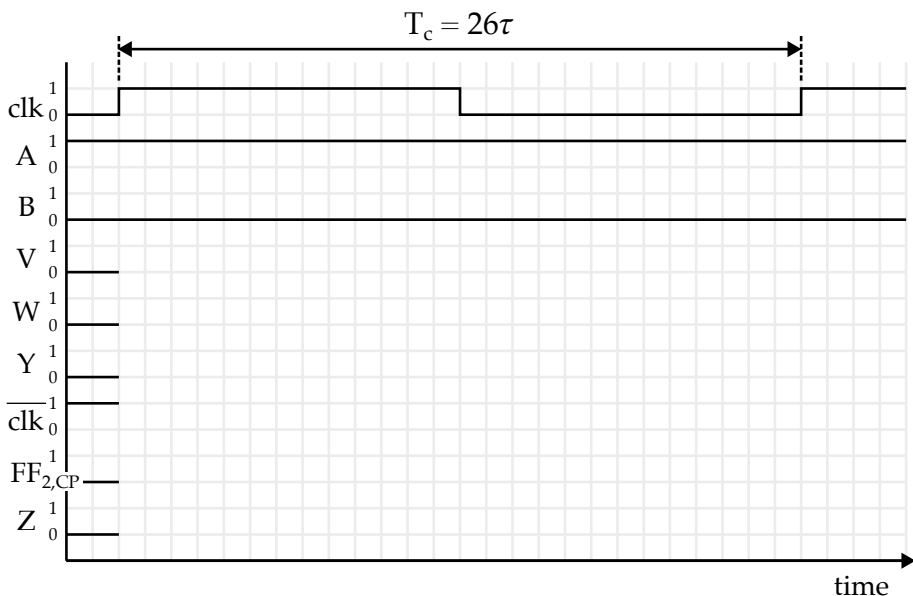
Path	Contamination Delay	Shortest Path?

Hold Time (t_{hold})

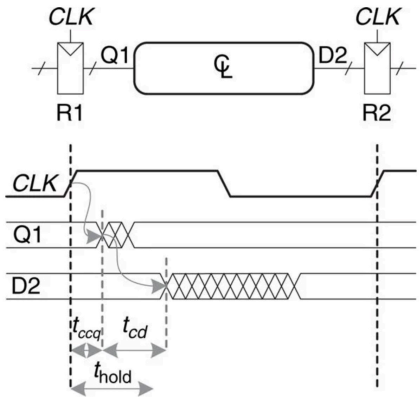
Path	Contamination Delay	Shortest Path?



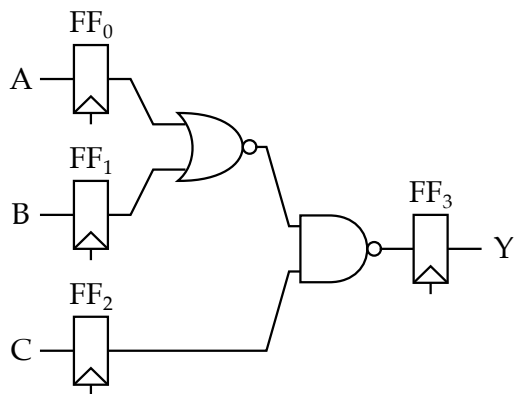
	t_{pd}	t_{cd}
XOR2	7τ	1τ
FF (t_{cq})	9τ	2τ
FF (t_{setup})	10τ	
FF (t_{hold})	1τ	
T_C	26τ	



Hold Time Constraint



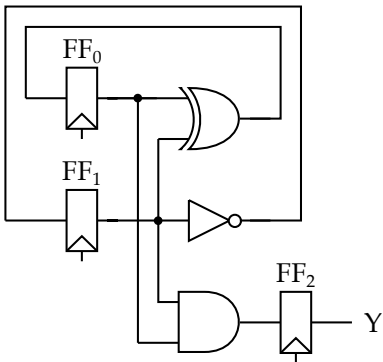
- Hold time constraint is a *race between the clock and data*
 - We want the clock to win, we want the clock to cause the leader latch to become opaque before the data is sampled
- Only way to fix a hold time violation
 - Increase contamination delay (t_{cd}) by adding buffers (i.e., two NOT gates)
- *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:
 - $\leq$ (satisfied, no timing violation)
 - $\nless$ (not satisfied, 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

- What is the critical path and setup time slack?
- What is the short path and hold time slack?
- Will this gate-level network *always* operate correctly?

