Lecture 7:

Clocking of VLSI Systems

Overview

Reading

Wolf 5.3 Two-Phase Clocking (good description)

W&E 5.5.1, 5.5.2, 5.5.3, 5.5.4, 5.5.9, 5.5.10 - Clocking

Note: The analysis of latch designs in 5.5.3 is not correct, don't be confused by it. Also the description of two phase clocking in 5.5.9-5.5.10 is not very good. The notes are probably better

Introduction

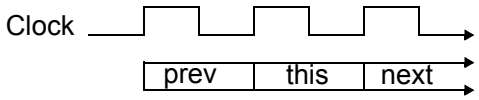
We will take a brief digression and talk about different methods of sequencing FSMs. This is usually done using clocks and storage elements (latches and flip-flops). This lecture looks at the function that clocks serve in a system, and the trade-offs between the different clocking methods. It presents 2-phase clocking, one of the safest clocking methods around, and the one we will use in this class.

Industry uses clocking methods that are less safe (either edge-triggered design or latch design using clock and clock_b) and the lecture will discuss these clocking methods as well.

Why Have Clocks

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. These previous outputs are the state bits of the FSM, 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.



Common View of Clock's Function

Clocks work with Latches or Flip-Flops to hold state

Latch

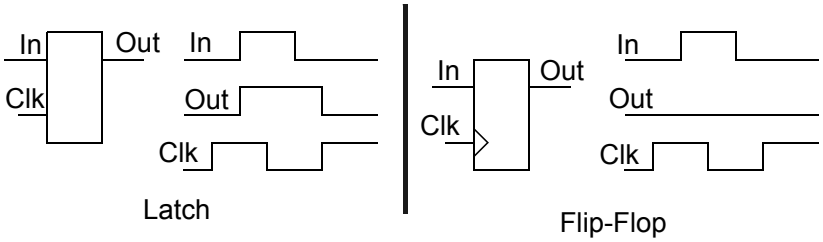
When the clock is high it passes In value to Output

When the clock is low, it holds value In had when the clock fell

Flip-Flop

On the rising edge of clock, it transfers the value of In to Out

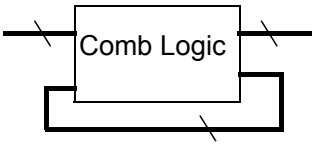
It holds the value at all other times.



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
  - All I really need to know is that there is some boundary
- If the delay of every path through my logic was **exactly** the same



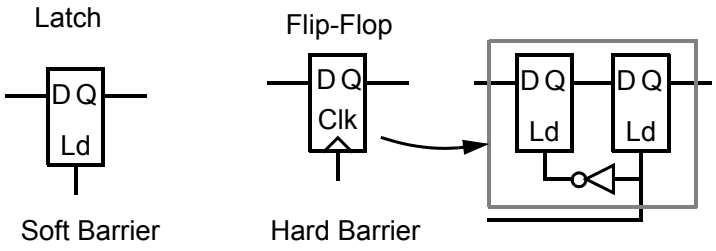
  - Then I would not need clocks
  - Signals stay naturally correlated in time

    People do this to a limited degree. It is called wave pipelining.
  - The 'state' is stored in the gates and the wires.

Alternative View

Clocks serve to slow down signals that are too fast

- Flip-flops / latches act as barriers

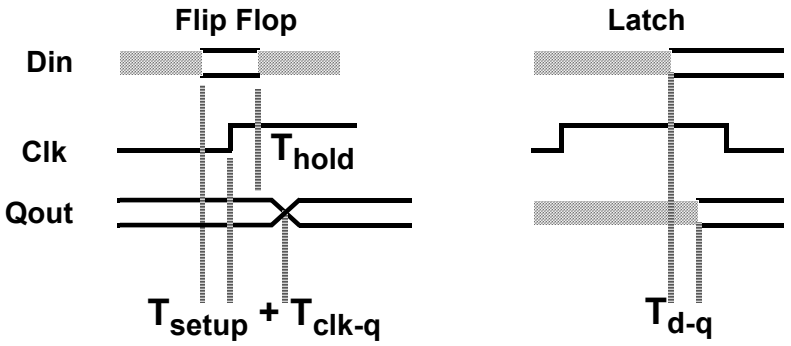


  - With a latch, a signal can't propagate through until the clock is high
  - With a Flip-flop, the signal only propagates through on the rising edge

