THE UNIVERSITY OF TEXAS AT DALLAS

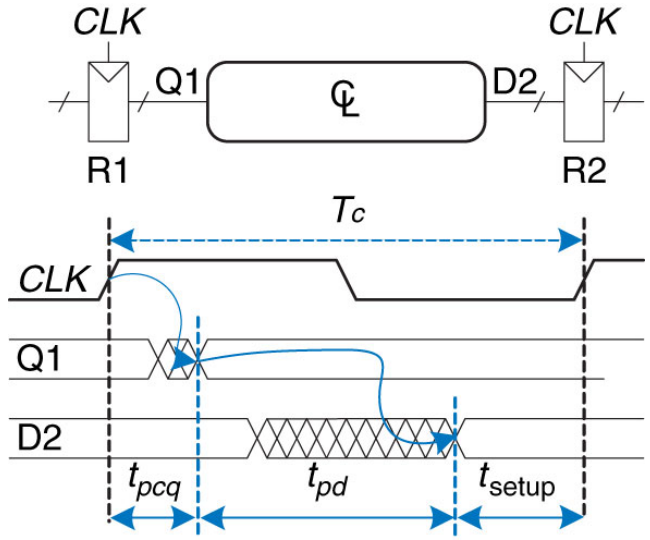
CS4341 Digital Logic & Computer Design

Lecture Notes 18

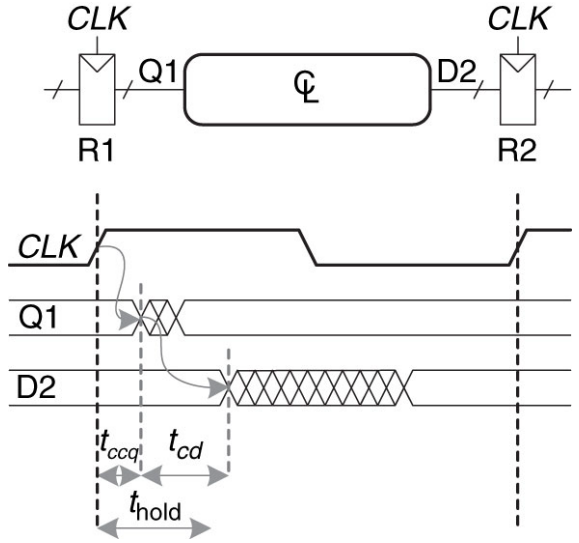
Omar Hamdy
Assistant Professor
Department of Computer Science

Review: Setup and Hold Time Constraints

- Setup Time Constraint:
  - $T_c \geq t_{pcq} + t_{pd} + t_{setup}$
  - $t_{pd} \leq T_c - (t_{pcq} + t_{setup})$

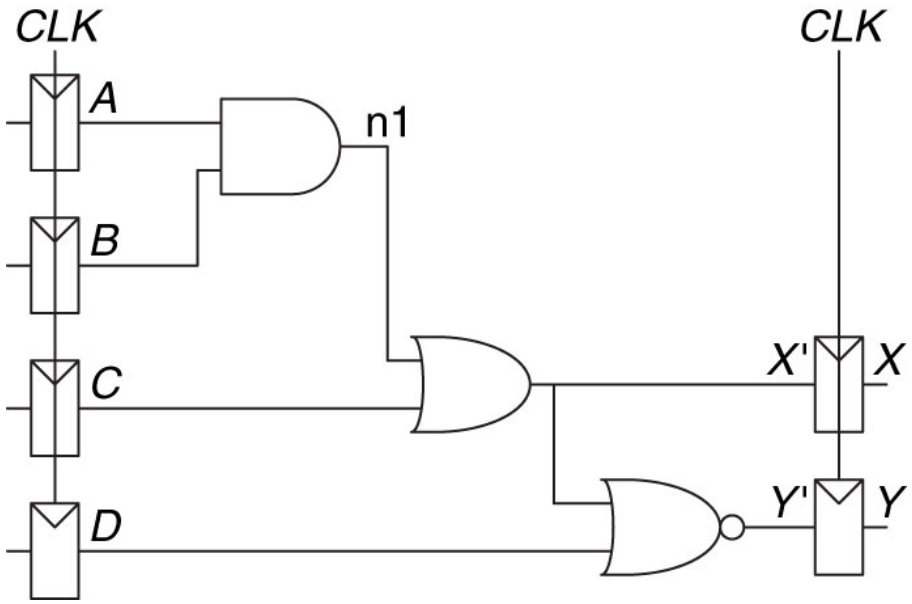


- Hold Time Constraint:
  - $t_{ccq} + t_{cd} \geq t_{hold}$
  - $t_{cd} \geq t_{hold} - t_{ccq}$



