ECE4740: Digital VLSI Design

Lecture 27: (A)synchronous circuits

Dos and don'ts (or do's and don'ts, or do's and don't's)

Safe (a)synchronous circuits

Careful with sequential logic

- Digital VLSI designs often fail because of timing issues and not wrong functionality
- Correct and deterministic operation can only be guaranteed if all signals settled before stored in flip-flop, latch, RAM, etc.
- There are essentially three ways
  - Synchronous clocking
  - Asynchronous clocking
  - (Self-timed clocking)

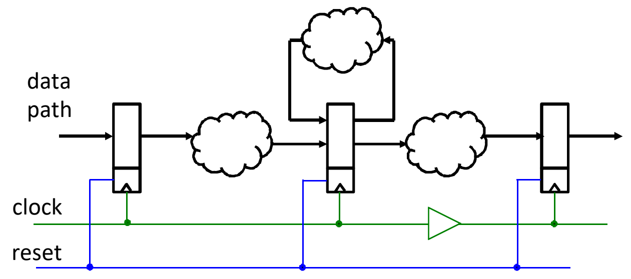
978

Synchronous clocking

- All storage operations and state transitions occur periodically at precise moments in time determined by a single clock
- Clock domain: Subcircuit where all clock signals maintain fixed frequency and phase relationships
- Clock boundary: the separation between two distinct so-called clock domains

979

Synchronous clocking (cont'd)



- Data path and clock/reset strictly separated
- Clock and reset nets may contain buffers

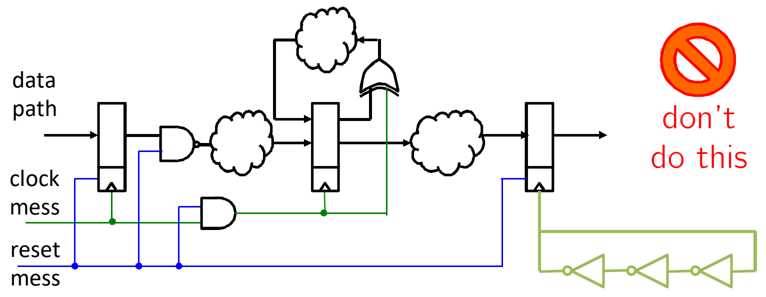
980

Asynchronous clocking

- Some or all of the storage elements are permitted to change their states independently from a global reference
- Such circuits may contain
  - Zero-latency feedback loops, ring oscillators
  - Asynchronous state machines (ASMs)
  - Logic gates on clock and reset nets
  - Unclocked bistables (e.g., SR latch)
  - Etc.

981

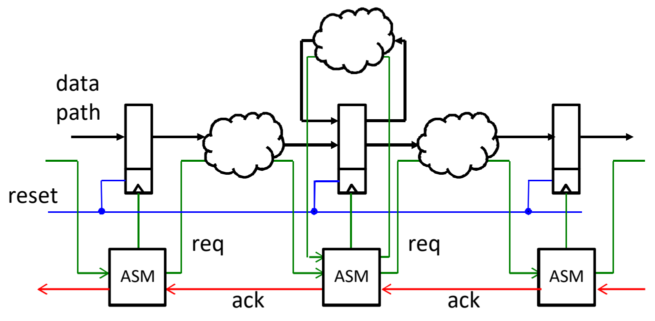
Asynchronous clocking (cont'd)



- Data path and clock/reset not separated
- Logic on clock/reset nets
- Multiple clock sources
- Zero latency feedback loops

982

(Self-timed circuits)



- Request and acknowledge signals control dataflow among blocks
- Each block runs as fast as it can

Why is careful clocking important

- Glitches/hazards are unwanted transients
  - Glitch is what you see, hazard is the cause
  - Causes: Reconvergent fan-outs, multiple inputs that change at different time instants, etc.
- Critical rule: clocks, asynchronous reset, write lines of RAMs etc. must always be glitch free

984

Potential failures if rule violated

- Unwanted transitions in state machines
- Unwanted reset to initial state
- Erroneous triggering of interrupts in processor
- Storage of bogus data in flip-flop or RAM
- Data loss or duplicates during data transfer
- Deadlocks in asynchronous communication
- Metastable behavior or marginal triggering
- And many more…

985

Rules for safe synchronous designs

- Strictly separate reset, clock, and information signals (data, control, test, etc.)
- Allow all signals to settle before storage
- No unclocked bistables (e.g., SR latch)
- No zero-latency feedback loops
- No logic on clock/reset signals*
- Distribute clock & async. reset by fanout tree
- Never use reset for functionality (gated reset)

