EE273 Lecture 12
Synchronization

February 26, 2001

William J. Dally
Computer Systems Laboratory
Stanford University
billd@csl.stanford.edu

Logistics

- No class on Wednesday 2/28 – class will be during problem session on Friday 3/2 instead.
- Prof Dally will have no office hours on Wednesday 2/28.
- Project feedback (second session)
  - Friday 9:00AM to 11:00AM – 15min appointments
- Reading
  - Sections 10.3
  - Complete before class on Friday 3/2

How are people doing on their projects?

Today's Assignment

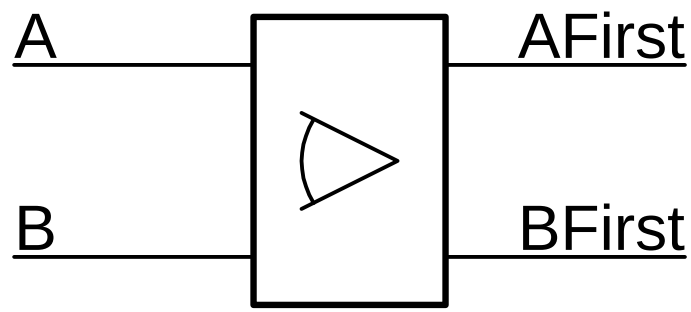
- Reading
  - Section 10.3
  - Complete before class on Wednesday 03/08

A Quick Overview

- Synchronization
  - determining an event order
  - used for
    - moving a signal into a clock domain
    - asynchronous arbitration
- Synchronization Failure
  - as the time between two signals decreases it becomes more difficult to tell which came first
  - synchronizer may hang in a metastable state, unable to decide
  - different parts of the circuit may interpret result differently
- Failure Probability
  - is proportional to fraction of *vulnerable* time
  - exponentially decreases with waiting period
  - exponentially increases with flip-flop time constant
  - failure rate is proportional to event rate

What is Synchronization?

- A *synchronizer* determines the order of events on two signals
- Which event came first?
  - Does it matter? Some times synchronization is unnecessary
- Often one signal is a clock
  - did the data go high before or after the clock went high?
- Why is this problem hard?



Uses of Synchronization

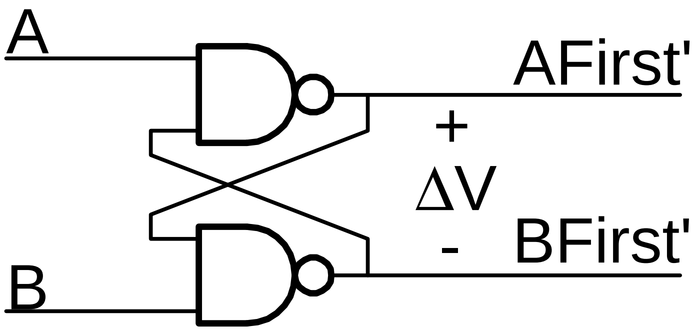
- Sampling asynchronous inputs with a clock
  - e.g., particle counter or pushbutton
- Crossing clock domains
  - sampling a synchronous signal with a *different* clock
  - this is an easier problem if both clocks are periodic
- Arbitration of asynchronous signals
  - e.g., request line for shared resource
  - game-show pushbutton

Synchronization Failure

- Which came first, event on A or event on B?
- The closer the race, the harder it is to call
- When the events are very close, the synchronizer may enter a *metastable* state
- The synchronizer may take an arbitrary amount of time to *exit* this state
- Synchronizer output may be interpreted inconsistently in the meantime