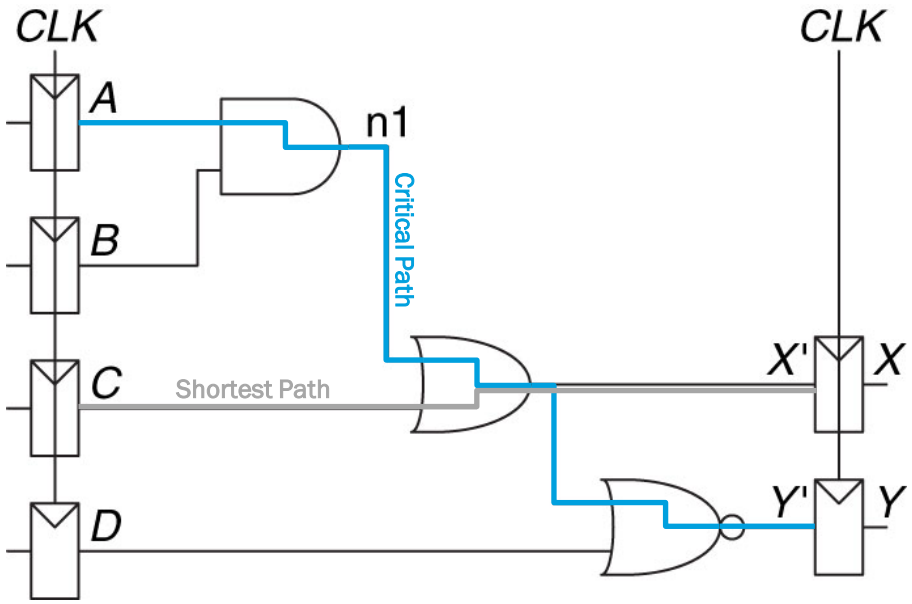
Timing Analysis Example

- The shown circuit has the following time specifications:
  - t_{ccq} = 30 ps
  - t_{pcq} = 80 ps
  - t_{setup} = 50 ps
  - t_{hold} = 60 ps
  - t_{pd} = 40 ps
  - t_{cd} = 25 ps
- What is the maximum clock frequency?
- Is there any hold time violations?

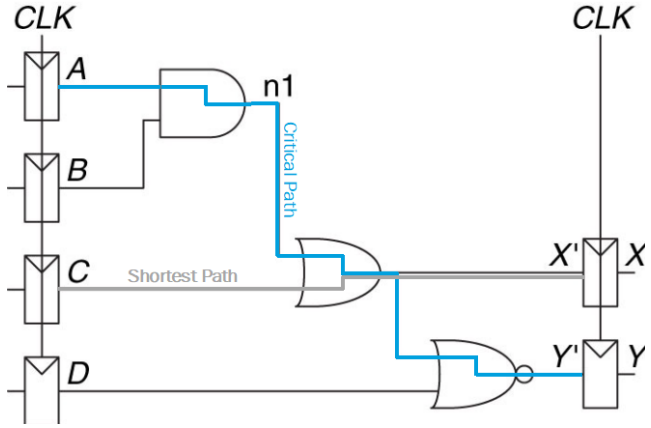


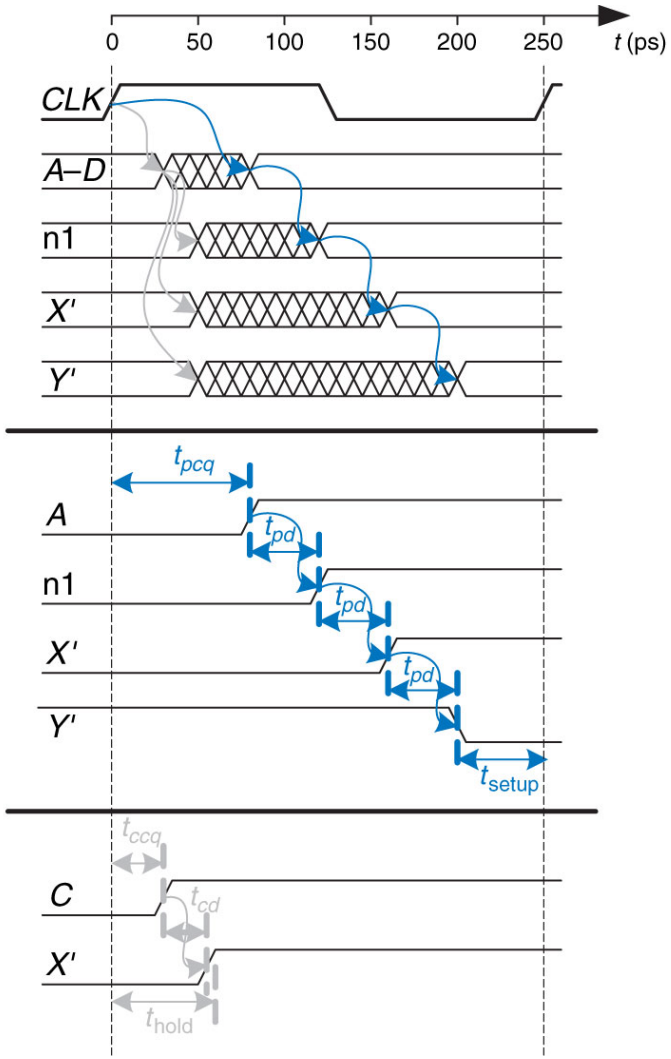
Solution: Critical and Shortest Paths

- Determine the critical path (sum of all t_{pd} on the longest path)
- Determine the short path (sum of all t_{cd} on the shortest path)



