EITF35: Introduction to Structured VLSI Design

Part 4.2.1: Learn More …

Liang Liu
liang.liu@eit.lth.se

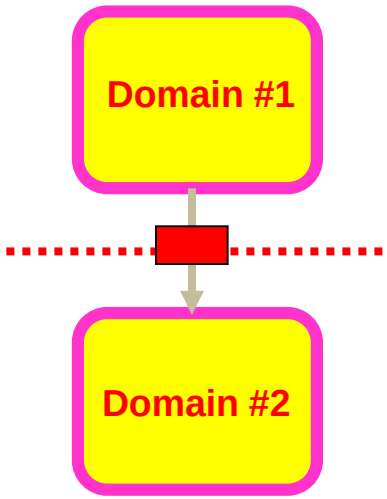
Outline

- Crossing clock domain
- Reset, synchronous or asynchronous?

Why two DFFs?

Crossing clock domain

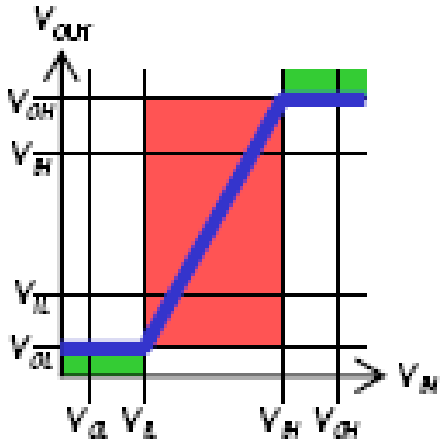
- **Multiple clock is needed in case:**
  - Inherent system requirement
    - ***Different clocks for sampling and processing***
  - Chip size limitation
    - ***Clock skew increases with the # FFs in a system***



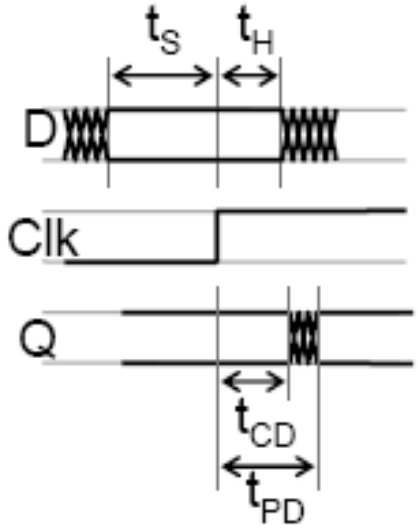
Multiple Clocks: Problems

- We have been setting very strict rules to make our digital circuits safe: using a forbidden zone in both voltage and time dimensions

Digital Values: distinguishing voltages representing "1" from "0"

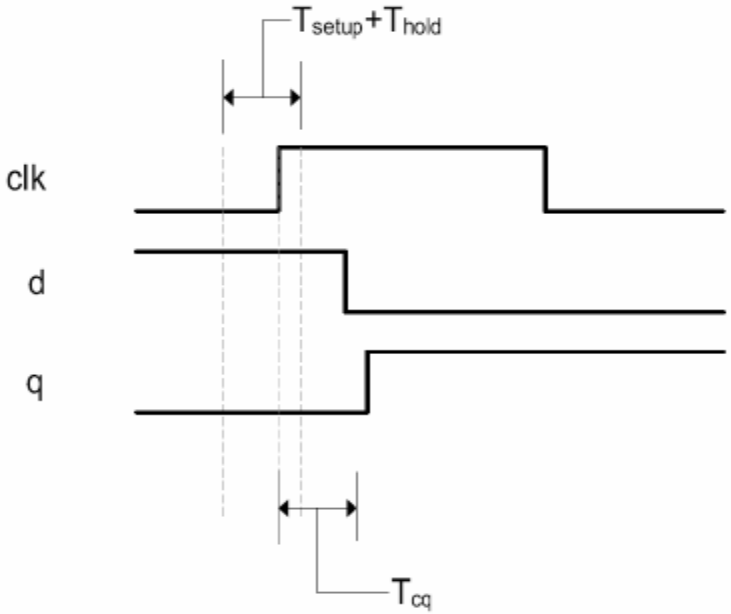


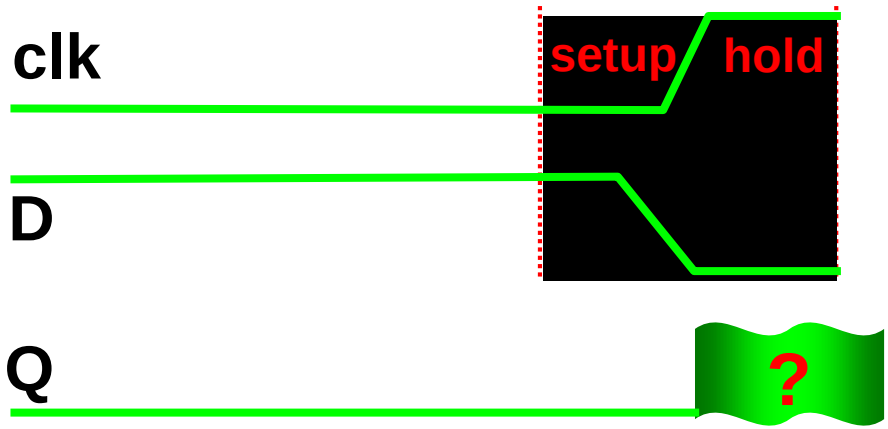
Digital Time: setup and hold time rules



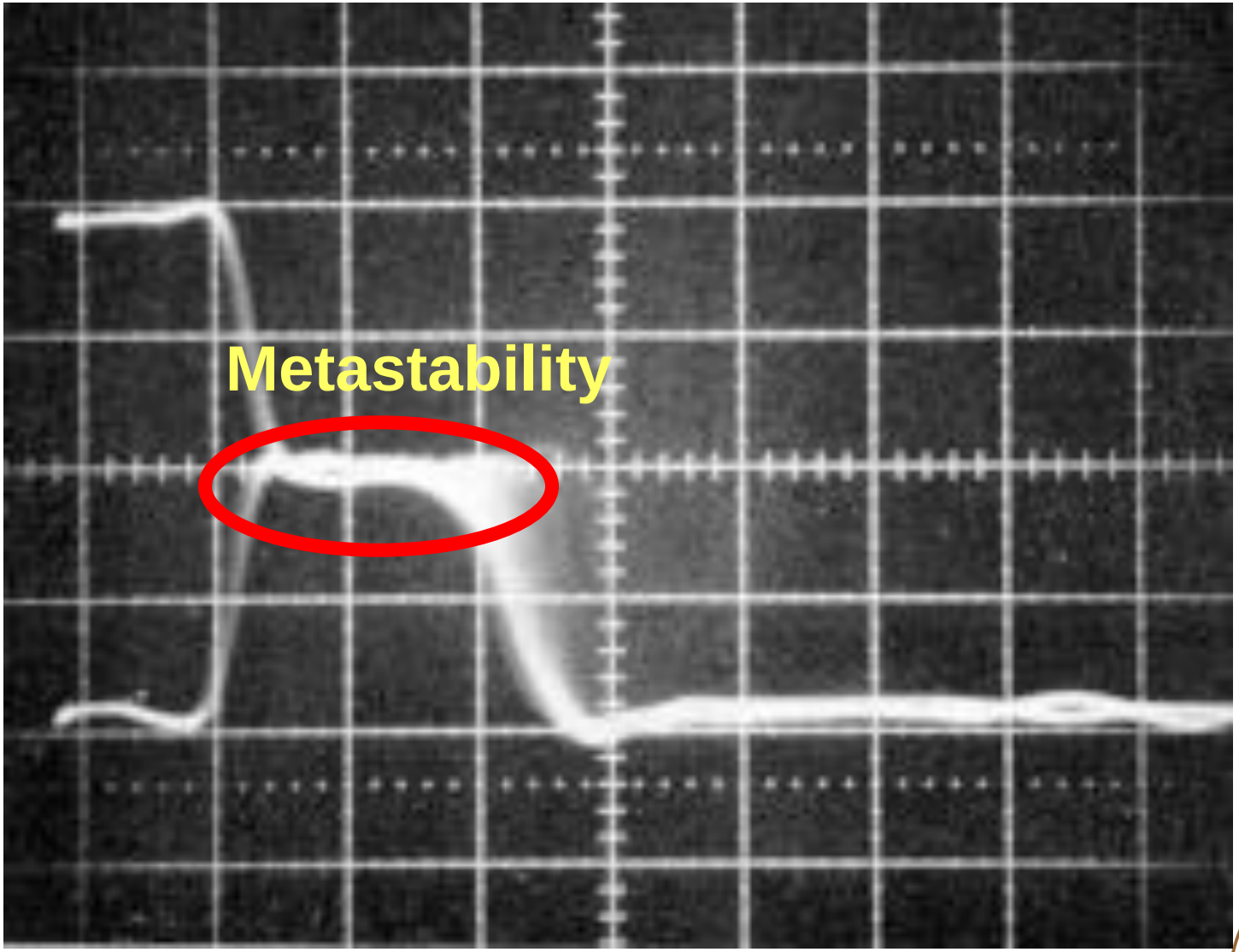
Metastability

- With asynchronous inputs, we have to break the rules: ***we cannot guarantee that setup and hold time requirements are met at the inputs!***
- What happens after timing violation?

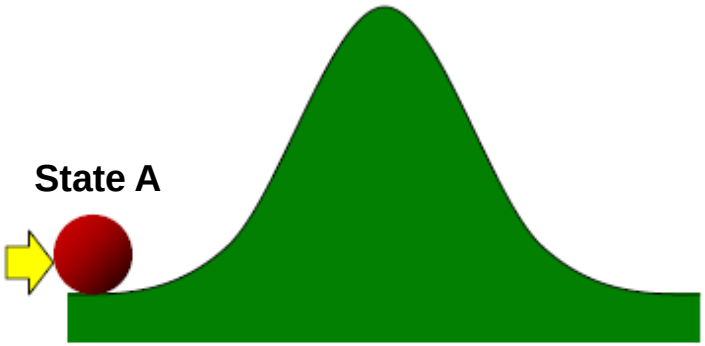




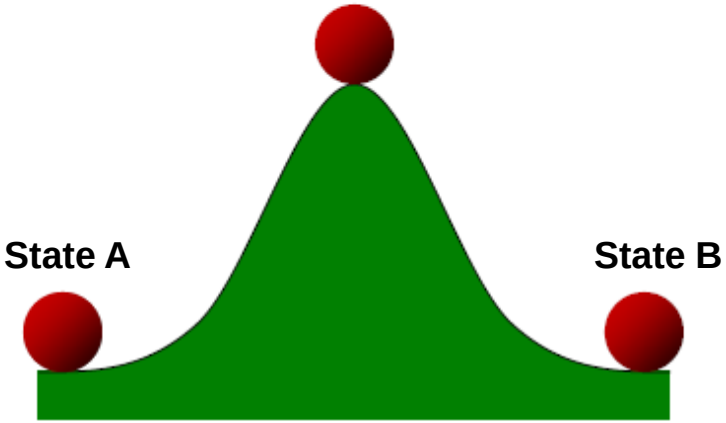
Metastability in Digital Logic



Mechanical Metastability



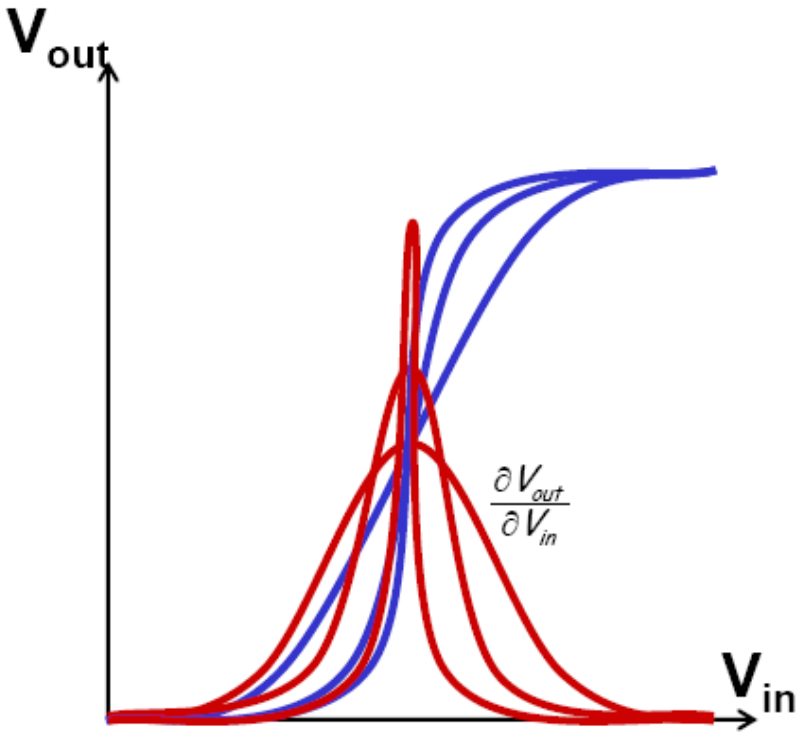
- **Launch a golf up a hill, 3 possible outcomes:**
  - Hit lightly: Rolls back
  - Hit hard: Goes over
  - Or: Stalls at the apex



