Lecture 6

Sequencing Digital Logic

Adapted from slides by Mark Horowitz

with significant contributions from Subhasish Mitra
and material from David Harris

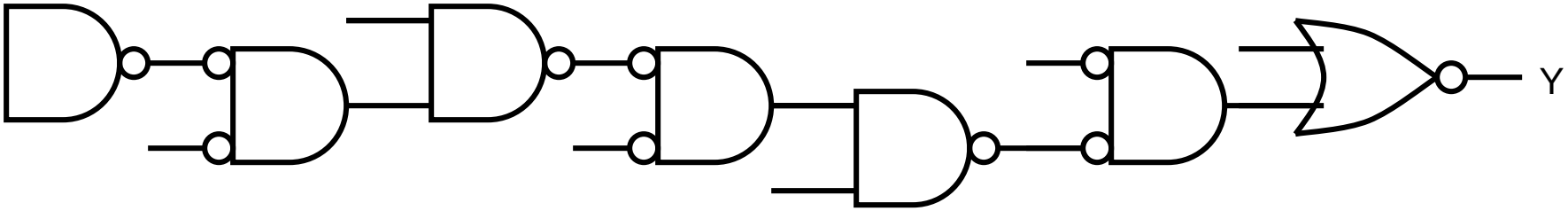
Overview

- **Reading**
  - W H 7.1, 7.2, 7.4

- **Introduction**

  Until now we covered combinational circuits, where the output is determined by the inputs. In most designs, you also want to build circuits where there are more than one calculation going on in the machine, or you want to build circuits where the output depends on both the inputs and some previous inputs. In these situations you need to do some sequencing. This is usually done using clocks and storage elements (latches and flip-flops). This lecture looks at the function that clocks serve in a system, and the trade-offs between the different clocking methods.

Problem with Combinational Logic: Most Gates are Idle Most of the Time



- If we just built combinational logic
  - Almost all the gates will be idle all the time
  - Only one useful transition in the logic at one time
    - That means most of the gates are really sitting idle
    - Seems like a waste

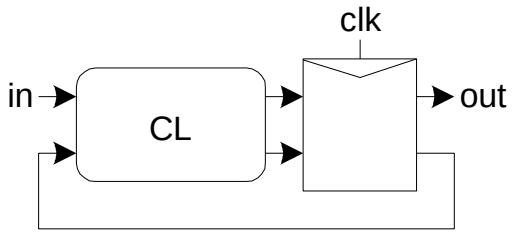
- Want to reuse the logic to make it more “efficient”
  - More work per unique time per square mm

Solution: Store State

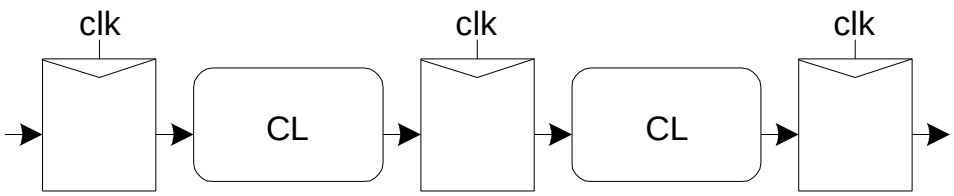
- *Combinational logic*
  - Output depends on current inputs
- *Sequential logic* (logic that uses state)
  - State:
    - The data that future computation will need to know
    - Updated on each cycle
  - Output depends on current inputs and state
  - Since state (and outputs) update each cycle
    - We are reusing hardware, which is good
    - Requires separating previous, current, future computations
    - Requires some barriers to make sure signals don't mix

Two Motifs: FSM and Data Pipelines

- When we build sequential logic
  - There are two abstract models I use
- Finite State Machine (FSM)
  - Abstraction for most control operations
  - Using state and inputs, compute outputs and next state
- Pipelined Dataflow
  - Given a complex operation, break it up over many clock cycles



Finite State Machine



Pipeline

Think About the Two Functions Differently

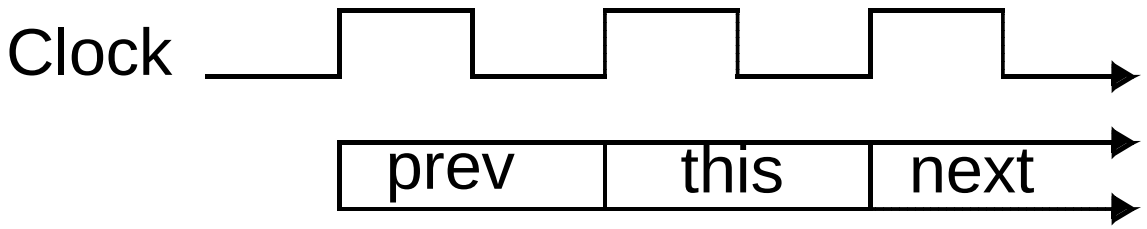
- FSM
  - State is critical, and generally not large
  - How the state is encoded might not matter that much
    - It is the abstract notion of where you are that matters
  - Case statements are very useful for FSM descriptions

- Pipelined dataflow
  - State is really just temporary values in flow
    - Might not really care which values are captured
  - Just want to start a new computation each cycle
    - Even though the functional unit delay is >> one cycle

Sequential Circuits Require Barriers

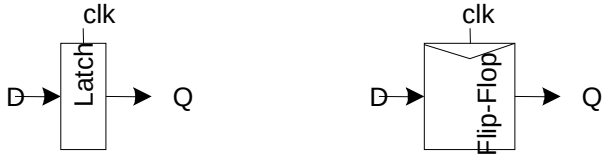
- The whole reason that we need clocks is that we want the output to depend on more than just the inputs. We want it to depend on previous outputs too, or we want to have multiple things going on in the hardware at the same time.

- The fact that there are multiple things going on in the machine at the same time, we need some means of distinguishing the different problems from each other. This is almost always done with the help of a clock, to provide reference points in time.

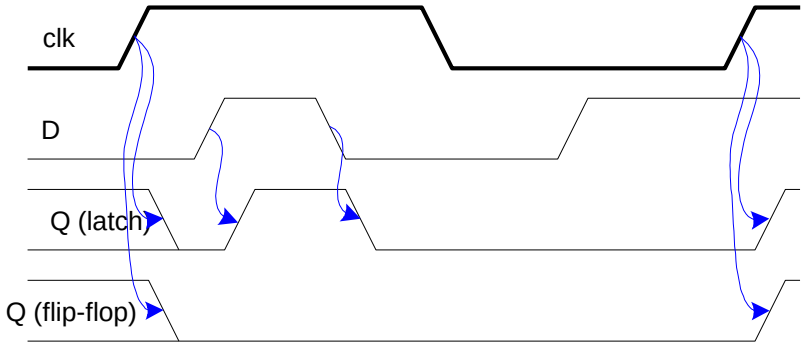


Conventional View: Sequencing Elements: State-Storage

- **Latch**: Level sensitive
  - a.k.a. transparent latch, D latch
  - When the clk is low, the previous input is stored
- **Flip-flop**: edge triggered
  - A.k.a. master-slave flip-flop, D flip-flop, D register
  - On rising edge of clk, the input is stored, transferred to output

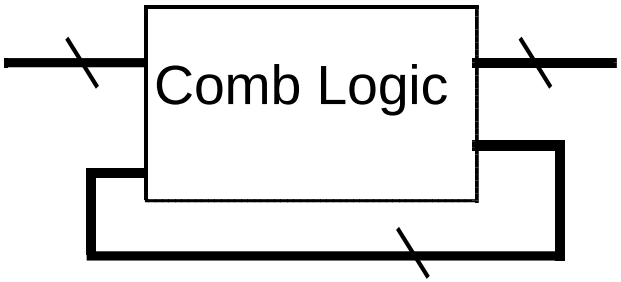


- Timing Diagrams
  - Transparent
  - Opaque
  - Edge-trigger



Important Point

- The real issue is to keep the signals correlated in time
- I don't really care where the boundaries are
  - All I want to know is that the signals don't mix