*terms and conditions may apply

Pros of synchronous clocking

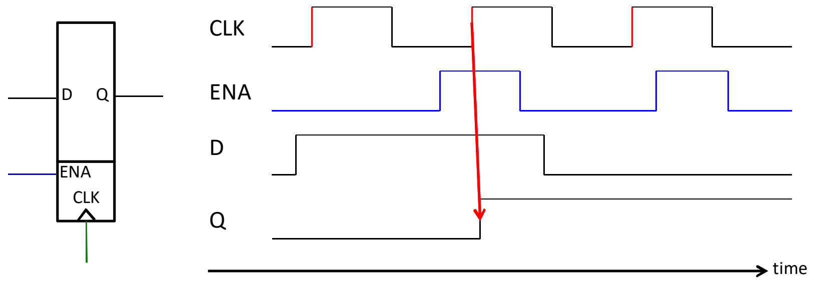
- Glitches do not compromise functionality
- No chance for inconsistent data
- Immunity to noise and interference
- All timing constraints are one-sided; enables to slow down or deactivate computation
- Deterministic behavior
- Enables separation of functional verification from timing analysis
- Automated tool support
- Simplified functional circuit testing and verification

But how about enable signals and clock gating?

Never have logic on clock signal!!!

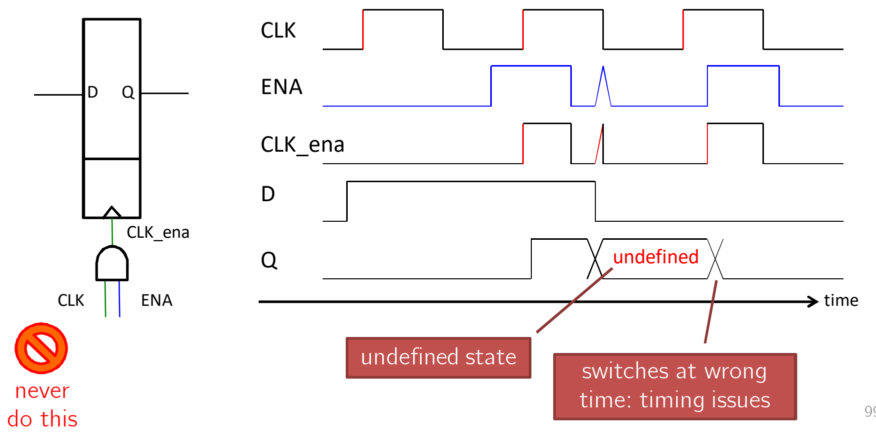
Rule: Never use logic on clock signal

- Very few exceptions: only if you know what you are doing!



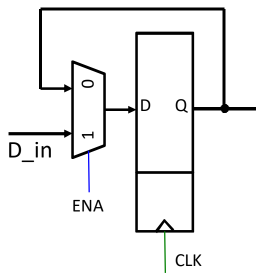
Problems with this approach

- Glitches on ENA (will) cause failures
- Timing issues



Safe flip-flop with ENA

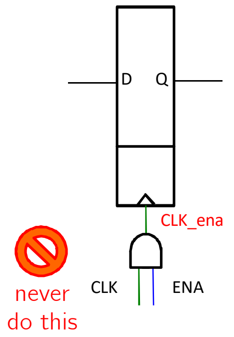
- Compliant with rules for synchronous design
- No logic on clock signal



- Not efficient from energy perspective: clock still active
  - Solution: safe clock gating

Clock gating

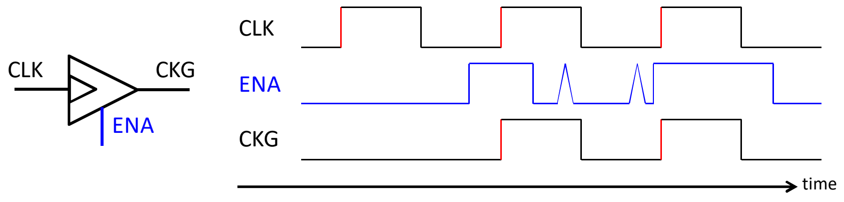
- Idea: Switch off clock to disable flip-flop(s)
- Significantly reduces dynamic power consumption of entire subcircuits



- AND gate on clock path disables activity of flip-flop
- Glitches cause failures

992

Clock gating (cont'd)