A NAND Arbiter

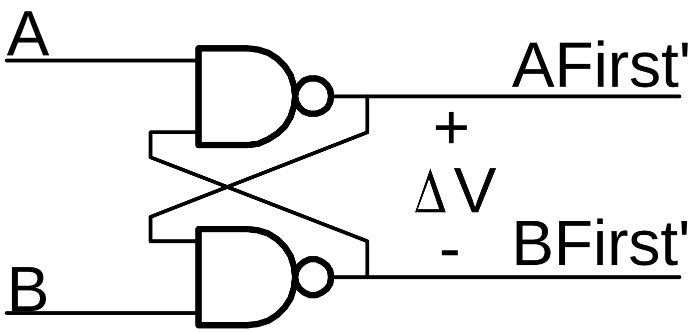
- Consider a NAND RS flip-flop
- We can attempt to use this as an arbiter.
  - If A arrives much earlier than B, AFirst’ goes low and locks B out
  - If B arrives much earlier than A, BFirst’ goes low and locks A out.
- What happens if A and B go high at nearly the same time?

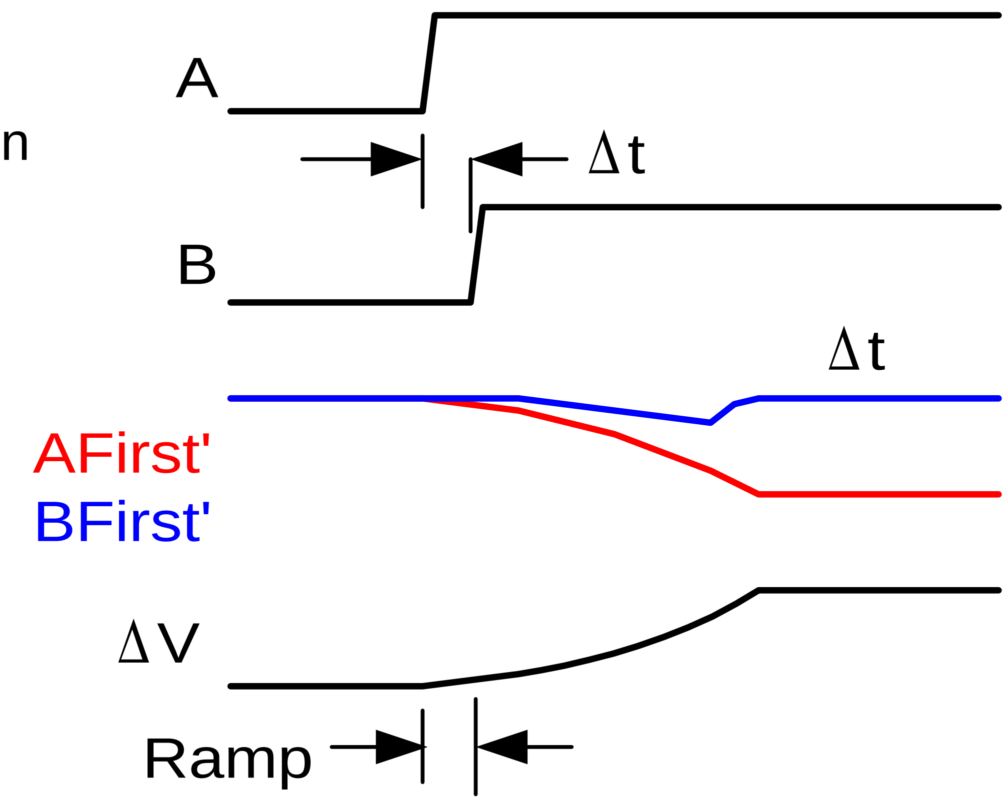


A NAND Arbiter Dynamics
Step 1: Initial Voltage

- When A and B go high at nearly the same time, difference in voltage is proportional to difference in time

