UiO : Department of Informatics
University of Oslo

IN3160 IN4160

Metastability and Clock domain crossing

Yngve Hafting 2021

Messages

- Self-test vs test bench..?
  - What is what and when do you use which?

- Next two Thursday lectures := Oblig 8 Workshop
  - *Will not be recorded…*
  - *Both lab and lecture room will be manned.*
  - *Bring your own laptop's*

  - 7.4: Guest lecture w. Espen Tallaksen
  - Thursdays after easter:
    - 3x Normal lecture (with Yngve)
      - (System design, Timing/pipelining, Interconnect/Memory)
    - 2x Exam prep with Mojtaba

Course Goals and Learning Outcome

https://www.uio.no/studier/emner/matnat/ifi/IN3160/index-eng.html

In this course you will learn about the **design of advanced digital systems**. This includes programmable logic circuits, a hardware design language and system-on-chip design (processor, memory and logic on a chip). Lab assignments provide practical experience in how real design can be made.

After completion of the course you will:

- understand important principles for design and testing of digital systems
- **understand the relationship between behaviour and different construction criteria**
- be able to describe advanced digital systems at different levels of detail
- be able to perform simulation and synthesis of digital systems.

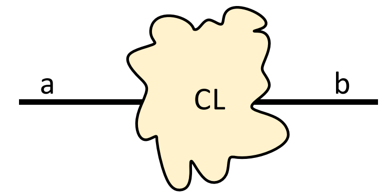
Goals for this lesson:

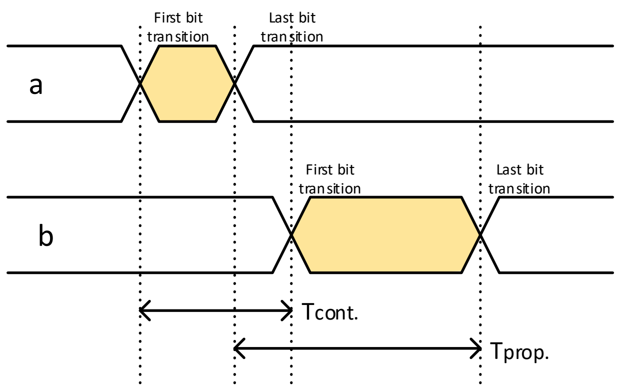
- be able to explain
    - how metastability occurs
    - how to deal with metastability in digital designs
- be able to calculate
    - error frequency for clock domain crossing
    - mean time between failure (MTBF) for brute force synchronizers
- know some common ways to safely transfer data between clock domains.

Contamination & Propagation delay

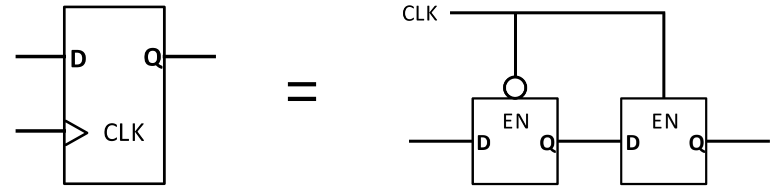
- **Contamination delay** is the
  – minimum time from the *first input* bit changes to the *first output* bit changes.

- **Propagation delay** is the time
  – from the *last input bit* changes until the *last output* bit changes

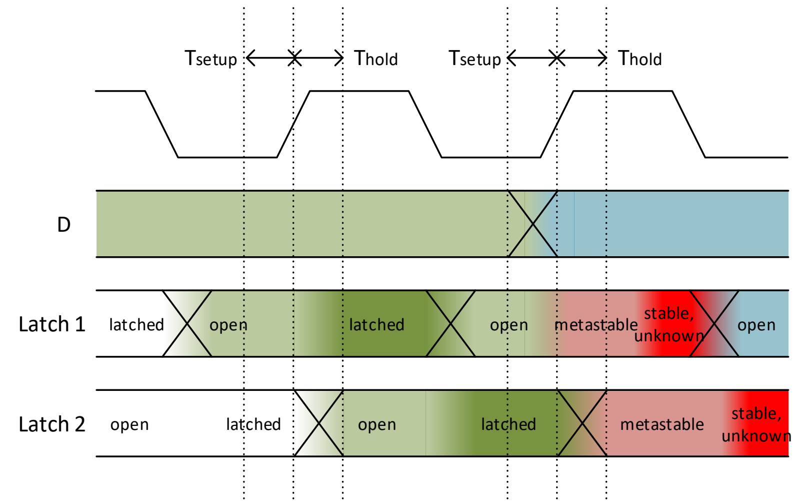




Flipflops, setup & hold



- A flipflop is 2 latches
  - EN on negated clock edge
- the input to the first latch must be ready before clock edge (T_{setup})
- the first latch may become metastable even if the input changes shortly after the clock edge (T_{hold})
- Transitions or metastability in the first latch will likely cause the second latch output to become metastable for an *unpredictable amount of time* before it settles *at an arbitrary state*.



Clock domain crossing

- Two unsynchronized systems interchanging data, will cause metastability

- Random asynchronous signal into clocked domain:

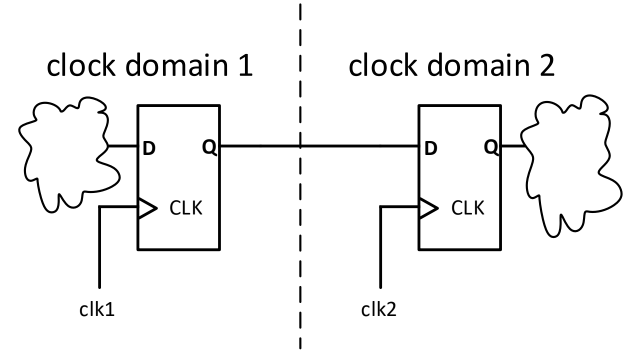
$$P_{error} = \frac{t_s + t_h}{t_{clk2}} = f_{clk2}(t_s + t_h)$$