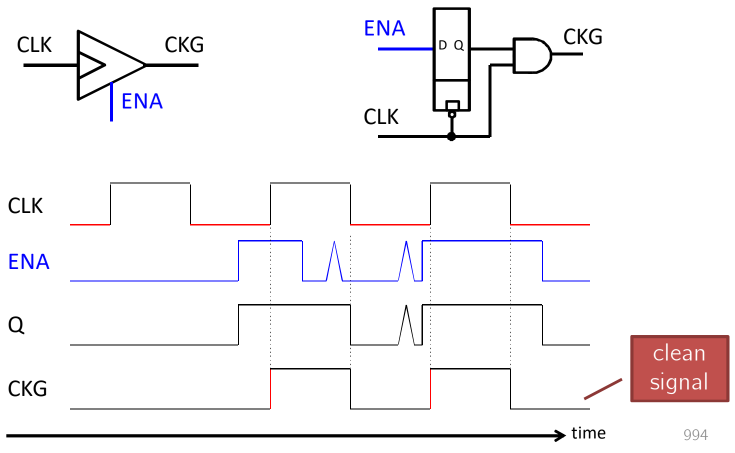


- Ensure that glitches and early ENA release signals do not contaminate CKG signals!
- Modern standard-cell libraries often include robust clock-gate cells

993

Safe clock gate implementation

- Single-edge triggered clock gate:



Every sequential circuit needs this!

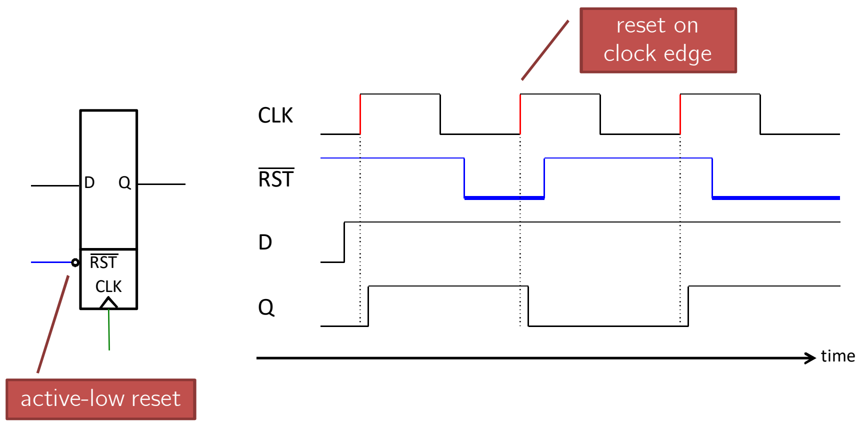
Reset

The reset signal

- Required to force circuit into predefined state (initialization)
- Determines when to enter a given state

- Usually applied at beginning of time/operation
- Rarely applied during operation
  → common exception: watchdog

Synchronous reset



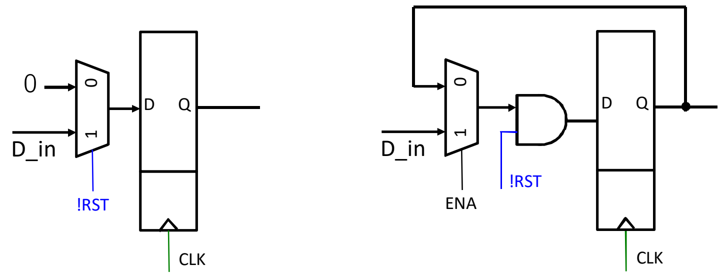
- Resets whenever !RST=0 and at clock edge
- Applied as any other input to flip-flop

Why is reset usually active low?

- During power-up, keeping !RST=0 is more safe as voltage high level where gates are operating correctly is not clearly defined
  - Low level (GND) is always clearly defined
- It is easier for external sources (e.g., switches) to safely provide active low signal
  - Active high requires V_{DD} available at at switch
- Also a historical reason from TTL circuits:
  - Could easily produce GND but not VDD
  - Can sink more current than source

998

Synchronous reset (cont'd)



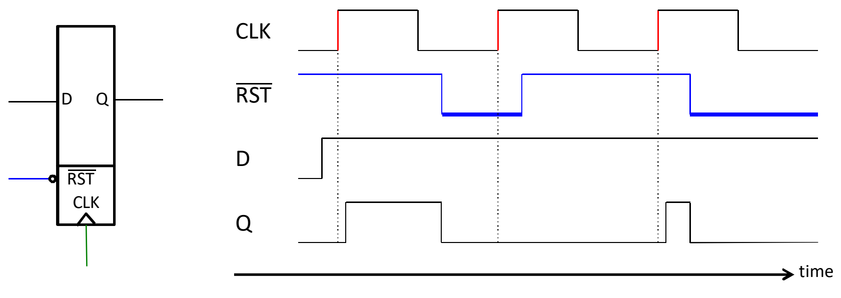
- !RST signal behaves as regular input signal
- Versions that set FF to 1 also exist

999

Synchronous reset (cont'd)

