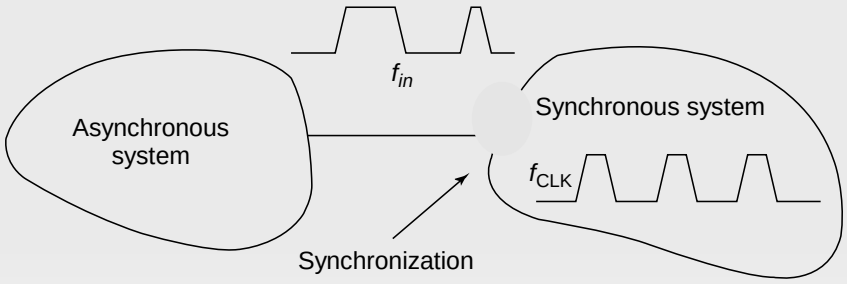
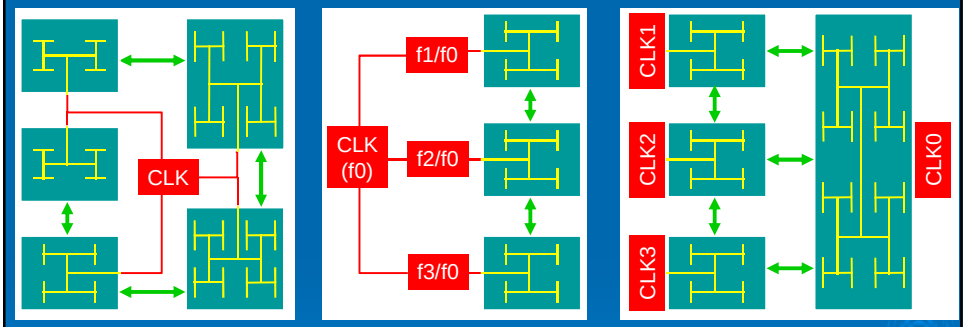
Asynchronous-Synchronous Interface



Need of Interfacing

- Asynchronous design is good, but should communicate with synchronous-latency penalty
- Synchronous system gets asynchronous input from keyboard
- Even if no asynchronous modules are used, synchronous modules operating at different clock rates or out of phase can have the same problem.

Multiple clock domains- *it becomes increasingly difficult to distribute a single global clock to all parts of the chip.*

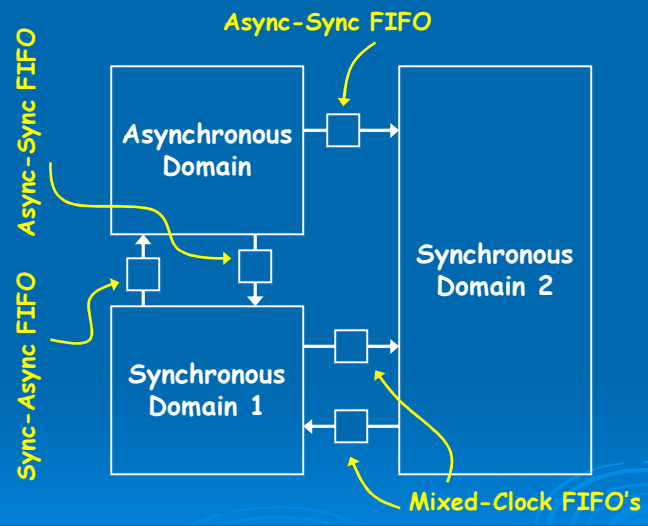


Single clock (Mesochronous)

Rational clock frequencies

Independent clocks (plesiochronous if frequencies closely match)

Mixed-Timing Interfaces



Chelcea & Nowick, 2001

Synchronization problem

- WHEN?--Occurs when a synchronous circuit must synchronize an asynchronous input.
- HOW?--if the clock edge arrives too close in time to data arriving from an asynchronous circuit, the circuit may enter a meta-stable state in which its output is at neither a logic 0 or logic 1 level, but rather, lies somewhere in between.