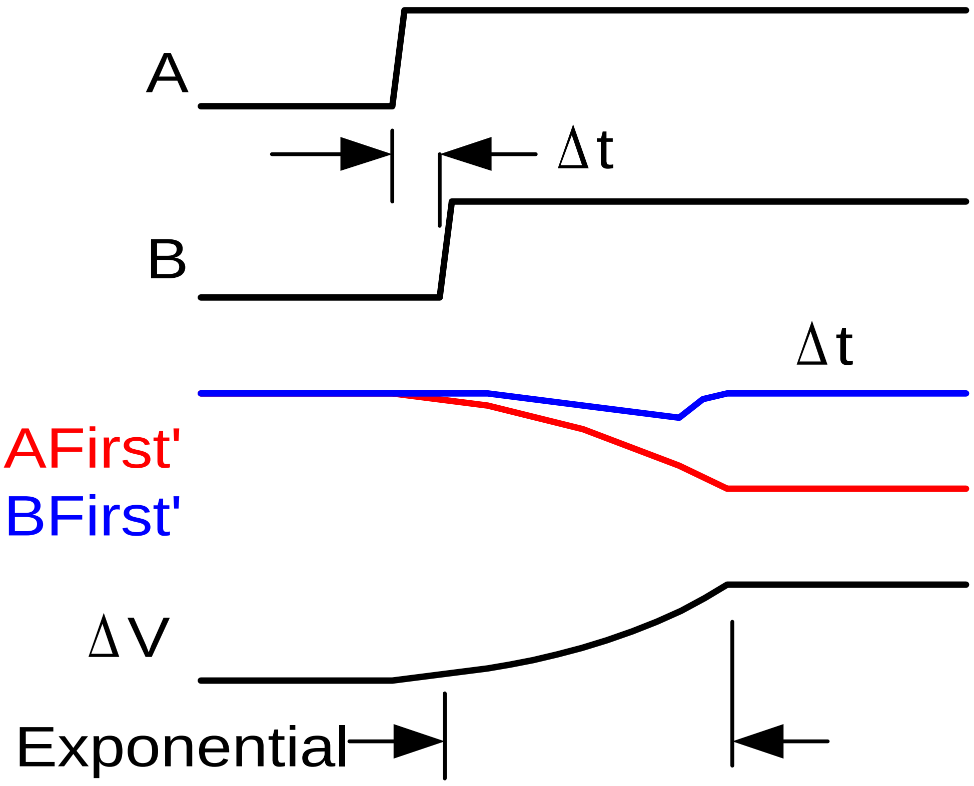
$$\Delta V_1 = K_S \Delta t$$





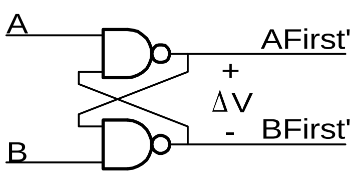
A NAND Arbiter Dynamics
Step 2: Exponential Regneration

- After A and B are both high, initial voltage difference is exponentially amplified.



