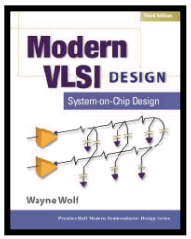
Slide Set 7

Latches and Clocking

Steve Wilton
Dept. of ECE
University of British Columbia
stevew@ece.ubc.ca

Overview

Reading: Wolf, Chapter 5



In this section, we will talk about charge storage, registers, and clocking schemes. We will talk about 2-phase clocking, one of the safest clocking methods around, and the one we will use in this class.

We'll also talk about less safe methods (edge-triggered design or latch design using clock and clock_b). These are less safe, but they are well understood.

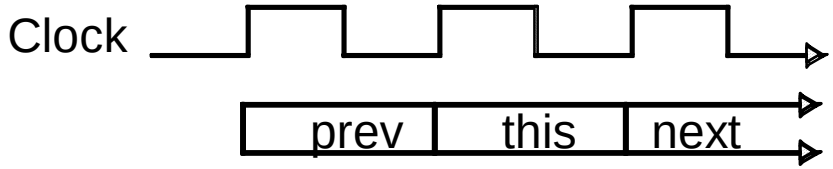
We will start by talking about charge storage which is fundamental to the operation of some of these latches.

Why have Clocks?

The whole reason that we need clocks is that we want the outputs to depend on more than just the inputs -- we want them to depend on previous outputs too. These previous outputs are encoded in the state bits of a state machine, and are the signals that cause lots of problems.

The state bits cause problems because we now need some sort of policy to define what previous and next mean. This is almost always done with the help of a clock, to provide reference points in time.

Build state machine from combinational logic (next state logic) and latches, FF (storage elements) to store the state information



Control latches and FF with a clock; **unfortunately we can have fast/slow signals (delay) and fast/slow clocks (skew)**

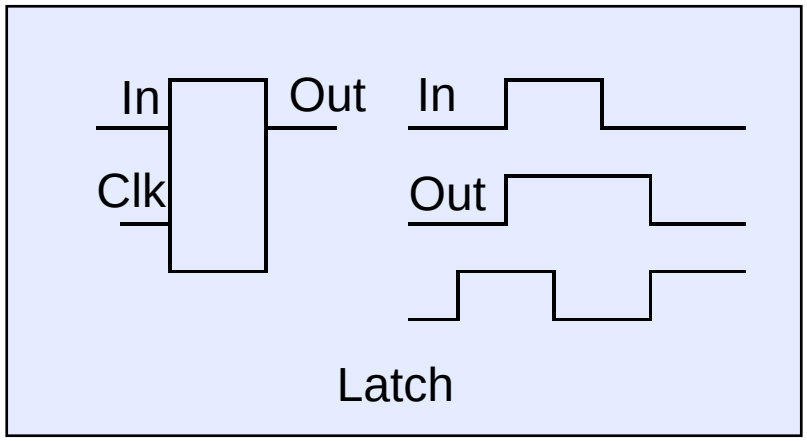
Latch vs. Flip-Flop

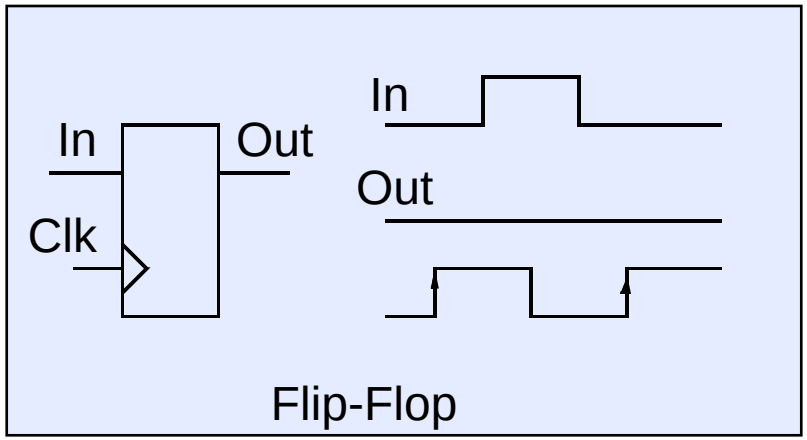
Latch (transparent)

- When the clock is high it passes In value to Output
- When the clock is low, it holds value In had when clock fell

Flip-Flop (non transparent)

- On the rising edge of clock, it transfers the value of In to Out
- It holds the value at all other times.





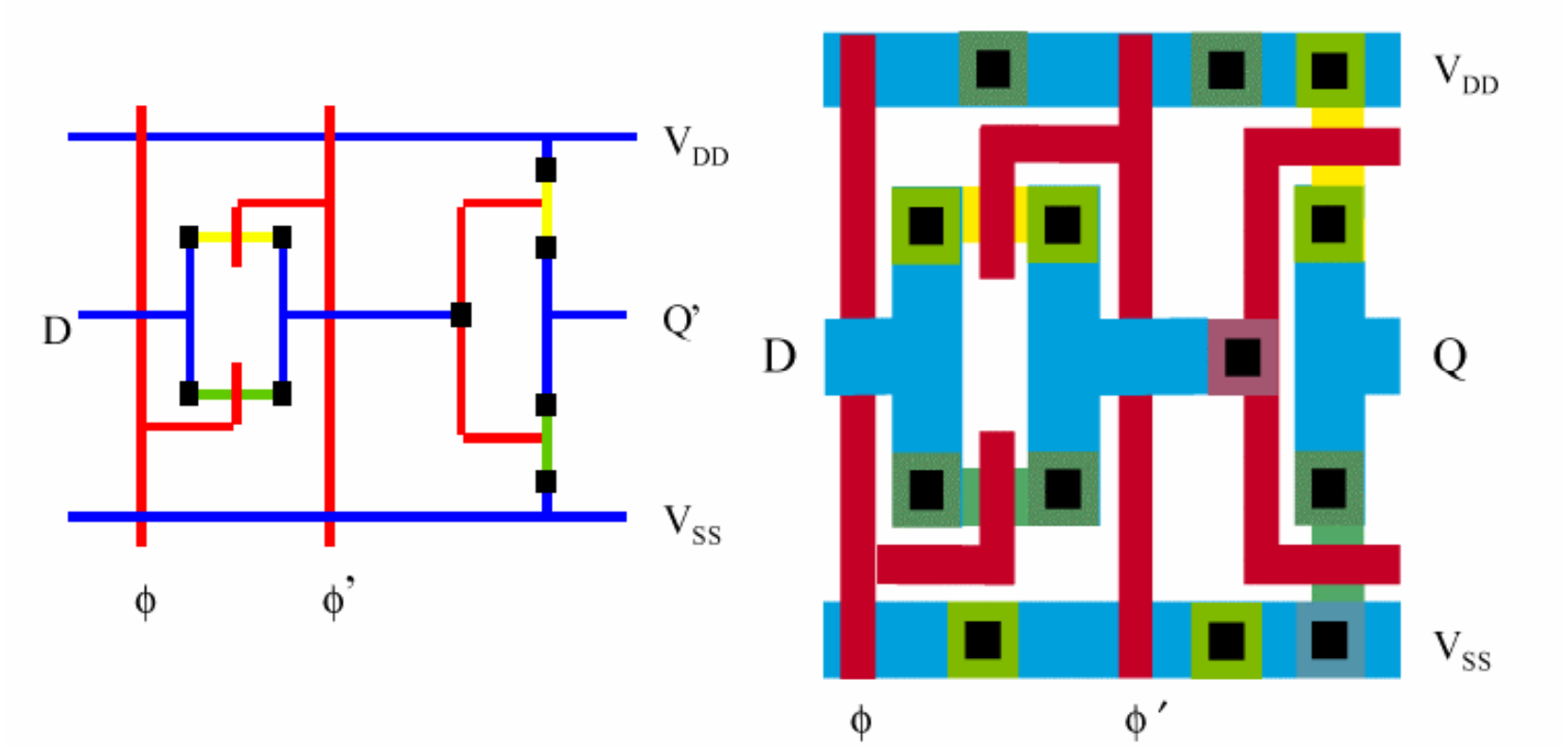

```
always@(Clk In)
   if( Clk ) Out = In;
```

```
always@(posedge Clk)
      Out = In;
```