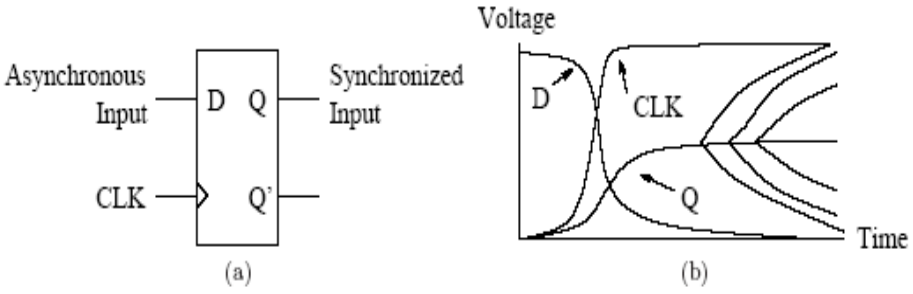


Fig. 1. (a) Simple, dangerous synchronizer. (b) Oscilloscope view of metastable behavior.

- Assume that Q is initially low and that D has recently gone high.
- If D goes low again at about the same time that CLK rises, the output Q may start to rise and then get stuck between the logic levels as it observes D falling. Should Q rise or fall?
- Actually, either answer would be okay, but the FF becomes indecisive.
- At some point, Q may continue to a logic 1 level, or it may drop to the logic 0 level.
- When this happens, however, is theoretically unbounded.

- Subsequent FF looks at the synchronized input, it sees an indeterminate value.
- This value may be interpreted by different subsequent logic stages as either a logic 0 or a logic 1. This can lead the system into an illegal or incorrect state, causing the system to fail.
- Such a failure is traditionally called a synchronization failure.

Background Work

- The problem was largely ignored until 1966.
- Even after that the synchronization problem was not widely known or understood as several asynchronous arbiters designed in the early 1970s suffered from metastability problems

Awakening

- Finally, in 1973 experimental evidence of the synchronization problem presented by Chaney and Molnar appears to have awakened the community to the problem
- After this paper, a number of papers were published that provided experimental evidence of metastability due to asynchronous inputs, and mathematical models were developed to explain the experimental results

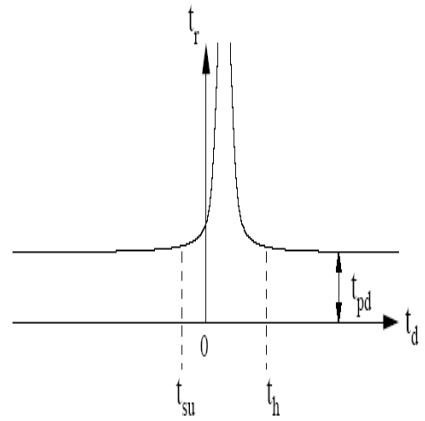
Hard fact

- Meta-stability in a BISTABLE is unavoidable

Probability of Synchronization Failure

- An acceptance of this fact and a careful analysis of this probability is crucial in designing a reliable system

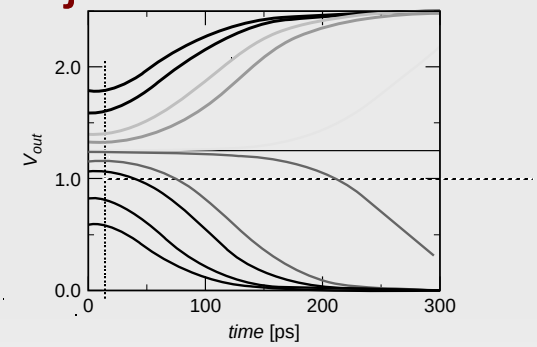
Representative plot based on measured data for the response time of a FF as a function of the arrival time of data, td, with respect to the clock



IMPORTANT OBSERVATIONS

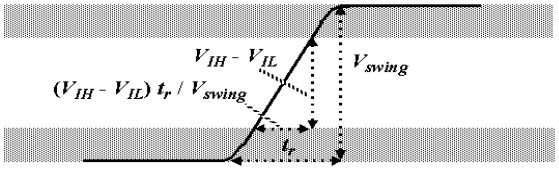
- If data only changes before the setup time, tsu, and after the hold time, th, of a flip-flop, the response time, tr, is roughly constant and equal to the propagation delay through the flip-flop, tpd.
- If, data arrives between the setup and hold times, the delay increases.
- In fact, if the data arrives at just the absolutely wrong time, the response time is unbounded.