    Note that all real flip-flops consist of two latch like elements (master and slave latch)

Clocking Overhead

Problem is that the latches and flops slow down the slow signals too



Flip-flop delays the slowest signal by the setup + clk-q delay

Latches delay the late arriving signals by the delay through the latch

Start by focusing on the most common clocking discipline - edge triggered flops

Clock Skew

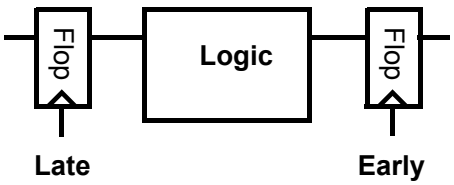
Not all clocks arrive at the same time

- Some clocks might be gated (ANDed with a control signal) or buffered
- There is an RC delay associated with clock wire

Causes two problems

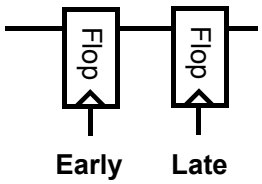
- The cycle time gets longer by the skew

$$T_{cycle} = T_d + T_{setup} + T_{clk-q} + T_{skew}$$



- The part can get the wrong answer

$$T_{skew} > T_{clk-q} - T_{hold}$$



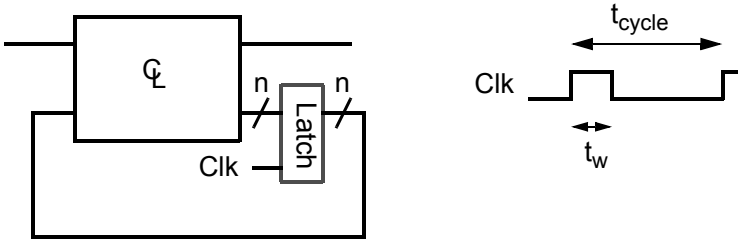
Clock Design

- Trade off between overhead / robustness / complexity
  - Constraints on the logic
    - vs.
  - Constraints on the clocks
- Look at a number of different clocking methods:
  - Pulse mode clocking
  - Edge triggered clocking
  - Single phase clocking
  - Two phase clocking
    - The one we will use
    - The most robust.

Pulse Mode Clocking

Two requirements:

- All loops of logic are broken by a single latch
- The clock is a narrow pulse
  - It must be shorter than the shortest path through the logic



- Timing Requirements

$t_{dmax} < t_{cycle} - t_{d-q} - t_{skew}$

$t_{dmin} > t_w - t_{d-q} + t_{skew}$

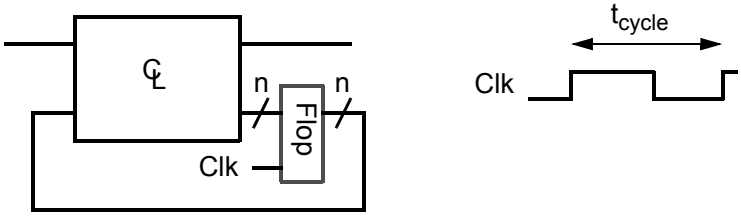
Pulse Mode Clocking

- Used in the original Cray computers (ECL machines)
- Advantage is it has a very small clocking overhead
  - One latch delay added to cycle
- Leads to double sided timing constraints
  - If logic is too slow OR too fast, the system will fail
- Pulse width is critical
  - Hard to maintain narrow pulses through inverter chains
- People are starting to use this type of clocking for MOS circuits
  - Pulse generation is done in each latch.
  - Clock distributed is 50% duty cycle
  - CAD tools check min delay

Not a good clocking strategy for a beginning designer

Edge Triggered Flop Design

- Popular TTL design style
- Used in many ASIC designs (Gate Arrays and Std Cells)
- Using a single clock, but replaces latches with flip-flops