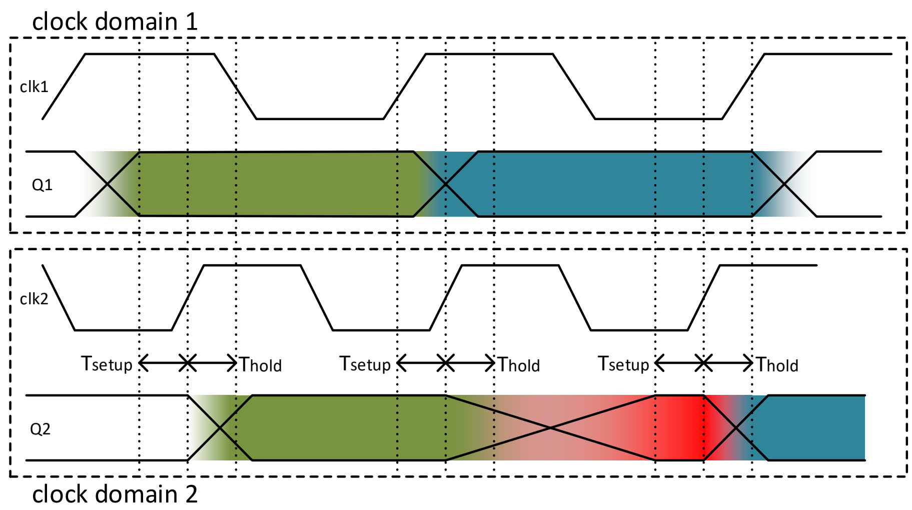
- For domain crossing:

$$f_{error} = f_{clk1} \cdot P_{error} = f_{clk1} \cdot f_{clk2}(t_s + t_h)$$

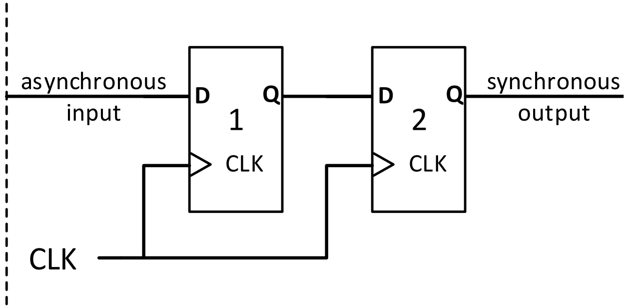
Ex: 25 MHz and 100MHz, ts = th = 100ps

$$f_{error} = 25MHz \cdot 100MHz \cdot (0.1 + 0.1)ns$$
$$= 500 \cdot 10^3 Hz$$
$$= 500kHz$$





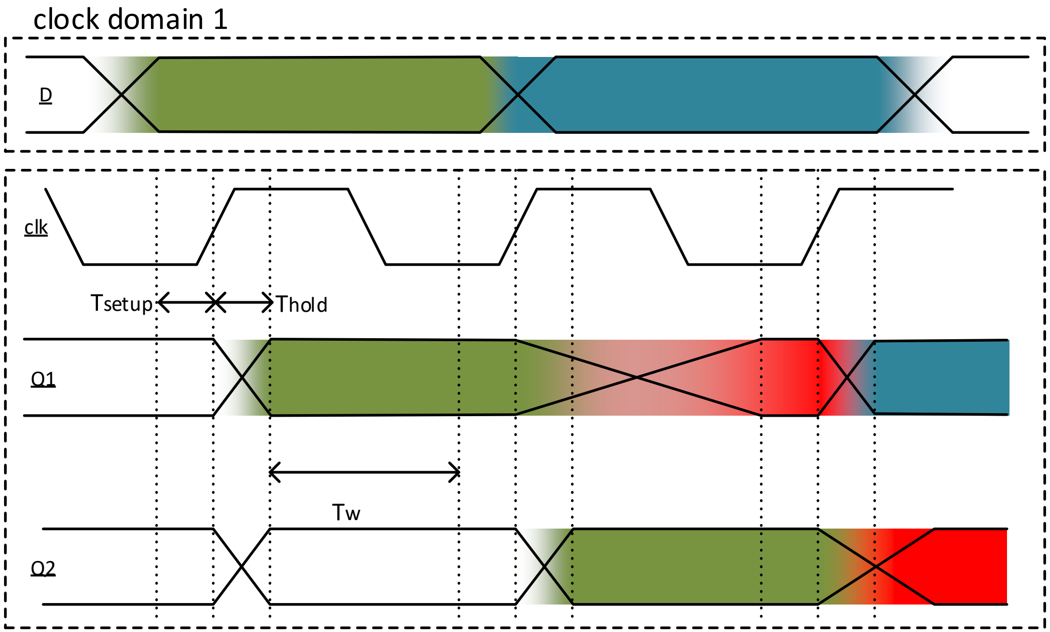
Brute force synchronizer: Probability of stability ($P_S = 1/P_U$)



- The odds of an FF being unstable after waiting for a certain time window (t_w) when metastable is given by the probability distribution function:

$$P_U = e^{\left(\frac{-t_w}{\tau_s}\right)}$$

- τ_s is the time constant for the CMOS technology in use
  - τ_s is typically in the range of 100ps

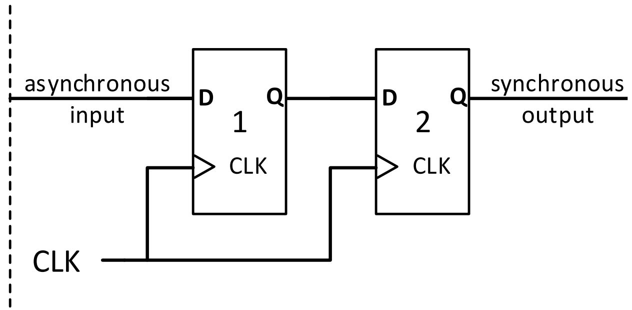


Using a 100 MHz brute force synchronizer, with Tc = 200 ps, τ_s = Ts = Th = 100ps we get

$$T_w = 10ns - (t_s + t_h) = 10ns - 200ps = 9.8ns \Rightarrow$$

The probability of failure is $P_U = e^{\left(\frac{-9.8}{0.1}\right)} = e^{(-98)} = 2{,}7 \cdot 10^{-43}$