Solution: Critical and Shortest Paths

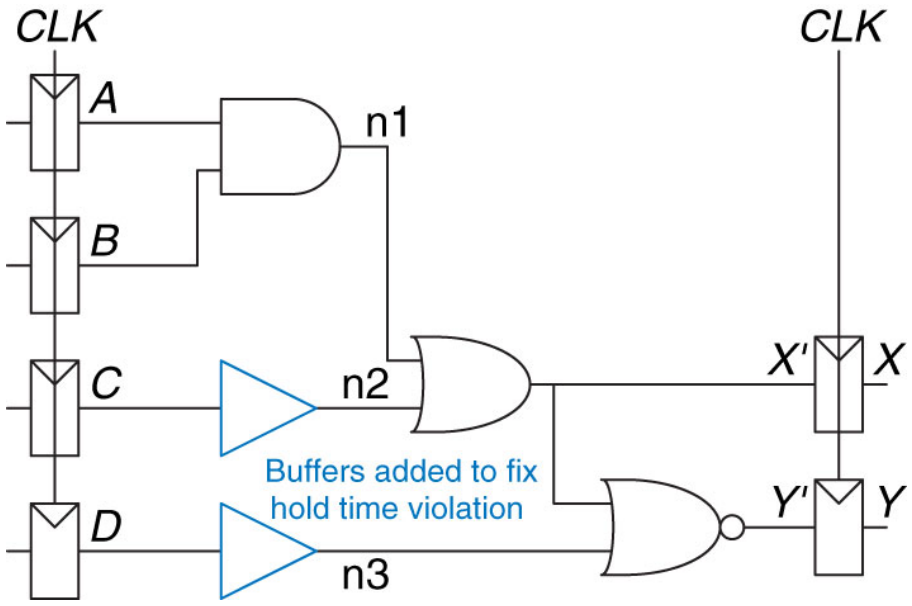




- Calculate $T_c \geq t_{pcq} + 3t_{pd} + t_{setup}$
- $T_c \geq 80 + 3x40 + 50 = 250$ ps
- Therefore, maximum clock frequency $f_c = 1/T_c \leq 4$ GHz
- For the short path, X' can start changing as early as $t_{ccq} + t_{cd}$
- Hence, X' will start changing as early as 30 + 25 = 55 ps, which violates the flip-flop hold time of 60 ps.

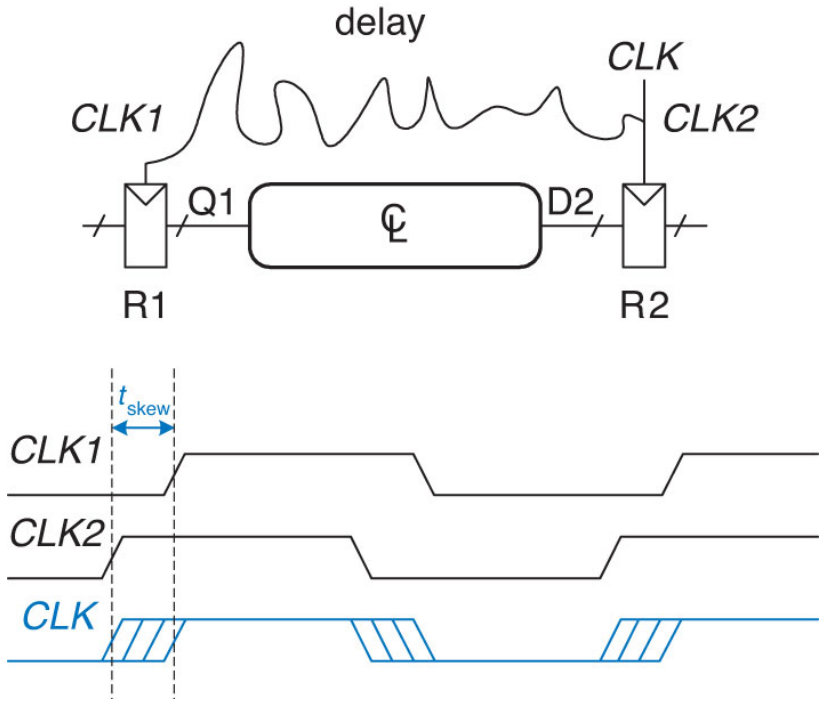
Timing Analysis Example 2

- The buffers in this circuit were added to slow down the short path to fix the hold time violation.
- What is the maximum clock frequency?
- Is there still any hold time violations?
- Solution:
  - Critical path is not affected (why)?
  - Hence, clock frequency is the same ≤ 4 GHz
  - For short path, X' will start changing after 30 + 2 x 25 = 80 ps. This is more than the required 60 ps hold time. Therefore, there is no violation in the circuit.