- Timing Constraints

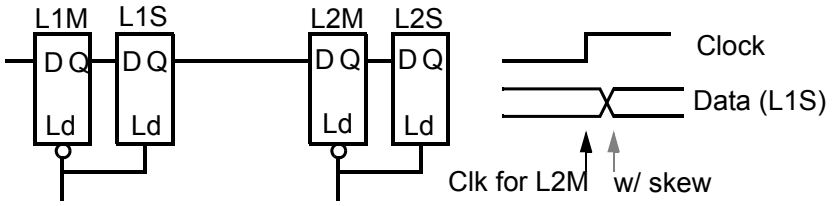
$$t_{dmax} < t_{cycle} - t_{setup} - t_{clk\text{-}q} - t_{skew}$$

$$t_{dmin} > t_{skew} + t_{hold} - t_{clk\text{-}q}$$

- If skew is large enough, still have two sided timing constraints

The Problem

- The same edge controls the enable of the output and the latching of the input – the rising edge of the clock.



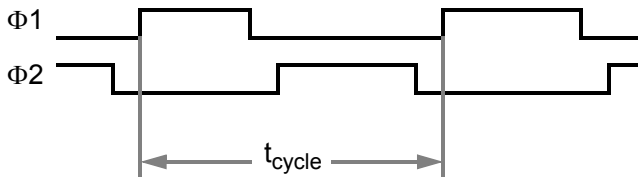
- If there is skew the L2M can close after the data from L1S changes
- Since both events are triggered from the same clock edge, a user can't change the clock and make the circuit work

  Changing the clock frequency will not fix this problem

  Need to change flop or skew – need to redo the chip

2 Phase Clocking

Use different edges for latching the data and changing the output



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

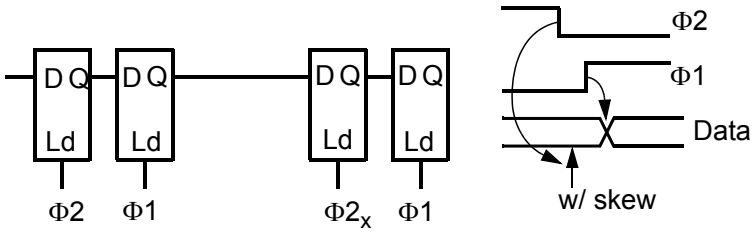
- Φ1 high
- Φ1 falling to Φ2 rising
- Φ2 high
- Φ2 falling to Φ1 rising

These two times can be small (less than zero if skew is 0)

Key is the spacing is under user control

2 Phase Clocking

Look at shift register again:

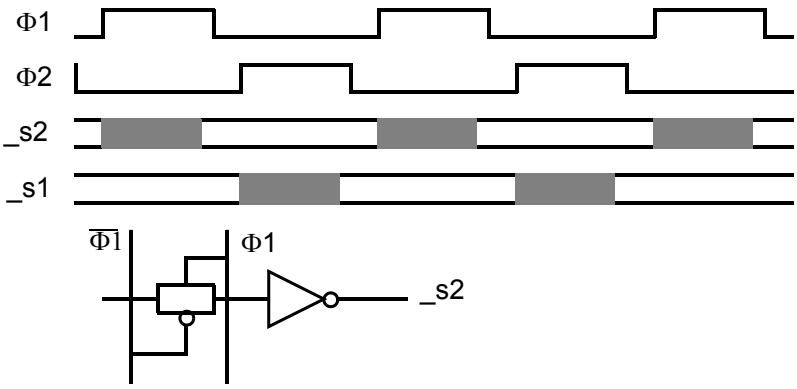


- If there is a large skew on the $\Phi 2_x$ clock, then the spacing between $\Phi 1$ and $\Phi 2$ can be increased to make sure that even with the skew, the $\Phi 2$ latch closes before the $\Phi 1$latch lets the new data pass.
- For some setting of the timing of the clock edges, the circuit will like a perfect abstract FSM.