$$\Delta V_1 = K_S \Delta t$$

$$\Delta V(t) = \Delta V_1 \exp\left(\frac{t}{t_S}\right)$$



A NAND Arbiter Dynamics
Settling Time

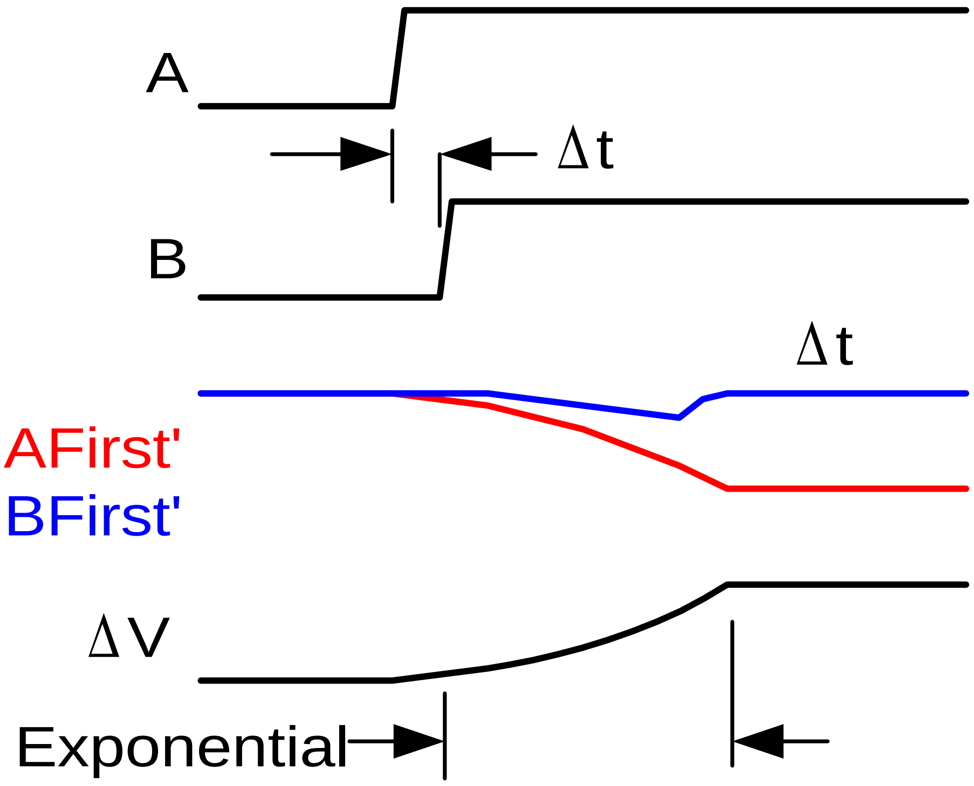
- After A and B are both high, initial voltage difference is exponentially amplified.

$$\Delta V_1 = K_S \Delta t$$

$$\Delta V(t) = \Delta V_1 \exp\left(\frac{t}{t_S}\right)$$

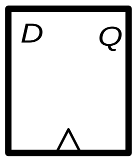
$$\Delta V_1 = \exp\left(-\frac{t}{t_S}\right)$$

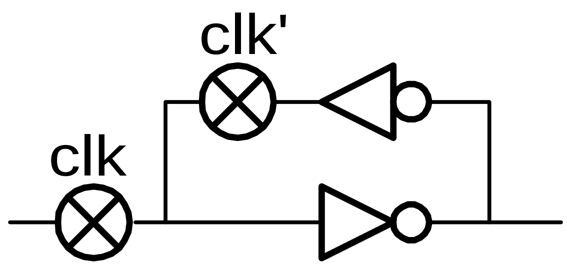
$$\begin{aligned} t_x &= -t_S \log(\Delta V_1) \\ &= -t_S \log(K_S \Delta t) \end{aligned}$$



Static Flip-Flop Dynamics are Similar

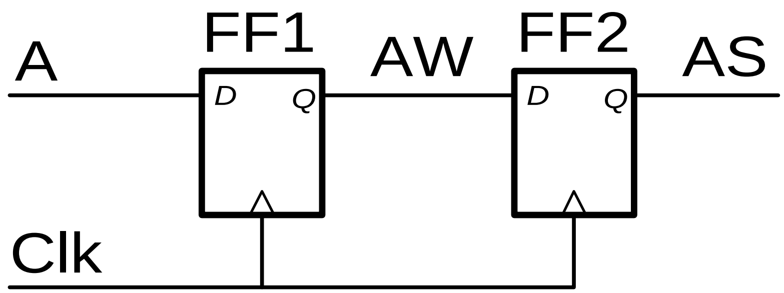
- Initial voltage difference depends on Δt
- Voltage difference increases exponentially after clock rises