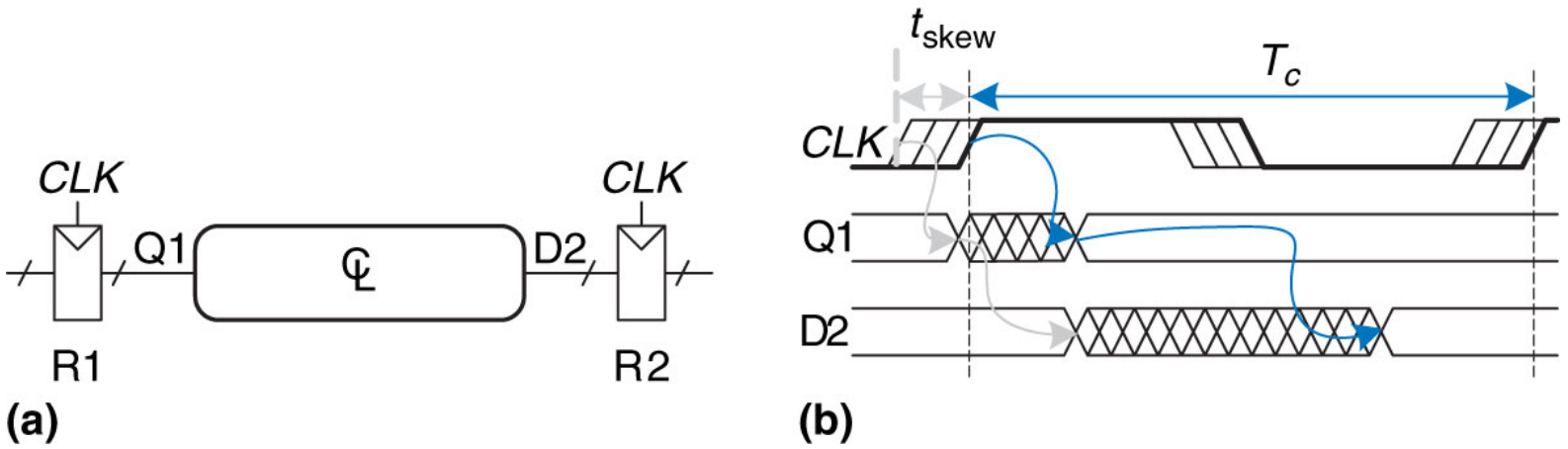
Clock Skew

- The clock itself can have its own timing problems caused by many reasons.
  - Wire length, noise, clock gating, etc
  - This variation in clock edges is called clock skew



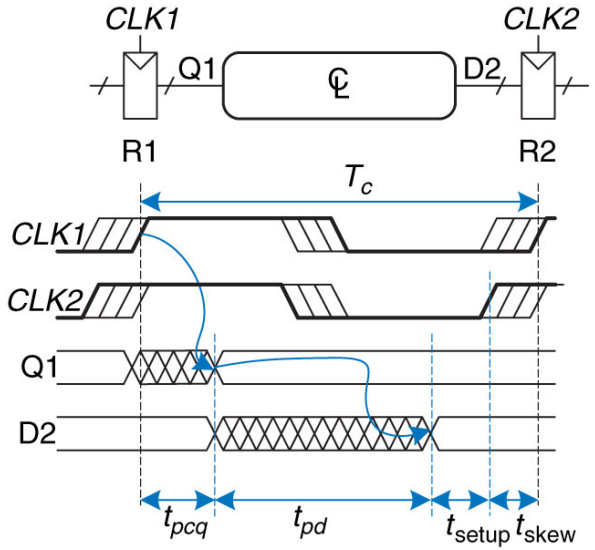
Clock Skew Time

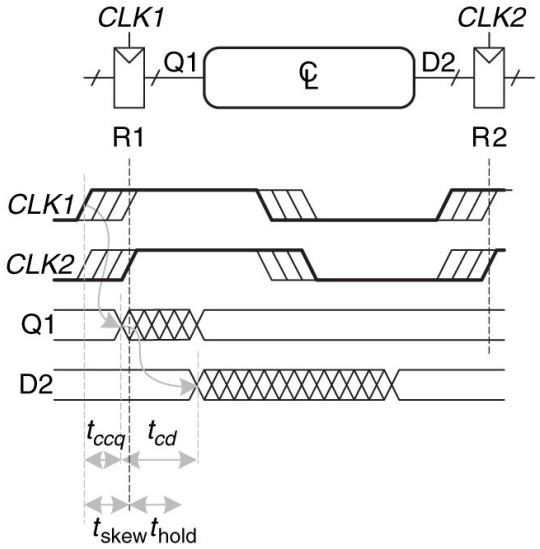
- The worst-case clock skew timing must be added to the timing calculations.
  - The dark clock line is latest time the clock signal might reach a register. All other lines means the signal might arrive up to t_{skew} earlier