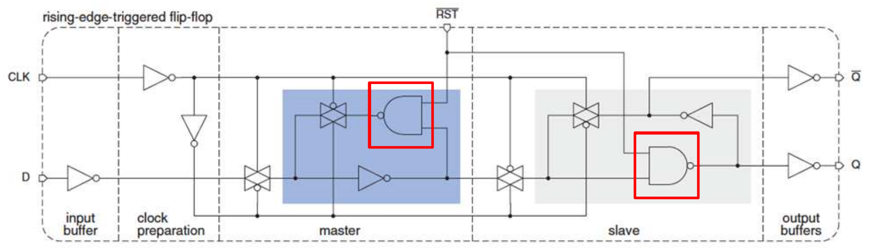
- Synchronous reset advantages
  - Circuit completely synchronous
  - (Sometimes smaller flip-flops)
- Synchronous reset disadvantages
  - Reset tree required to ensure all resets occur in same clock cycle
  - May require pulse stretch to ensure that all flip-flops see !RST signal at rising clock edge
  - Requires a clock to be present

Asynchronous reset



- Resets state whenever !RST is low
- Clock edges do not matter

Asynchronous reset (cont'd)



- NAND gate either sets output to 1 (if !RST low) or inverts A

!RST	A	!(RST*A)
0	0	1
0	1	1
1	0	1
1	1	0

From H. Kaeslin, "Digital Integrated Circuit Design," Cambridge Univ. Press, 2008

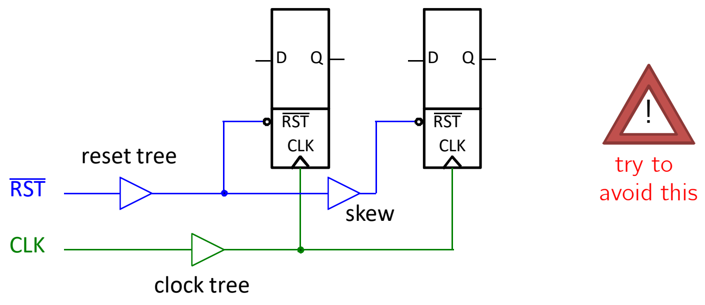
1002

Asynchronous reset (cont'd)

- Asynchronous reset advantages
  - Reset has priority over any other signal
  - Reset happens without clock present
  - Data paths are always clear of reset signals
  - Synthesis tools understand what is going on
- Asynchronous reset disadvantages
  - Reset de-assertion (=release) must occur within the same clock cycle for all flip-flops
  - Reset line is sensitive to glitches at any time

1003

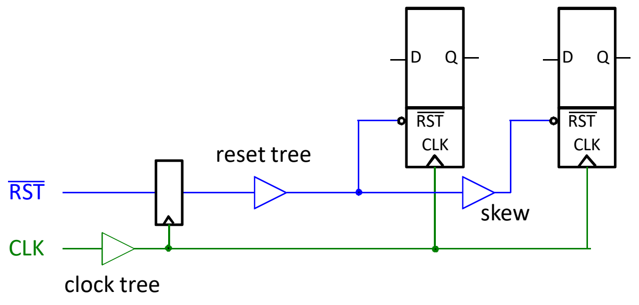
Fully asynchronous reset



- One must be careful that reset de-assertion (!RST from 0 to 1) happens in same cycle
- This approach should be avoided

1004

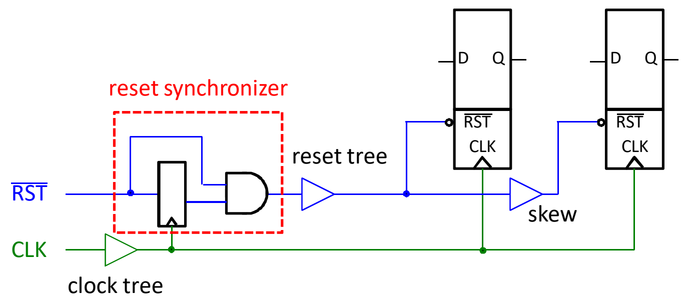
Fully synchronized async. reset



- Reset and de-assertion happens in synchronous way (w.r.t. the clock signal)
- Can use tools to generate reset tree