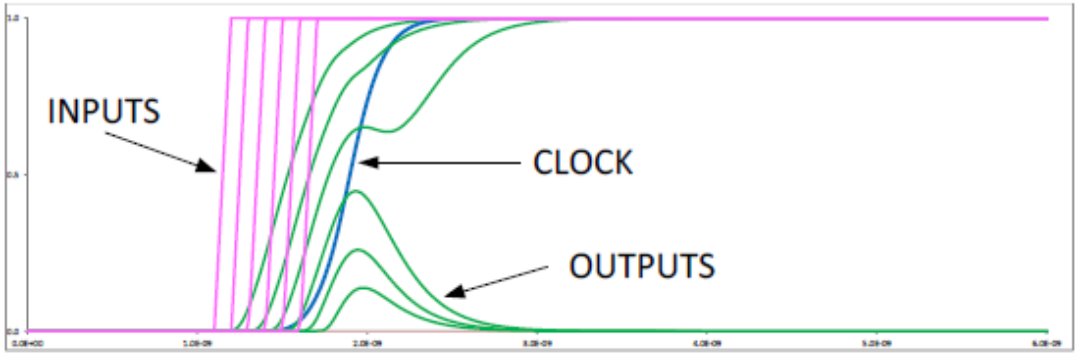
- **That last outcome is not stable:**
  - A gust of wind
  - Brownian motion
  - ***Can you tell the eventual state?***

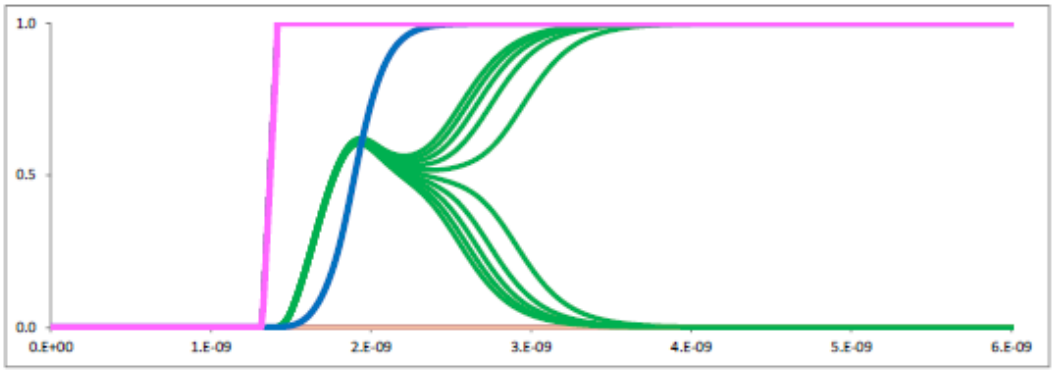
Metastability in Digital Logic

- **Our hill is related to the VTC (Voltage Transfer Curve).**
  - The higher the gain thru the transition region
  - The steeper the peak of the hill
  - The harder to get into a metastable state.
- **We can decrease the probability of getting into the metastable state, but we can't eliminate it…**



Metastability in Digital Logic



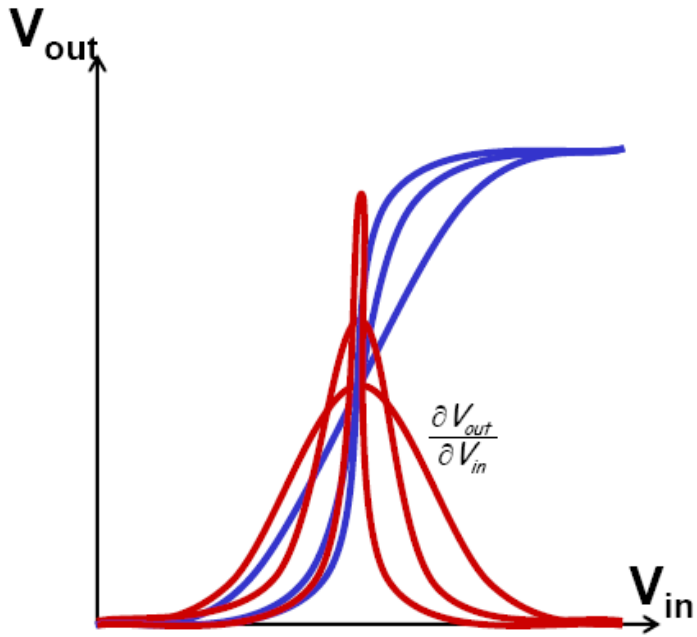


- Fixed clock edge
- Change the edge of inputs
- The input edge is moved in steps of 100ps and 1ps
- The behavior of outputs
  - 'Three' possible states
  - Will exit metastability

How long it takes to exit Metastability?

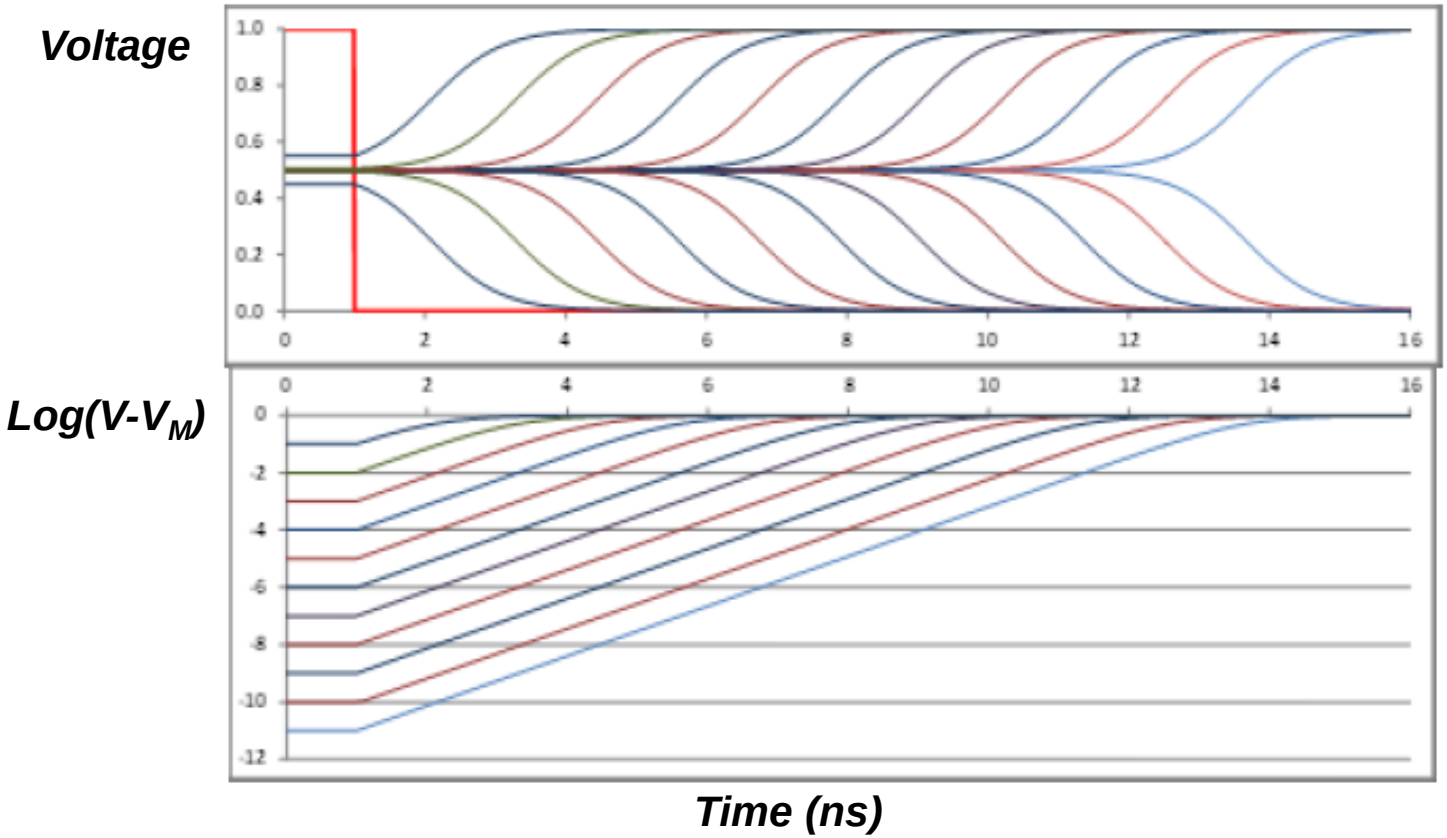
Exit Metastability

- Define a fixed-point voltage, V_M, (always have) such that $V_{IN} = V_M$ implies $V_{OUT} = V_M$
- Assume the device is sampling at some voltage V_0 near V_M
- The time to settle to a stable value depends on $(V_0 - V_M)$; its theoretically infinite for $V_0 = V_M$



Exit Metastability

- The time to exit metastability depends *logarithmically on* $(V_0 - V_M)$
- The *probability* of remaining metastable at time T is $e^{-T/\tau}$



MTBF: The probability of being metastable at time S?