Accounting for Clock Skew

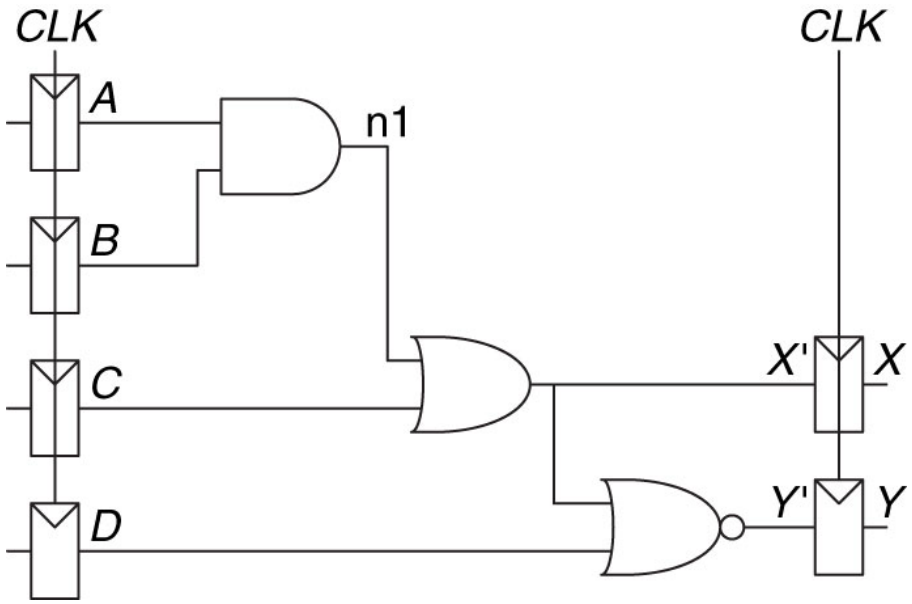
- Clock skew needs to be accounted for in T_c, t_{pd}, t_{cd} calculations.
  - $T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$
  - $t_{pd} \leq t_c - (t_{pcq} + t_{setup} + t_{skew})$
  - $t_{cd} \geq t_{hold} + t_{skew} - t_{ccq}$





Timing Analysis with Clock Skew Example

- Study the impact of a 50 ps clock skew on the same circuit analysis
- Solution
  - Critical path and short path remain the same with only calculation difference
  - $T_c \geq t_{pcq} + 3 \times t_{pd} + t_{setup} + t_{skew}$
  - $T_c \geq 80 + 3 \times 40 + 50 + 50 = 300$ ps
  - Therefore, $f_c = 1/T_c \leq 3.33$ GHz
  - t_{cd} is the same as 55 ps, but the hold time is now 60 + 50 = 110 ps. Hence, the hold violation became much worse.
  - How can we fix this problem?
    - More buffers, or flip-flops with shorter hold time.



Parallelism

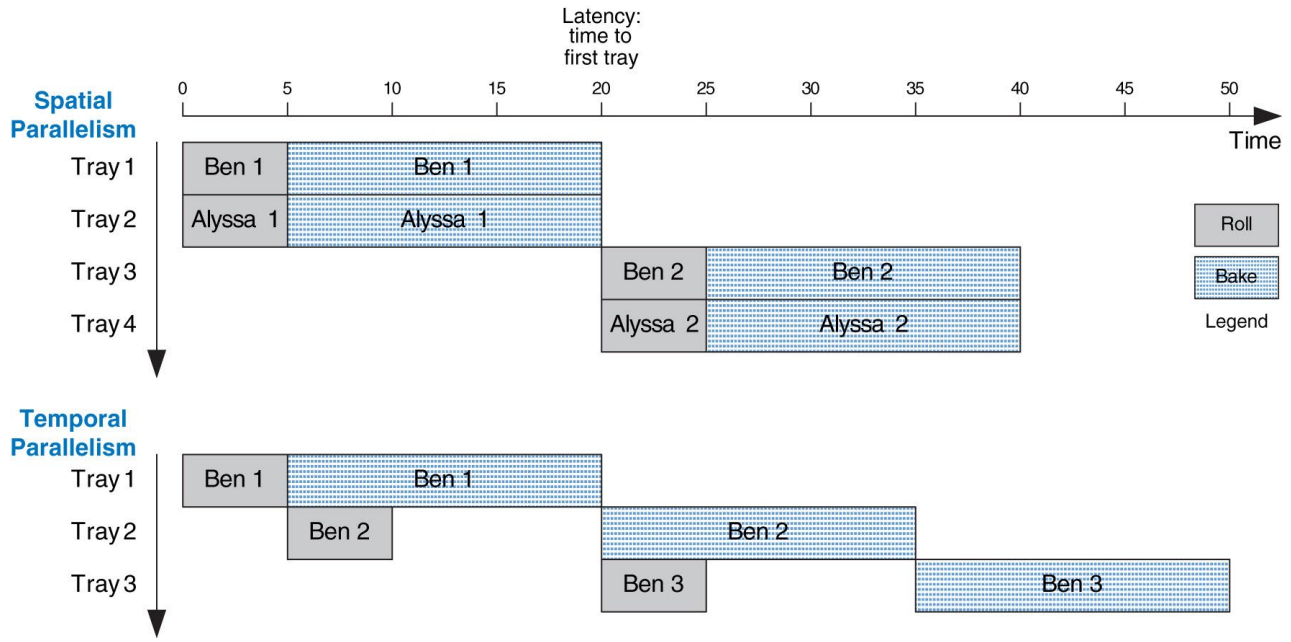
- We start with some key definitions:
  - Token: a group of inputs that are processed to produce a group of outputs
  - Latency: The latency of a system is the time required for one token to pass through the system from start to end.
  - Throughput: The throughput is the number of tokens that can be produced per unit time
- How can we increase the system throughput?
  - One method is to try to decrease the latency, so more tokens can be processed per unit time
  - Another method is to try processing multiple tokens at the same time (parallelism)