Board Notes: - Dynamic Storage

- Latches

Layout of a dynamic latch:



source: David Bayer, OSU

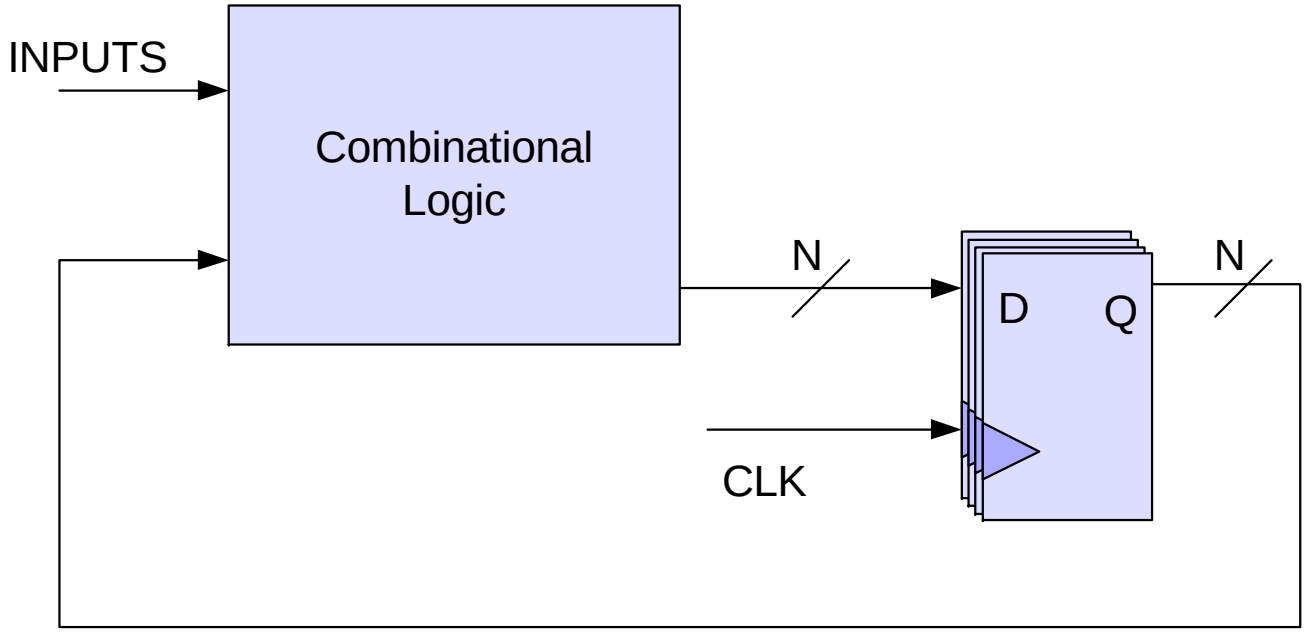
How do we use these flip-flop/latches in a system?

We will see two methods:

1. Edge-Triggered Clocking
   - Use the single clock flip-flop we just saw
   - Easy, and most industrial designs use this
   - But, there are a few things you have to watch out for…