1005

Async. reset, synch. de-assertion

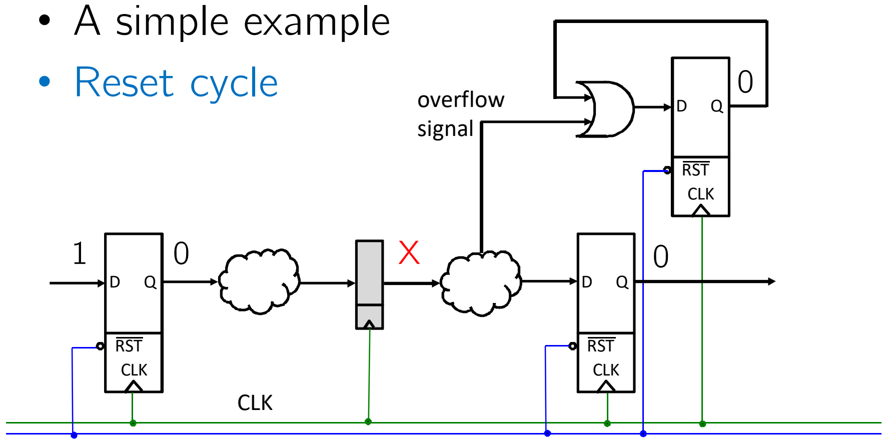


- Reset is applied in fully asynchronous way
- De-assertion in synchronous way
- Can use tools to generate reset tree

1006

Include reset in ALL flip-flops*

- A simple example
- Reset cycle

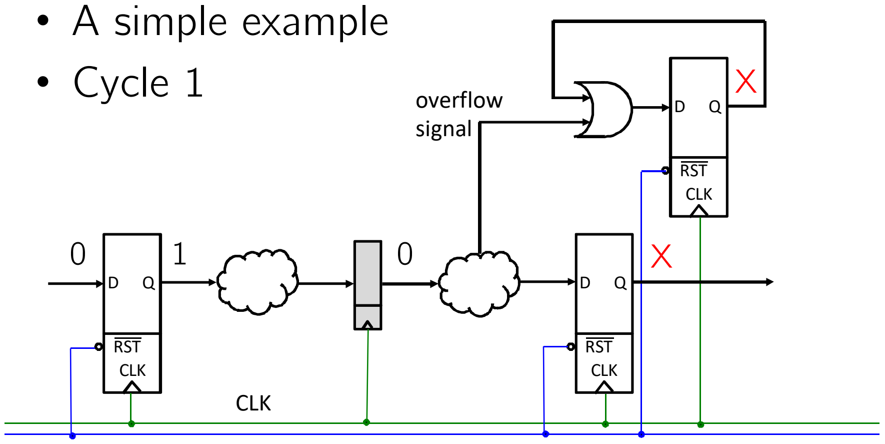


*unless you are a pro; but even then, think thrice (and also think about this slide)

1007

Include reset in ALL flip-flops*

- A simple example
- Cycle 1

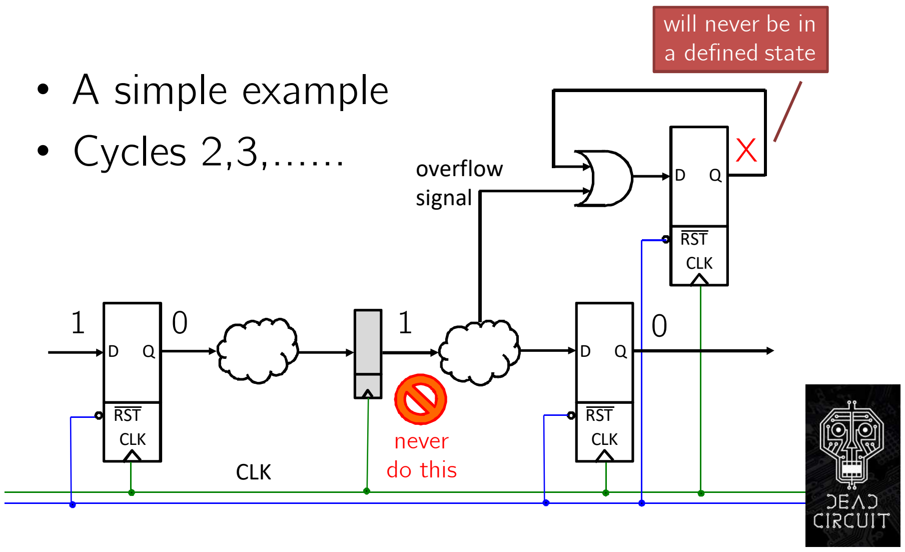


1008

*unless you are a pro; but even then, think thrice (and also think about this slide)

Include reset in ALL flip-flops*

- A simple example
- Cycles 2,3,……



1009

*unless you are a pro; but even then, think thrice (and also think about this slide)

Things to remember

- Glitches must be avoided on reset signal
- No logic on reset signal allowed
- Never use reset to implement functionality
- Careful with timing on reset signal
- Distribute reset signal by fanout tree
- All flip-flops should have a reset input
  - Simplifies design for test
  - Avoids unknown states that remain forever