Parallelism

- Parallelism has two forms:
  - Spatial parallelism: multiple copies of the hardware are provided so that multiple tasks can be done at the same time.
  - Temporal parallelism (pipelining): a task is broken into stages, like an assembly line, and hence multiple tasks can be spread across the stages.
  - Although each task must pass through all stages, a different task will be in each stage at any given time so multiple tasks can overlap.
  - Simple example: assume preparing a tray of cookies (token) requires 5 minutes rolling in the tray and 15 minutes baking. Study the spatial parallelism if we use two ovens and two helpers, vs a temporal parallelism approach.
  - Use 1 hr as the time unit

Parallelism

- Single process: latency is 1/3 hr and throughput is 3 trays/hr
- Spatial parallelism: Latency is the same, but throughput is double the single process of 6 trays/hr
- Pipelining: Once a tray is put in the oven, work on rolling the next. Latency is the same, and throughput is 4 trays/hr (why)?

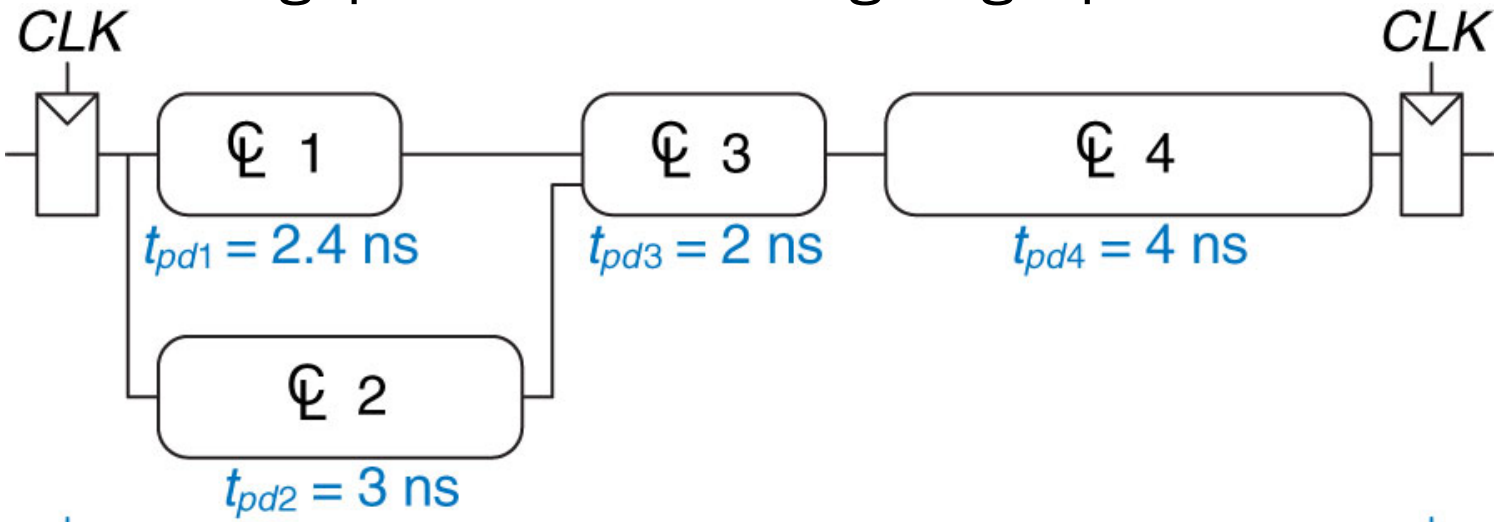


Parallelism

- In digital circuits, if a task has latency L, in a single process system, the throughput is 1/L
- In a spatially parallel system with N copies of the hardware, the throughput is N/L
- In pipelined system, ideally the task is broken into N equal stages, hence throughput is N/L (why)?
- However, it is not practical to assume there will be N equal stages. If the longest step takes L_1, then the throughput is $1/L_1$ in pipelines system.
- Pipelining is advantageous over spatial parallelism because it can speed up the performance without duplicating the hardware.