2. Two-Phase Clocking
   - Use the two-clock latch we just saw
   - More flexible timing, but a but tricky….

Edge-Triggered Systems

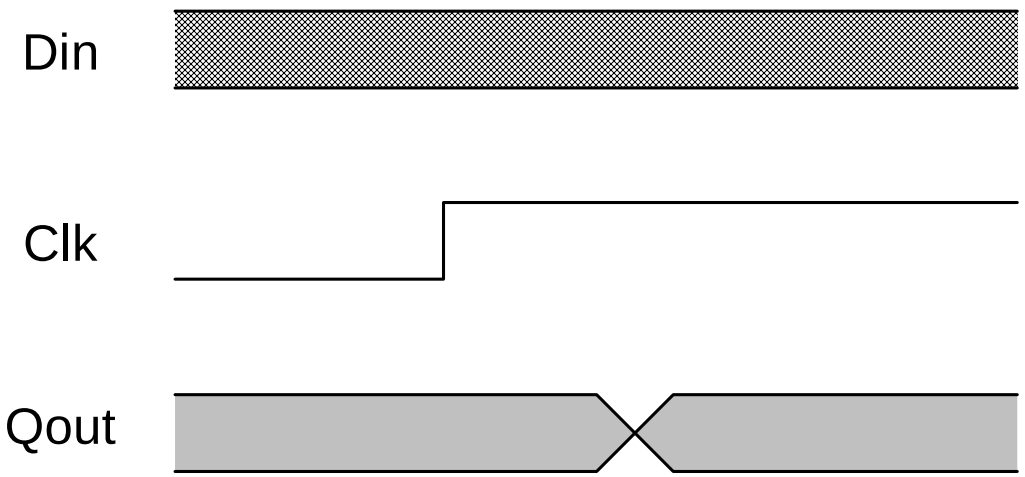


State is stored in the edge-triggered flip-flops

Let's talk a bit about the timing of this type of circuit…

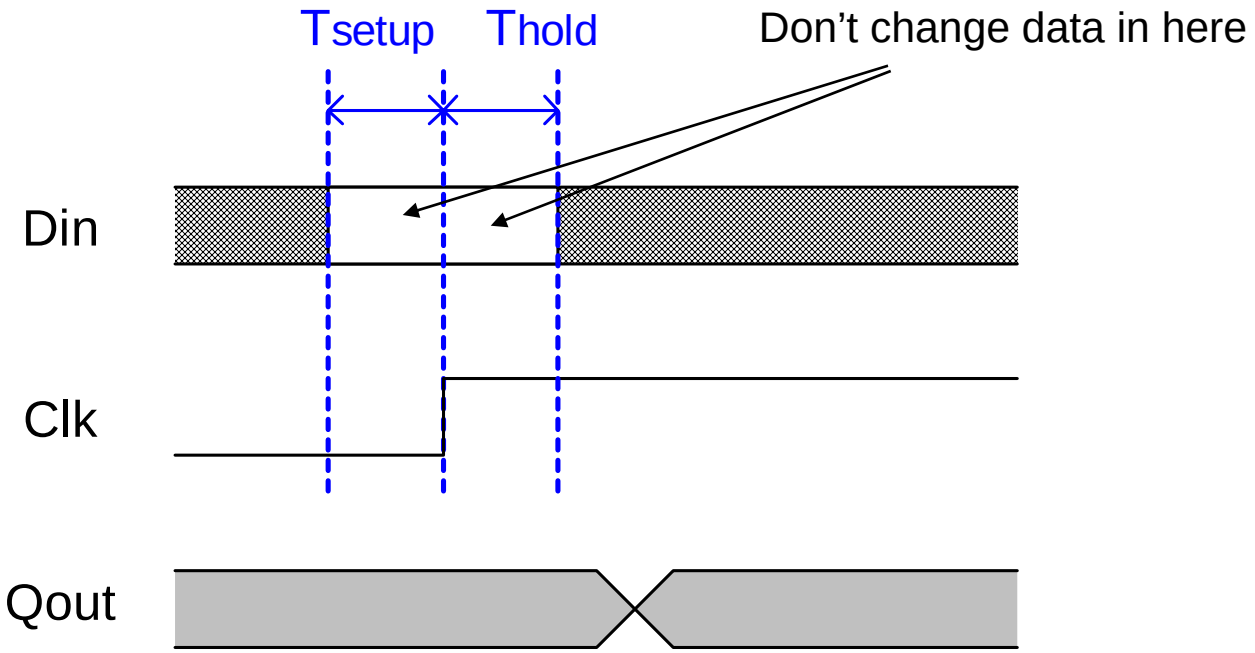
Clocking Overhead

Flip-Flops have setup and hold times that must be satisfied:



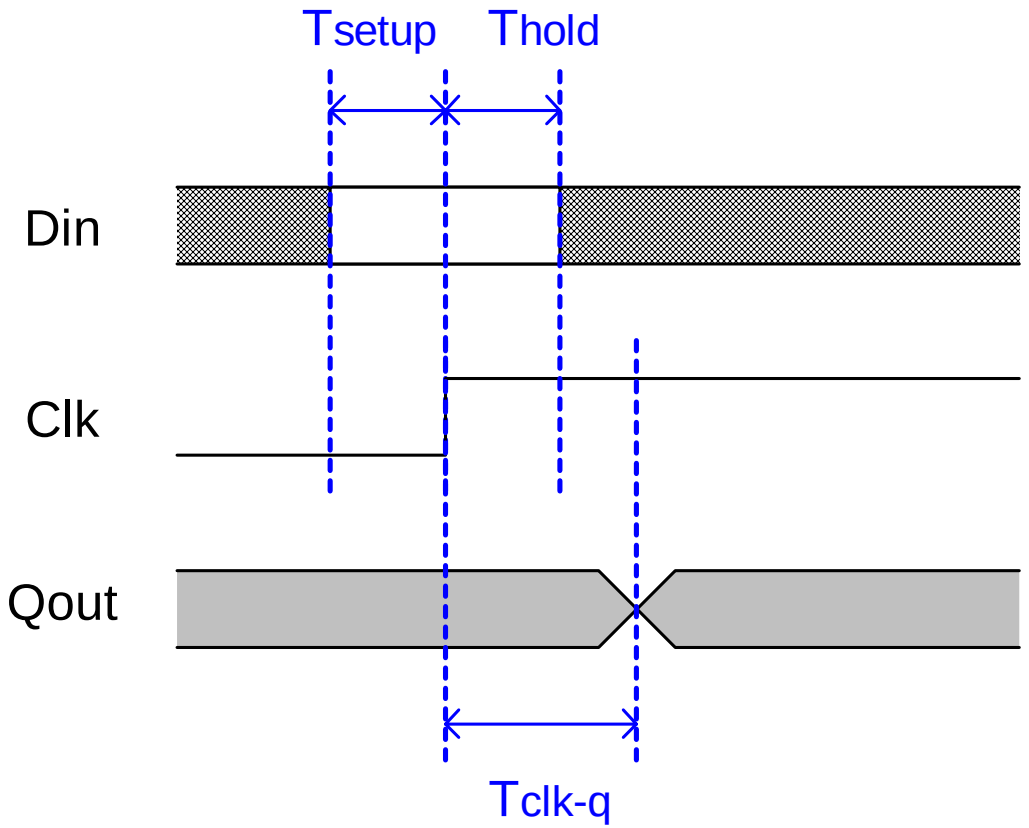
Clocking Overhead

Flip-Flops have setup and hold times that must be satisfied:



Clocking Overhead

Flip-Flops have setup and hold times that must be satisfied:



Timing Constraints

So, for correct operation…

1. The clock period must be long enough for the flip-flop outputs (current state) to feed-back through the combinational logic, and arrive back at the inputs of the flip-flops