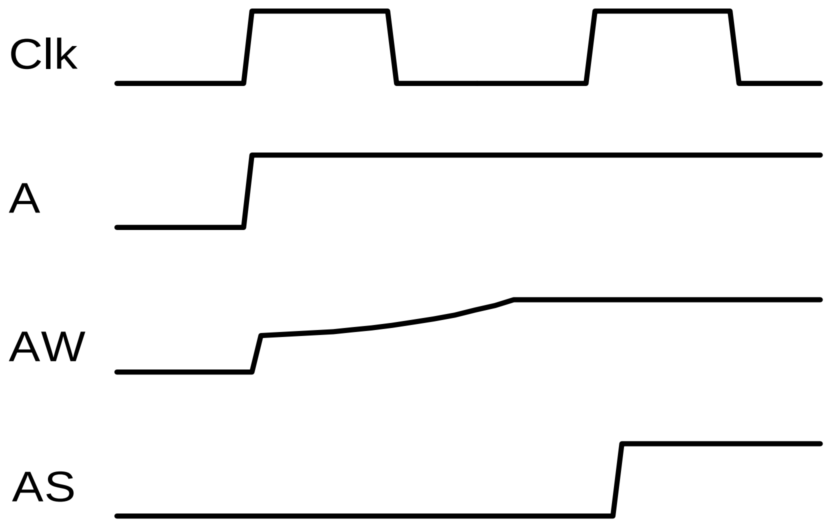




A Brute-Force (Waiting) Synchronizer

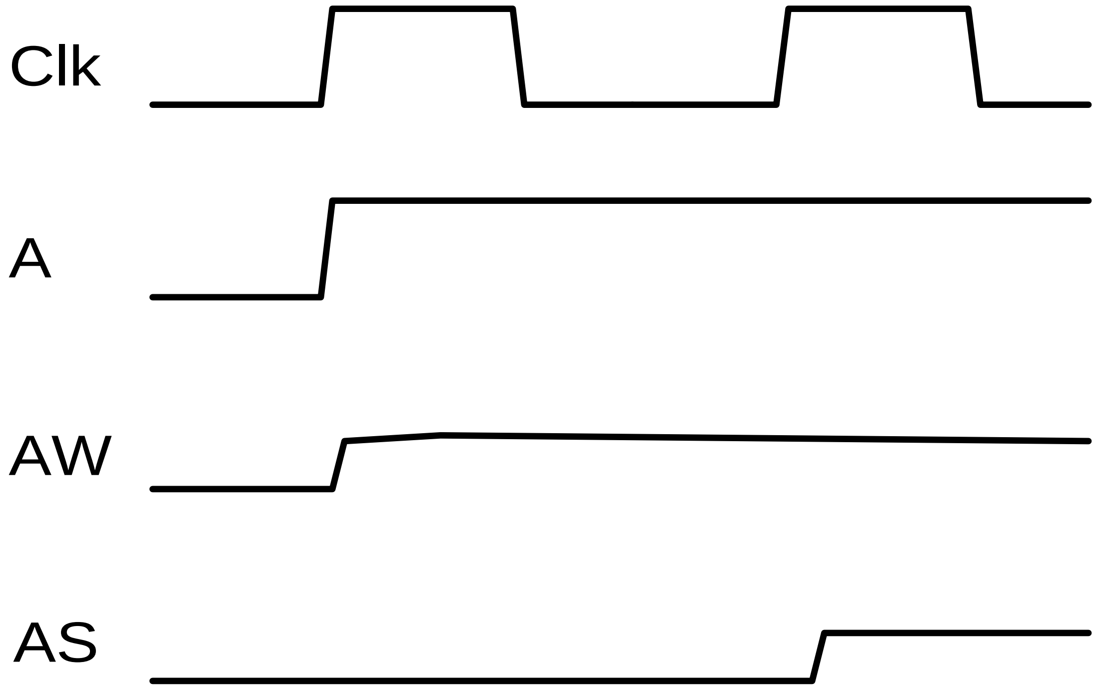
- To sample an asynchronous signal with a clock
- Sample signal with FF1
  - may go into a metastable state
- Wait for possible metastable stages to decay
  - time t_w
- Sample output of FF1





Synchronization Failure

- What happens if FF1 is still in a metastable state when FF2 is clocked?
- What is the probability that this will happen?