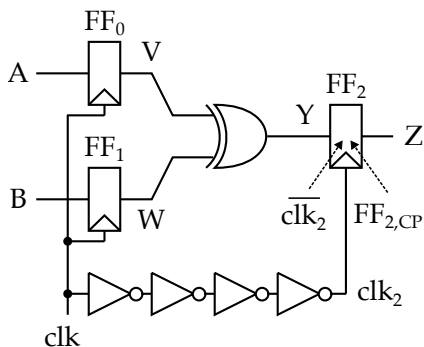


	t_{pd}	t_{cd}
NOT	1τ	1τ
AND2	3τ	1τ
XOR2	7τ	1τ
FF (t_{cq})	9τ	2τ
FF (t_{setup})	10τ	
FF (t_{hold})	1τ	
T_C	23τ	

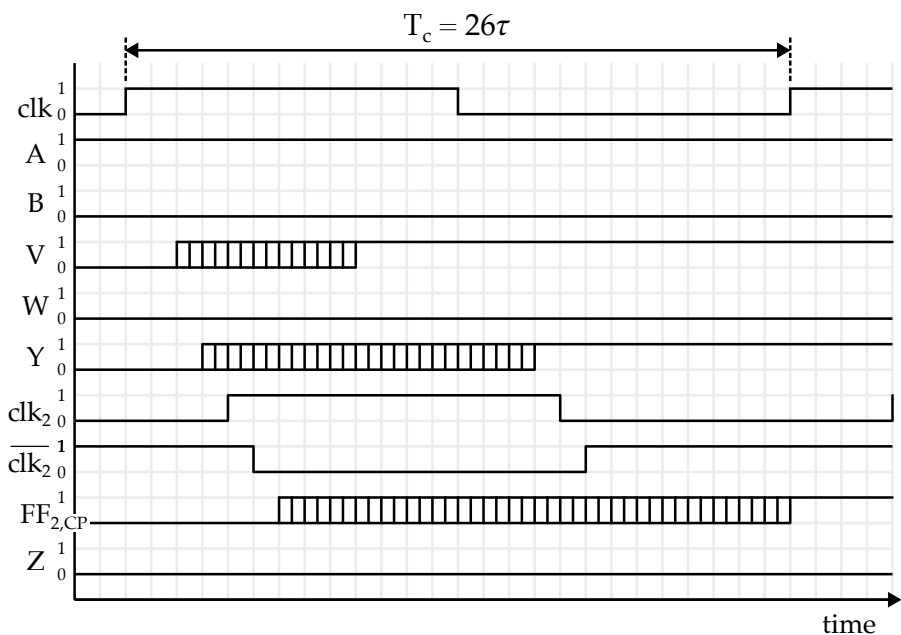
Path Start Point	Path End Point	Constraint	Constraint

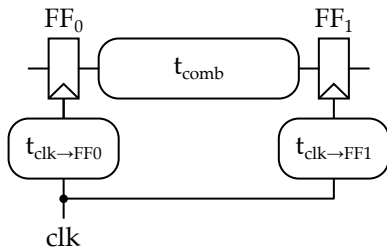
- What is the critical path and setup time slack?
- What is the short path and hold time slack?
- Will this gate-level network *always* operate correctly?