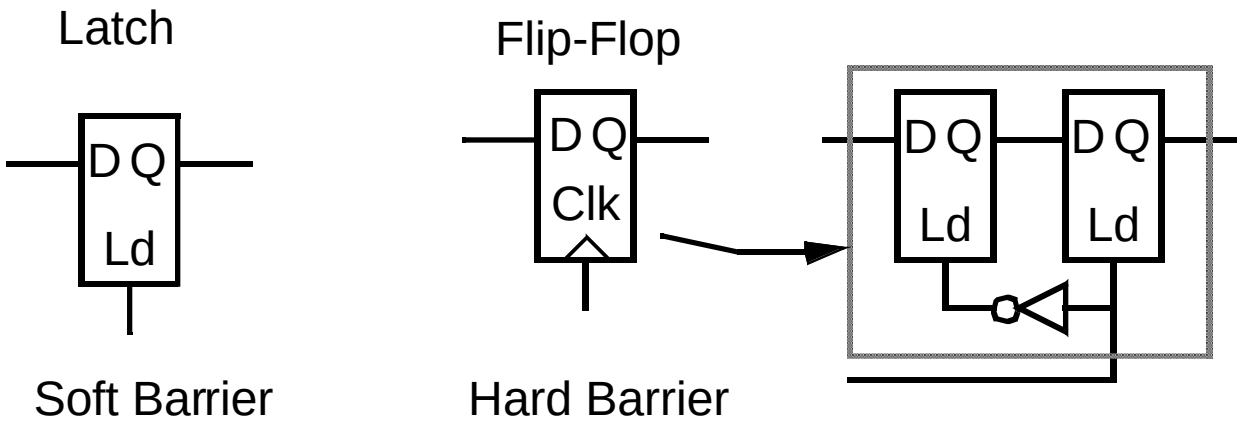
- If the delay of every path through comb. logic was **EXACTLY** the same (EXTREMELY hard to guarantee in practice)
  - Then I would not need clocks
  - Signals stay naturally correlated in time
  - The 'state' is stored in the gates and the wires.

Sequencing

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- This is called *wave pipelining* in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones. As we will see later, the tokens that are too fast can cause trouble (also called hold time problem).

So the Alternative, More Correct View

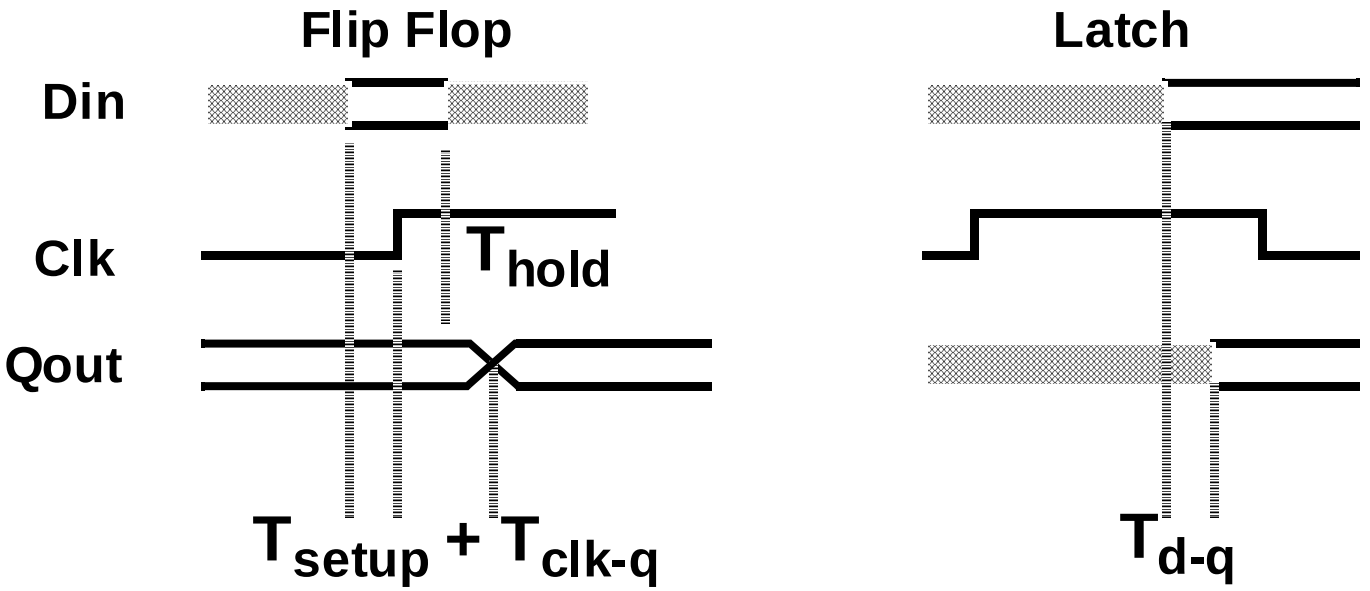
- Clocks serve to slow down signals that are too fast
- Flip-flops / latches act as barriers



  - With a latch, a signal waits until the clock is high
  - With a Flip-flop, the signal propagates on the rising edge
  - Note that all real flip-flops consist of two latch like elements (master and slave latch)

Clocking Overhead

- Problem: latches and flip-flops slow down slow signals too



  - Flip-flop delays the slowest signal by the setup + clk-q delay
  - Latches delay signals by the delay through the latch
  - So pipelining a FU increases its latency (adds flop delays)

Clocking / Sequential Element Design

- Trade off between overhead / robustness / complexity

  Constraints on the logic

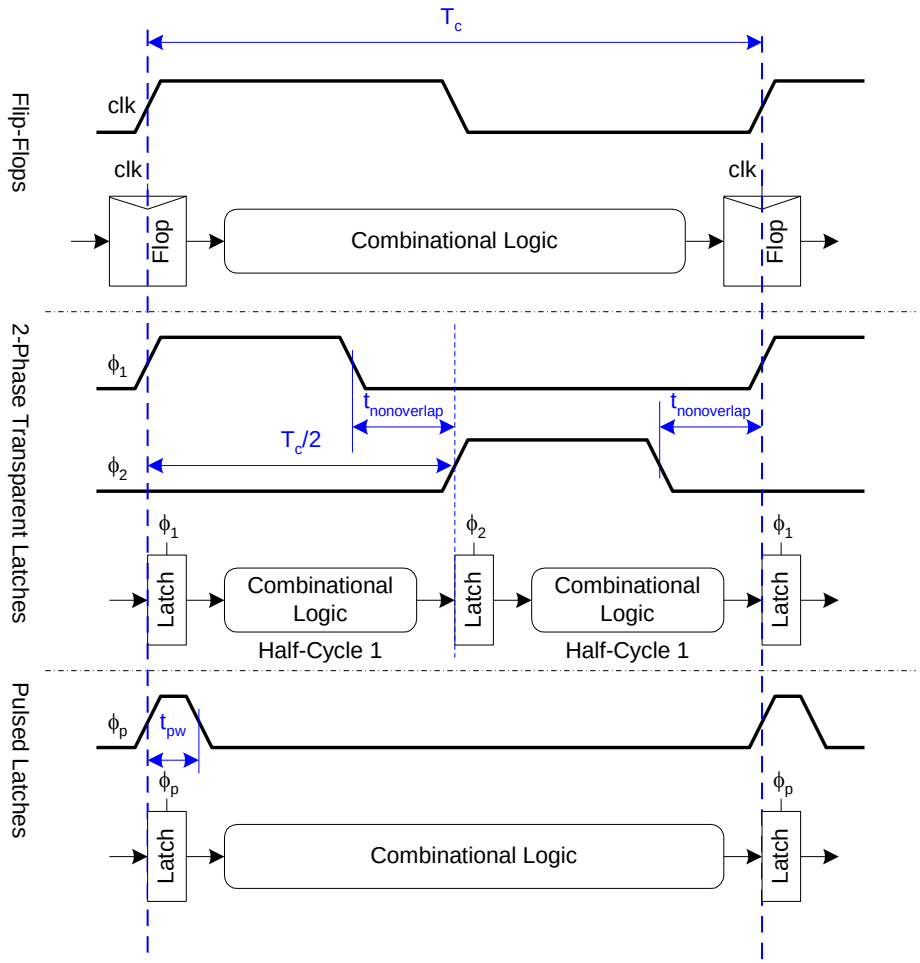
  vs.