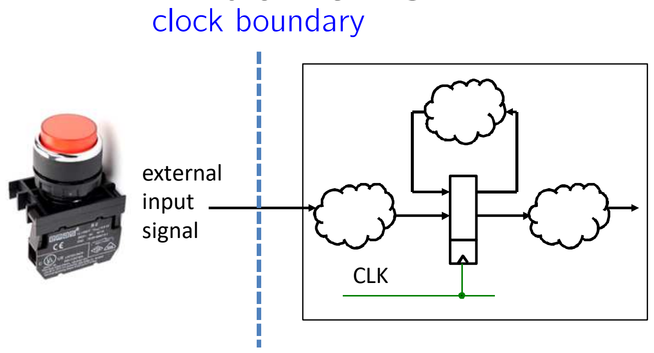
1010

Acquisition of asynchronous data

Synchronization

1011

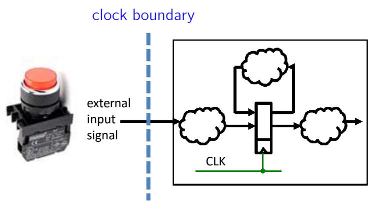
Signals between two clock domains

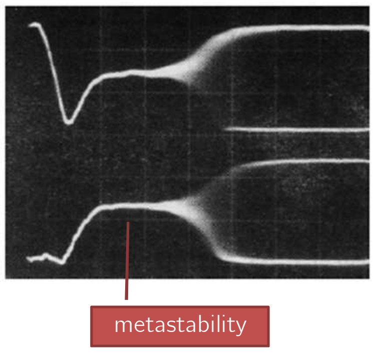


- External input signal is not synchronized to the CLK signal within the RHS circuit
- Can also be signal from another clock domain!

1012

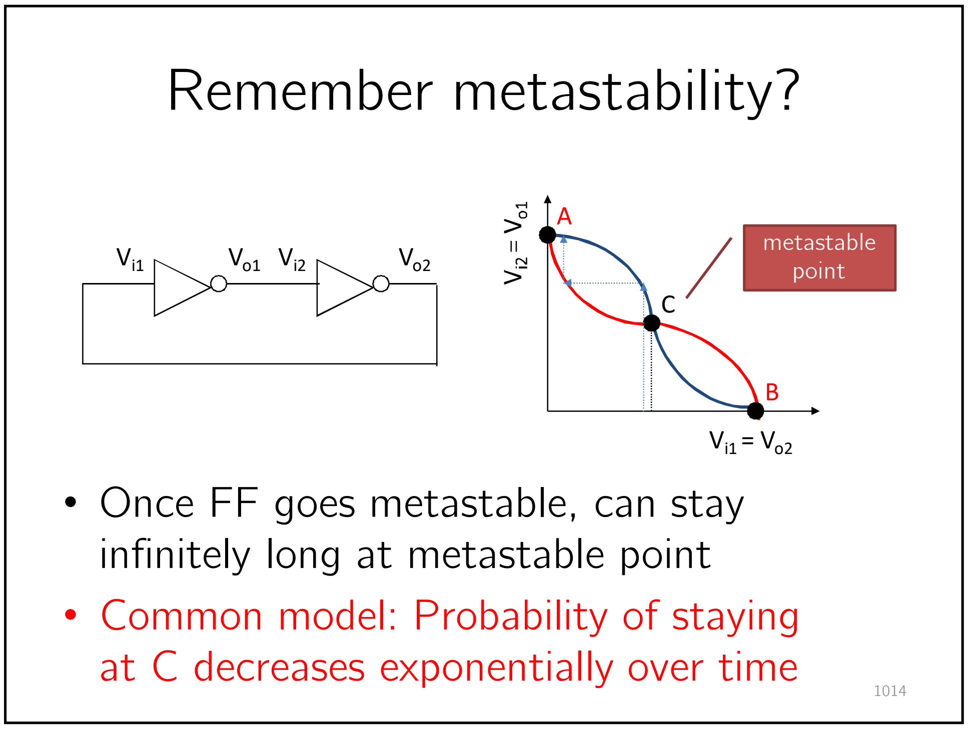
Signal may not be sampled correctly



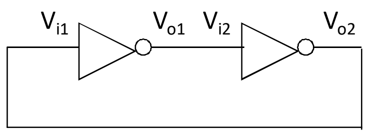


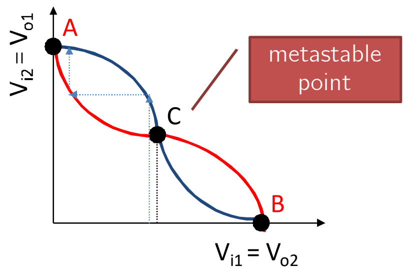
- Signal may violate setup/hold timing
- Signal may remain metastable for long time