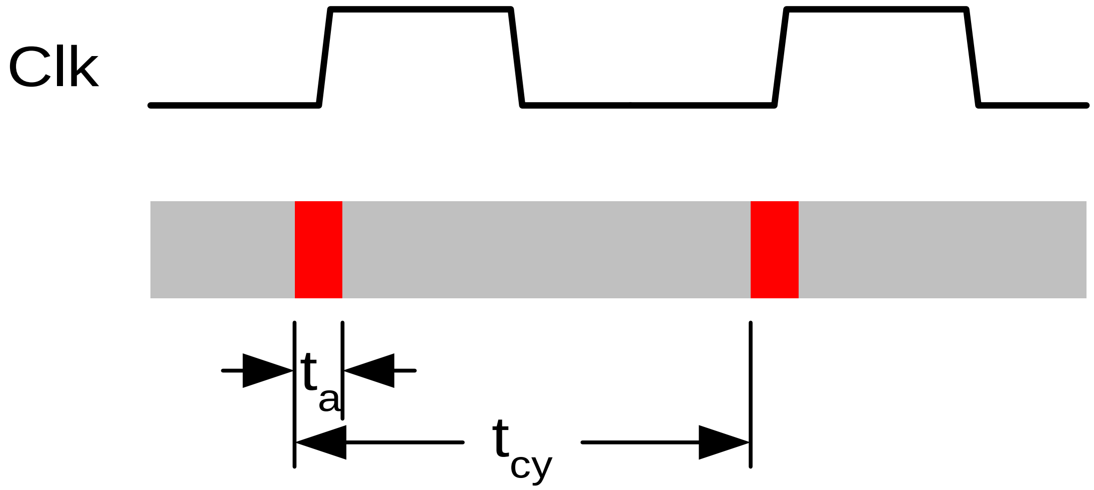


Calculating Synchronization Failure (The Big Picture)

P(failure) = P(enter metastable state) x P(still in state after t_w)

Probability of Entering a Metastable State

- FF1 may enter the metastable state if the input signal transitions during the *aperture* time of the flip flop
- Probability of a given transition being in the aperture time is the fraction of time that *is* aperture time



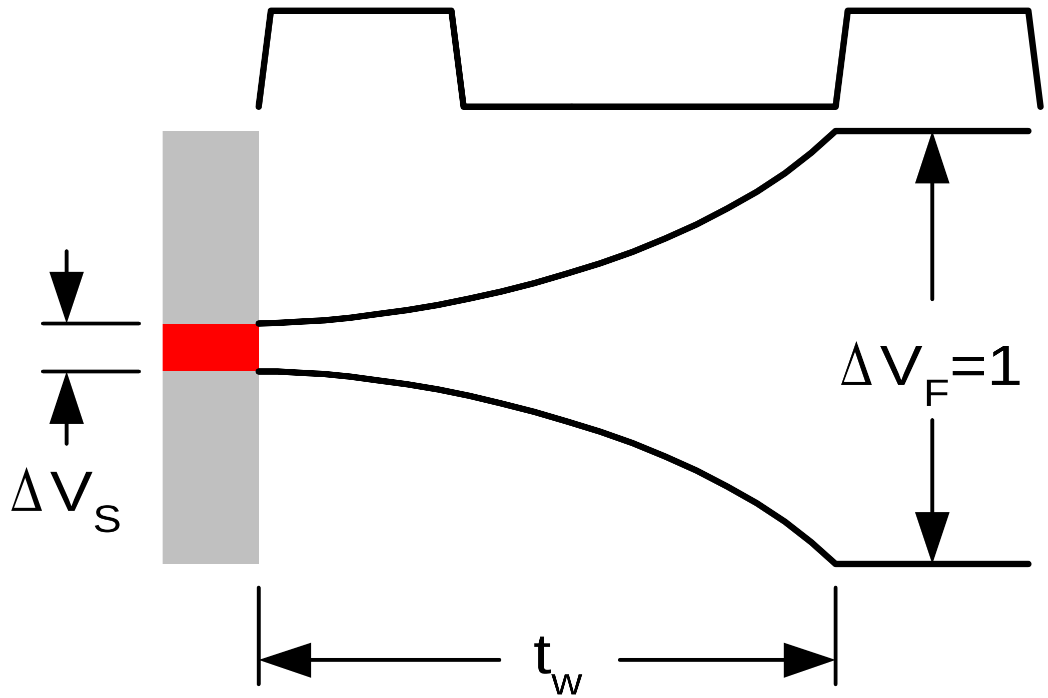
$$P_E = \frac{t_a}{t_{cy}} = f_{cy} t_a$$