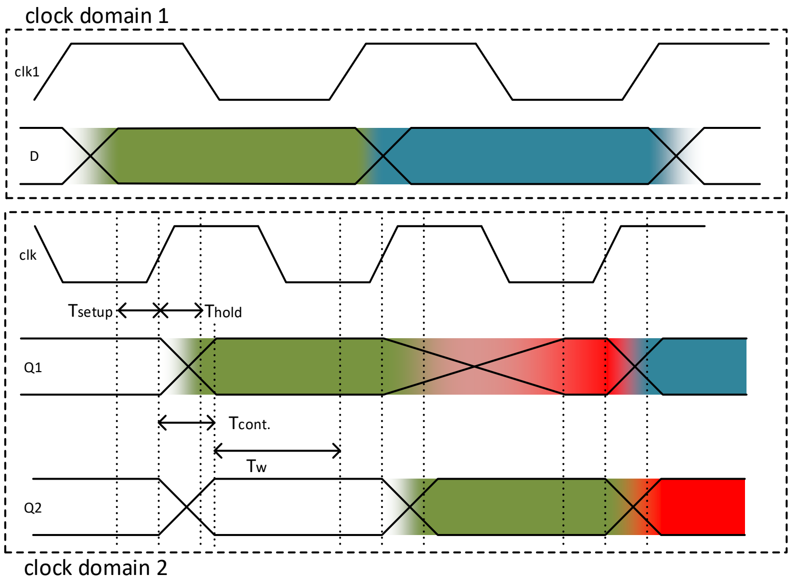
MTBF in a brute force synchronizer



- Mean Time Between Failure (MTBF)
  = 1/(Metastability frequency):

  Metastability frequency = Prob. of failure * error frequency =>

$$MTBF = \frac{1}{f_{error} \cdot P_U}$$

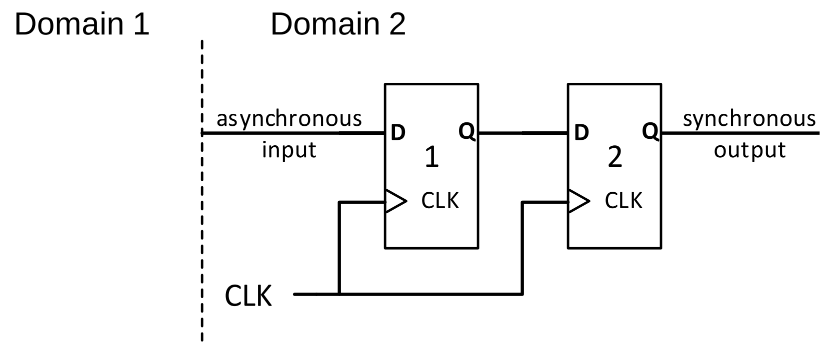


- MTBF for our 25-100 MHZ clock domain crossing
  (P_U=$2{,}0 \cdot 10^{-42}$, $f_{error} = 500kHz$) becomes

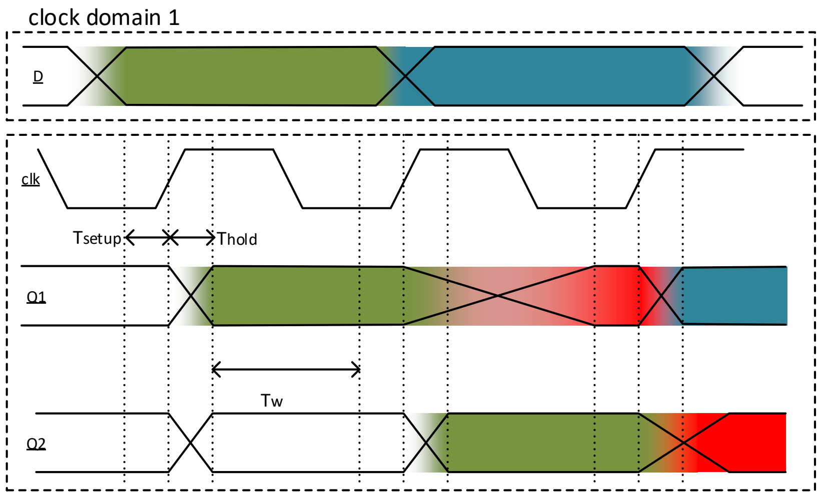
$$\frac{1}{2{,}7 \cdot 10^{-43} \cdot 500kHz} = 7{,}3 * 10^{39}s = 2{,}3 \cdot 10^{32}years$$

Sek/år = 3600*24*365 = 31 536 000

Summary



- $f_{error} = f_{clk1} \cdot P_{error} = f_{clk1} \cdot f_{clk2}(t_s + t_h)$
  - Der $P_{error} = \frac{t_s+t_h}{t_{clk2}} = f_{clk2}(t_s + t_h)$
- $P_U = e^{(\frac{-t_W}{\tau_S})}$
  - $T_W = T_{cycle} - (t_{hold} + t_{setup})$
    - $T_{cycle} = \frac{1}{f_{clk2}}$, τ_S *-settling time is technology dependant*
- $MTBF = \frac{1}{f_{error} \cdot P_U}$
- *Note: we assume* $f_{clk2} > f_{clk1}$



Brute force synchronizer

- The goal is to avoid propagating metastability
  - It is not to ensure correct data
  - brute force synchronizer ensures longest possible settling time