  Constraints on the clocks

- Look at a few different clocking methods:
  - Edge triggered clocking (flip-flops)
    - What almost everyone does
  - Two phase clocking
    - Sometimes used to help in special cases
  - Pulse mode clocking
    - (aka Single phase clocking, pulsed latch clocking)
    - Used in high performance systems

Sequencing Methods

- Flip-flops
- 2-Phase Latches
- Pulsed Latches



Problems to Avoid

- Remember the point on clocks
  - Keep signal correlated in time
  - So there are two issues to consider:

- Max path
  - Ensure that the cycle time is slower
    - Than the slowest path in the logic
  - This can be always be fixed by running the clock slower

- Min path
  - Ensure that a fast signal does not race through a barrier
  - Can’t be fixed by changing cycle time!

Clock Skew: Just to Make Things More Difficult

- State elements are distributed all over the chip
  - Need to get the clock signal to all of them
  - Build fancy clock distribution circuits
    - Try to make all clock arrive at the same time
- Clock distribution circuits are not perfect
  - So there is skew in clocks

Makes the two problems worse

- The cycle time can get shorter by the skew
  - This is makes the long path problem harder

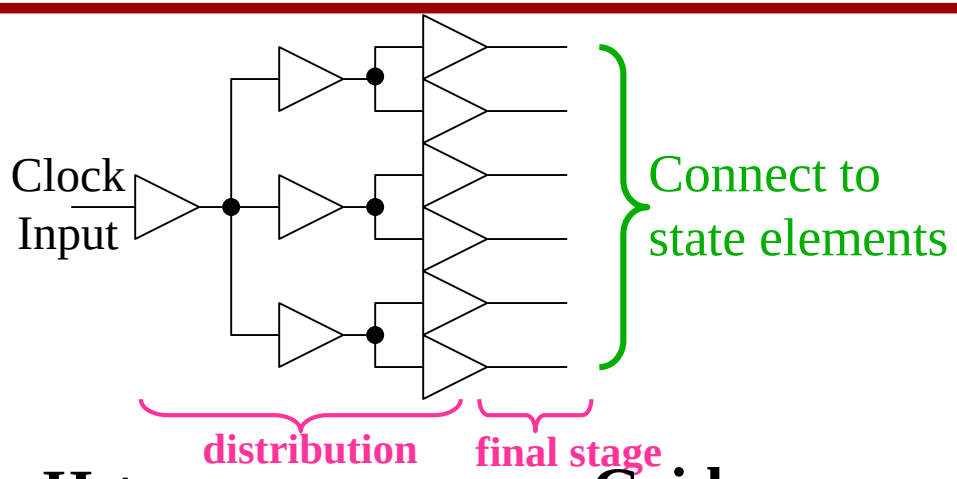
- The launching clock can be early (makes cycle time longer)
  - This makes the short path problem worse

Clocking Distribution

- Goal: Low skew
  - Deliver clock to state elements at exactly the same time
  - Any variation between 2 state elements is called skew
- Modern designs can have up to 200,000 or more state elements (I have seen designs with > 500,000 state elements)
  - Not including caches, State elements spread all over the chip
- Requires buffering to deliver clock to all state elements
  - Single large driver wouldn't be able to drive the load
- Requires complex wiring network to deliver clock
  - Need to route signals and place buffers such that every state element is clocked at exactly the same time

Clocking Distribution

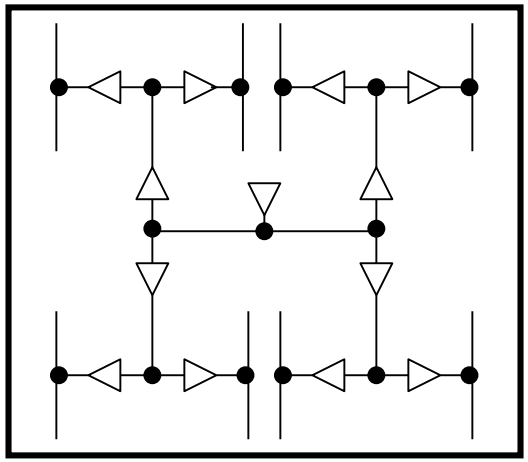
- Balance delay to all points
  - Build a tree
  - Often connect nodes at the same level in the tree



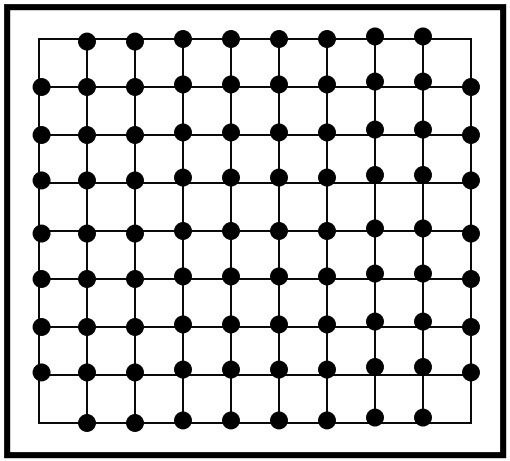
- Tree layout can be different
  - Distributed (H-tree)
  - In 1-D (Spine)

- Clock gating means you can't short the end nodes

H-tree



Grid



Thinking About the Worst-Case

- Need to guarantee that the circuit will work
  - Even in the worst-case situation
- But the delay of your cells can vary
  - With fabrication, data, operating conditions
- So you need to determine the worse case delay
  - Sometimes this will be the minimum delay
  - Sometimes this will be the maximum delay
  - So need to know both
- We will first ignore clock skew, but will add its effect at the end

Setup and Hold Times

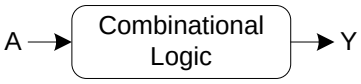
- Setup time
  - The latest a signal can arrive at the input to a flop or latch
  - Before the latching edge (rising for flop, falling for a latch)
  - And be guarantee to be captured by the element
  - Might capture data if it is later, but can't guarantee it will
  - Can be negative!

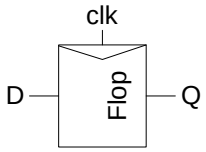
- Hold time
  - The earliest a signal can arrive at an input to flop or latch
  - After the latching edge and be guaranteed not to affect output
  - Setup - Hold time is always positive

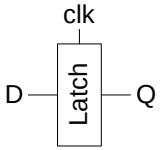
Timing Diagrams for Max and Min Delay

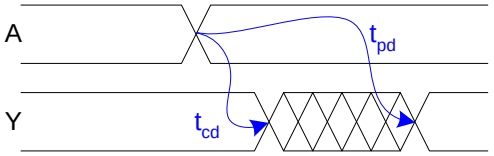
Contamination (min) / Propagation (max) Delays

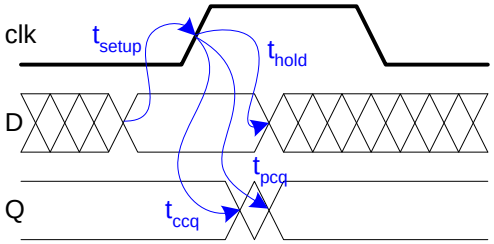
t_{pd}	Logic Prop. Delay
t_{cd}	Logic Cont. Delay
t_{pcq}	Latch/Flip-Flop Clk-Q Prop Delay
t_{ccq}	Latch/Flip-Flop Clk-Q Cont. Delay
t_{pdq}	Latch D-Q Prop Delay
t_{cdq}	Latch D-Q Cont. Delay
t_{setup}	Latch/Flip-Flop Setup Time
t_{hold}	Latch/Flip-Flop Hold Time