Terminology

We will give signals timing types, so it will be easier to know which latch to use:

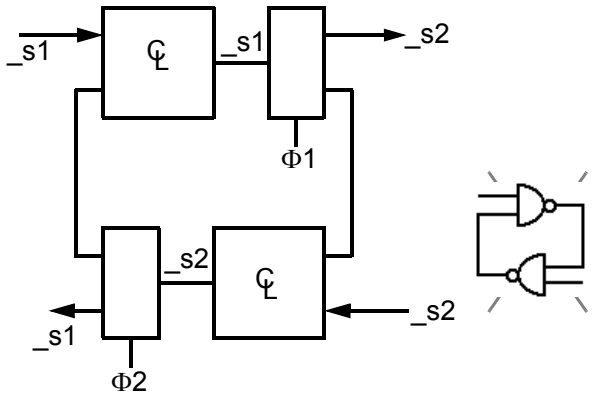
- Output of a $\Phi 1$ latch is stable $\Phi 2$ (_s2) – good input to $\Phi 2$ latch
- Output of a $\Phi 2$ latch is stable $\Phi 1$ (_s1) – good input to $\Phi 1$ latch



- Signal is called stable2, since it is stable for the entire $\Phi 2$ period

General 2 Phase System

Combination logic does not change the value of timing types.



No static feedback in the combination logic is allowed either. This makes the system not sensitive to logic glitches.

Why 2 Phase Clocking

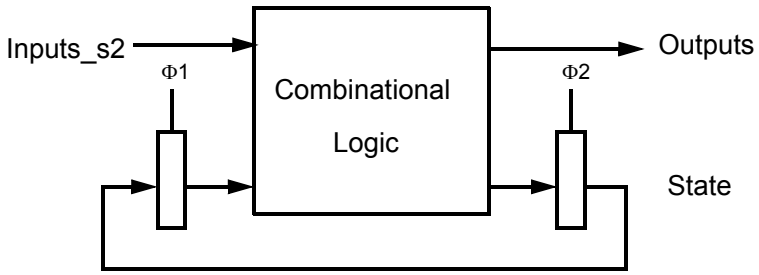
It is a constrained clocking style:

- Synchronous design
- Two clocks
- Constrained composition rules

But gives this guarantee:

- If you clock it slow enough (with enough non-overlap between edges)
  - Model as FSM
  - It will be a level sensitive design
    - no race, glitch, or hazard problems
    - no skew problems
  - One sided timing constraints
    - Logic can't be too fast

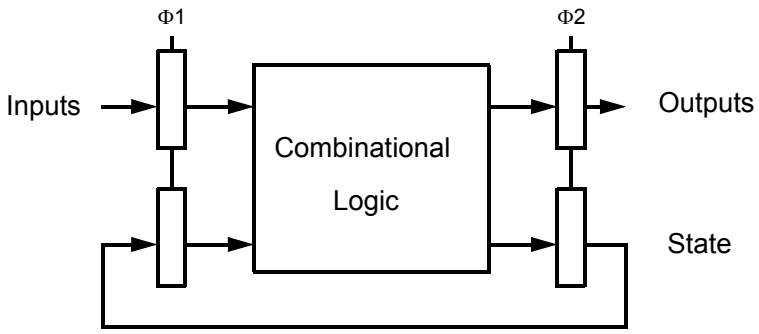
A Generic Mealy Machine



This is a Mealy machine, since the outputs are a direct function of the inputs. Note that the inputs and the state inputs to the combination logic need to have the same timing type (_s2 in this case).

Note: Mealy outputs change in the same cycle as the input change, but this means they might change late in the cycle.

A Generic Moore Machine



This is a Moore machine, since both the state and the outputs are registered (you could consider the outputs to be more state bits). But since you call only some of the outputs state bits, you can implement a Mealy state transition diagram this way.