Synchronizer: Output Trajectories



Single-pole model for a flip-flop

$$v(t) = V_{MS} + (v(0) - V_{MS})e^{t/\tau}$$



$$N_{sync}(0) = \frac{P_{init}}{T_\phi} = \frac{\left(\frac{V_{IH} - V_{IL}}{V_{swing}}\right) t_r}{T_{signal}} \frac{1}{T_\phi}$$

$$N_{sync}(T) = \frac{P_{init} e^{-T/\tau}}{T_\phi} = \frac{(V_{IH} - V_{IL}) e^{-T/\tau}}{V_{swing}} \frac{t_r}{T_{signal} T_\phi}$$

Example

T_f = 10 nsec = T

T_{signal} = 50 nsec

t_r = 1 nsec

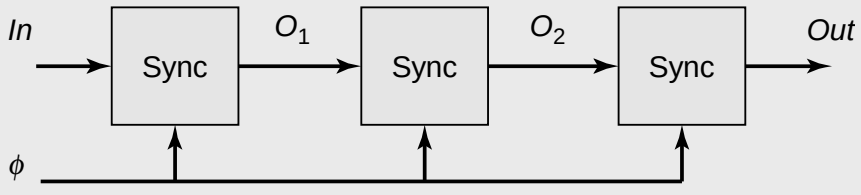
t = 310 psec =tau

$V_{IH} - V_{IL}$ = 1 V (V_{DD} = 5 V)

N(T) = 3.9 10^{-9} errors/sec
MTF (T) = 2.6 10^{8} sec = 8.3 years
MTF (0) = 2.5 μsec

N(0)= 400,000

Cascaded Synchronizers Reduce MTF



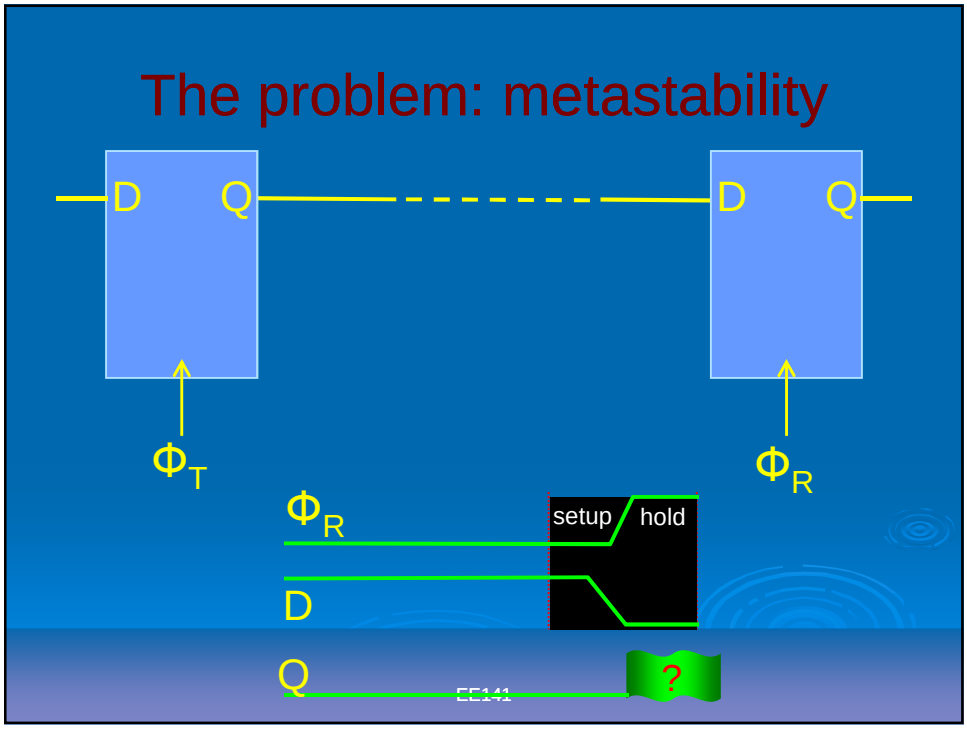
MODELING

- The probability that the data arrives at a time td which falls between tsu and th is --