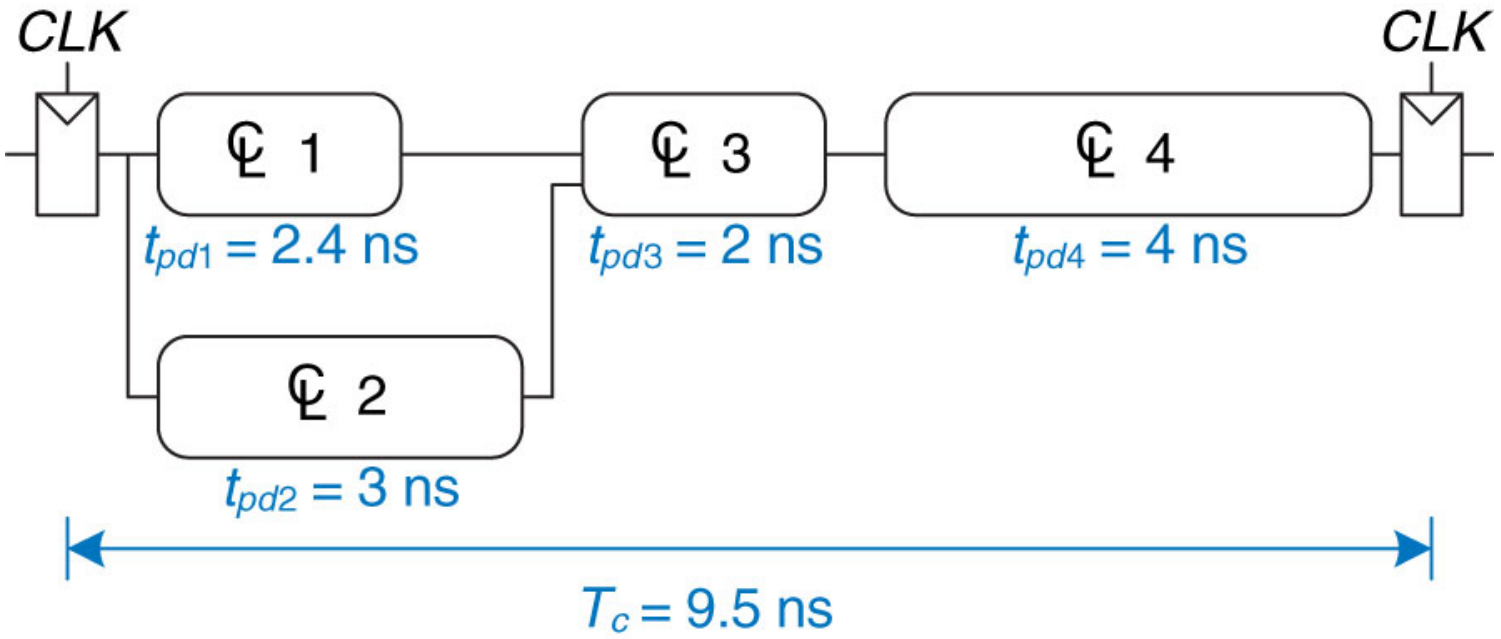
Pipelining Implementation

- In pipelining, the combinational circuit is divided into smaller stages by insertion of registers.
- Registers are important to prevent tokens from catching up with one another.
- Calculate the throughput of the following single process circuit