1013



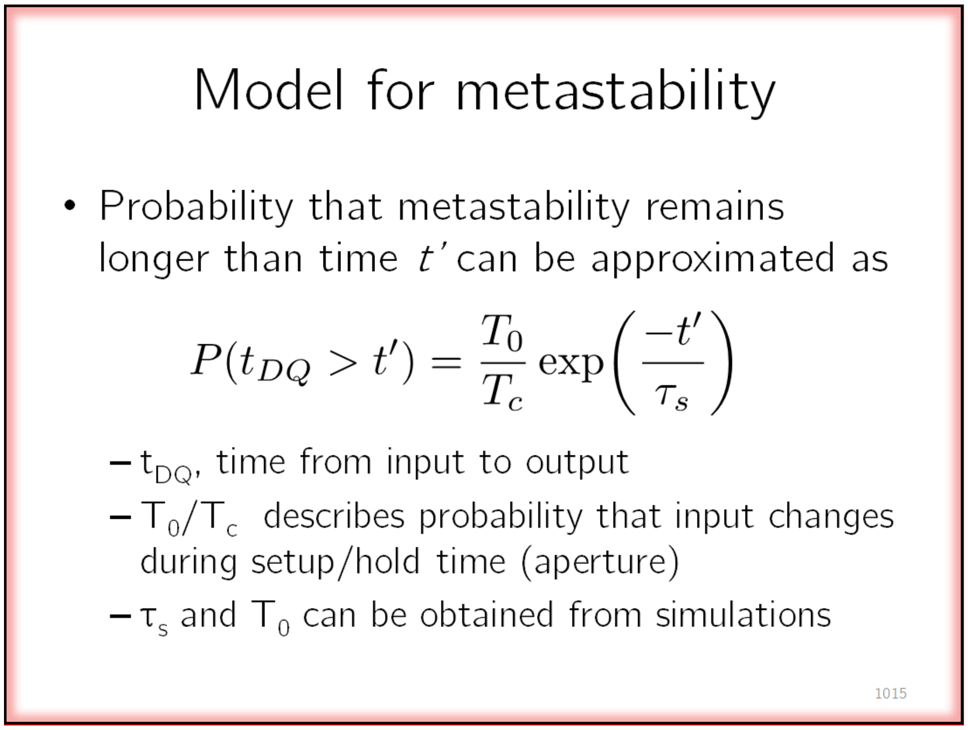
Remember metastability?





- Once FF goes metastable, can stay infinitely long at metastable point
- Common model: Probability of staying at C decreases exponentially over time

1014



Model for metastability

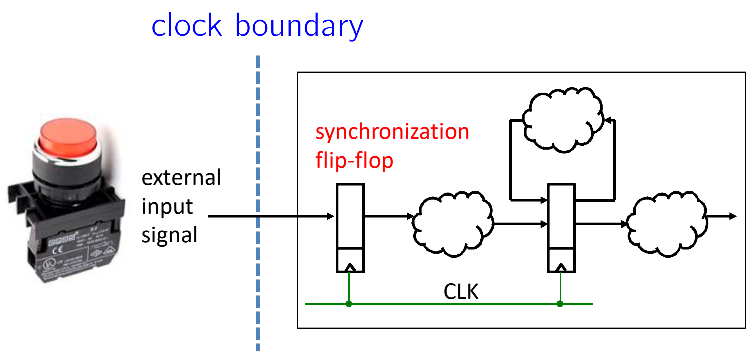
- Probability that metastability remains longer than time t' can be approximated as

$$P(t_{DQ} > t') = \frac{T_0}{T_c} \exp\left(\frac{-t'}{\tau_s}\right)$$

  - t_{DQ}, time from input to output
  - T_0/T_c describes probability that input changes during setup/hold time (aperture)
  - τ_s and T_0 can be obtained from simulations

1015

The "solution": Synchronizer FF



- Synchronization flip-flop reduces likelihood of metastability but does not solve the problem

1016

Probability of synchronizer failure

$$P(\text{failure}) = N \frac{T_0}{T_c} \exp\left(\frac{-(T_c - t_{\text{setup}})}{\tau_s}\right)$$

- Models probability of failure per second
  - *N* average number of asynchronous input changes per second
  - T_c = clock period
  - T_{setup} = setup time

1017

Mean-time between failure (MTBF)

$$MTBF = \frac{1}{P(\text{failure})} = \frac{T_c}{NT_0}\exp\left(\frac{T_c - t_{\text{setup}}}{\tau_s}\right)$$

- MTBF is a design parameter
- Set to 1e19 seconds (lifetime of universe)
- MTBF can be increased by
  - Fewer switching events
  - Longer period, smaller setup time