$$P(t_d \in [t_{su}, t_h]) = \frac{t_h - t_{su}}{T}$$

- If we assume that the latch is given some bounded amount of time, tb, to decide, then the probability of a synchronization failure is related to the probability that the response time, tr, exceeds tb
- if td falls in this range, the probability that tr > tb can be expressed as follows:

$$P(t_r > t_b \mid t_d \in [t_{su}, t_h]) = \frac{1}{k + (1-k)e^{(t_b - t_{pd})/\tau}}$$



How to live with metastability ?

- Metastability cannot be avoided, it must be tolerated.
- Having a decent MTBF (≈ years) may result in a tangible impact in latency
- Purely asynchronous systems can be designed failure-free
- Synchronous and mixed synchronous-asynchronous systems need mechanisms with impact in latency
- But latency can be hidden in many cases …

Different approaches

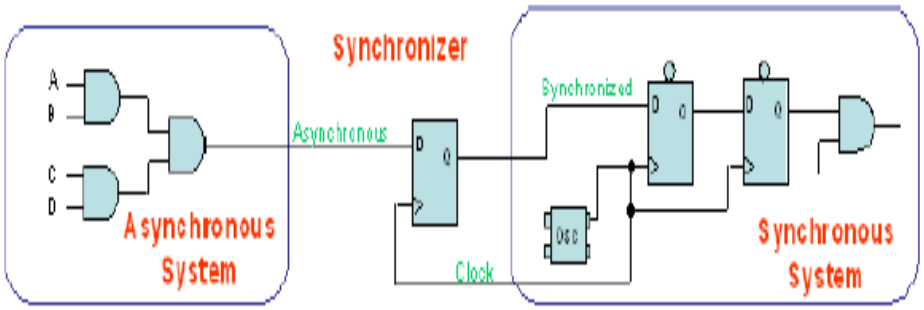
- Pausible Clocks (Yun & Donohue 1996)
- Predict metastability-free transmission windows for domains with related clocks (Chakraborty & Greenstreet 2003)
- Use the waiting time in FIFOs to resolve metastability (Chelcea & Nowick 2001)
- And others …
- The term “Globally Asynchronous, Locally Synchronous” is typically used for these systems (Chapiro 1984)

Synchronizers and Arbiters

- Arbiter: Circuit to decide which of 2 events occurred first
- Synchronizer: Arbiter with clock ϕ as one of the inputs
- Problem: Circuit HAS to make a decision in limited time - which decision is not important
- Caveat: It is impossible to ensure correct operation
- But, we can decrease the error probability at the expense of delay

Problem: Introducing an asynchronous signal into a digital {synchronized} system, using Flip-Flops.
The outcome is Intermittent or random failures during operation.

How to avoid metastability in ICs: Add an additional Flip Flop in the design to Synchronize the incoming asynchronous signal with the new clock domain, which will reduce the Mean-Time-Between-Failure [MTBF].



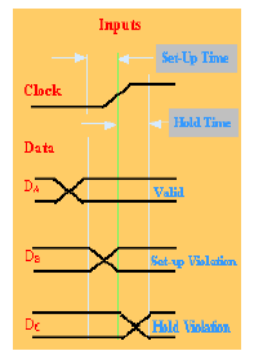
Resolve Time: The amount of time the Flip Flop's output must return to a valid level before it's used.
This is 1/{clock frequency} - path delay. The output must be valid by the next clock, minus any chip or routing delay.
Path Delay = Tcko + Troute + Tsu;
.... Tcko = Clock to Output time of the flip flop,
.... Troute = Any trace delay between the the Q of the flip flop and the next device reading that data,
.... Tsu = any Set-Up time required by the next device reading the data.

Skew {Clock or data}: The change in time of one signal compared to another, caused by timing delays or propagation delays. ~The timing differences developed by different devices performing the same function.

Ambiguity: The uncertainty in the amount of time it takes for a valid logic signal to change from one state to another.