- **Two conditions have to be met concurrently**
  - An FF enters the metastable state
  - An FF cannot resolve the metastable condition within S
- **The rate of failure** $p(failure) = p(enter\ MS) \times p(time\ to\ exit > S)$

$$Rate(failures) = T_W F_C F_D \times e^{-S/\tau}$$

  - T_W: time window around sampling edge incurring metastability
  - F_C: clock rate (assuming data change is uniformly distributed)
  - F_D: input change rate (input may not change every cycle)
- **Mean time between failures (MTBF)**

$$MTBF = \frac{e^{S/\tau}}{T_W F_C F_D}$$

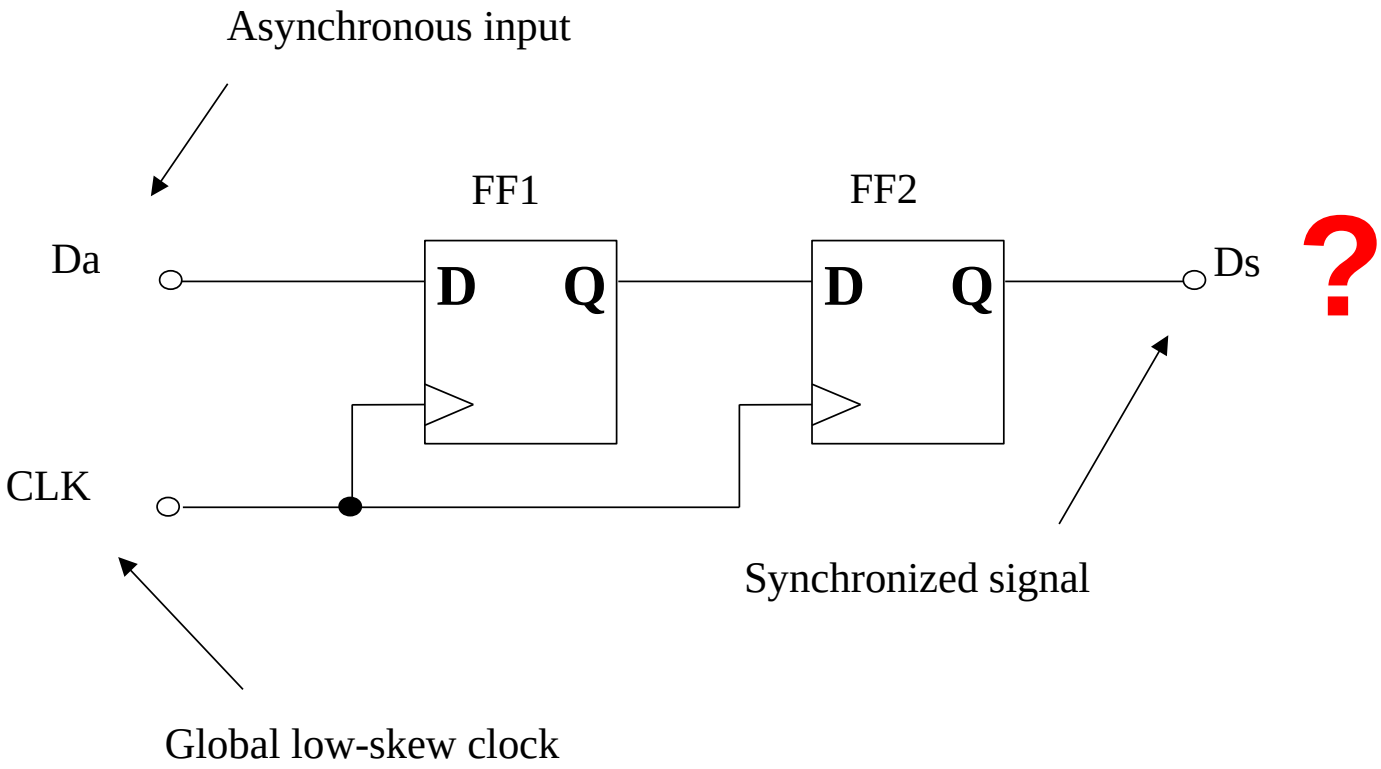
MTBF (Mean Time Between Failure)

- **Let's calculate an ASIC for 28nm CMOS process**
  - τ: 10ps (different FFs have different τ)
  - T_W=20ps, F_C=1GHz
  - Data changes every ten clock cycles
  - Allow 1 clock cycle to resolve metastability, **$S=T_C$**

$MTBF=4\times10^{29}$ year !

[For comparison:
Age of oldest hominid fossil: $5x10^6$ years
Age of earth: $5x10^9$ years]

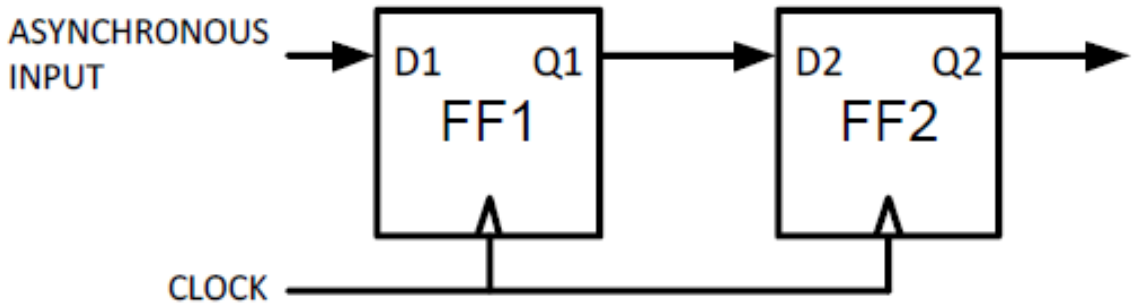
The Two-Flip-Flop Synchronizer

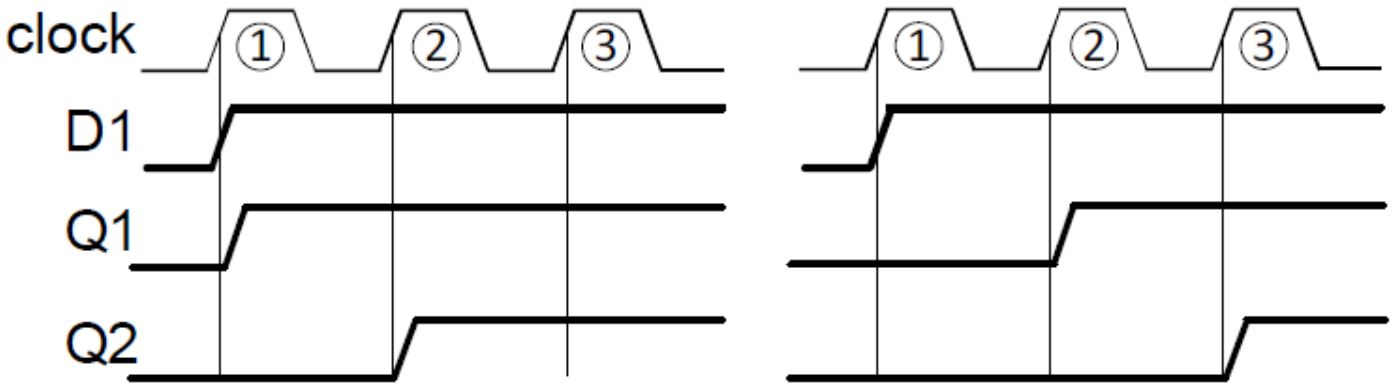


$$S = T_C$$

The Two-Flip-Flop Synchronizer

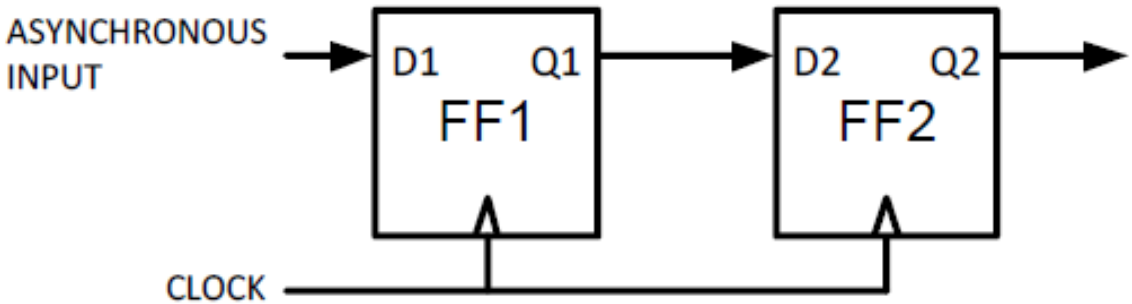
- **Possible Outcomes**

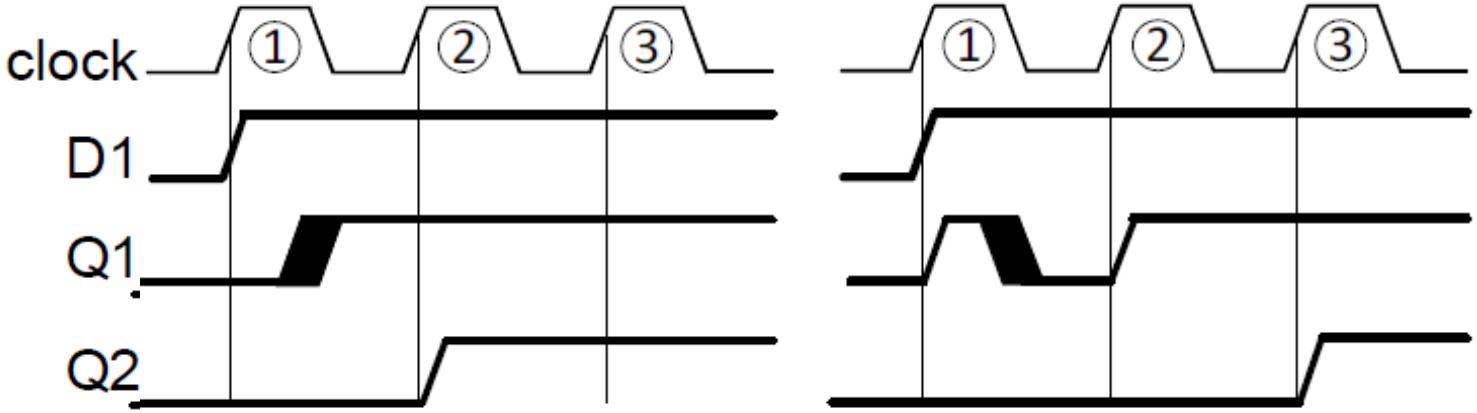




The Two-Flip-Flop Synchronizer

- **Possible Outcomes**



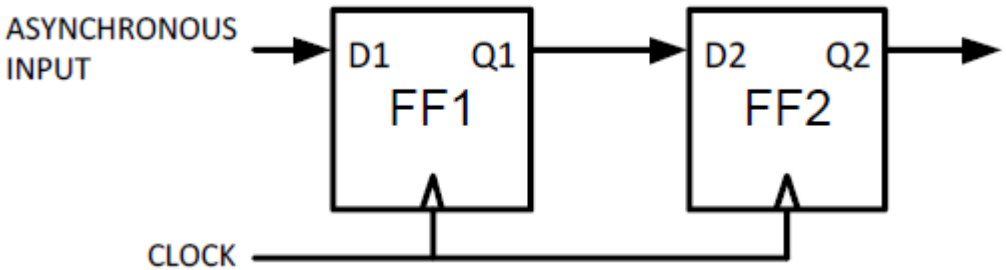


Open Question: What is the limitation?