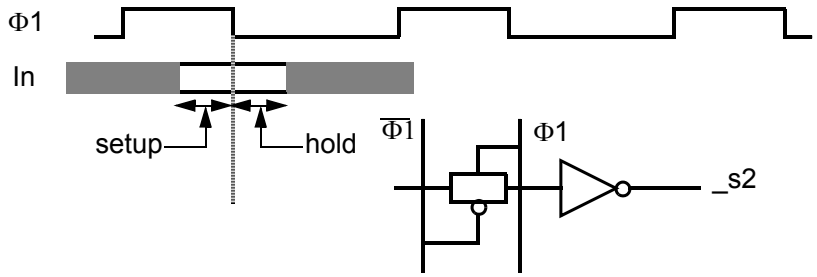
Note: All Moore outputs change a cycle after the inputs change

More Timing Type

Look a little more closely at latches, to come up with a more complete set of timing types (more than _s1 _s2 signals) that we can use in our synchronous designs.

- Look at a latch since this the critical element

  What is the weakest requirement on the input to a latch?



  Signal must settle before Φ1 falls, and not change for some time after Φ1 falling, even for a skewed Φ1 (this is usually called the setup and hold times of the latch)

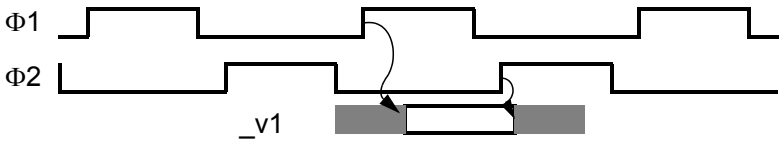
Valid Signals

The weakest input to a latch is called a valid signal (_v1 _v2)

- For a valid signal we need to be sure we can guarantee it meets the setup and hold requirements of the latch

  To do this we need to have the signal settle off an edge that comes before Φ1 falling. The closest edge is Φ1 rising.

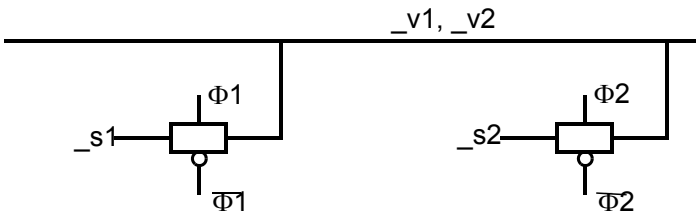
  The signal should not change until an edge occurs that comes after Φ1 falling. The closest edge is Φ2 rising.



- If we changed the input on Φ1 falling, most of the time the circuit would work fine. But if it failed, we can't change the clock timing to make the circuit work -- Φ1 falling controls the changing of the input, and the closing of the latch. Since we can't guarantee it would be ok a signal that changes on Φ1 falling would not be a _v1 signal.

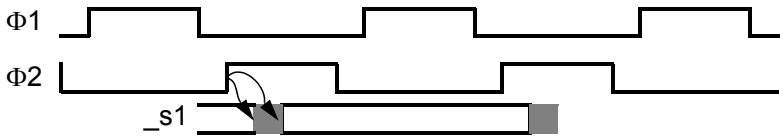
+ Use of a Valid Signal

- Very useful for precharged logic, but that comes later in the class
- Is not needed for standard combinational logic with latches
  - This should always give stable signals
- Can't use stable signals if you want to drive two signals/cycle on a wire (multiplex the wire), since the value has to change twice. There are many wrong ways to do it, and only one right way, which is shown below. The values become _v signals.



Stable Signals

Have even larger timing margins than valid signals[1]



- A _s1 signal starts to change sometime after Φ2 rises
- A _s1 signal settles sometime after Φ2 rises

  Input to the latch must be a _v2 (settles after Φ2 rises)

  Output of a latch settles some small delay after input settles

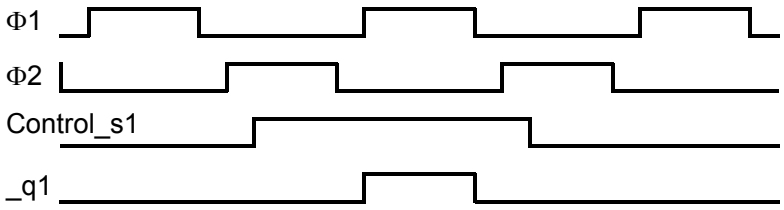
1. Please note that combinational logic does not change the value of the timing type, even though it does increase the delay of the signal path. The timing types have to do with the clocking guarantee that we are trying to keep. This promise is that the circuit will work at some frequency. A _s1 signal might not settle until after Φ1 rises when the part is run at high-frequency, but the label means that you can make that signal stabilize before Φ1 rises if you need to by slowing the clock down.