Metastability Window: The specific length of time, during which both the data and clock should not occur. If both signals do occur, the output may go metastable.

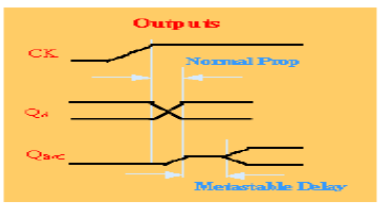
Why metastability occurs



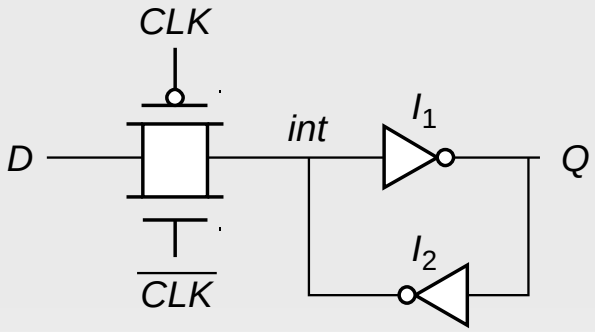
- Time between two white line is metastability window
- Async. sig. comes any time violating setup or hold time constraints. Causing metastable state

Possible behavior of latch

Five possible conditions exist:
One: No timing violation occurred, and the output moves to the appropriate state (high or low).
Two: A timing violation does occur, and the output oscillates between the valid states (for a long time), or until its needed.
Third: A timing violation does occur, and the output moves to the wrong state.
Forth: A timing violation does occur, and the propagation delay is increased. Causing the next device in the chain to see the wrong value.
Fifth: A timing violation does occur, and no meta-stable behavior occurs. The output moves to the correct state with no oscillation or increase in propagation delay. However this condition is problematic because the gamble is taken each time the device is clock. This condition may persist for thousands of clock cycles, but may fail on the next clock cycle. For a fail-safe design stay in condition one, any other condition will result in failure.



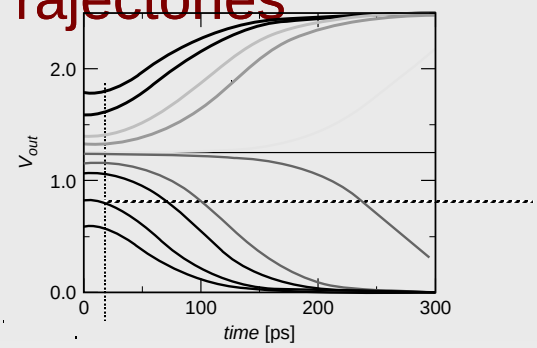
Avoid metastability--A Simple Synchronizer



- **Data sampled on rising edge of the clock**
- **even if input not valid, Latch will eventually resolve the signal value,**

but ... this might take infinite time!

Synchronizer: Output Trajectories

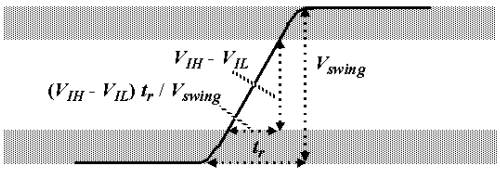


Single-pole model for a flip-flop

$$v(t) = V_{MS} + (v(0) - V_{MS})e^{t/\tau}$$

Modelling

Mean Time to Failure

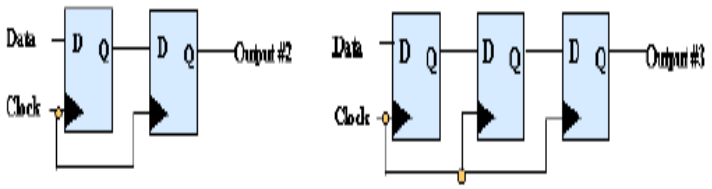


$$N_{sync}(0) = \frac{P_{init}}{T_\phi} = \frac{\left(\frac{V_{IH} - V_{IL}}{V_{swing}}\right) t_r}{T_{signal}} \frac{1}{T_\phi}$$