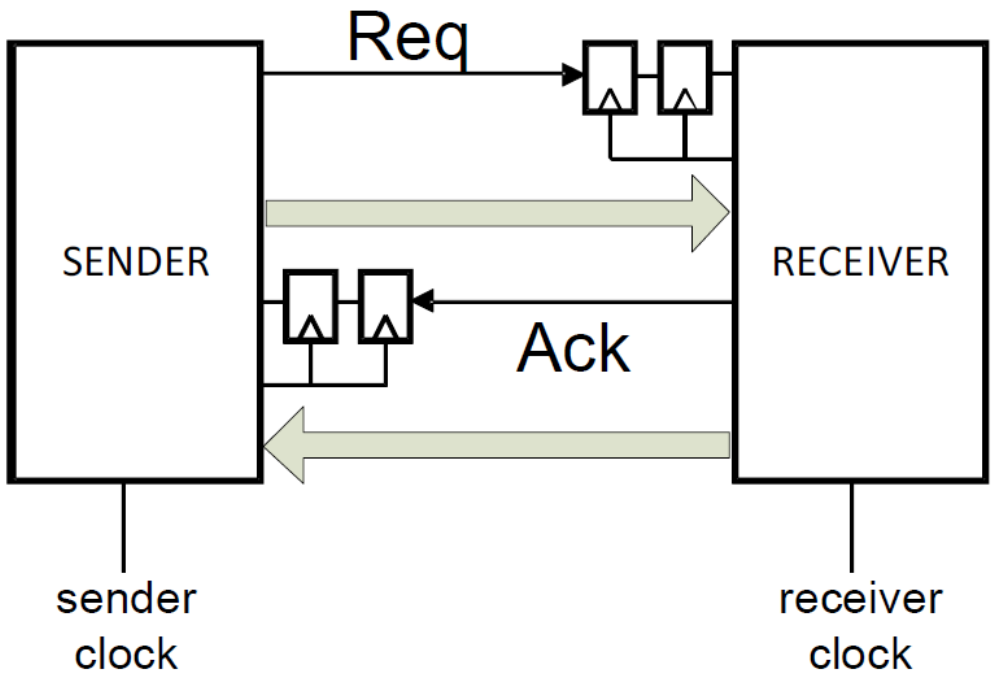
The Two-Flip-Flop Synchronizer

Problems

- Just ensures that the receiving system does not enter a metastable state
- Not guarantee the **"function"** of the received signal
  - ***Uncertainty Remains: Q2 goes high either one or two cycles later than the input***
- D1 mush stay high for at *least two cycles*.
- How about data bus (***multiple bits***) crossing clock domain?
  - ***Some bits may pass through the synchronizer after one cycle while others may take two cycles.***



A Complete Synchronizer

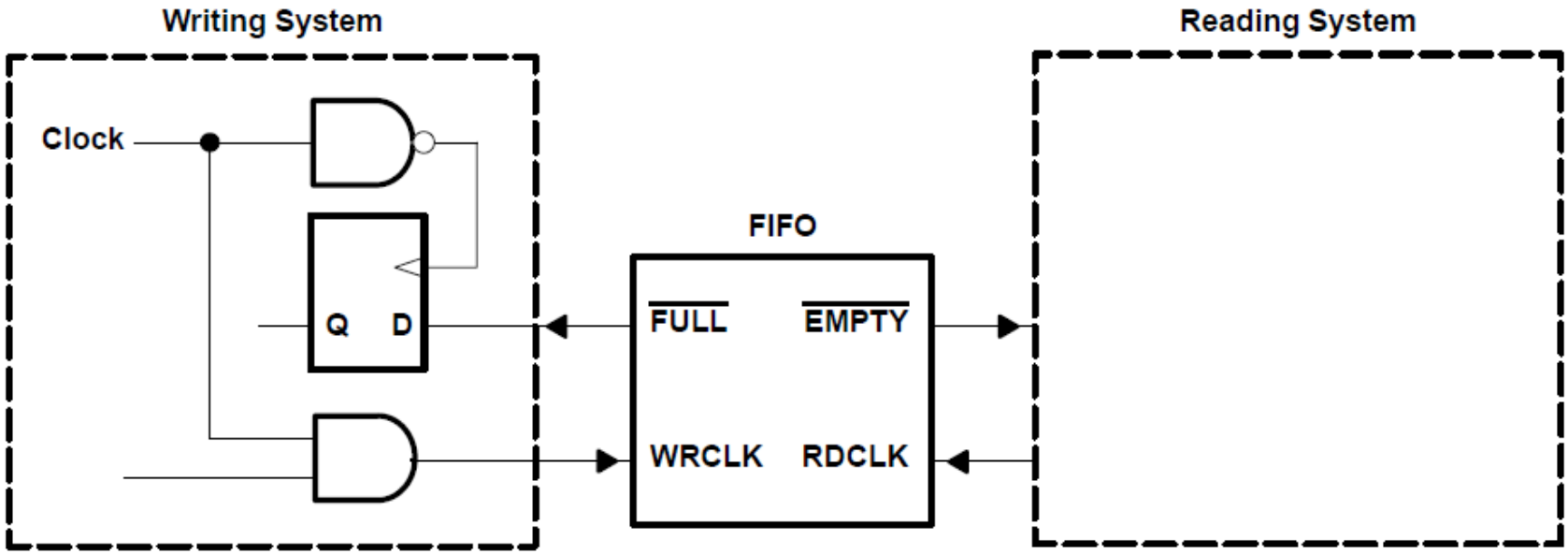


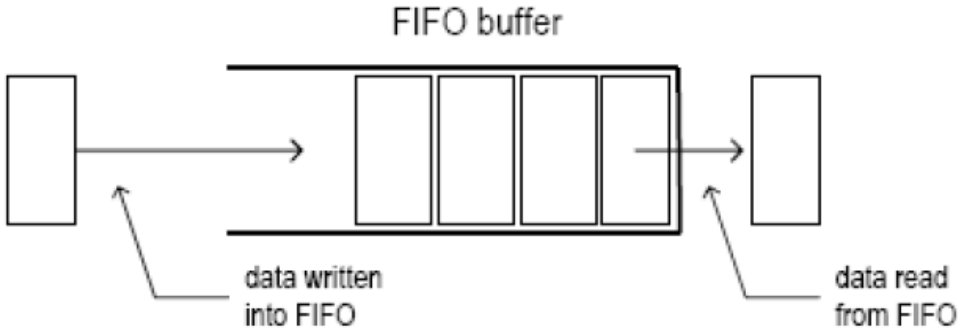
- The sender place data on the bus
- The sender sends Req, Req gets synchronized by the top synchronization circuits
- The receiver gets data and sends back ACK
- Ack gets synchronized by the sender, and only then is the sender allowed to start a new cycle again.

FIFO

FIFO (first in first out) Buffer

- "Elastic" storage between two subsystems



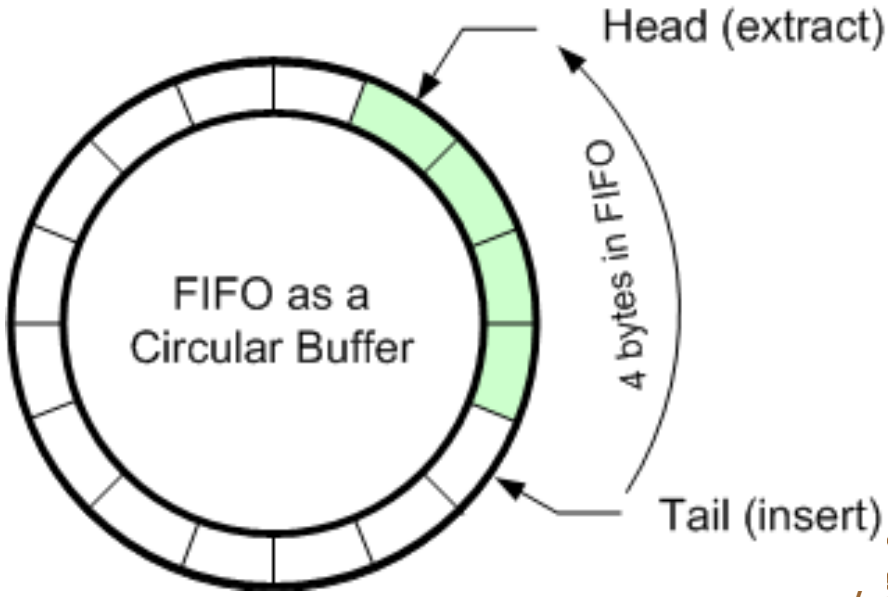


Circular FIFO

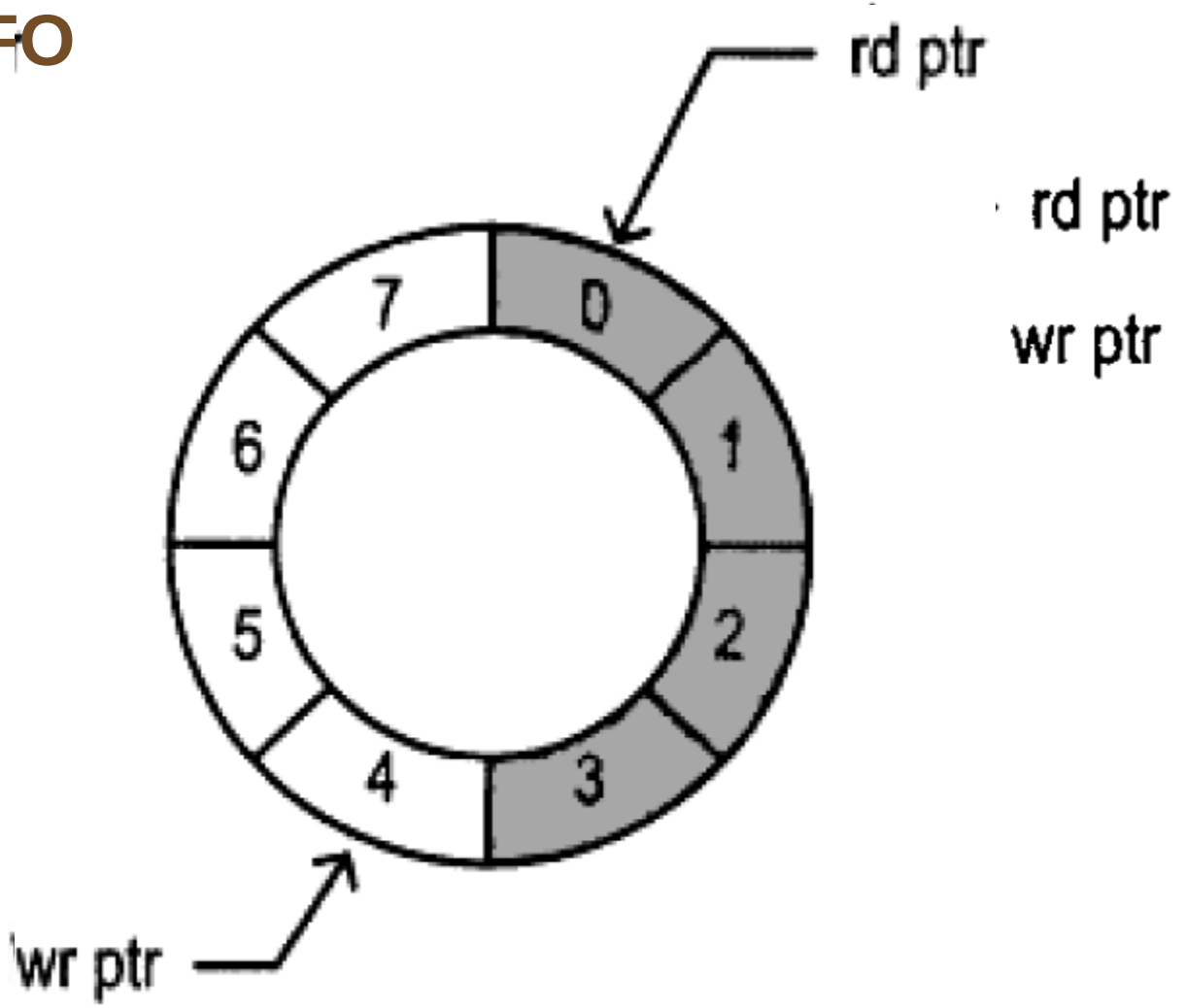
How to Implement a FIFO?