1018

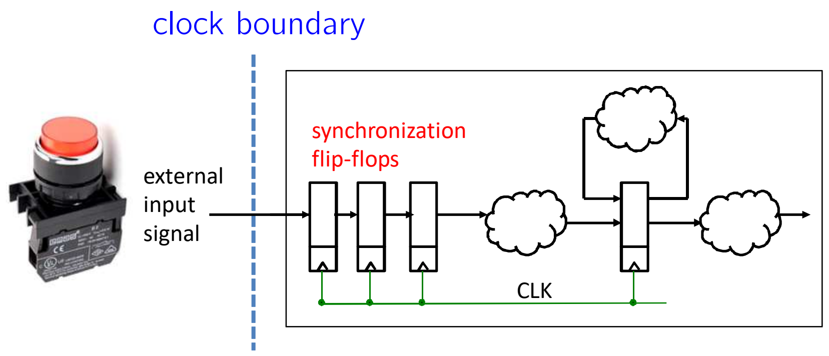
Example: single synchronizer

$$MTBF = \frac{1}{P(\text{failure})} = \frac{T_c}{NT_0}\exp\left(\frac{T_c - t_{\text{setup}}}{\tau_s}\right)$$

- Example parameters in 0.25um process:
  - τ_s = 20 ps
  - T_0 = 15 ps
  - N = 50 MHz
  - T_c = 0.5ns (ignore setup time)
- What is the MTBF?

1019

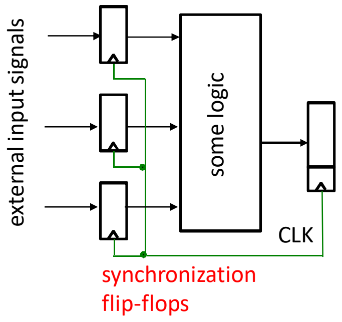
Solution: Multiple synchronizer FFs



- Cascading multiple synchronization flip-flops reduces probability of failure (increases MTBF)

1020

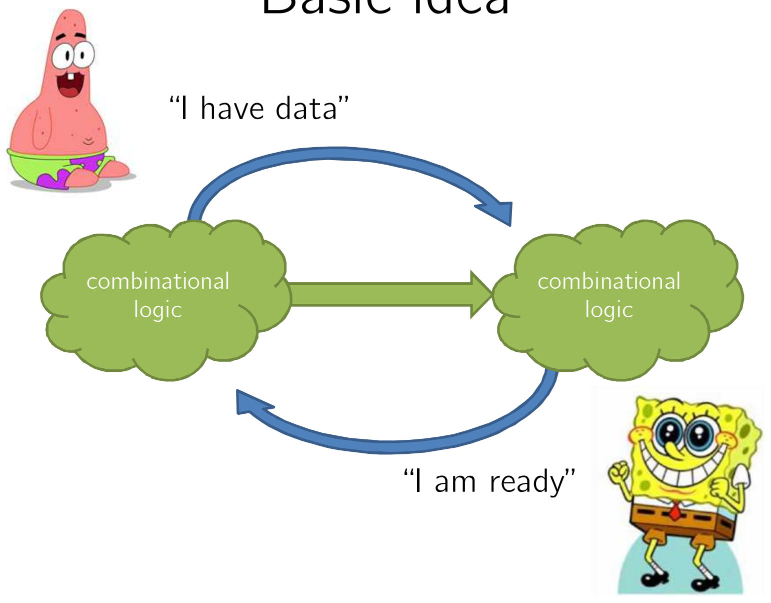
How about multiple parallel signals?



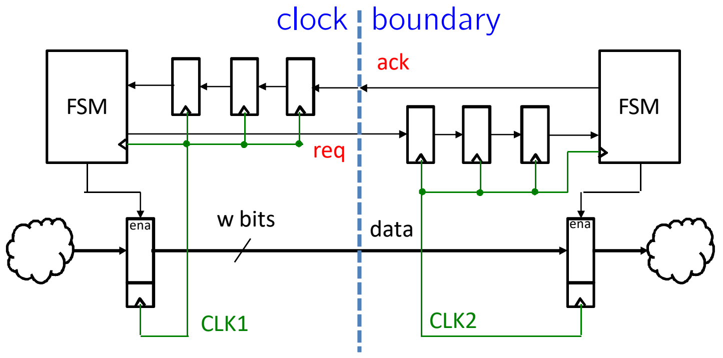
- Problem: "fast bits" and "slow bits" may be sampled in different periods
- Multiple sequential sync. FFs do not help!

1021

Basic idea



Solution: Handshaking protocols



- Two or four phase protocols possible
- Synchronizer FFs only on req and ack signals