Probability of Staying in the Metastable State

- Still in metastable state if initial voltage difference was too small to be exponentially amplified during wait time
- Probability of starting with this voltage is proportion of total voltage range that is 'too small'

$$\Delta V_S = \Delta V_F \exp\left(\frac{-t_w}{t_S}\right)$$

$$P_S = \exp\left(\frac{-t_w}{t_S}\right)$$

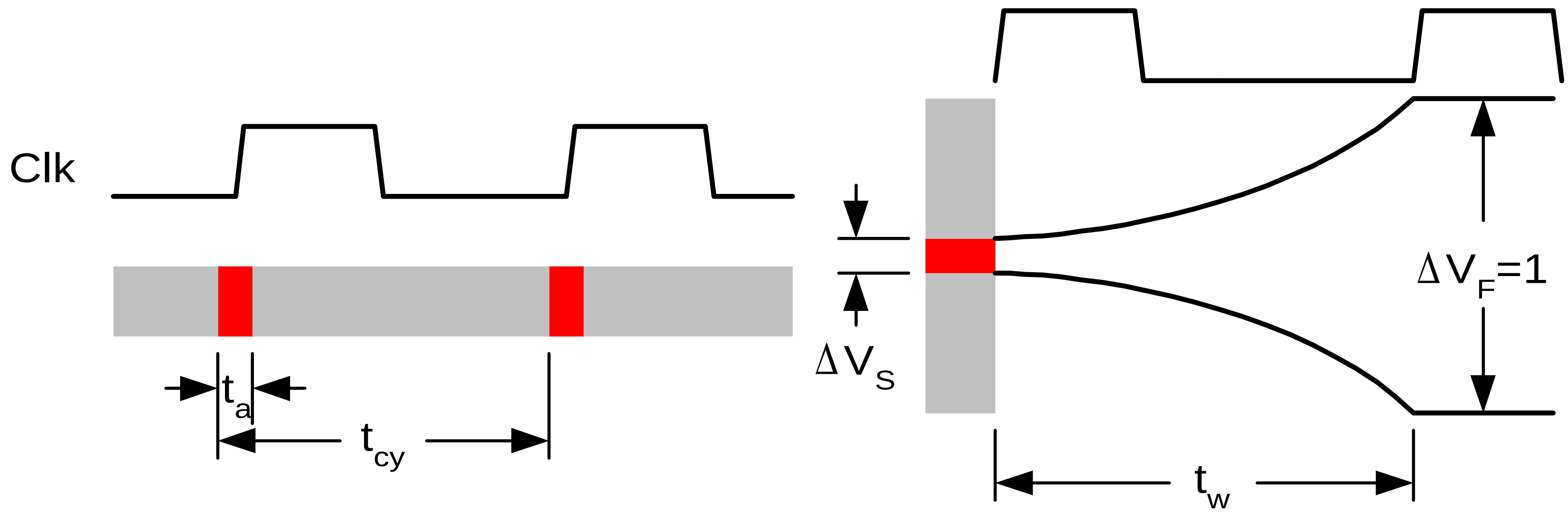


Failure Probability and Error Rate

- Each event can potentially fail.
- Failure rate is given by the event rate times the failure probability

$$P_F = P_E P_S = t_a f_{cy} \exp\left(\frac{-t_w}{t_S}\right)$$

$$f_F = f_e P_F = t_a f_e f_{cy} \exp\left(\frac{-t_w}{t_S}\right)$$



Example Failure Rate Calculation

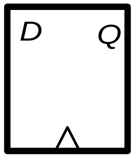
- Suppose a 500MHz clock samples a 10MHz asynchronous signal
- Flip-flops have aperture and regeneration time of 100ps
- What is the probability of synchronization failure?
- What is the failure frequency?