- Circular queue implementation
- Use two pointers and a “generic storage”
  - ***Write pointer*: point to the empty slot before the *head* of the queue**
  - ***Read pointer*: point to the *tail* of the queue**

"First in? First out!"



Circular FIFO

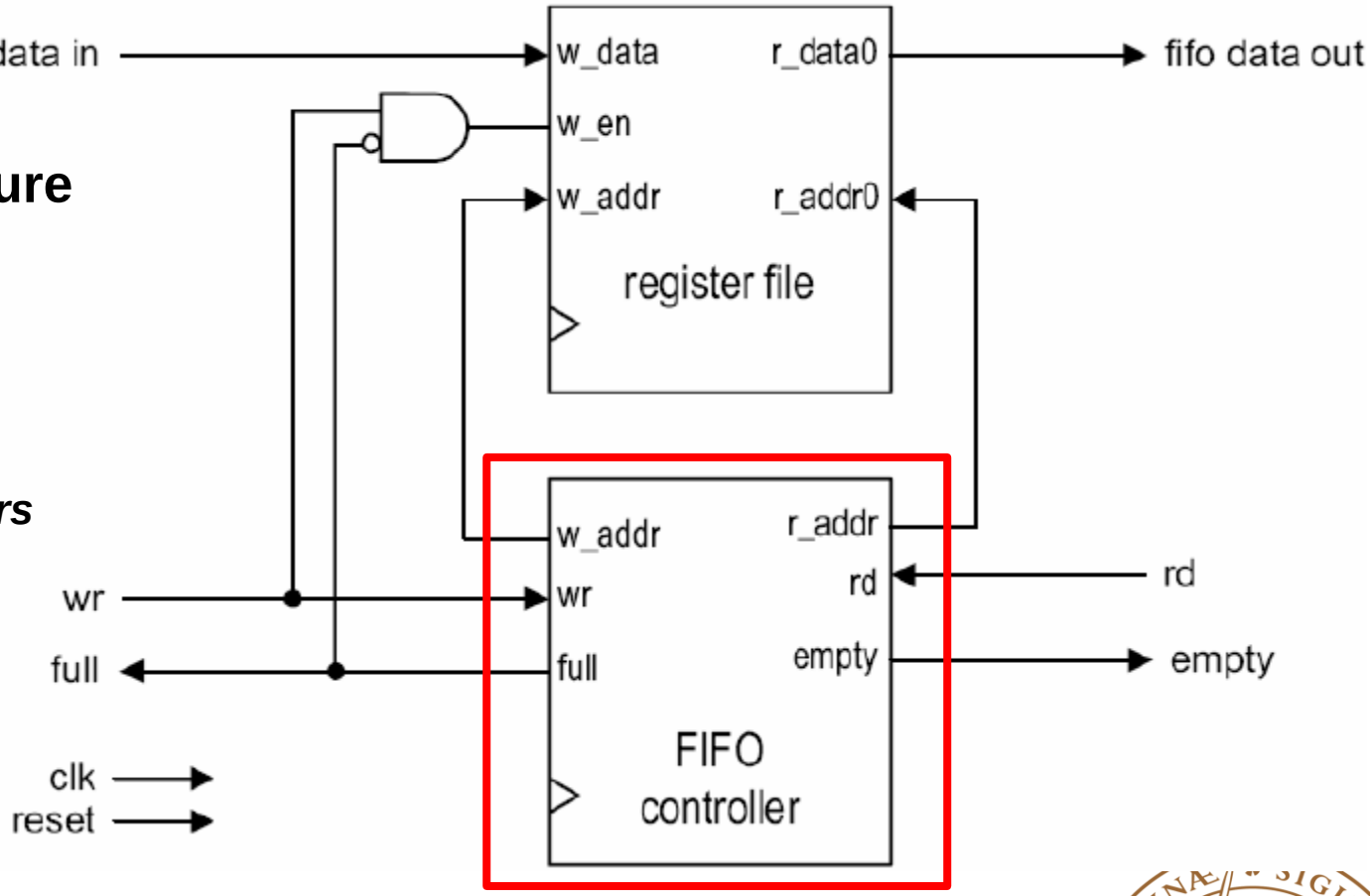


(c). 4 more writes

FIFO Implementation

- **Overall Architecture**
  - Storage Elements
    - ***Reg. file***
  - **FIFO Controller**
    - ***Read and write pointers: 2 counters***
    - ***Status circuit:***

      full, empty



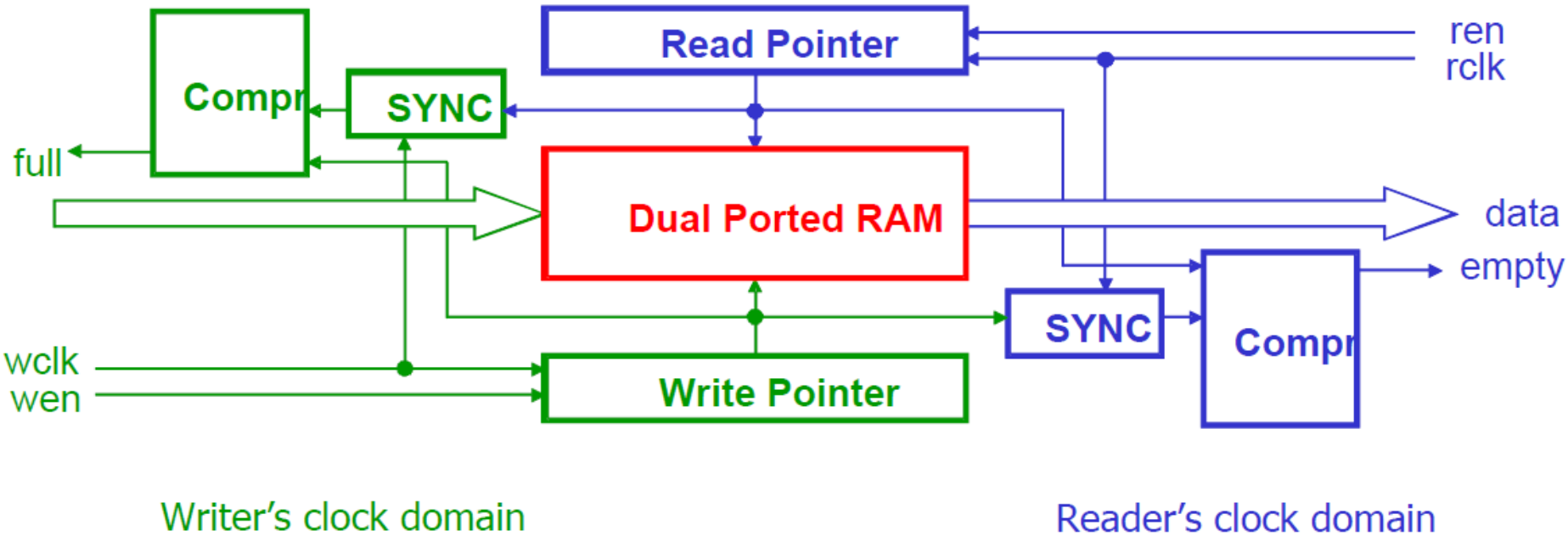
FIFO Implementation: Controller

- **Augmented binary counter:**
  - Increase the counter by 1 bits
  - Use LSBs for as register address
  - Use **MSB** to distinguish full or empty

Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

A Complete Synchronizer



- ***Control signals*** **are 'synchronized',** ***data*** **pass through storage elements**
- **Assertion delay, it takes** ***two clock*** **cycles for the control signal passing through synchronizer**
- **Usually available in libraries**
- **The key question,** ***how large the RAM should be?***
- **"When in doubt, double it"**

Outline

- Crossing clock domain

- **Reset, synchronous or asynchronous?**

Reset Design Strategy

- Force the SoC into a known state for stable operations
- In general, every flip-flop in an SoC (ASIC) should be resetable whether or not it is required by the system
- Reset might be eliminated for high-performance pipeline FFs
- Many design issues must be considered before choosing a reset strategy for an ASIC design

Synchronous or Asynchronous?

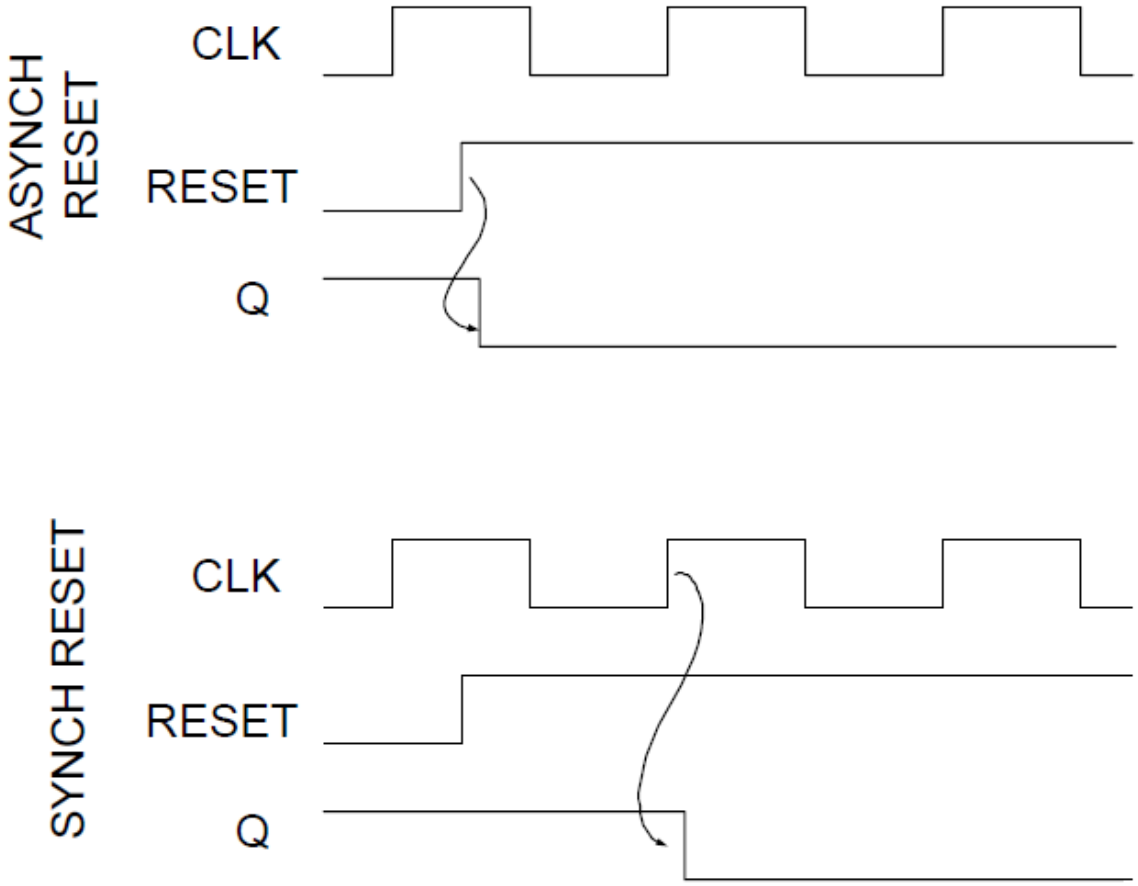
General Coding Style: FFs

- Coding for synchronous and asynchronous reset

```
architecture rtl of goodFFstyle is
   signal q1 : std_logic;
begin
   process (clk, rst_n)
   begin
      if (clk'event and clk = '1') then
        if (rst_n = '0') then
          q1 <= '0';
        else
          q1 <= d;
        end if;
      end if;
    end process;
end rtl;
```

```
architecture rtl of goodFFstyle is
   signal q1 : std_logic;
begin
  process (clk)
  begin
      if (clk'event and clk = '1') then
        if (rst_n = '0') then
          q1 <= '0';
        else
          q1 <= d;
        end if;
      end if;
    end process;
end rtl;
```