- Can not be used for multiple bits...
  - metastability ensures data arrives at different clock edges

- ... unless data is sent using Gray code.
  - only one data bit changes at a time

How to ensure data travels safely between clock domains

- Handshake (only using brute force on control signal)
- Use of FIFOs ()

UiO : Department of Informatics
University of Oslo

IN3160

Metastability

Synchronization of n-bit data bus

Convergence and divergence in CDC path

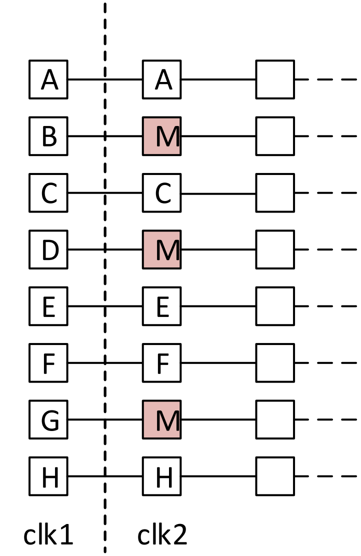
Outline

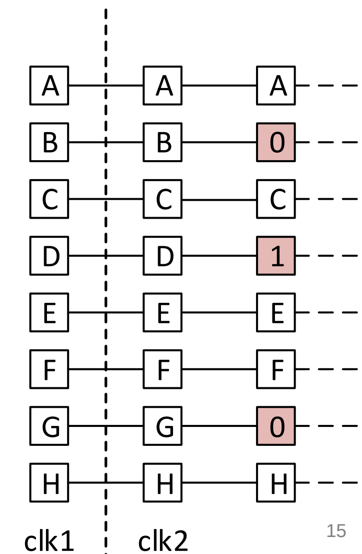
- Multiplexer-based and enable synchronizer
- Handshake synchronizer
- FIFO synchronizer
- Memory synchronizer
- Example design with enable synchronizer
- Convergence and divergence in CDC

The N-bit problem

- Using 2DFF (double flopping) synchronizer for data wider than 1-bit may lead to functional error.
  - Some bits arrive before others

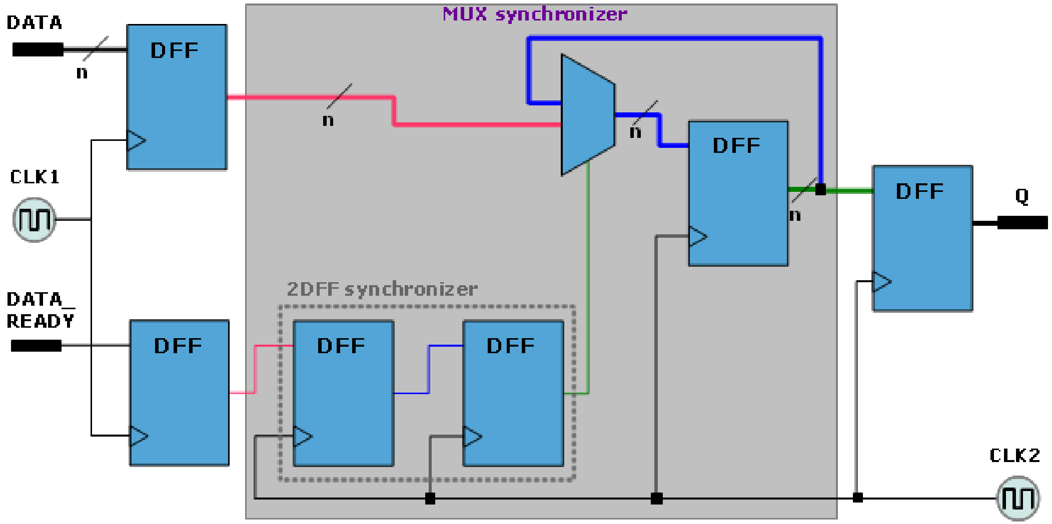
- Use synchronizers based on:
  - *Multiplexer or enable signal*
  - *Handshake*
  - *FIFO* (First In First Out buffer)





UiO : Department of Informatics
University of Oslo

Multiplexer-based Synchronizer

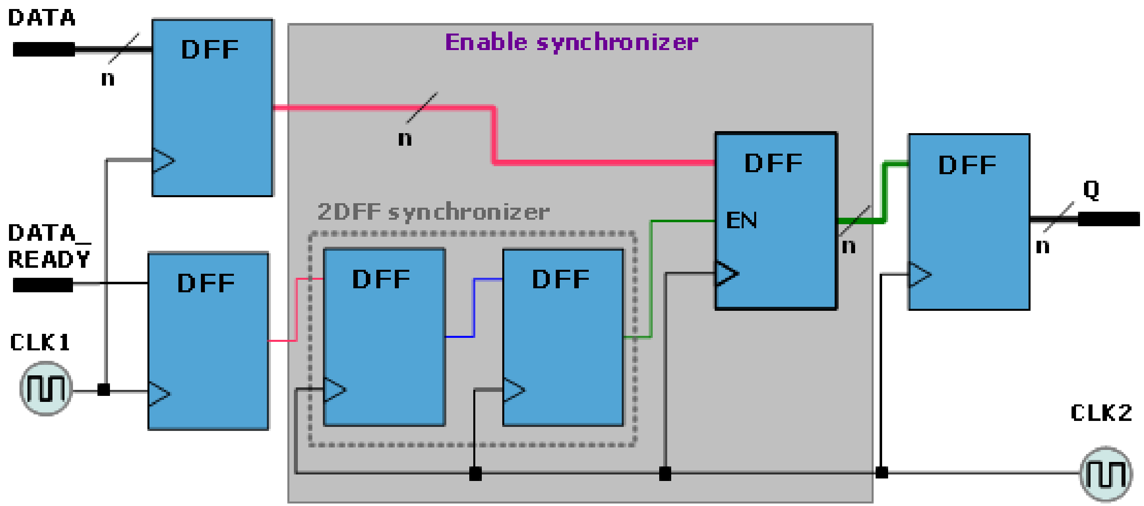


"DATA_READY" from the source domain is synchronized using a 2DFF synchronizer.
Then the synchronized control signal enables the select pin of the multiplexer.