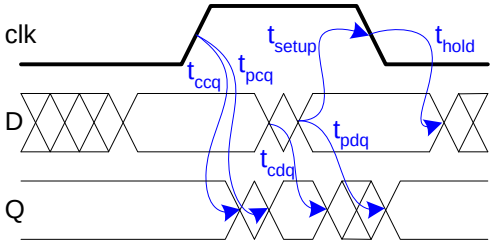




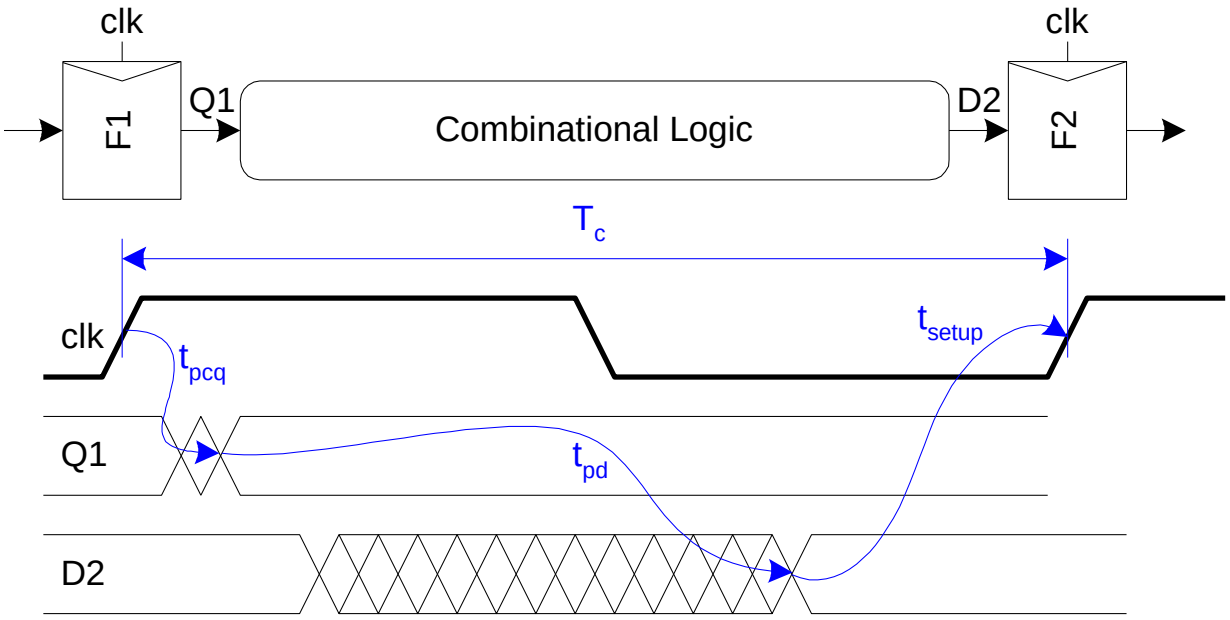






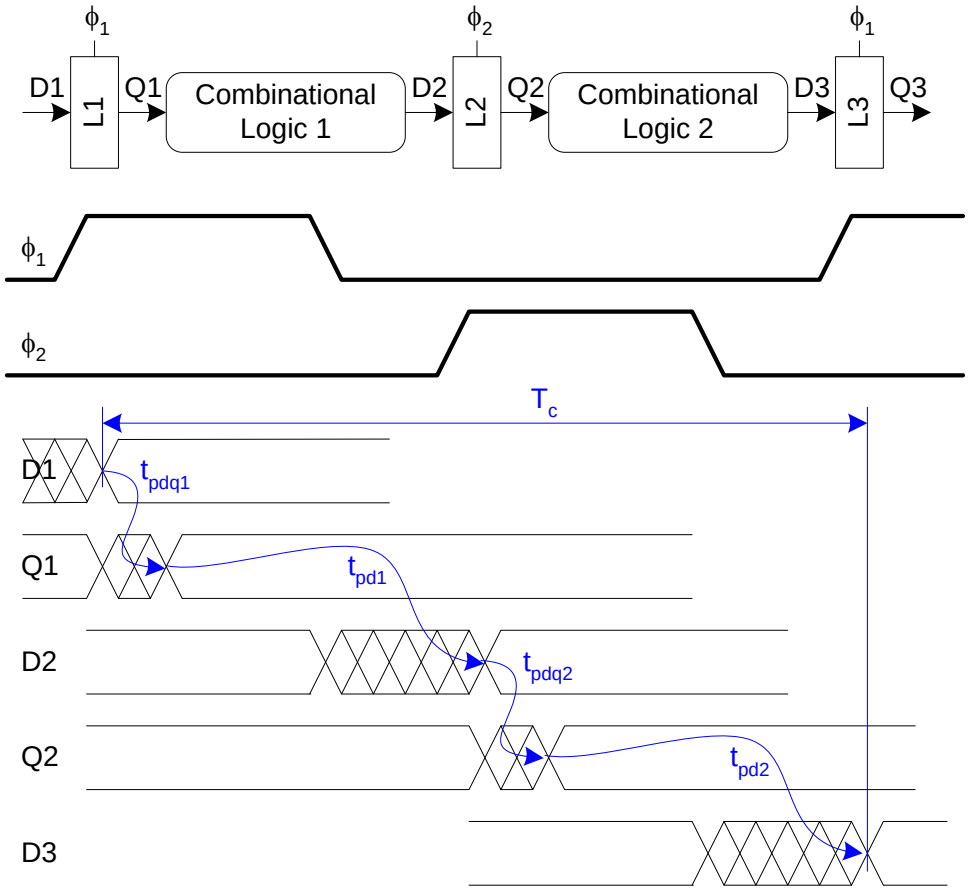


Max-Delay: Flip-Flops



$$t_{pd} \leq T_c - \underbrace{\left(t_{\text{setup}} + t_{pcq}\right)}_{\text{sequencing overhead}}$$

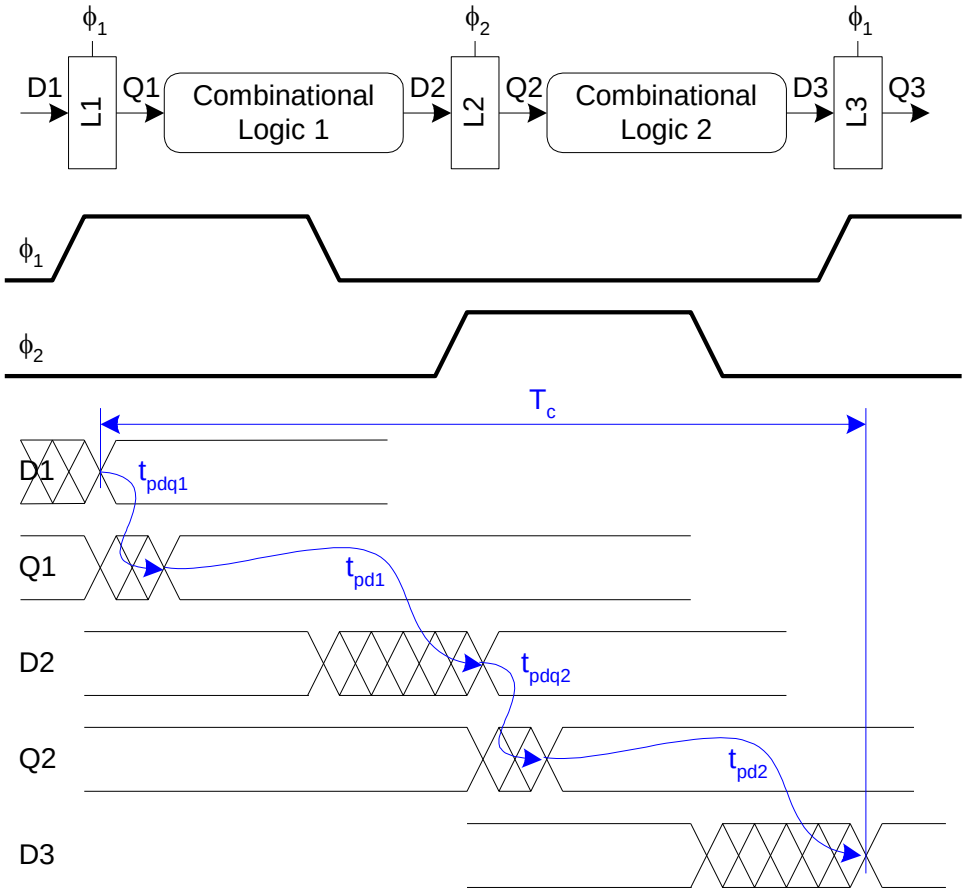
Max Delay: 2-Phase Latches



$$t_{pd} = t_{pd1} + t_{pd2} \leq T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}}$$