Synchronous vs. Asynchronous Reset



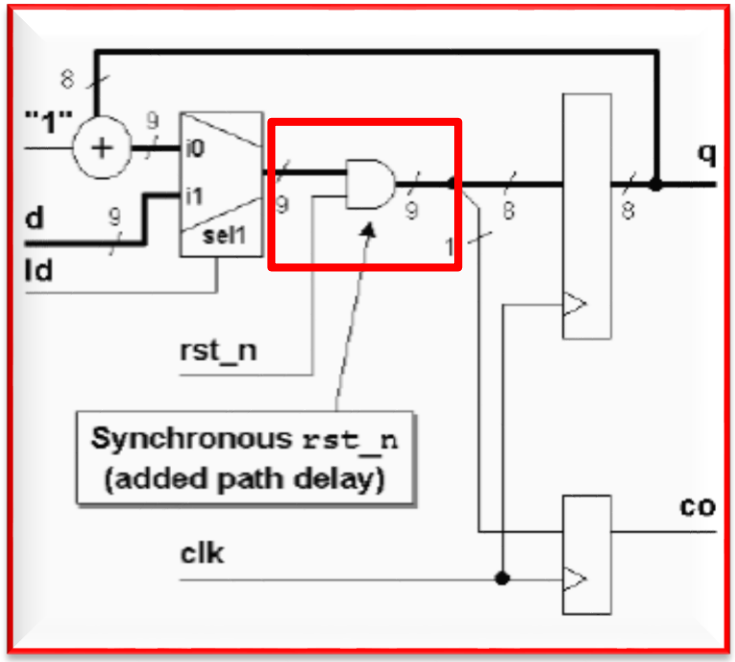
Synchronous vs. Asynchronous Reset

- 80% designs using synchronous reset (investigation by *Sunburst Design, Inc*)
- "we all know that the best way to do resets in an ASIC is to strictly use synchronous resets"
- "asynchronous resets are bad and should be avoided"

- There are *both advantages and disadvantages* to using either synchronous or asynchronous resets.
- The designer must use an approach that is appropriate for the design.

Synchronous Reset

reset is not part of the sensitivity list.

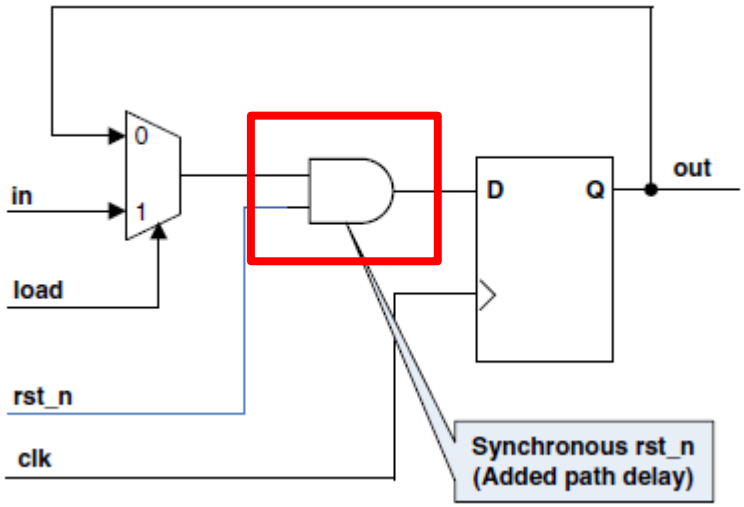


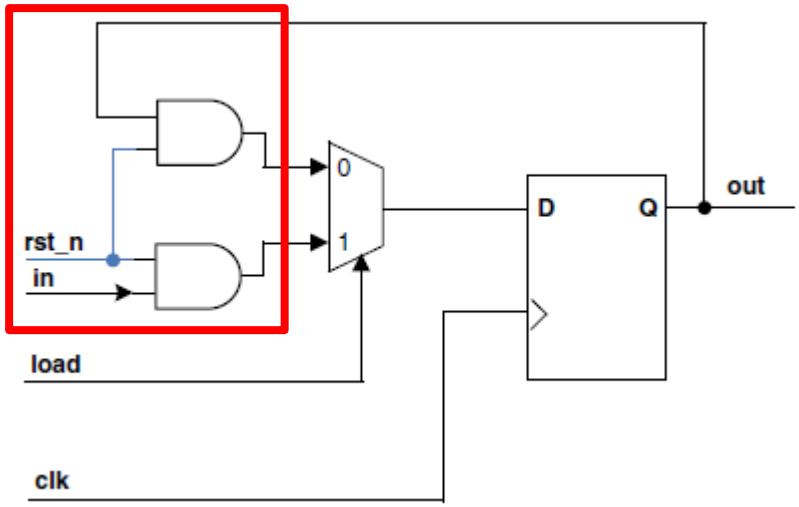

```
architecture rtl of ctr8sr is
  signal count : std_logic_vector(8 downto 0);
begin
  co <= count(8);
  q  <= count(7 downto 0);

  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (rst_n = '0') then
        count <= (others => '0');        -- sync reset
      elsif (ld = '1') then
        count <= '0' & d;                -- sync load
      else
        count <= count + 1;              -- sync increment
      end if;
    end if;
  end process;
end rtl;
```

reset is part of the input path

Synchronous Reset: one problem

- Synthesis tool may not easily distinguish the reset signal from any other data signal
- The synthesis tool could alternatively have produced the circuit
- If synthesis tool can distinguish reset, it will put reset as close to FFs as possible





What … If 'load=X'

Only a problem in simulation

Synchronous Reset

Advantage

- Generally insure that the ***circuit is 100% synchronous***.
- Will synthesize to ***smaller flip-flops***
- Ensure that reset can only occur at an active clock edge. The clock works as a filter for ***small reset glitches***.

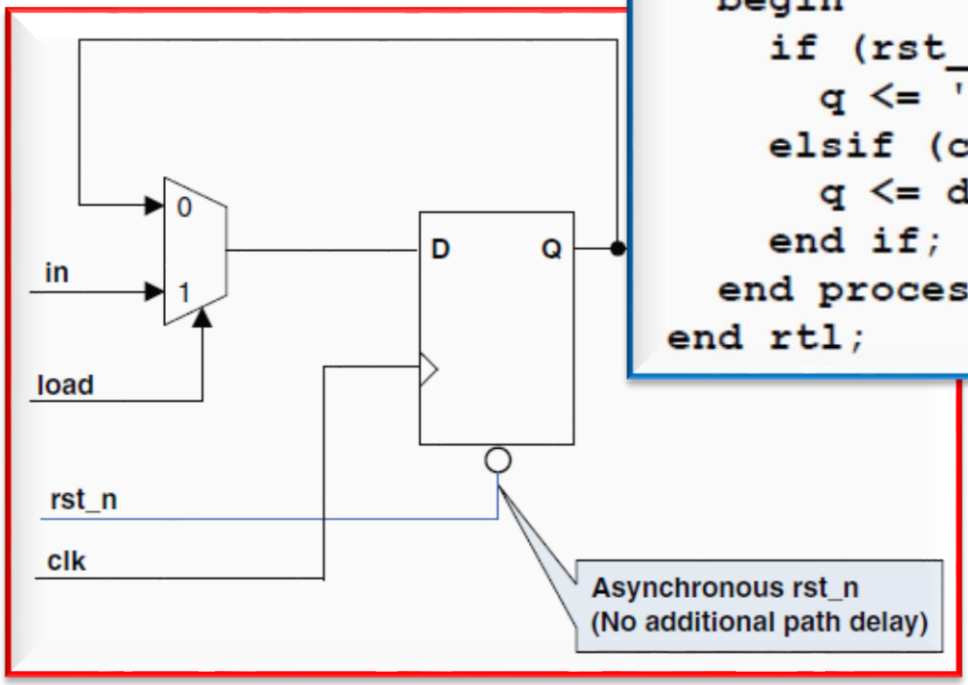
Disadvantage

- May need a pulse stretcher to guarantee a reset ***pulse width*** wide enough to ensure reset is present during an active edge of the clock
- Will require a clock in order to reset the circuit, e.g., if you have a gated clock to save power, the ***clock may be disabled***

Asynchronous Reset

reset is part of the sensitivity list.

```
architecture rtl of asyncresetFFstyle is
begin
  process (clk, rst_n)
  begin
    if (rst_n = '0') then
      q <= '0';
    elsif (clk'event and clk = '1') then
      q <= d;
    end if;
  end process;
end rtl;
```



reset is not part of the input path

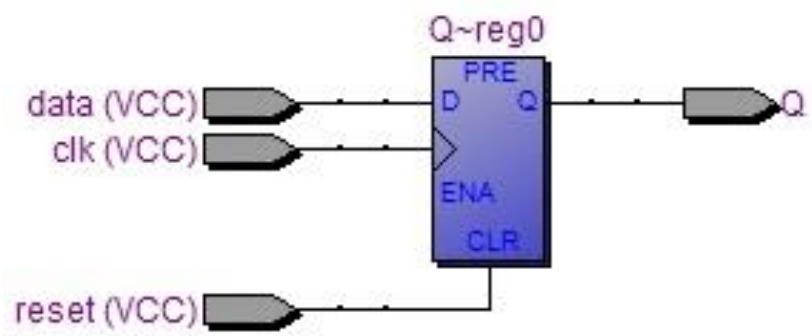
Asynchronous Reset

Advantage

- The ***data path*** is guaranteed to be ***clean***
  - designs that are pushing the limit for data path timing, cannot afford to have added gates and additional net delays in the data path due to synchronous resets
- The most obvious advantage favoring asynchronous resets is that the circuit can be reset with or ***without a clock*** present

Disadvantage

- If the asynchronous reset is ***released at or near the active clock edge*** of a flip-flop, the output of the flip-flop could go ***metastable*** and thus the reset state of the SoC could be lost.
- ***Spurious resets*** due to noise or glitches on the board or system reset

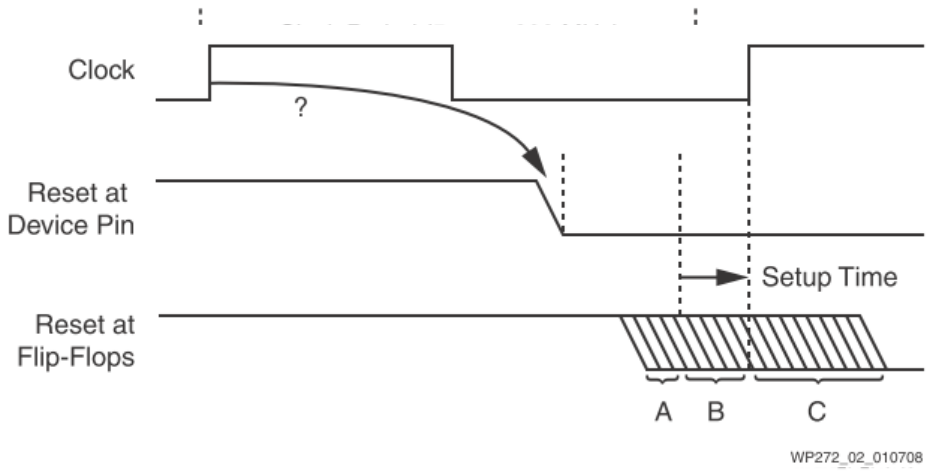


Reset Timing

Reset Removal Problem

Reset recovery time:

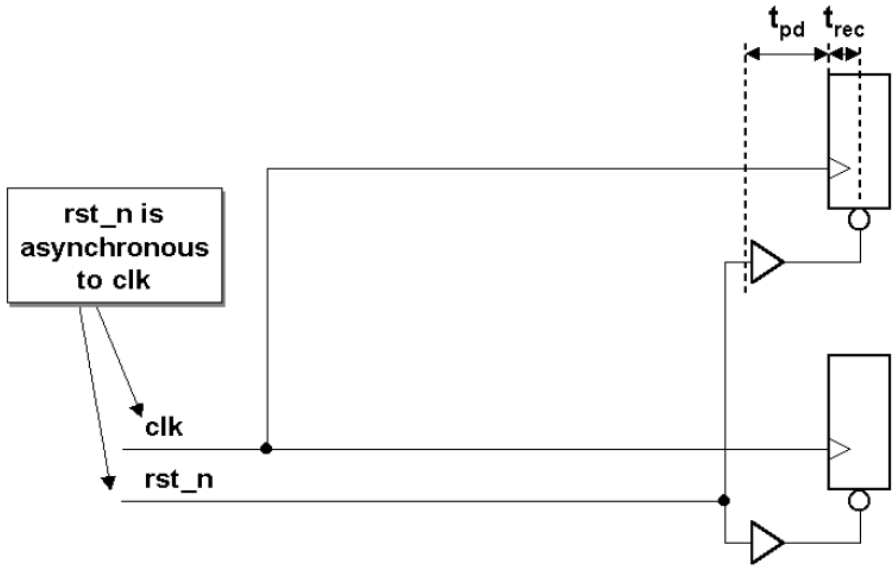
- Time between when ***reset is de-asserted*** and the time that the ***clock signal goes high*** again.
- Synchronous reset, ***both the leading and trailing edges*** of the reset must be away from the active edge of the clock.



Active high reset

Reset Removal Problem

- **Reset removal traversing different clock cycles**
  - Cause some registers or flip-flops to exit the reset state ***before others***
  - Reset Buffer Tree



Reset Removal Problem

- **Reset removal traversing different clock cycles**
  - Cause some registers or flip-flops to exit the reset state ***before others***
  - Especially for high-rate clock and big chip
  - Reset Buffer Tree

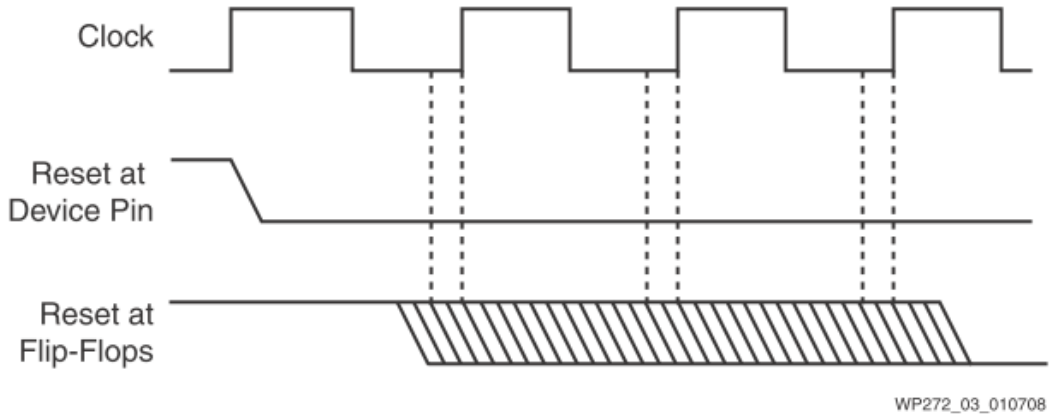


Figure 3: **Reset Timing Diagram - High Clock Rate**

Active high reset

But… Does it really matter?

Single pipeline stage?

- After a few cycles, the entire pipeline will be operational
- Any incorrect data will be flushed out of the system
- In fact, there is little point in having a reset at all

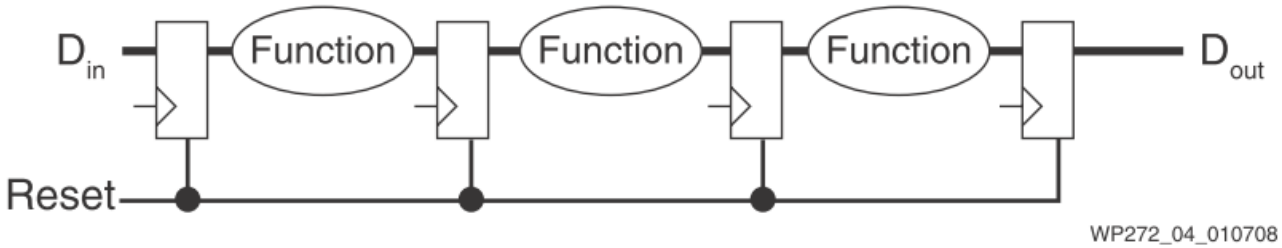
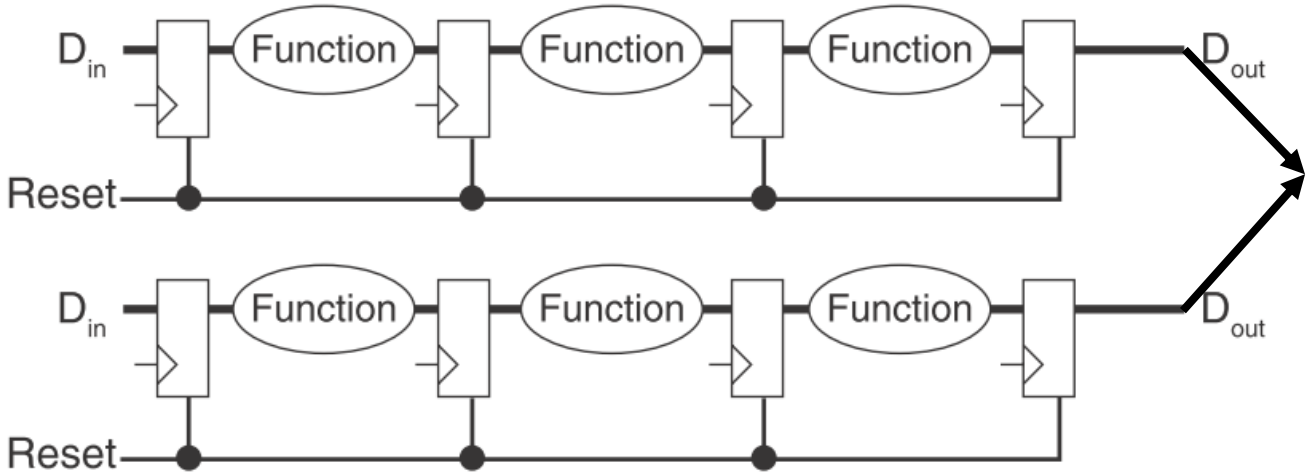


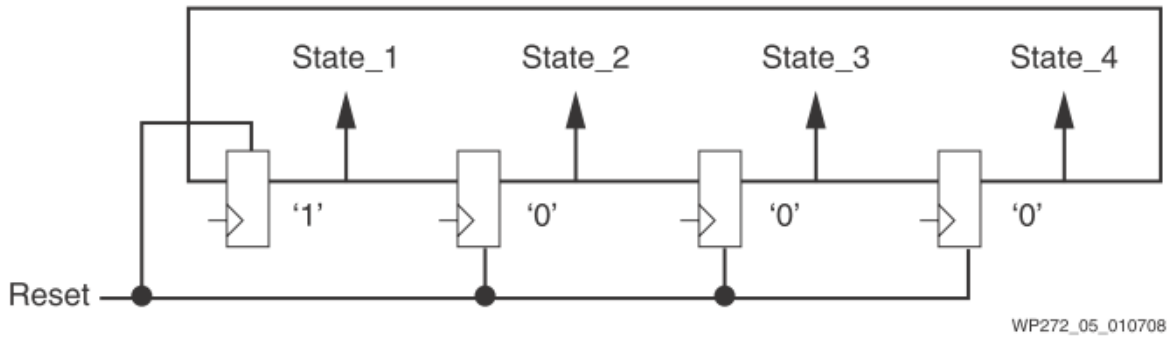
Figure 4: **Reset for a Pipeline**

But… Does it really matter?