Time to do this: Tclk-q + Td + Tsetup

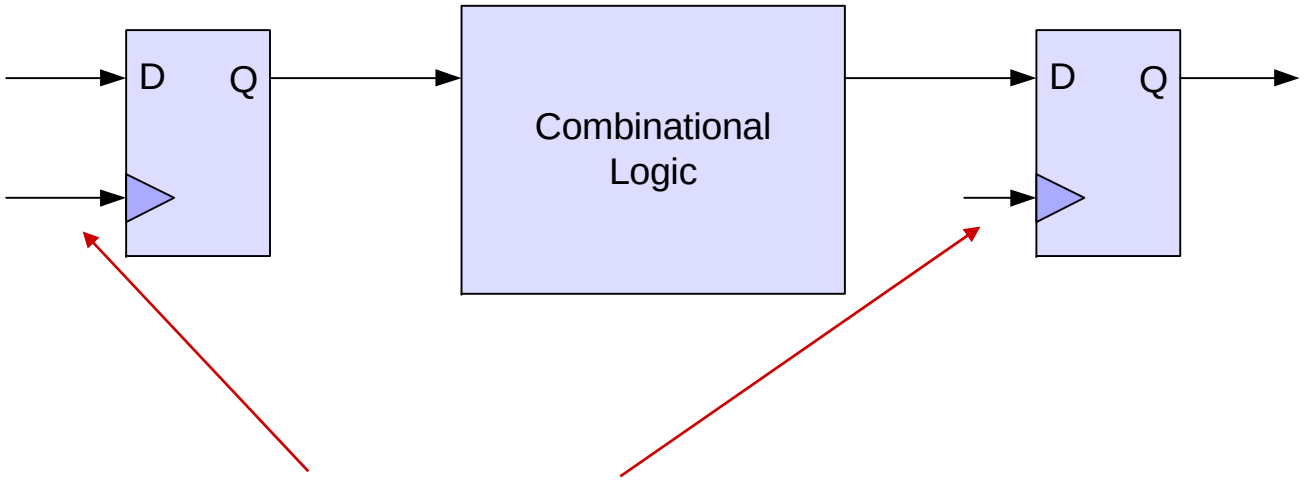
setup time of destination flip-flop

clock to output delay of source flip-flop

delay of combinational logic block

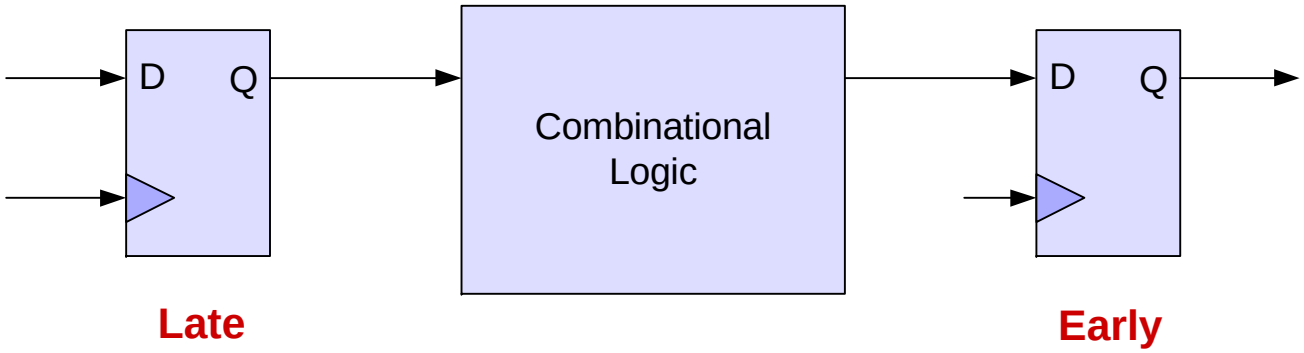
For correct operation: Tclk > Tclk-q + Td + Tsetup

Clock Skew



Even though these are the same clock there may be skew. Skew is the difference in arrival times of the clock signals. Skew is due to unequal routing delays in the clock distribution network

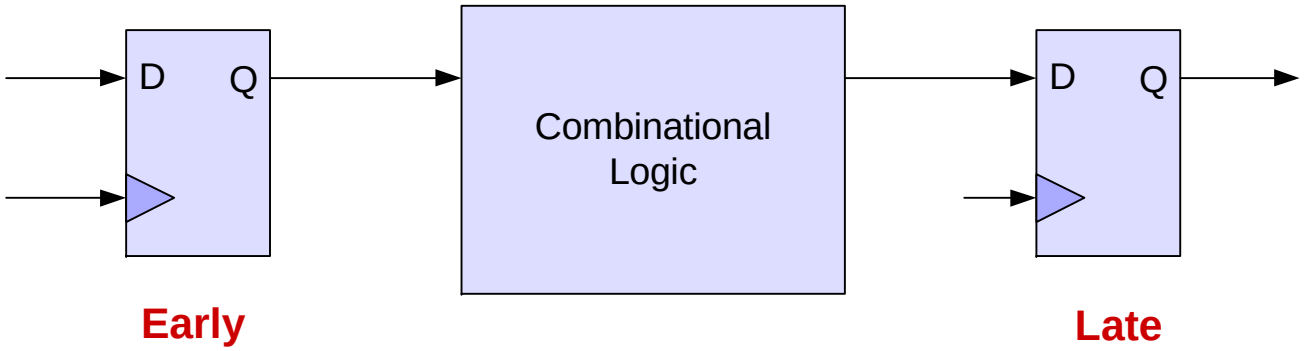
Clock Skew



Increase clock period to account for worst-case skew:

For correct operation: Tclk > Tclk-q + Td + Tsetup + Tskew

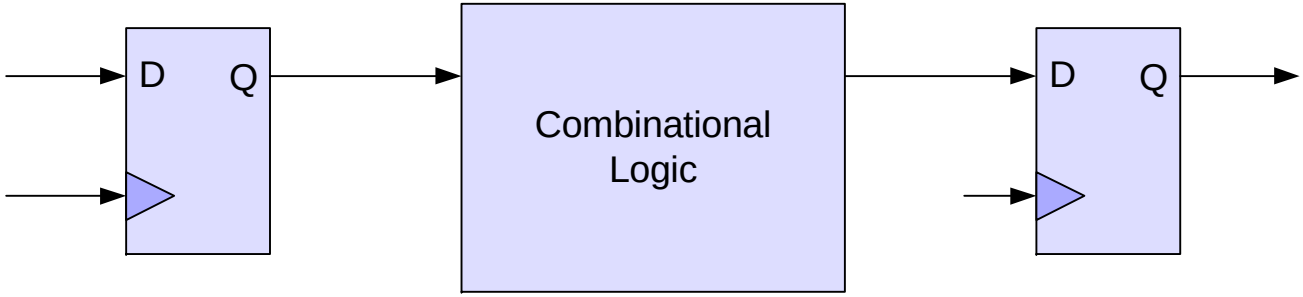
Clock Skew



The part will fail if Tclk-q + Td < Tskew + Thold

Note: if the circuit fails, we can't just adjust the clock speed until it works!

Clock Skew



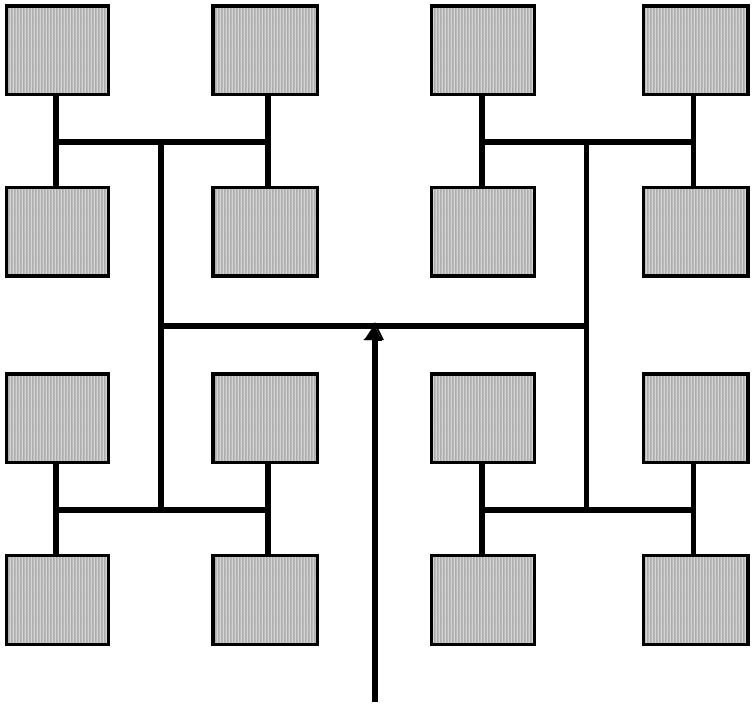
So there is a two-sided timing constraint for correct operation:

$$T_{clk} > T_{clk\text{-}q} + T_d + T_{setup} + T_{skew}$$

$$T_{clk\text{-}q} + T_d > T_{skew} + T_{hold}$$

Skew is a very real problem. Clock distribution networks are important, and we will talk about them soon.