"DATA_READY" arrives with a delay which is sufficient for the data to get stable,
it signals to the multiplexer that the "DATA" is ready to be transferred to the destination domain.

The source domain *must* keep the data constant when the "DATA_READY" signal is active.

Enable Synchronizer



In the enable scheme a control signal "DATA_READY" from the source domain is synchronized using a 2DFF synchronizer.

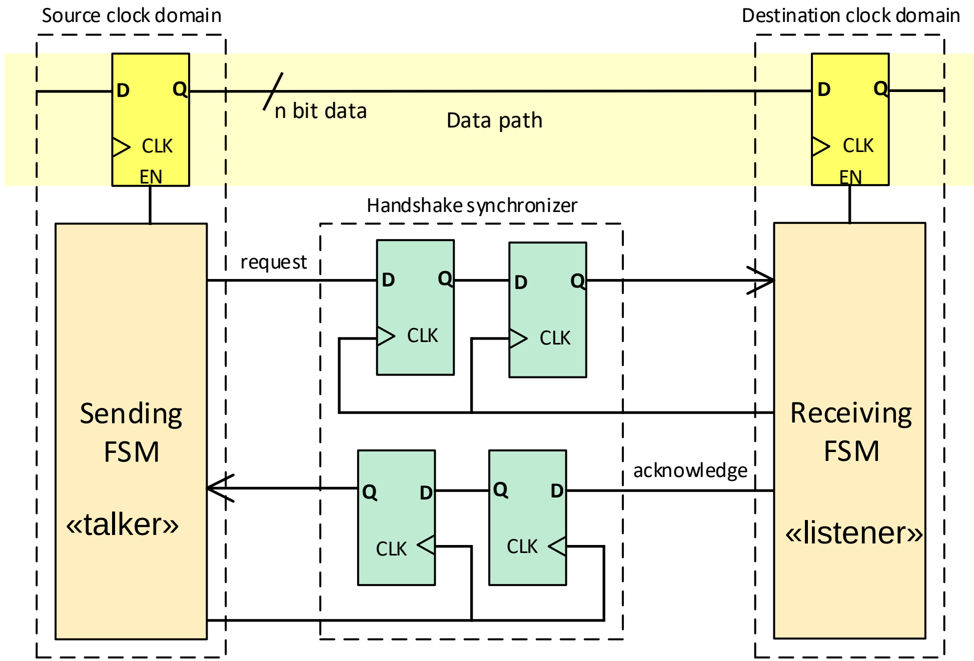
Then the synchronized control signal drives the enable pin of the first flip-flop of the destination domain.

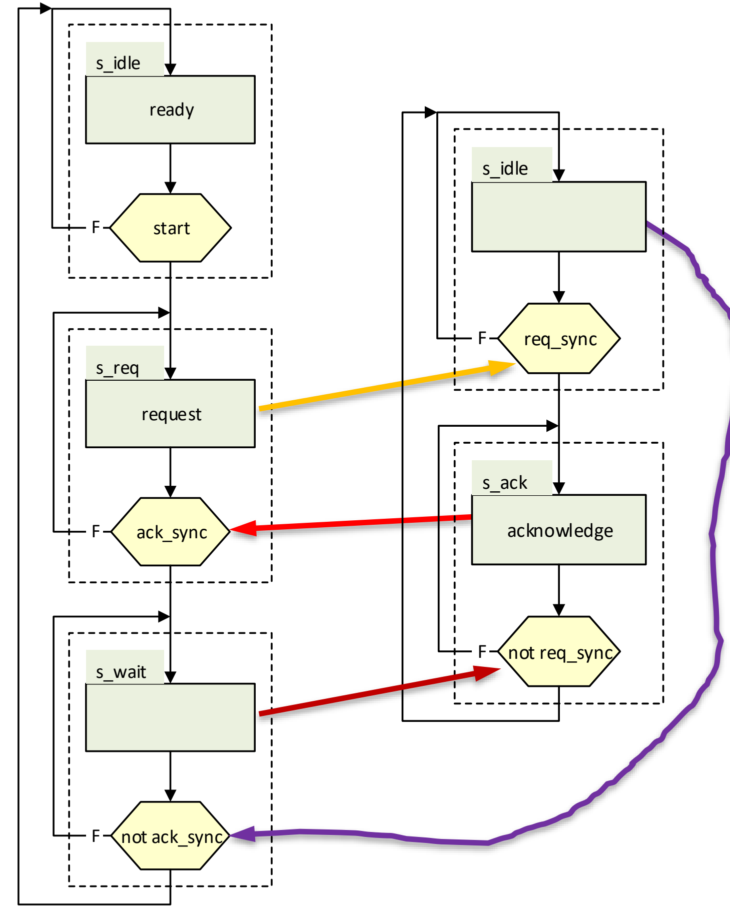
This is essentially the same solution as the previous,
using built-in ENABLE multiplexer in each DFF rather than an external

Design Principles

- MUX-select signal or FF-enable input should be driven by the synchronized control signal
- Data should be held static signal during transfer
- Select/enable synchronizers allows control of the data transfer for all bits of the bus
  - individual bits of the data bus are not synchronized separately
    - They cannot be read before (we must assume) they are ready

Handshake Synchronizer





When data are available (start):

1. The talker asserts the request signal
2. When the request signal is synchronized by the listener
   – It asserts enable and acknowledge when the synchronized request signal arrives
3. When the talker receives the synchronized acknowledge signal
   – It deasserts the request signal and waits until it is deasserted
4. The listener deasserts acknowledge when it receives the deasserted request

Source “RTL Hardware Design Using VHDL” Chapter 16.7-16.8 by P.P.Chu

FIFO Synchronizer

When the handshake synchronization cannot be applied, FIFO synchronizer may be used.