- **Parallel pipeline**

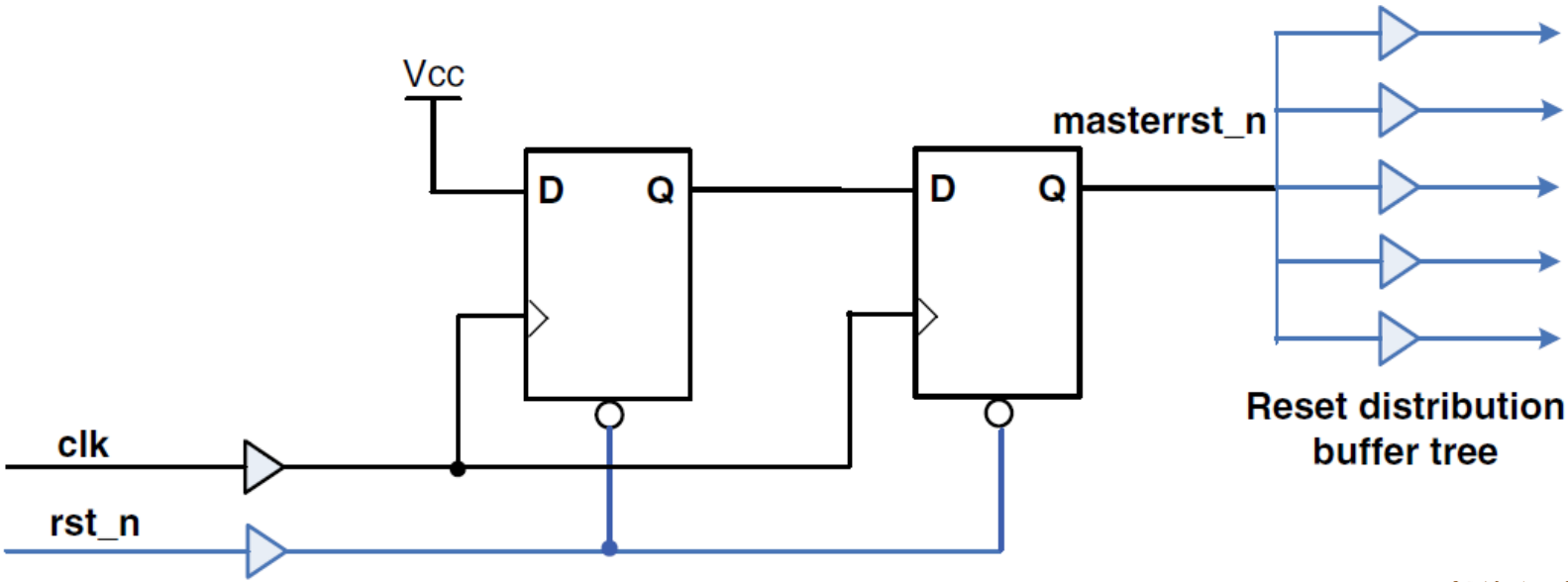


- **Pipeline with feedback**



Reset Synchronizer

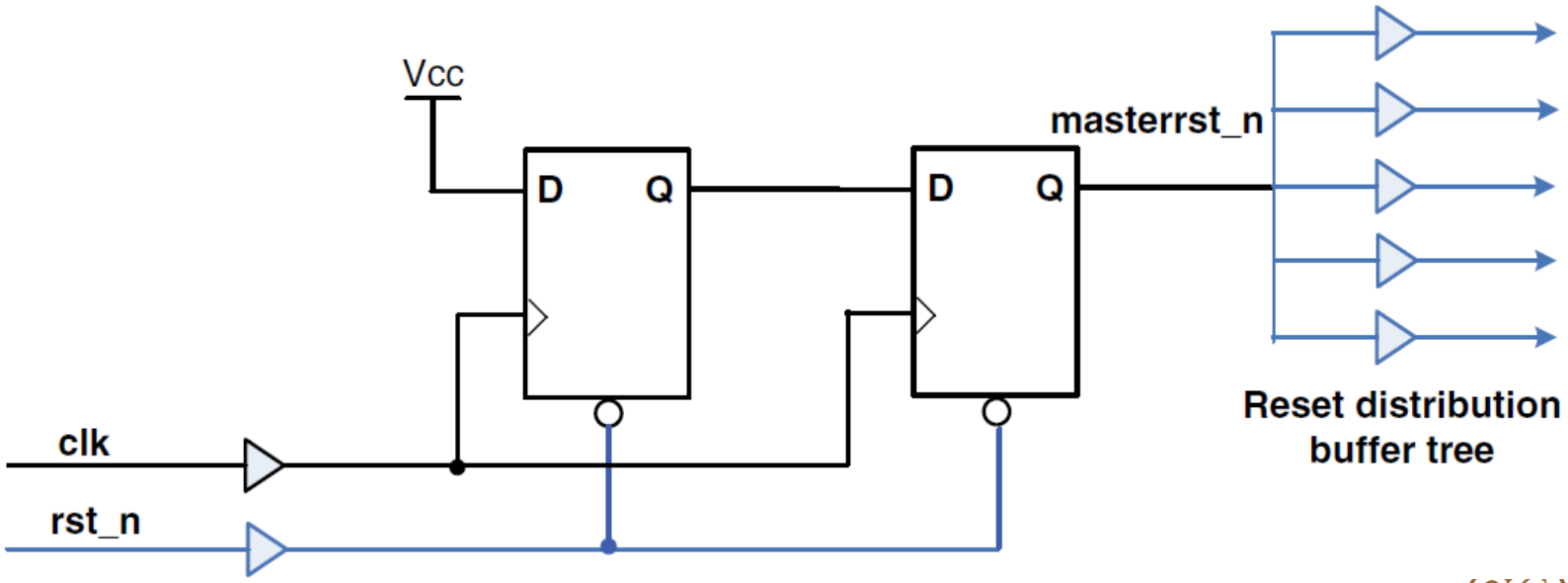
- An external reset signal asynchronously resets *a pair of master reset flip-flops*, which in turn drive the master reset signal *asynchronously*
- Take advantage of the best of both asynchronous and synchronous reset styles.
- Takes *two rising clock edges* after reset removal to synchronize removal of the master reset.
- *No metastability problems* on the second flip-flop when reset is removed



Active low reset

Reset Glitch Filter

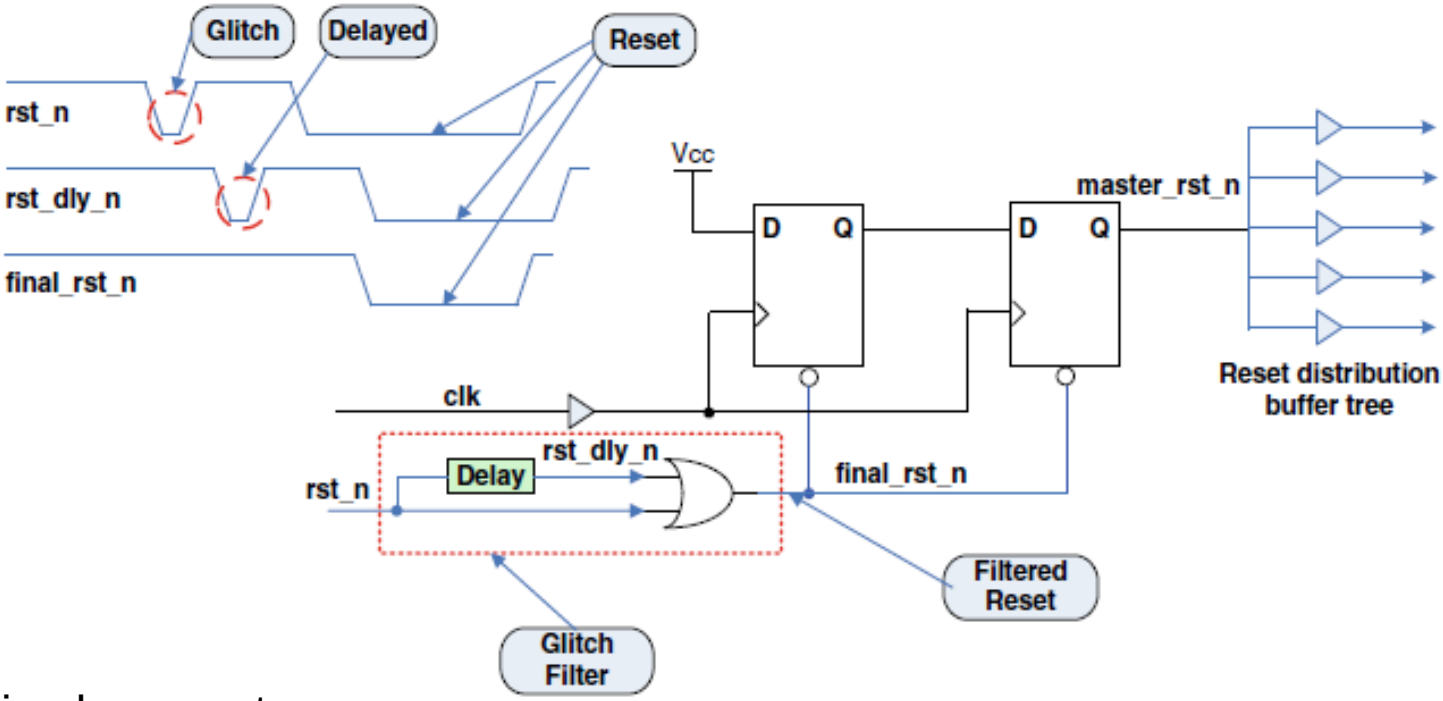
- **Any input wide enough to meet the minimum reset pulse width for a flip-flop will cause the flip-flop to reset**



Active low reset

Reset Glitch Filter

- **Any input wide enough to meet the minimum reset pulse width for a flip-flop will cause the flip-flop to reset**
- **Delay line:**
  - Vendors provide a delay hard macro that can be hand instantiated
  - Instantiate a slow buffer in a module



Active low reset

Xilinx Reset

One of the commandments of digital design states, "Thou shalt have a master reset for all flip-flops so that the test engineer will love you, and your simulations will not remain undefined for time eternal."

So, some may be surprised to learn that applying a global reset to your FPGA designs is not a very good idea and should be avoided. Clearly, this is a controversial issue, so let's take a look at the reasons why such a design policy should be considered.

Xilinx Reset: covering 99.99% of cases

Initialization after configuration (power-on reset)

- Has the same effect as a global reset
- It also initializes all RAM cells
- All program and data areas are defined even before the processor executes the first instruction

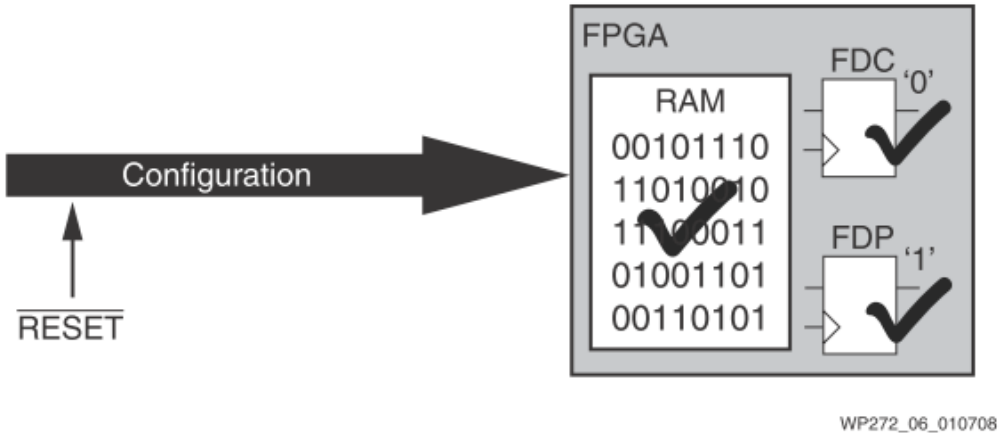
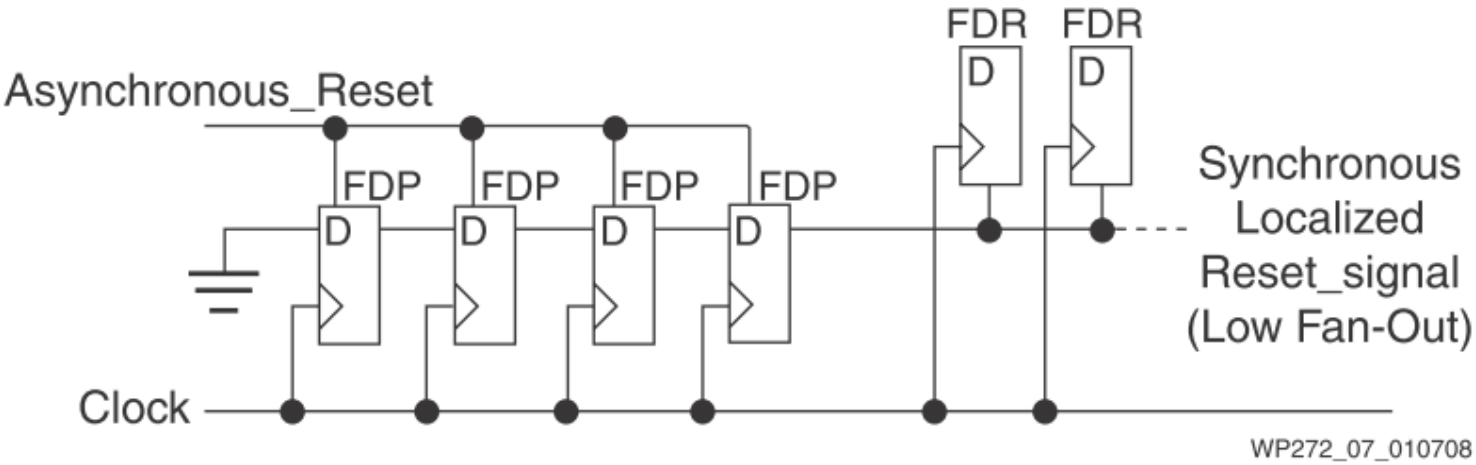


Figure 6: **FPGA Configuration**

Xilinx Reset: Strategy for the 0.01% of Cases



Reference Readings

Simulation and Synthesis Techniques for Asynchronous FIFO Design

“Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use?”

Get Smart About Reset: Think Local, Not Global
http://www.xilinx.com/support/documentation/white_papers/wp272.pdf

Get your Priorities Right – Make your Design Up to 50% Smaller
http://www.xilinx.com/support/documentation/white_papers/wp275.pdf

Lecture

- No lecture tomorrow
- Sept. 28th Monday (8.15-10.00)

Design for Test (DFT)

Erik Larsson

Associate Professor

- Sept. 29th Tuesday (8.30-10.00)

Igor Tasevski

ERICSSON