Clock Distribution



H-Tree Network

Observe: Only Relative Skew is Important

Example: DEC Alpha 21164

Clock Frequency: 300 MHz - 9.3 Million Transistors

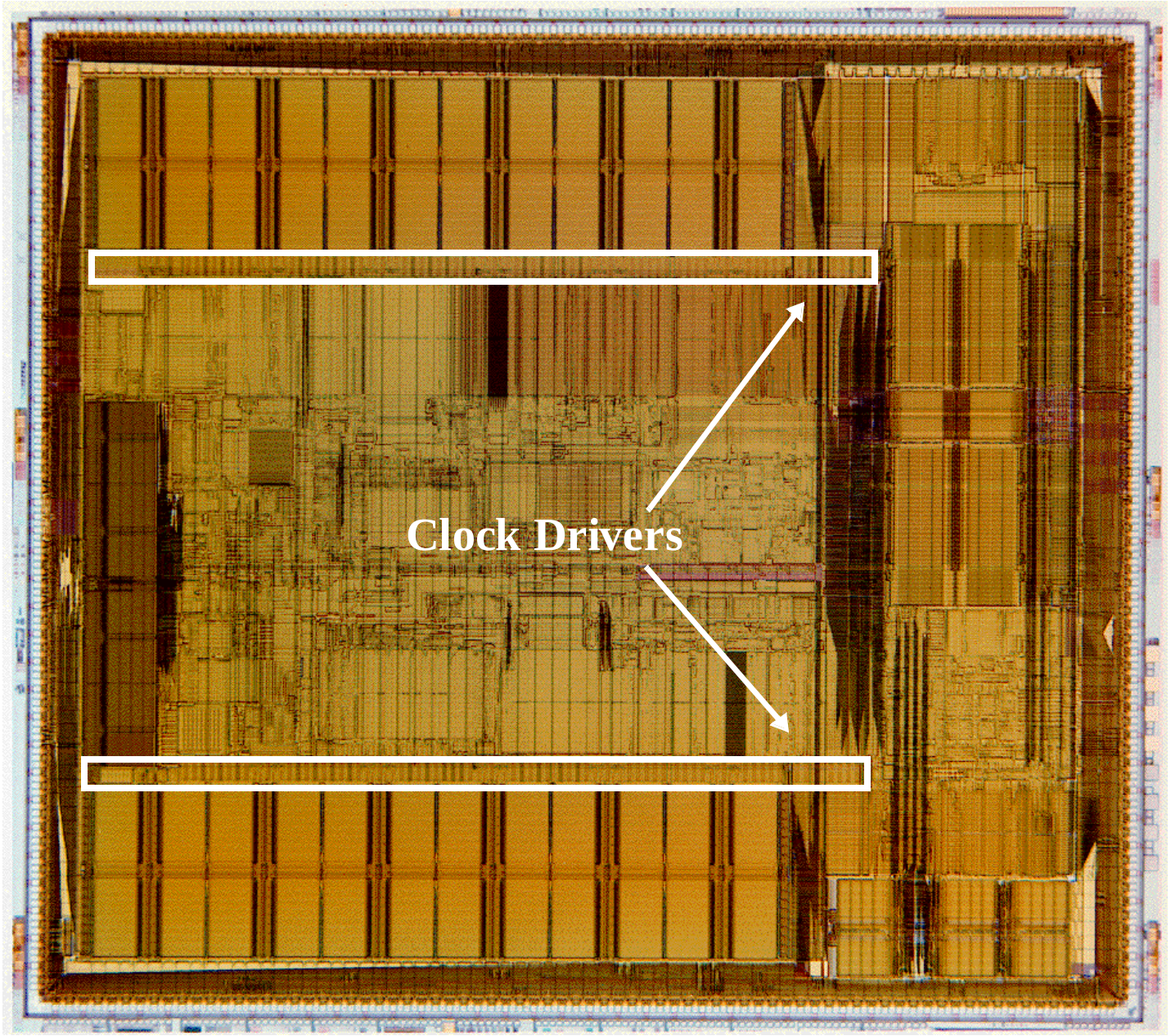
Total Clock Load: 3.75 nF

Power in Clock Distribution network : 20 W (out of 50)

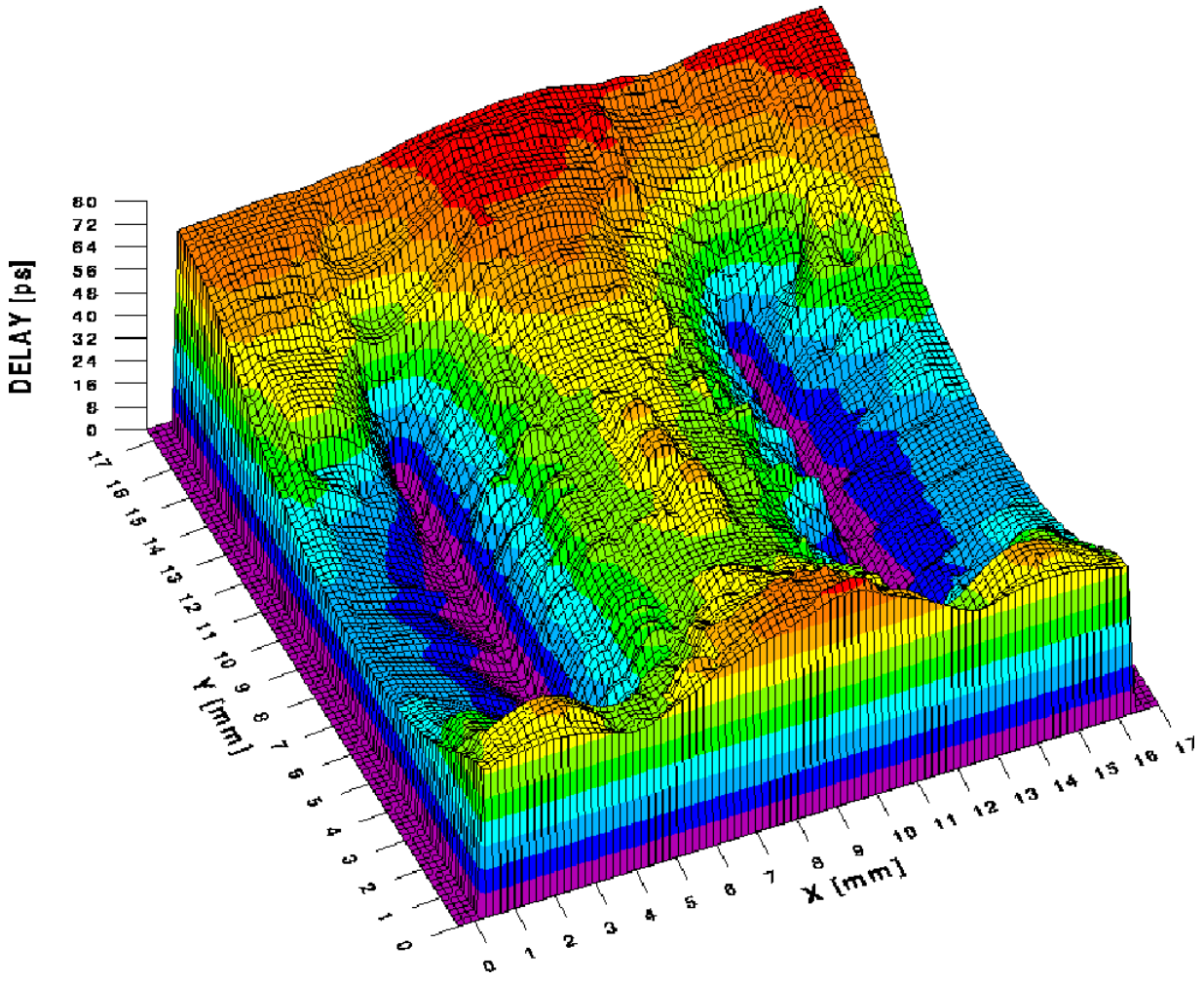
Uses Two Level Clock Distribution:

- **Single 6-stage driver at center of chip**
- **Secondary buffers drive left and right side clock grid in Metal3 and Metal4**

Total driver size: 58 cm!