- Not quite right
  - True for cycle
  - Can borrow time
    - More later
- Want to balance delays

Max Delay: 2-Phase Latches



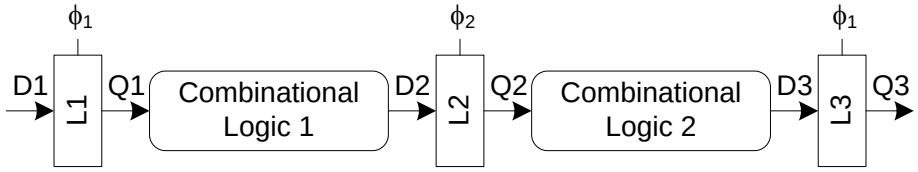
$$t_{pd} = t_{pd1} + t_{pd2} \leq T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}}$$

- Setup time is to falling edge
  - Provides flexibility

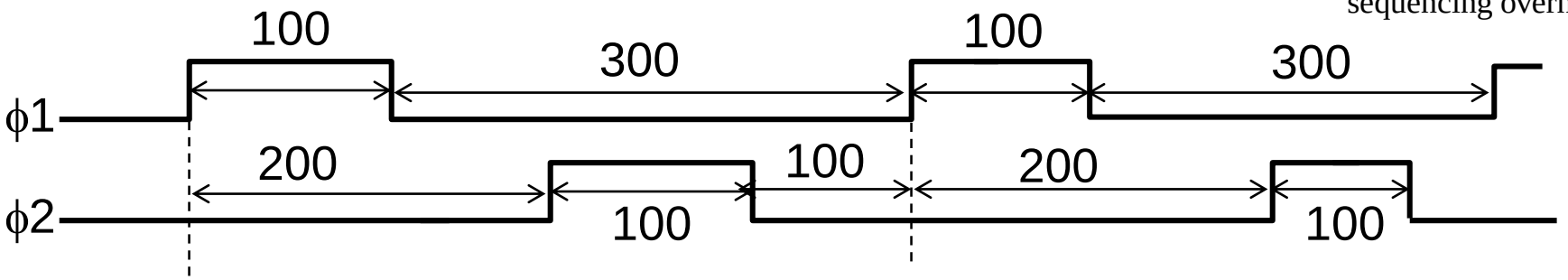
- $T_{pd1} <= T_c - t_{nonoverlap} - t_{setup} - t_{pcq}$
- $T_{pd2} <= T_c - t_{nonoverlap} - t_{setup} - t_{pcq}$

- Phi2 is usually not Phi1
  - Non overlap time is small
  - Remove CL2
    - Get flop system!

- Max Delay: 2-Phase Latches



$$t_{pd} = t_{pd1} + t_{pd2} \leq T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}}$$

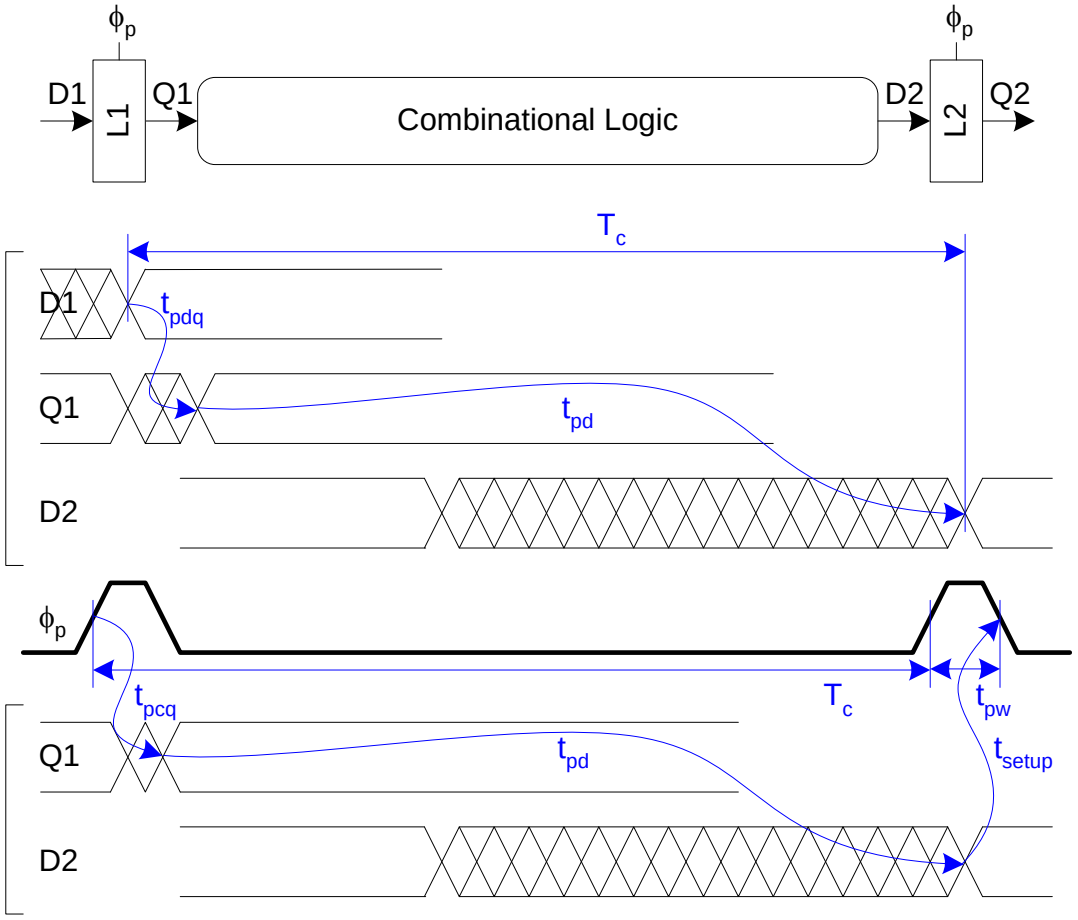


Suppose $t_{pdq} = t_{pcq} = 0$ time units. From above expression: $T_{pd} <= 400$

But $T_{pd1} < = 300$ time units, $T_{pd2} <= 300$ time units

Normally Phi2 is not Phi1 so the non-overlap time is small

Max Delay: Pulsed Latches



For a loop:

$$t_{pd} \leq T_c - \underbrace{\left(t_{pdq}\right)}_{\text{sequencing overhead}}$$

Not a loop:

$$t_{pd} \leq T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} - t_{pw}\right)}_{\text{sequencing overhead}}$$