- Faster, burst type sending

- multiple signals are passed between clock domains;

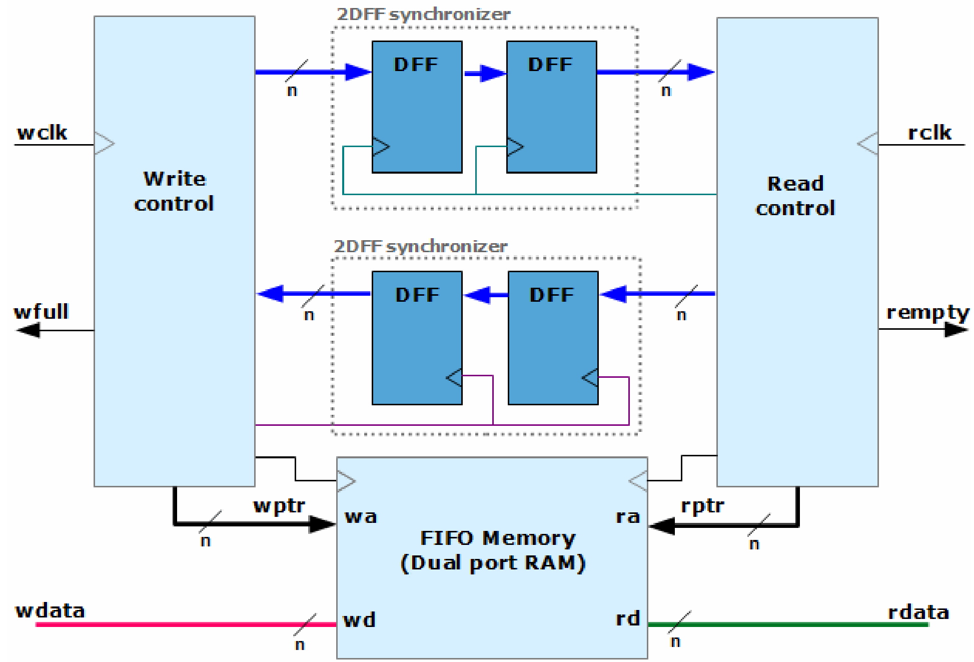
- gray code counters are used to detect full and empty state;

- signals released by these counters are synchronized via 2DFF synchronizers;

- the read and write pointers are passed to the corresponding address pins of the FIFO;

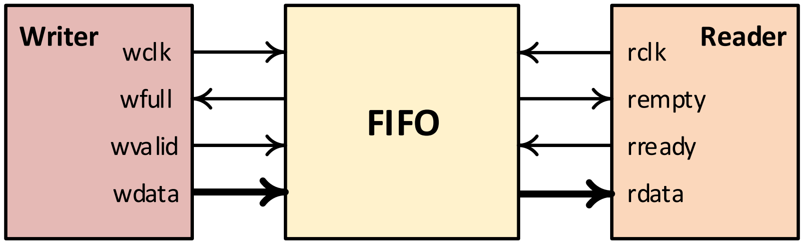
- the producing clock-domain logic never writes when the FIFO is full;

- the receiving clock-domain logic never reads when the FIFO is empty.



First In First Out (FIFO) synchronizer

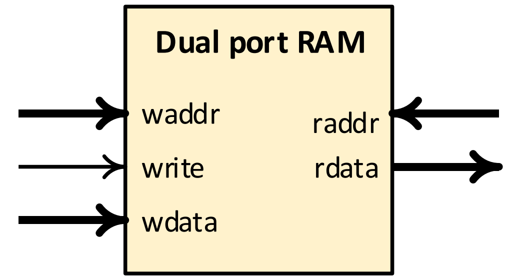
- FIFO is clocked by both sides...
  - Details on next slide(s)
- ...Either side can have the fastest clock period
  - within the FIFO capabilities
- Data is buffered in a dual port RAM
  - Enables burst read and write
  - The FIFO maintains pointers to the data
- More complex than a simple handshake
  - Details on next slide(s)
- Large buffers may be less suitable for real-time data
  - Small FIFOs can be useful...



Unwrapping the FIFO synchronizer: *Dual port RAM*

- RAM is asynchronous..
  - Data is latched, not FlipFlop'ed
  - Read and write can be done simultaneously...
    - Data should not be changed while being read

  - The FIFO makes sure..
    - Separate read- and write- address-pointers are used
      - Ensures data out is stable
    - Writing cannot be done if the RAM is full
    - Reading is prohibited if the RAM is empty