3.3. Clock Skew



	t_{pd}	t_{cd}
NOT	1τ	1τ
XOR2	7τ	1τ
FF (t_{cq})	9τ	2τ
FF (t_{setup})	10τ	
FF (t_{hold})	1τ	
T_C	26τ	

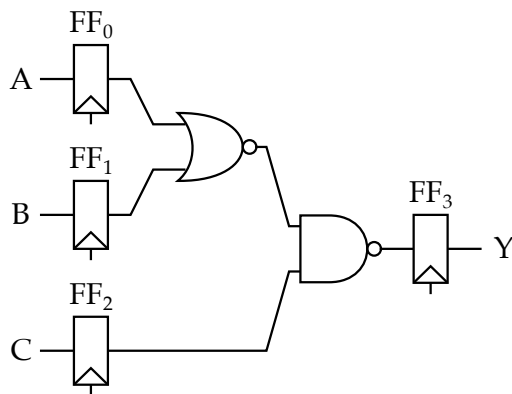




Setup Time Constraint with Clock Skew

Hold Time Constraint with Clock Skew

- Timing constraints are a *race between the clock and data*
 - Setup time constraint $\rightarrow$ we want the data to win
 - Hold time constraint $\rightarrow$ 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 constraint
 - 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 constraint
 - 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

- What is the critical path and setup time slack?
- What is the short path and hold time slack?
- Will this gate-level network *always* operate correctly?

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)

```

1 set_max_delay -from [all_inputs] -to [all_outputs] 20
2 set_min_delay -from [all_inputs] -to [all_outputs] 0
3
4 create_clock -name clk -period 20 [get_ports {clk}]
5
6 set_input_delay -add_delay -clock clk -max 0 [all_inputs]
7 set_input_delay -add_delay -clock clk -min 0 [all_inputs]
8 set_output_delay -add_delay -clock clk -max 0 [all_outputs]
9 set_output_delay -add_delay -clock clk -min 0 [all_outputs]

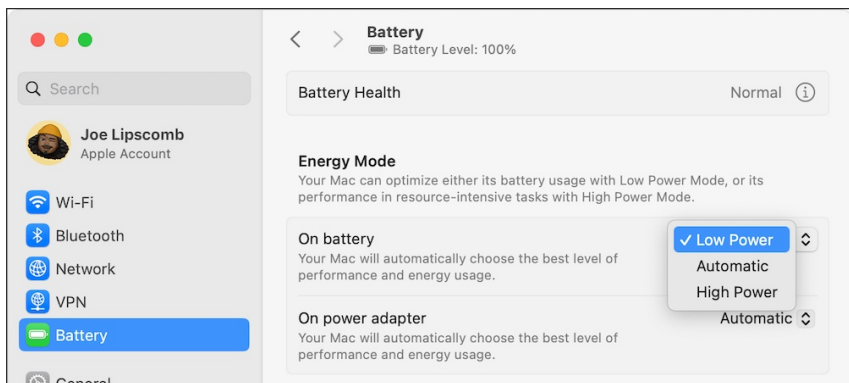
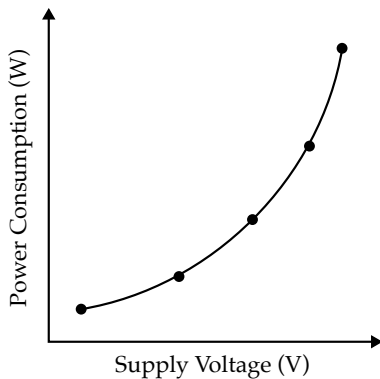
```

The screenshot shows the Xilinx Vivado timing analysis tool interface. The 'Set Operating Conditions' window is open, showing 'Slow 1100mV BSC Model'. The 'Summary of Paths' table is displayed, with a red circle highlighting the first three rows. A red box labeled 'Paths in our Design' points to the table. Below the table, the 'Path #1: Setup slack is 6.125' is shown. The 'Data Arrival Path' table is also visible, with a red box labeled 'How Our Path Propagates' pointing to it.