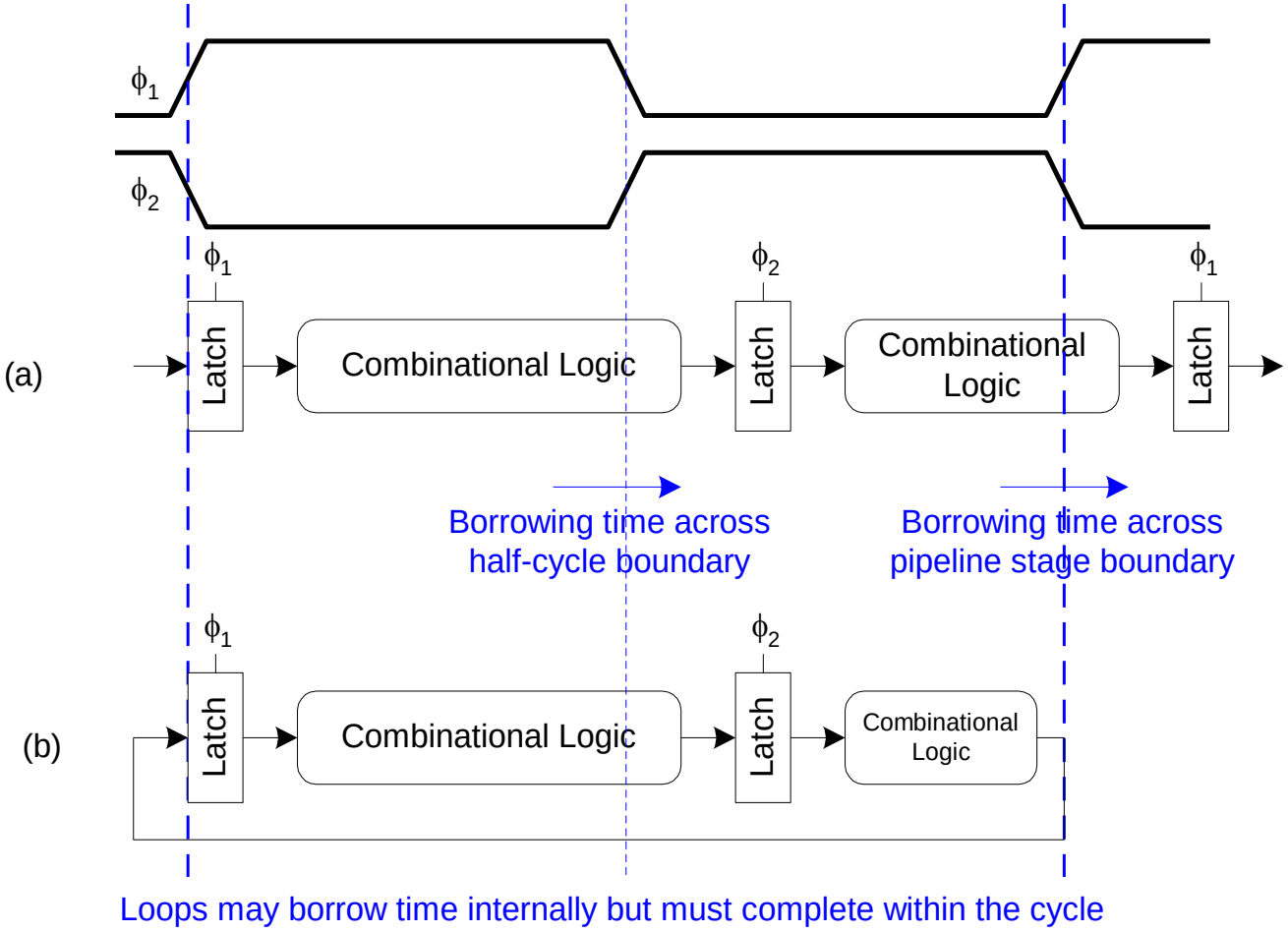
If this is larger than T_c you are borrowing time from another cycle

Time Borrowing

- In a flip-flop-based system:
  - Data launches on one rising edge
  - Must setup before next rising edge
  - If it arrives late, system fails
  - If it arrives early, time is wasted
  - Flip-flops have hard edges

- In a latch-based system
  - Data can pass through latch while transparent
  - Long cycle of logic can borrow time into next
  - As long as each loop completes in one cycle

Time Borrowing Example



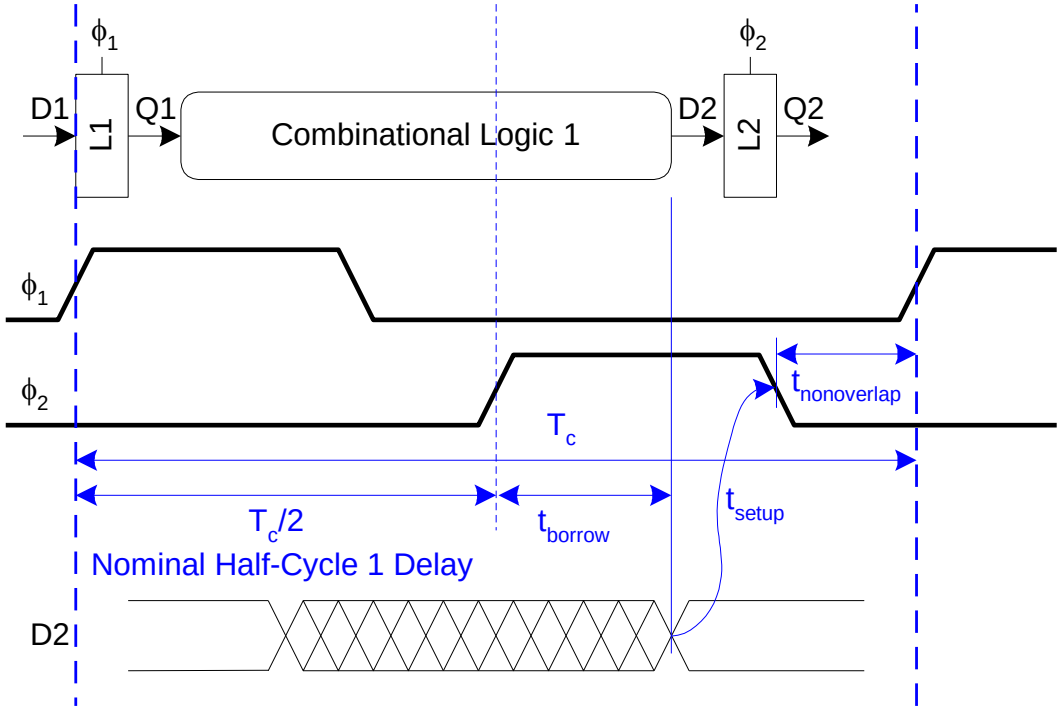
How Much Borrowing?

2-Phase Latches

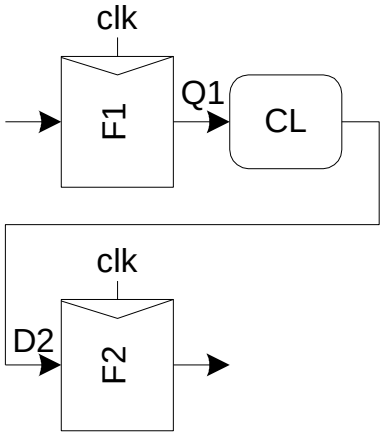
$$t_{\text{borrow}} \leq \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}} \right)$$

Pulsed Latches

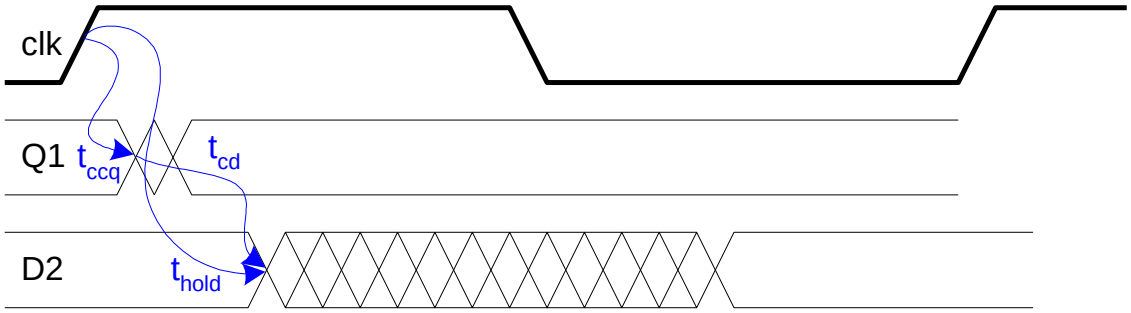
$$t_{\text{borrow}} \leq t_{pw} - t_{\text{setup}}$$