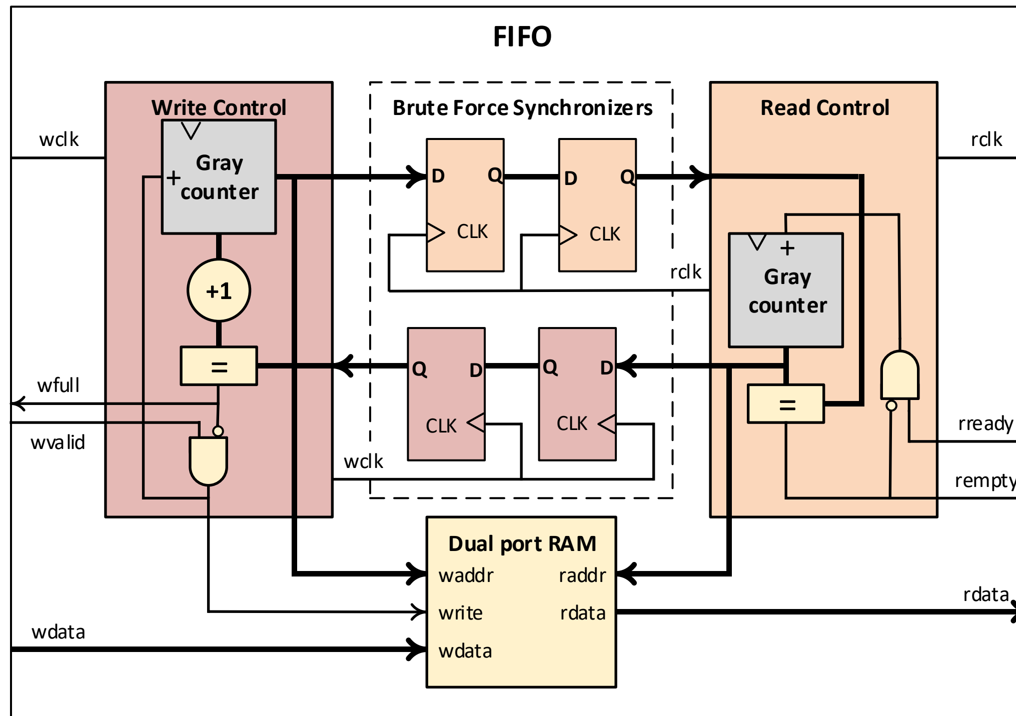


Unwrapping the FIFO-synchronizer:
Read and write control

- Write control
  - Gray counter
    - Counts up on wvalid
      - Except when wfull
    - Write address is count value
  - FIFO is full when write address is one step behind read address

- Read control
  - Gray counter
    - Counts up for every rready
      - Except when rempty
    - Read address is count value
  - FIFO is empty when read address is the current write address.

- The gray code sequence must be the same in both counters



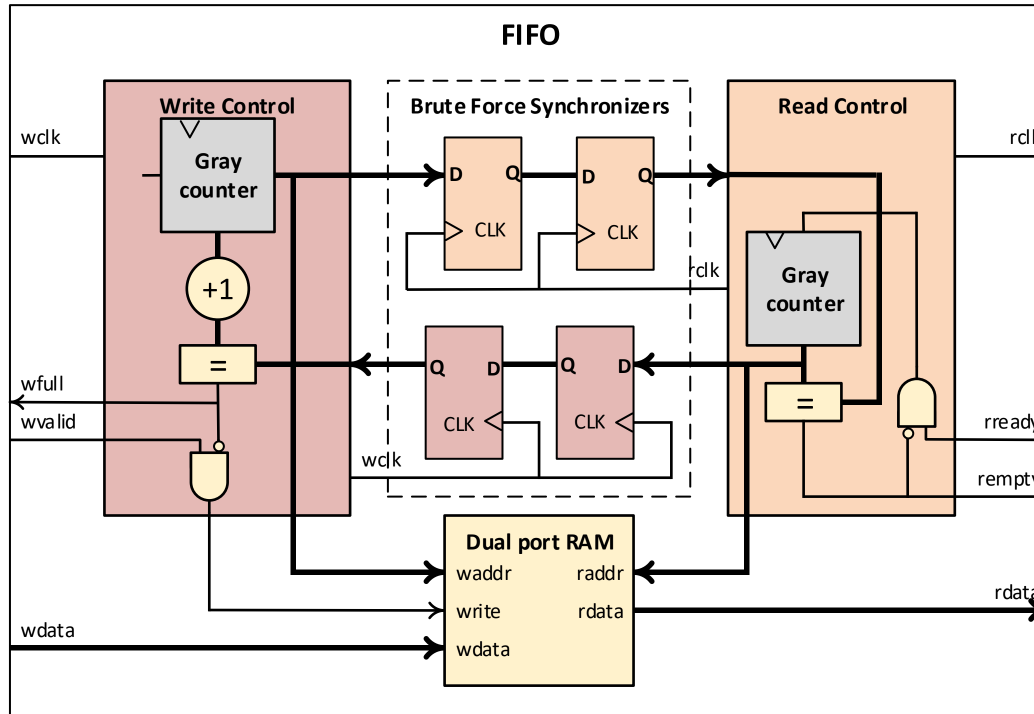
-See DHA 29.4

Unwrapping the FIFO-synchronizer:
Gray code / Gray counters

- Gray code changes only one bit at a time
  - Example sequences:
    - 00-01-11-10
    - 000-001-011-111-110-100
    - 000-001-011-010-110-111-101-100
  - the «n-bit problem» of synchronization is not an issue

- Gray code *can* be used for fault detection
  - Check if more than one bit is flipped.
    - *This is not needed* in a FIFO
      - we can only have metastability in the last bit being flipped
        (assuming all FFs are made with the same technology)
      - The read count will never be worse than one behind actual count.
  - Ex. Usage: Discovering errors in rotary encoders (Gray/ Quadrature)

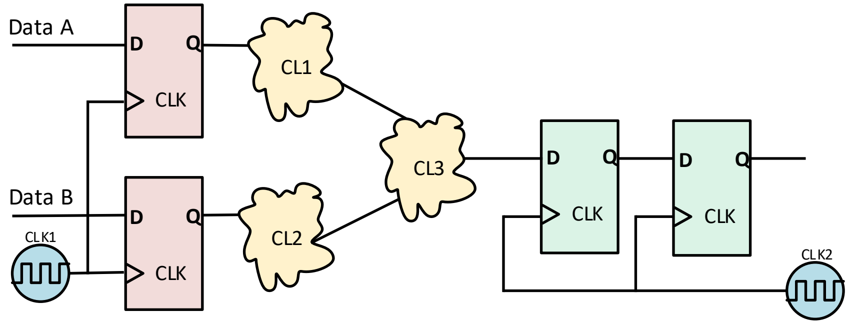
FIFO that blocks writing and reading when full/empty