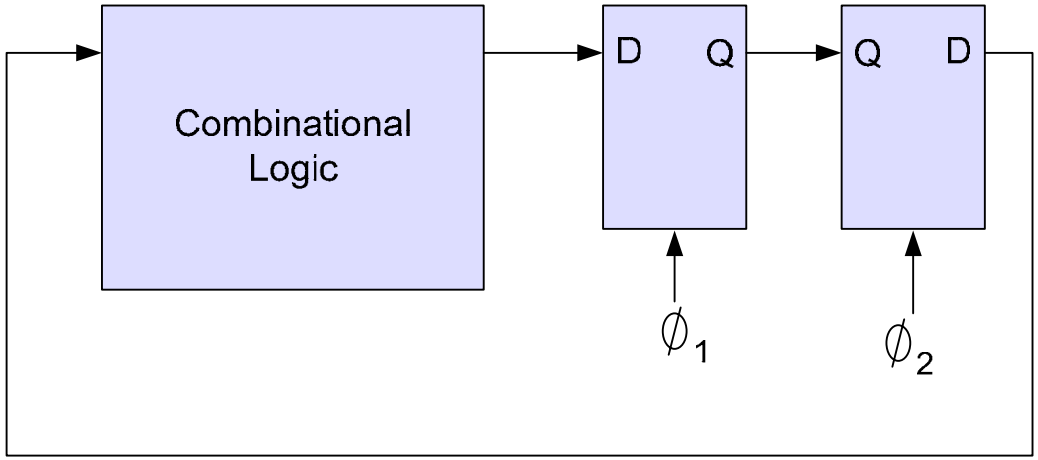


Clock Skew in Alpha Processor



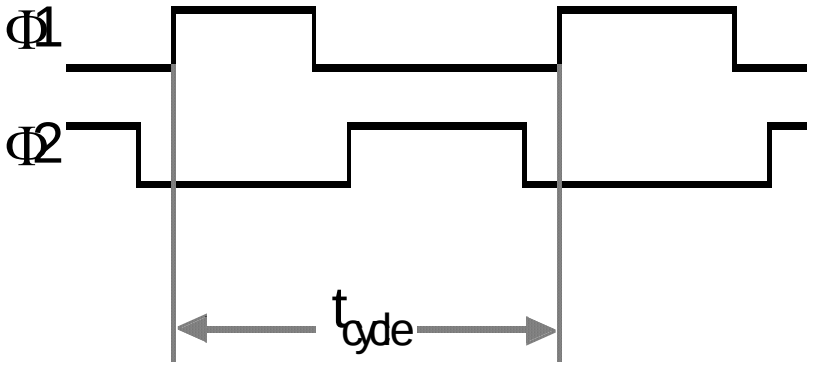
Two-Phase Clocking

Separate flip-flop into two latches:



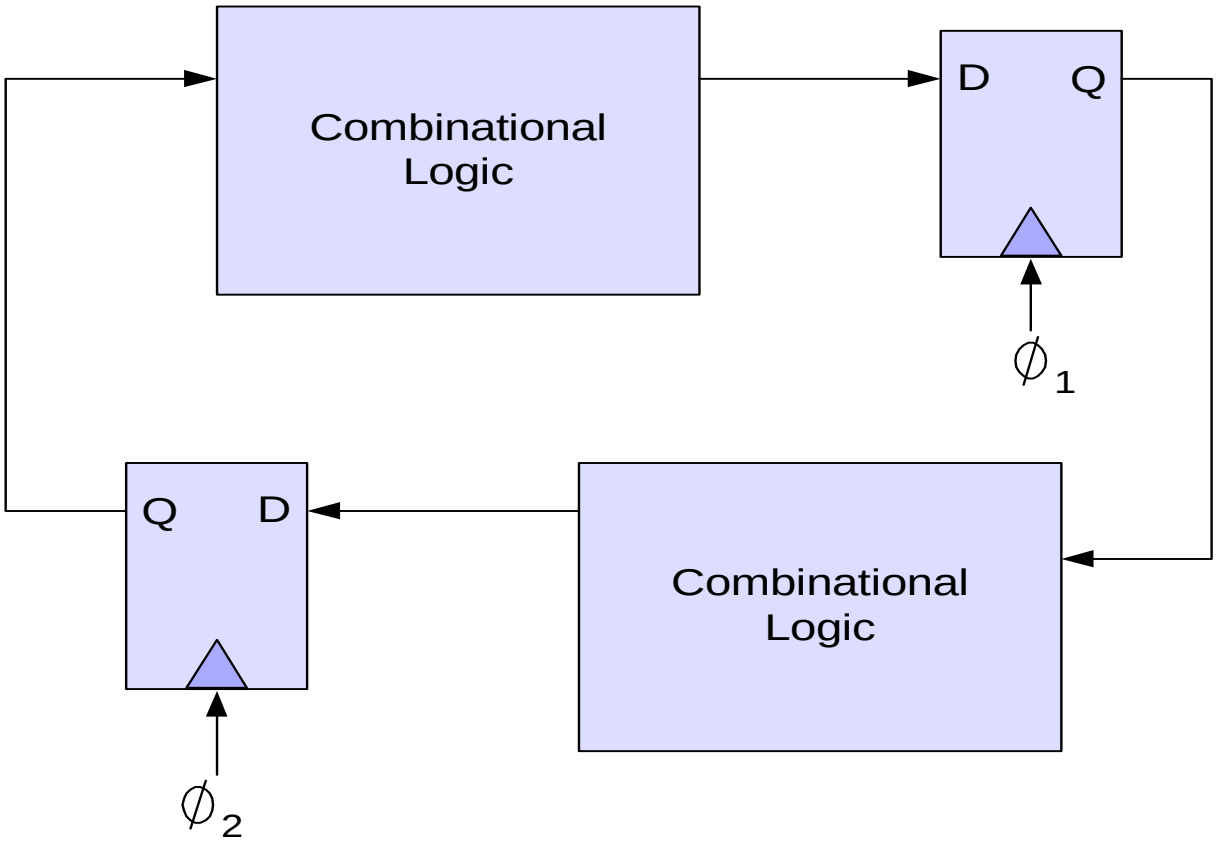
What's new here?