Min-Delay: Flip-Flops



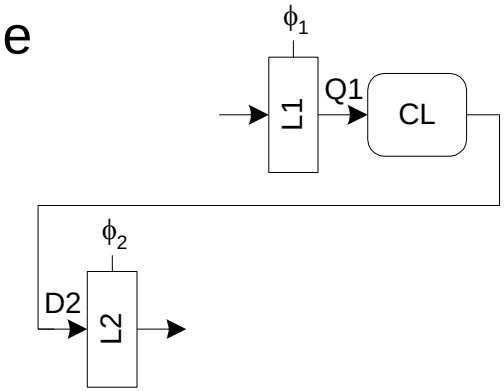
Hold time problems can have MAJOR impact on product time-to-market.

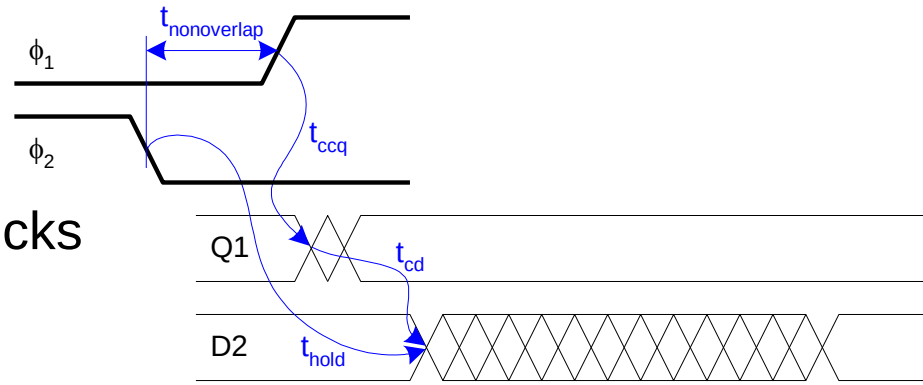


$$t_{cd} \geq t_{\text{hold}} - t_{ccq}$$

Min-Delay: 2-Phase Latches

- Now see reason for non-overlap time
  - Effectively reduces hold time
- Time constraint is from
  - Phi2 falling to Phi1 rising
- If you independently control both
  - Can fix any hold time issue!



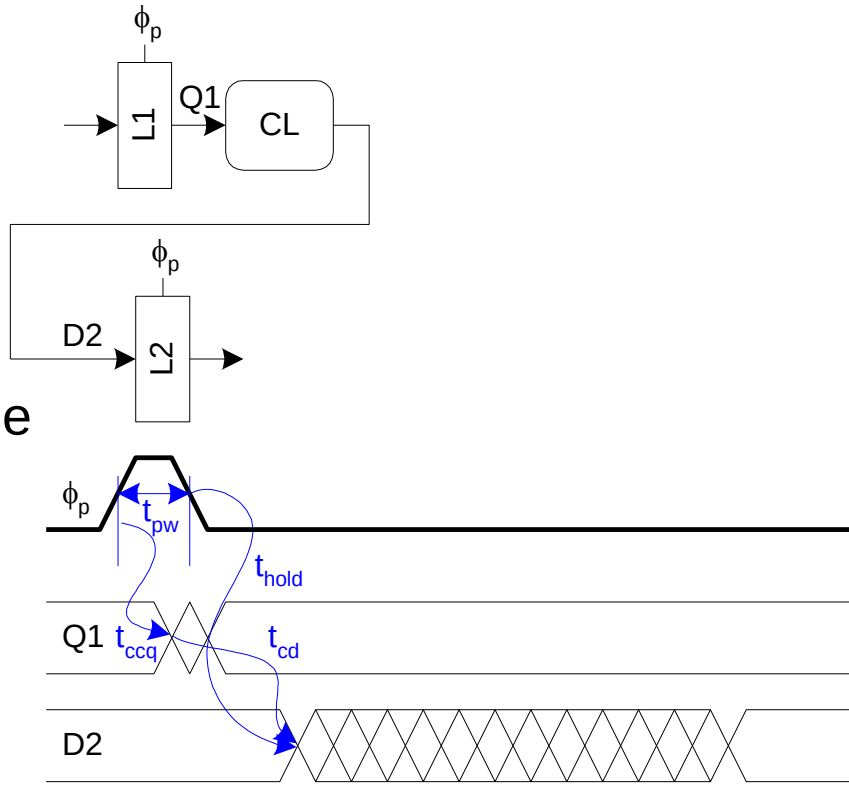


- Sounds great until
  - You remember you need two clocks
  - Just one clock is a huge pain
  - So generally use Phi and Phi_b
  - No independent control!

$$t_{cd1}, t_{cd2} \geq t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}}$$

Min-Delay: Pulsed Latches

- What is good for max delay
  - Transparency window
  - Hurts for min delay
- Hold time increased by pulse width
  - Signal is launched on rising edge
  - Latched by falling edge
- Pulse widths are generally short
  - Generated internal to latch



$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{pw}$$

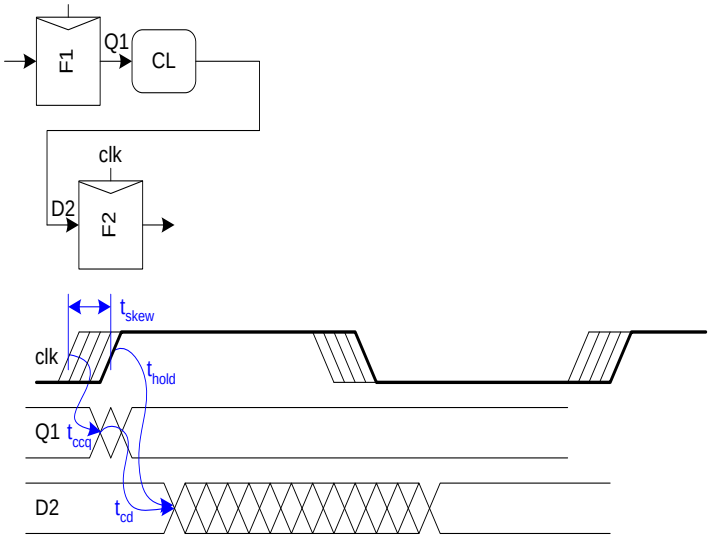
Skew Just Makes Both Constraints Worse

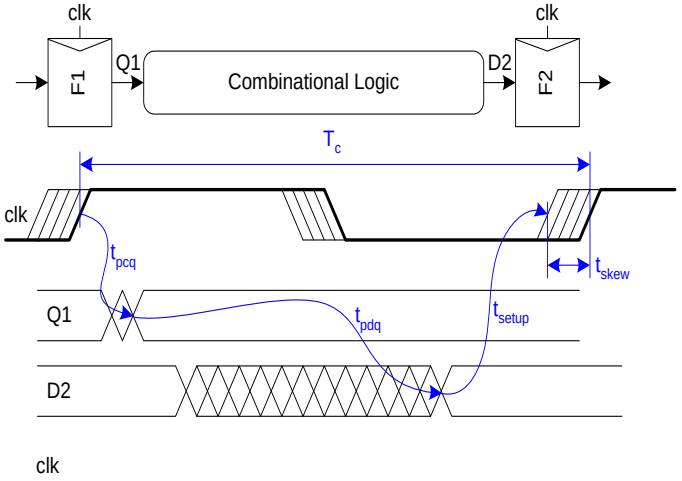
- Since we don't know which flop gets which clock
  - Need to make sure circuits works in all cases