Qualified Clocks

These are signals that have the same timing as clocks, but they don't occur every cycle. They are formed by ANDing a '_s1' signal with Φ1 giving _q1, or ANDing a '_s2' signal with Φ2 giving a _q2 signal.

- The control signal needs to be a stable signal to prevent glitches on the qualified clocks.
- Qualified clocks can only be used as the clock input to a latch

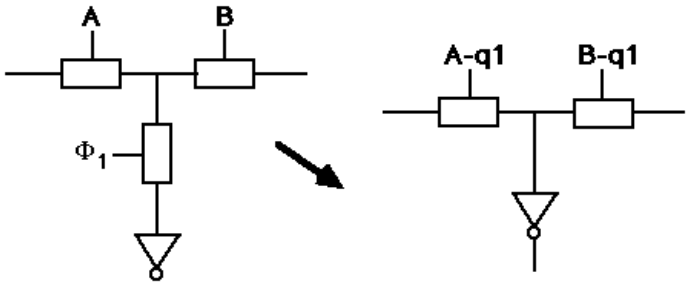
  It is not a valid data input (why)



Qualified (gated) Clocks

They provide a conditional load

- Allow the merging of a mux and clock



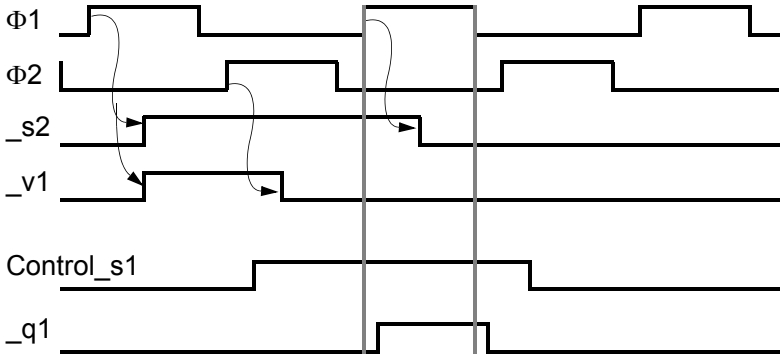
- Remember you need to generate both true and complement control for the transmission gates

  Control * Clk

  $\overline{\text{Control * Clk}}$

- Clocking Types Review

The figure shows the timing of all the signals we have discussed with little arrows that indication with clock edge caused the signal to change. Remember the pictures, and the timing types are what the signals look like at slow clock frequencies



Verilog Coding Rules

- Follow 2 phase clocking in all your verilog code
- Append a timing type to all signals in your design
  - _s1 _s2 _v1 _v2 _q1 _q2
  - For weird timing types use _w
- For pipelined machines you can add an additional character to indicate which pipestage that signal is from
  - For example if a machine has an Instruction Fetch and an Execute cycle, then you could use _s1i for signals related to the fetch, and _s1e for signals related to the exec stage.
- Standard verilog latch

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

Verilog Rules

- Remember that combinational logic does not change the timing types
- Combining a valid and stable signal always leaves a valid signal
- Combinational logic should not have both _s1 and _s2 inputs, since it will not have a good timing type for its output[1]
- There is a program, vcheck, that will check your verilog for clocking / signal labelling problems.

1. Actually it does have a defined timing type (I will let you figure it out), but it is probably a logical bug, so it is not allowed.

Disadvantage of 2 Phase Clocking

- 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 $\overline{\text{Clk}}$ - $\Phi 2$, and go from there.
  - (Note in this class we will use $\Phi 1$ and $\Phi 2$ for clocks)

Advantage of Latches Over Flops

If you are going to use Clk and Clk_b and control skew, why not 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
  - Gives the designer more rope

    Need to CAD tools to make sure s/he uses it wisely
  - Can borrow time to allow a path to be longer than clock period
  - Can tolerate clock skew -- skew does not directly add to cycle time