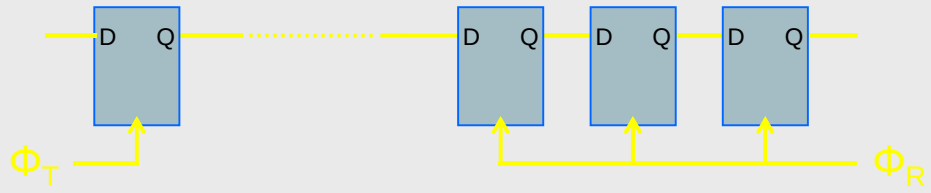
$$N_{sync}(T) = \frac{P_{init} e^{-T/\tau}}{T_\phi} = \frac{(V_{IH} - V_{IL}) e^{-T/\tau}}{V_{swing}} \frac{t_r}{T_{signal} T_\phi}$$

Multi-Stage Synchronizer

Adding a second Flip Flop to the design will reduce the chance of the output going Metastable.
The output from the first flip flop may go valid, before the second flip flop is clocked. Adding yet another flip flop will reduce the probability that its output will be unstable even more. A 74AS4374 from TI provides a 'D' type, Dual stage synchronizer.



Classical "synchronous" solution



Mean Time Between Failures

- f_Φ: frequency of the clock
- f_D: frequency of the data
- t_r: resolve time available
- W: metastability window
- τ: resolve time constant

$$MTBF = \frac{e^{t_r/\tau}}{2 \cdot f_\phi \cdot f_D \cdot W}$$

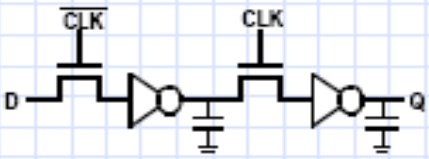
Example

# FFs	MTBF
1 FF	15 min
2 FF	9 days
3 FF	23 years

Synchronizer design

Flip Flop design is important?

- Dynamic FFs not suitable for synchronizers since they have no regeneration



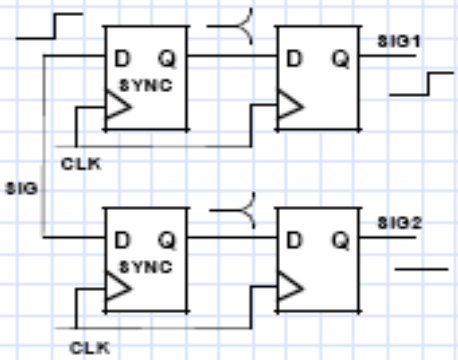
CMOS Dynamic FF

TSFF (Svenson)

- Special _SYNC_ FFs should be used for the primary synchronizer if available

Synchronization Pitfall

- Never synchronize the same signal in multiple places! Inconsistency will result!



Conclusions

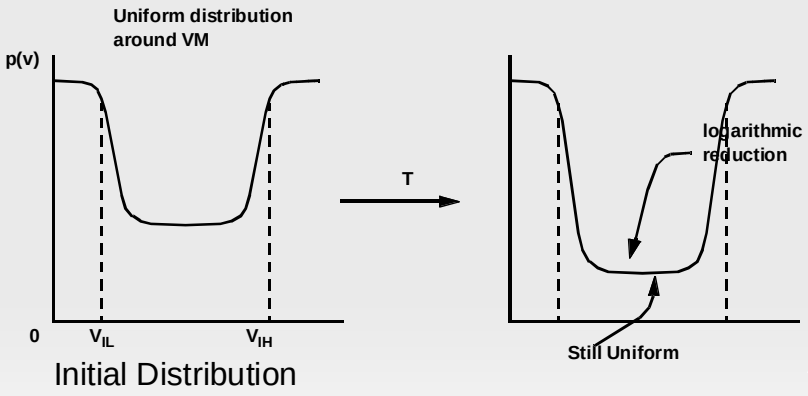
- Synchronizers are important. Synchronization failure is deadly and difficult to debug
- Synchronization requires careful design. Most CAD and logic tools CANNOT catch bad synchronizer designs.
- Design of synchronizer depends on performance level needed. Basic synchronizer of back-to-back FFs is the core design all others are based on.