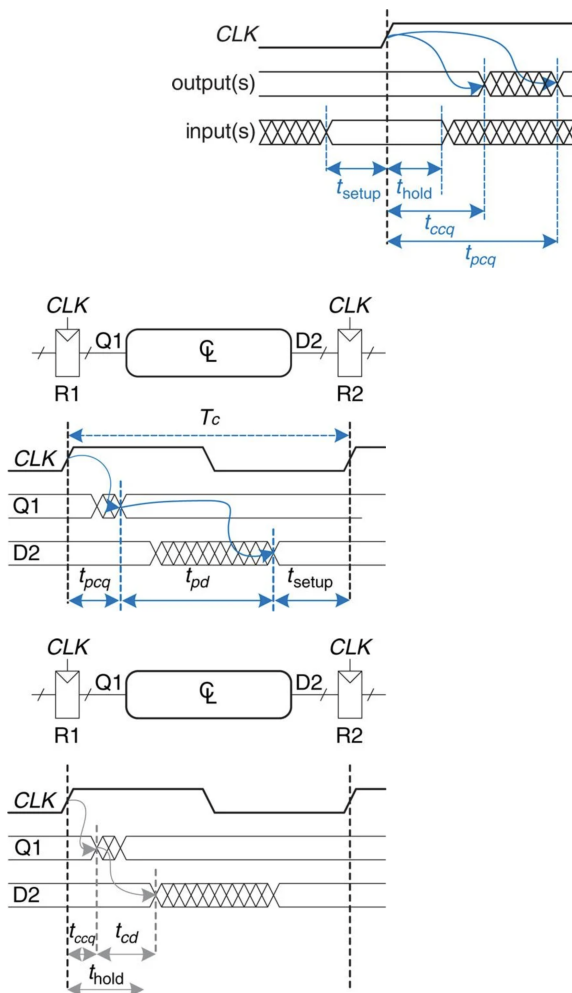
Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Setup Delay
6.125	SW[2]	LEDR[0]	CLOCK_50	CLOCK_50	20.000	0.000	13.875
6.168	SW[1]	LEDR[0]	CLOCK_50	CLOCK_50	20.000	0.000	13.832
6.180	SW[0]	LEDR[0]	CLOCK_50	CLOCK_50	20.000	0.000	13.820

Total	Incr	RF	Type	Fanout	Location	Element
0.000	0.000					clock path
0.000	0.000	R				clock network delay
0.000	0.000	F	IxT	1	PIN_T13	SW[2]
13.875	13.875					data path
0.000	0.000	FF	IC	1	IOBUF_X34_Y0_N1	SW[2]-input[i]
4.559	4.559	FF	CELL	1	IOBUF_X34_Y0_N1	SW[2]-input[o]
6.852	2.293	FF	IC	1	MLABCELL_X34_Y2_N30	my_detector[out-0]dataf
6.933	0.081	FF	CELL	1	MLABCELL_X34_Y2_N30	my_detector[out-0]combout
10.321	3.388	FF	IC	1	IOBUF_X0_Y18_N79	LEDR[0]-output[i]
13.875	3.554	FF	CELL	1	IOBUF_X0_Y18_N79	LEDR[0]-output[o]
13.875	0.000	FF	CELL	0	PIN_AA2	LEDR[0]

- 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)

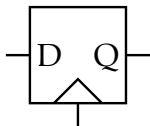


3.5. Summary of Sequential Gate-Level Timing

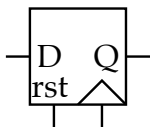


4. Verilog Modeling

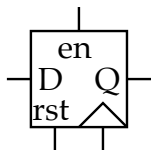
- Although possible to use *gate-level modeling* for sequential logic, far more common to use *register-transfer-level modeling* (RTL)



```
1 module DFF_RTL
2 (
3     input  logic clk,
4     input  logic d,
5     output logic q
6 );
7
8     always_ff @( posedge clk ) begin
9         q <= d;
10    end
11
12 endmodule
```



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