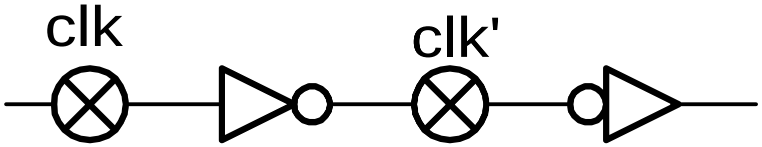
t_a	1.0E-10
f_cy	5.0E+08
τ_s	1.0E-10
t_w	2.0E-09
P_E	5.0E-02
P_S	2.1E-09
P_F	1.0E-10
f_e	1.0E+07
f_F	1.0E-03

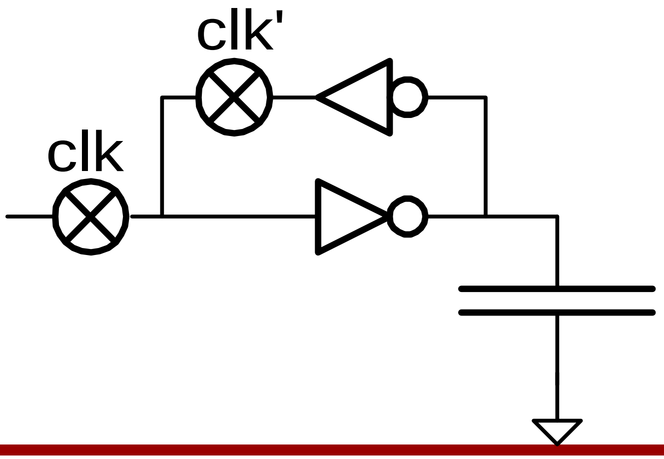
Common Pitfalls

- Its easy to get a synchronizer design wrong
- The two most common pitfalls are:
  - using a non-restoring (or slowly restoring) flip-flop
    - τ_S needs to be small
  - not isolating the flip-flop feedback loop

$$P_F = t_a f_{cy} \exp\left(\frac{-t_w}{t_S}\right)$$

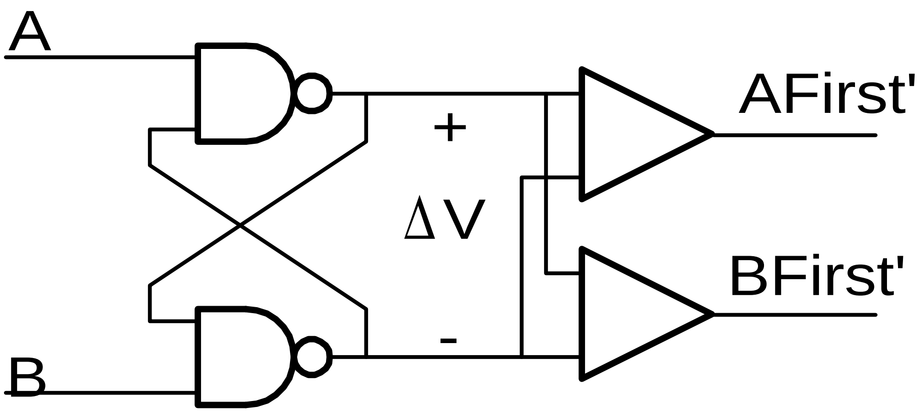






Completion Detection

- It is not possible to bound the amount of time needed for a synchronizer to settle.
- It is, however, possible to detect when the synchronizer has settled!
- This is only useful if the downstream logic can use this *asynchronous* completion signal



Next Time

- Synchronizer Design