Words to the wise

- Be wary of synchronizer schemes designed by others
  - Synopsys Designware DW04_sync multi-bit synchronizer DOES NOT WORK as a synchronizer
  - Synthesizers might use dynamic FFs as synchronizers _ they don_t know the difference.
  - Auto-placement tools must be told to place synchronizer FF pairs close together

BE PARANOID

Influence of Noise



Low amplitude noise does not influence synchronization behavior

The probability of escape from metastability with time is given by:

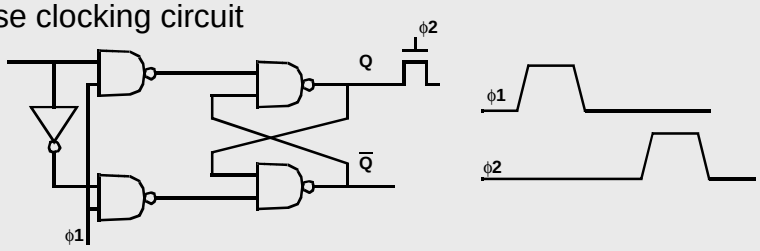
$$P_e(t) = 1 - e^{\frac{-t}{\tau}}$$

This function does not change with the addition of noise because of the uniform distribution of initial conditions.

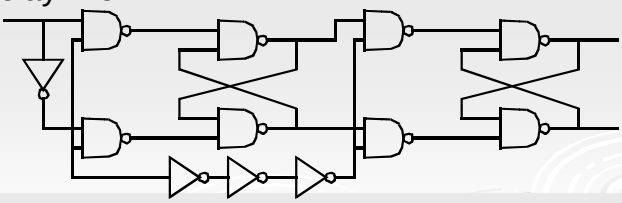
For each noise contribution that moves a trajectory away from metastability, there will, on average, be another compensating noise contribution that moves a trajectory towards metastability. The result, in a statistical measurement, is that the event histogram will be unchanged

Typical Synchronizers

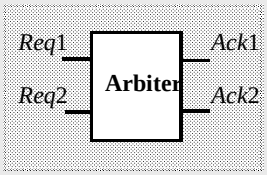
2 phase clocking circuit



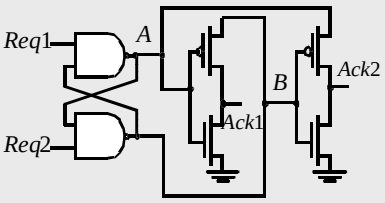
Using delay line



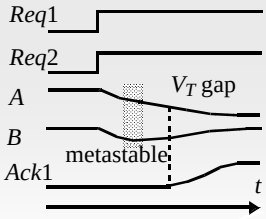
Arbiters



(a) Schematic symbol

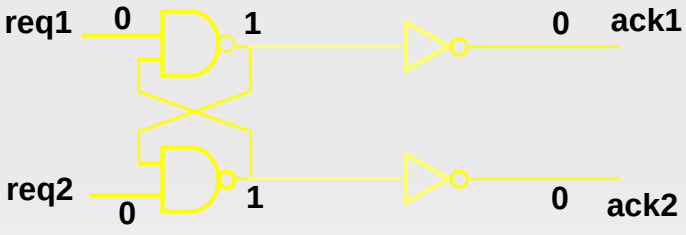


(b) Implementation

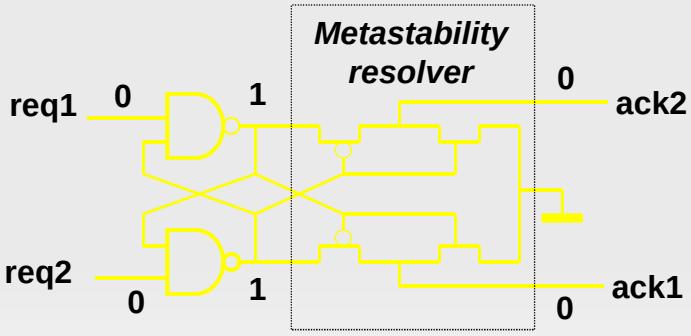


(c) Timing diagram

Mutual exclusion element



Mutual exclusion element



An asynchronous data latch with MS resolver can be built similarly