1. Use latches, not FF's
2. Use two non-overlapping clocks



Two-Phase Clocking

Can break up logic into two pieces::

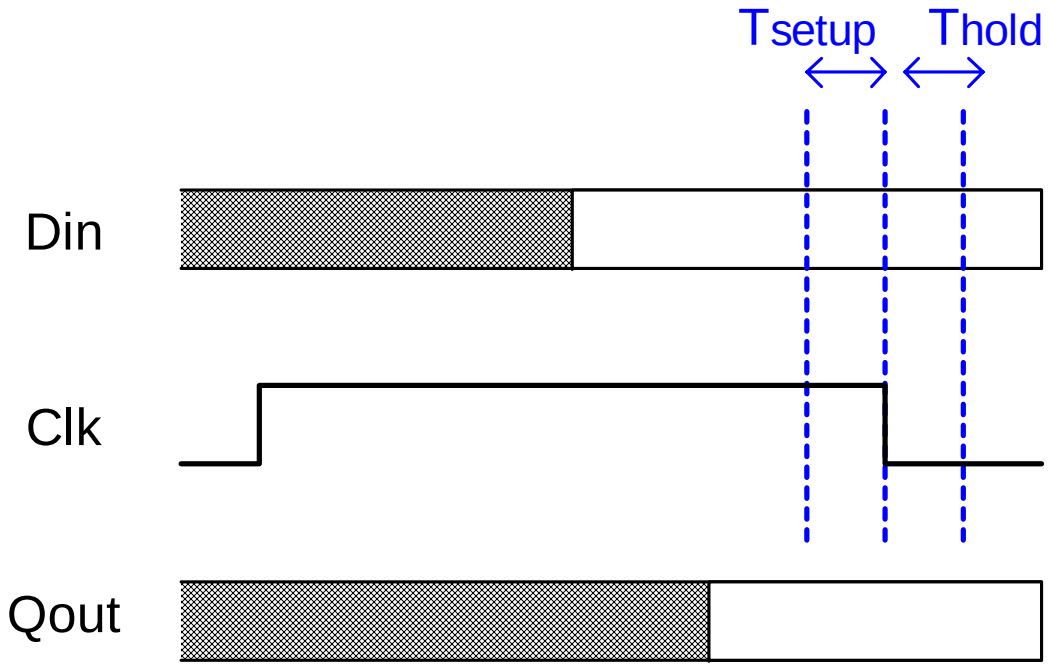


Clocking Overhead

Latches also have setup and hold times that must be satisfied:

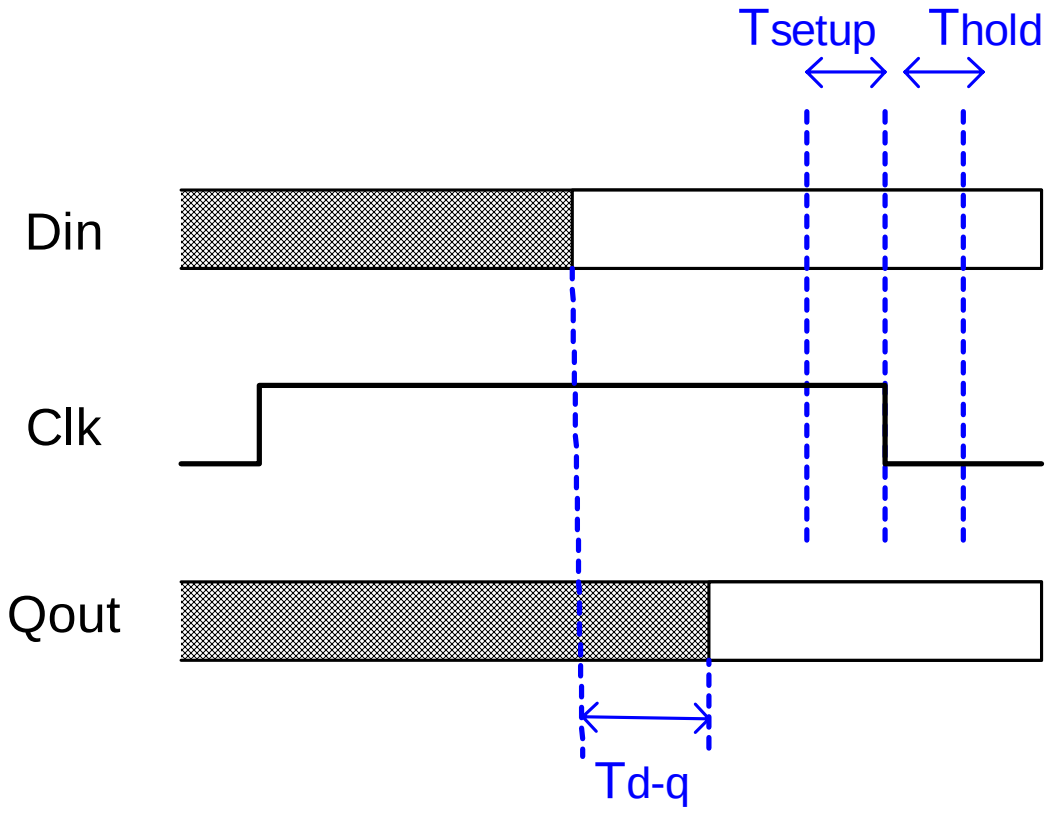
Latches (Clocking Overhead)

Latches also have setup and hold times that must be satisfied:



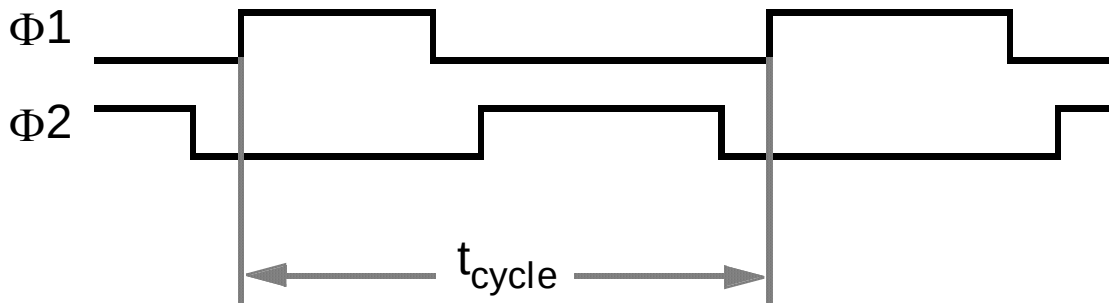
Latches (Clocking Overhead)

Latches also have setup and hold times that must be satisfied:



Two-Phase Clocking

Use different edges for latching the data and changing the output



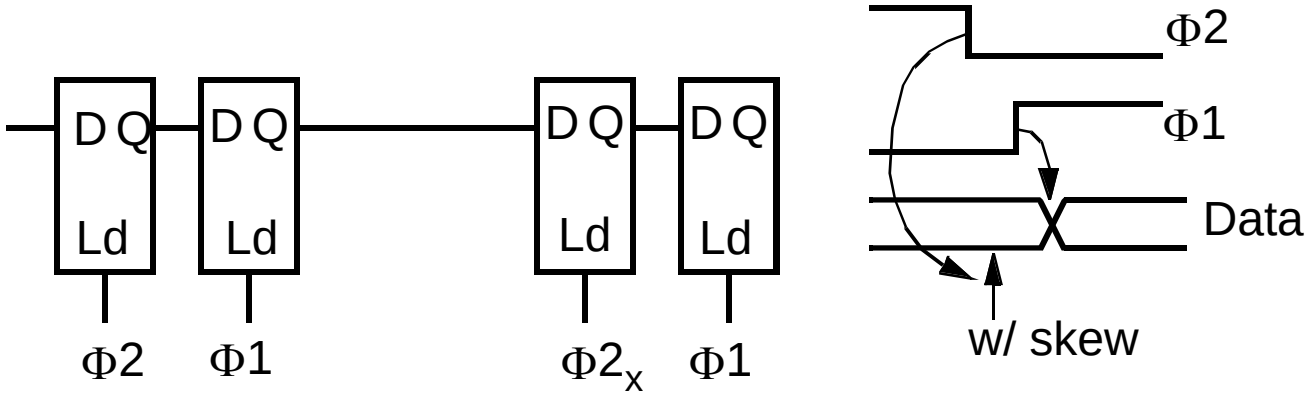
There are 4 different time periods, all under user control:

- F1 high
- F1 falling to F2 rising
- F2 high
- F2 falling to F1 rising

Can be small

Two-Phase Clocking

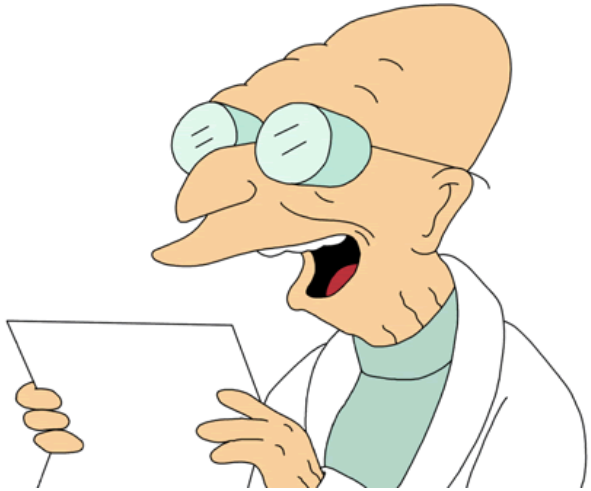
Look at shift register :



- If there is a large skew on the Φ2x clock, then the spacing between Φ1 and Φ2 can be increased to make sure that even with the skew, the Φ2 latch closes before the Φ1 latch lets the new data pass.
- New problem: should I use a Φ1 or a Φ2 latch at the output of my combinational block? How can I make sure that I don't make a mistake because I have two different types of latches?

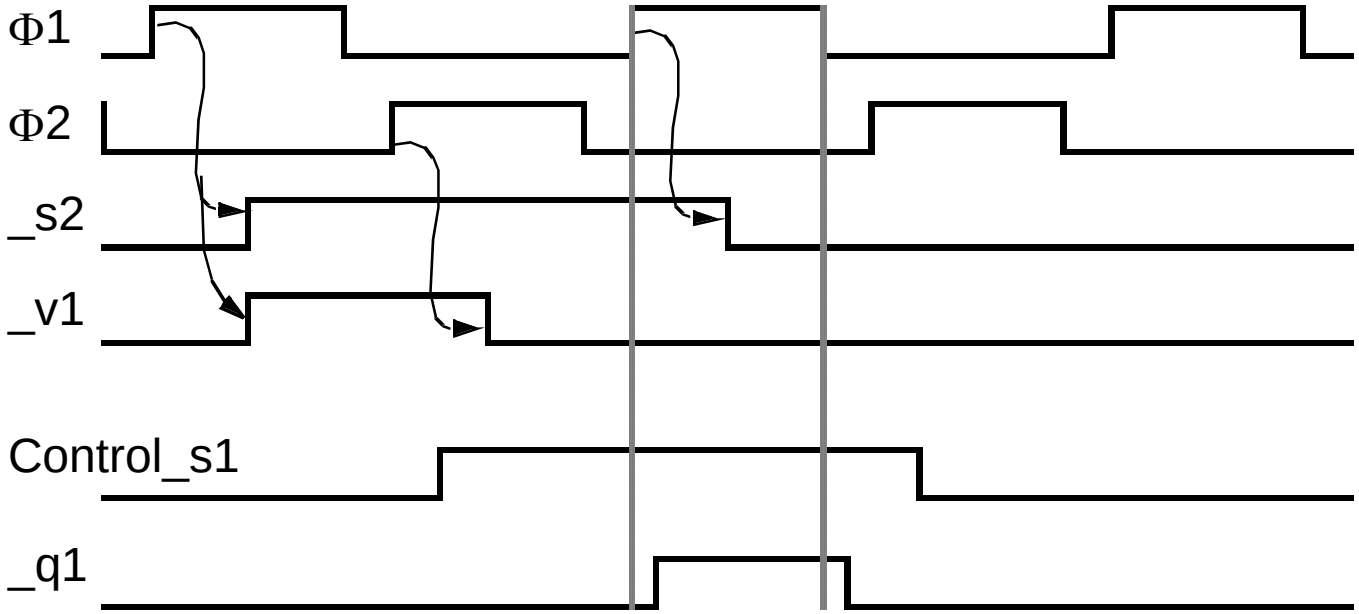
Board Notes: - A Naming Convention for Signals:

_s, _v, _q



Clocking Types: Summary

The following summarizes the clocking types we have discussed:



Verilog Coding Rules

If you decide to use 2-phase clocking, be sure to label every signal with one of _s1, _s2, _v1, _v2, _q1, _q2

- For weird timing-types, use _w

Standard Verilog Latch:

```
always @(Phi1 or Data_s1)
        if (Phi1)  Q_s2 = Data_s1
```

Remember:

- Combinational logic does not change timing types
- Combining a valid and stable signal gives a _____ signal
- Combining a stable signal and a clock gives a _____ signal
- Combinational logic should not have both _s1 and _s2 inputs

Disadvantages of 2-Phase Clocks

- Need four clocks in general
  - Need true and complement of both clocks
- Still need low skew for good performance
  - The skew increases the cycle time of the machine
  - Need low skew between all the clocks for good performance
    - Want to have $\Phi 1$ and $\Phi 2$ close to coincident
- Many systems use clock and its complement instead of 2 phases
  - Needless to say they are very careful about clock skew
  - For these systems it is still useful to maintain 2 phase timing types, since it ensures you connect all logic to the right latches
  - Call Clk - $\Phi 1$ and Clk\ - $\Phi 2$, and go from there.
  - (Note in this class we will use $\Phi 1$ and $\Phi 2$ for clocks)

Advantages of Latches over Flops

Why not just go back to flops?

- Many people do
  - Most designs in industry are based on flops
  - Very easy to verify timing
    - Each path between flops must be less than cycle time
  - Tools check for skew and hold time violations
    - Short paths are padded (buffers are added to slow down the signals)
  - Skew in flop based systems affects the critical path
- Latch designs are more flexible than a flop design
  - Need to CAD tools to make sure it works properly
  - Can borrow time to allow a path to be longer than clock period

    (see text for details)
  - Can always slow down clock until the circuit works