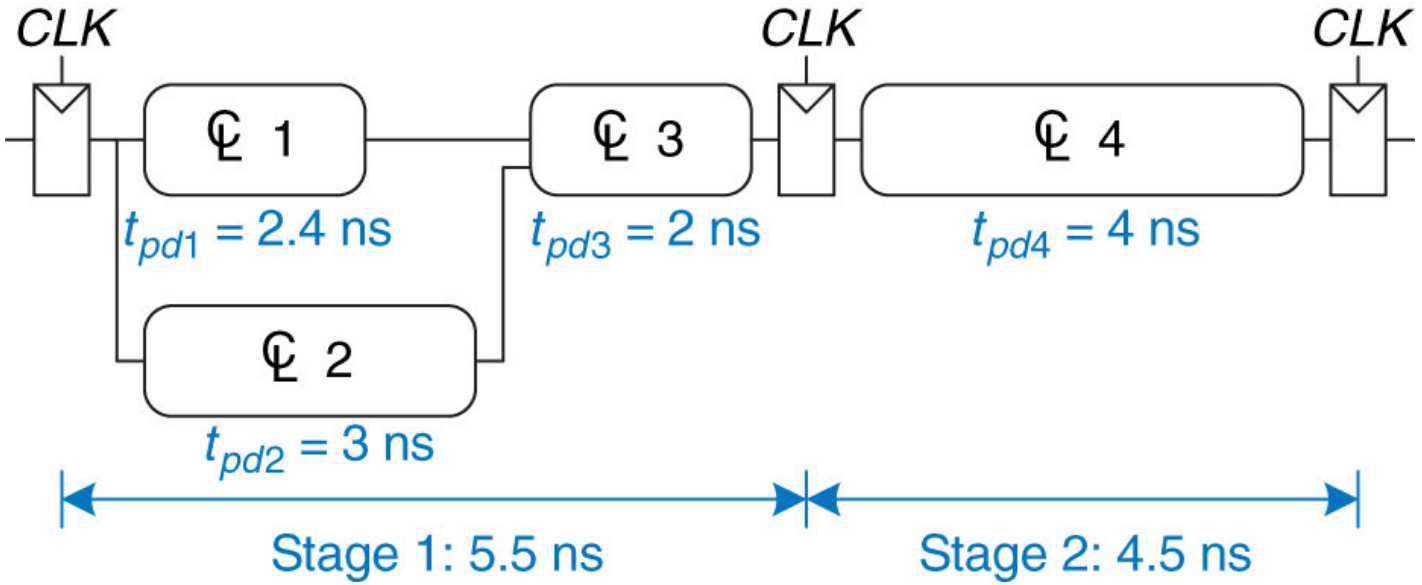
Throughput Calculation – No Pipelining

- First, we calculate the latency, which is Tc in this circuit (why)?
  - $T_c = t_{pcq} + t_{pd} + t_{setup}$
  - t_{pd} = sum of all t_{pd} on the critical path (CL 2, 3 and 4) = 3 + 2 + 4 = 9 ns
  - Assume t_{pcq} = .3 ns and t_{setup} = 0.2 ns
  - T_c = 9 + .5 = 9.5 ns ➔ throughput $1/T_c$ = 105 MHz



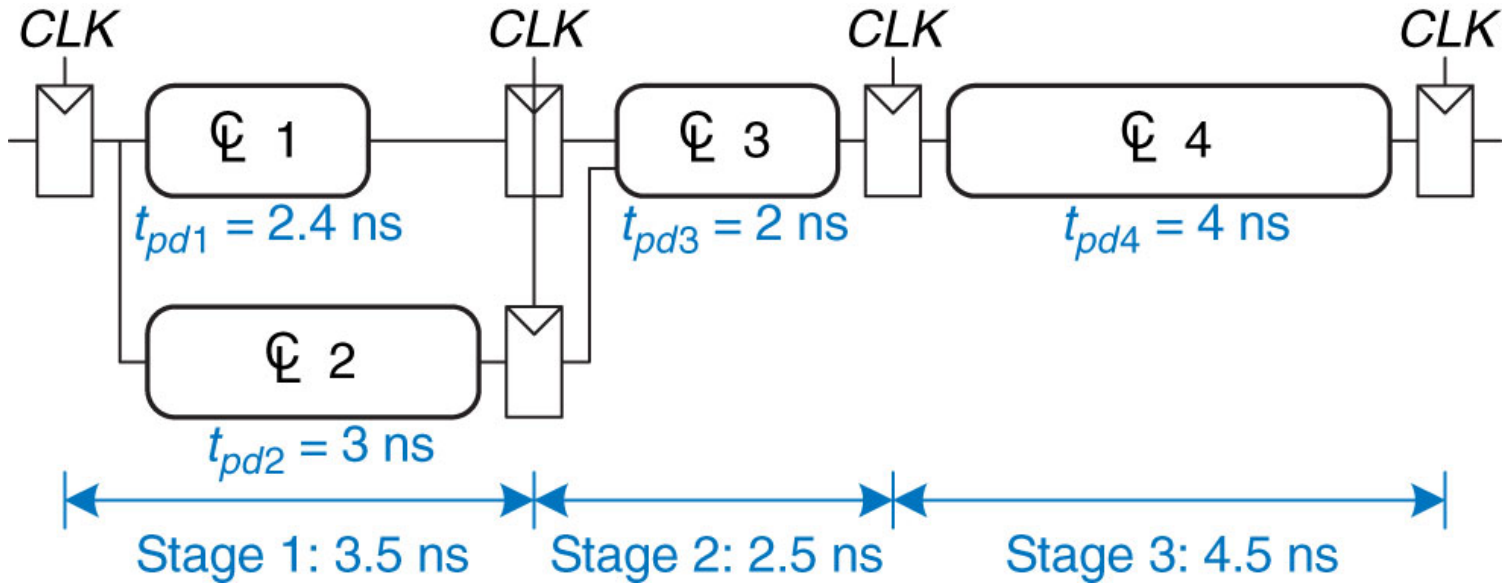
Throughput Calculation – 2 Stage Pipeline

- First, we calculate the latency as T_c in each stage.
  - T_{c1} = 5 + .5 = 5.5 ns
  - T_{c2} = 4 + .5 = 4.5 ns
  - System clock has to follow the slower T_c, hence, Circuit T_c = 5.5 ns
  - Therefore, throughput $1/T_c$ = 182 MHz. However, latency L is now 5.5 x 2 = 11 ns (why?)



Throughput Calculation – 3 Stage Pipeline

- Again, we calculate the latency as T_c in each stage.
  - T_{c1} = 3 + .5 = 3.5 ns
  - T_{c2} = 2 + .5 = 2.5 ns
  - T_{c2} = 4 + .5 = 4.5 ns
  - Therefore, throughput $1/T_c$ = 1/4.5ns = 222 MHz. However, latency L = 13.5 ns



Pipelining Limitation

- Pipelining is very powerful technique to increase the throughout. However, it assumes there are no dependencies between tokens (or tasks).
- In many cases, pipelining is of no significant use due to the dependencies between processes.
- Imagine in the cookie example if we had only one tray to use!!
- So in some situations, we experience the latency delay with no real throughput enhancement.

To Do List

- Review lecture notes
- Create an account on EDA Playground
- Study chapter 4