- For long paths checks
  - Assume that the launching flop has a late clock
  - Assume latching flop gets an early clock
  - Cycle time is reduced by skew

- For short path checks
  - Assume the opposite, launching flop is early
  - And latching flop is late
  - And since you can't fix it by changing clock
    - Assume a more conservative (larger) value of skew in this case

Skew: Flip-Flops



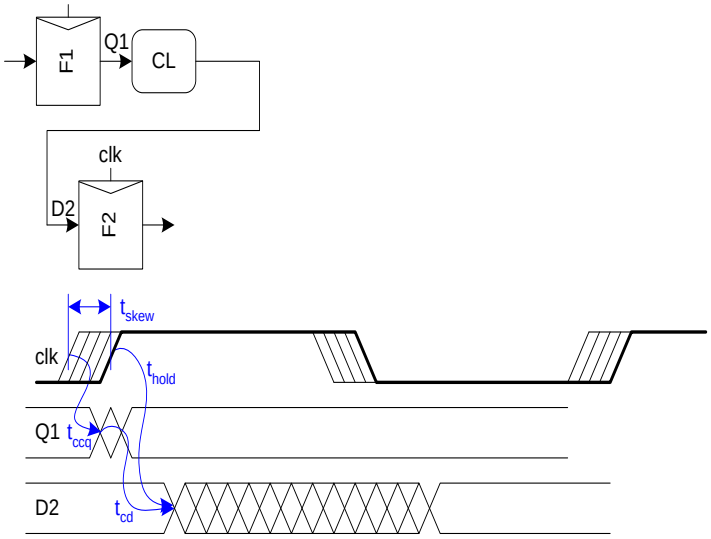


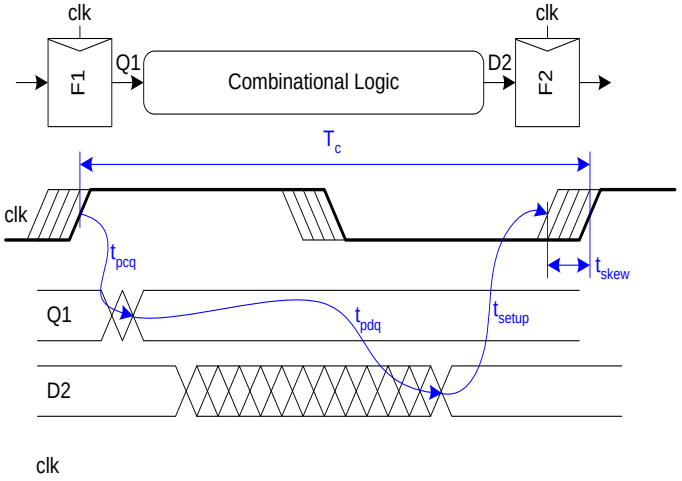
$$t_{pd} \leq T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} + t_{\text{skew}}\right)}_{\text{sequencing overhead}}$$

Worst case when clock arrives later at F1 and earlier at F2

$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{\text{skew}}$$

Skew: Flip-Flops





$$t_{pd} \le T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} + t_{\text{skew}}\right)}_{\text{sequencing overhead}}$$

$$t_{cd} \ge t_{\text{hold}} - t_{ccq} + t_{\text{skew}}$$

Worst case when clock arrives earlier at F1 and later at F2

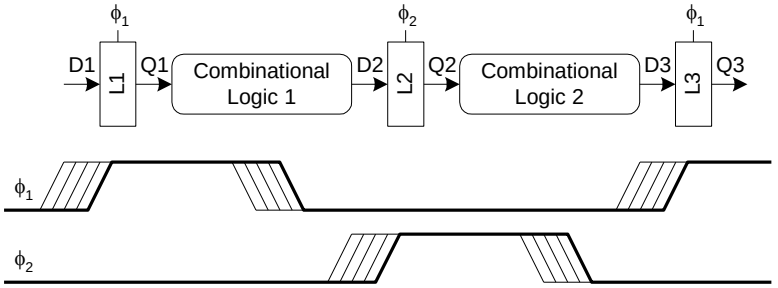
Skew: Latches

2-Phase Latches

$$t_{pd} \le T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}}$$

$$t_{cd1}, t_{cd2} \ge t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}} + t_{\text{skew}}$$

$$t_{\text{borrow}} \le \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}} + t_{\text{skew}}\right)$$



Pulsed Latches

$$t_{pd} \le T_c - \underbrace{\left(t_{pdq}\right)}_{\text{sequencing overhead}}$$

$$t_{cd} \ge t_{\text{hold}} + t_{pw} - t_{ccq} + t_{\text{skew}}$$

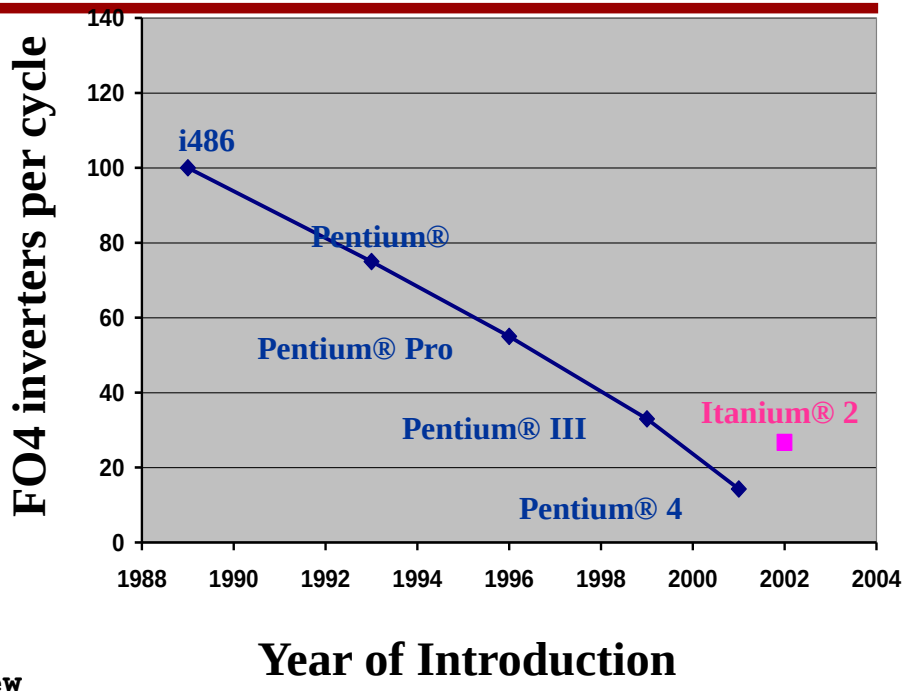
$$t_{\text{borrow}} \le t_{pw} - \left(t_{\text{setup}} + t_{\text{skew}}\right)$$

Skew Tolerance

- Both 2 phase and pulsed latch system are tolerant to skew
  - Small amounts of skew don't effect max path
- This is a result of these systems being able to borrow time
  - If latching clock is early borrow time from next cycle
  - It is longer than expected anyhow
- Skew tolerance is very important in short tick machines
  - Since skew is a larger percent of the cycle time
- Unfortunately hold time issues are harder in these systems
  - And if you blow the hold-time, the system will fail

Clocking Trends

- Pipelines were getting deeper
  - Fewer logic stages/clk
  - Enables faster frequency
    - Good for performance
    - Bad for power
- Signals have max & min issues
  - Difficult to design below 16 FO4
- Recently the trend has reversed
  - Core 2 is around 20+ FO4



Max: $T_d(max) < T_{cycle} - T_{su} - T_{skew}$

Min: $T_d(min) > T_{hold} + T_{skew}$

Pentium® 4 processor (gate delay):

$T_{cycle} = 14$

$T_{su} = 2$; $T_{hold} = 1$

$T_{skew} = 3$ (global signal)

→ $4 < T_d < 9$