Keeping track of large designs

- Problems that may arise when
  - Using combinational logic... (Hazards)
    - ...before storing asynchronous input in flipflops
    - ...driving output signals
  - Using two external signals in a module
    - Convergence in clock domain crossing (CDC) path
  - Using the same external signal in multiple modules (N-bit problem)
    - Divergence in clock domain crossing path

Convergence in CDC path problem (=Hazards)



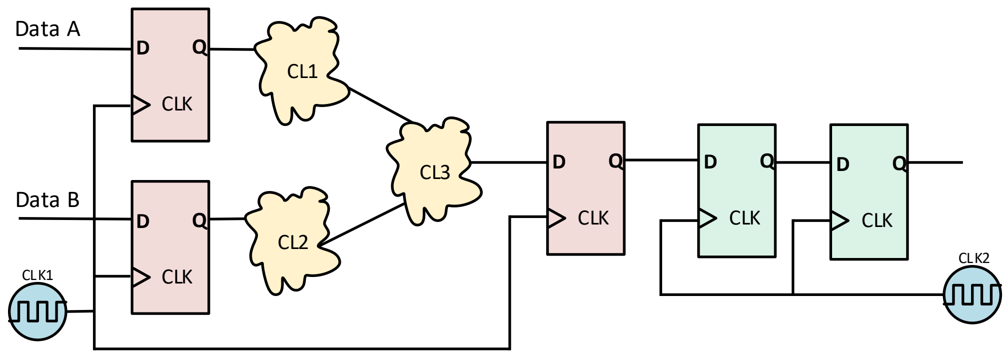
Convergent logic in the source domain may cause glitch to be passed to the destination clock domain.

Though both flip-flops of the source clock domain (CLK1) may sample the output signals at the same moment, the incorrect signal may be transferred to the destination clock domain (CLK2) because the propagation delay of the comb_logic1 may differ from the delay of the comb_logic2.

Thus, it is impossible to ensure that glitch is not propagated to the destination clock domain.

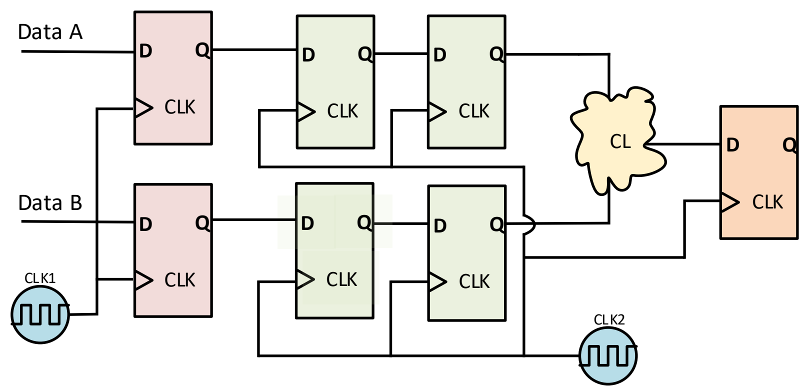
This may be obscured by multiple layers - if we allow CL in structural modules

Convergence in CDC path solution: Always store output values using FFs



To avoid such problems the convergent output data from the combinational logic should always be registered in the source domain first.

Convergence with synchronized signals problem = The N-bit signal problem

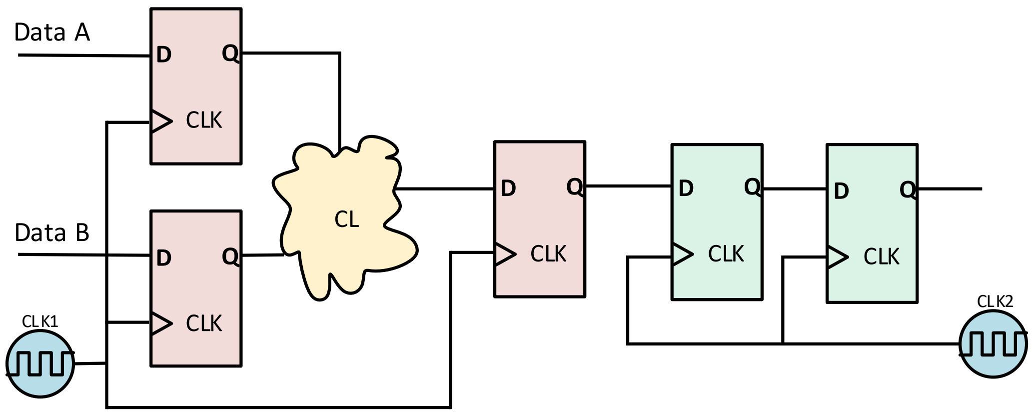


- When two signals are synchronized independently and then come to the same combinational logic in the destination domain, *functional errors* may occur.

- Both Q1 and Q2 signals from the CLK1 source domain are synchronized in the CLK2 domain. However, after the synchronization these signals converge on combinational logic.
  - Due to different meta-stable settling times data coherence can be lost and incorrect combination of the Q1 and Q2 values can reach the combinational logic leading to functional errors.

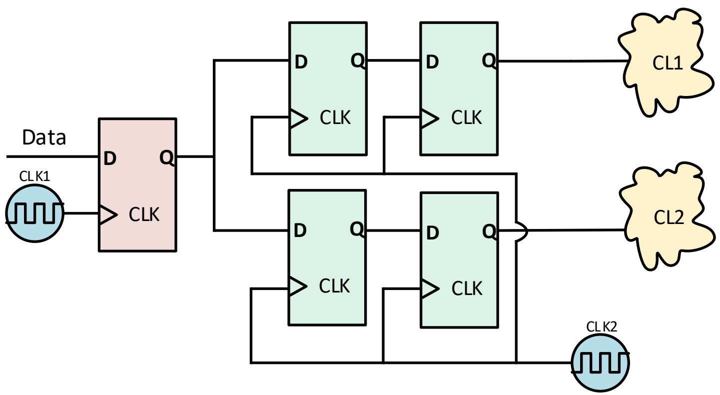
- Solution: next page

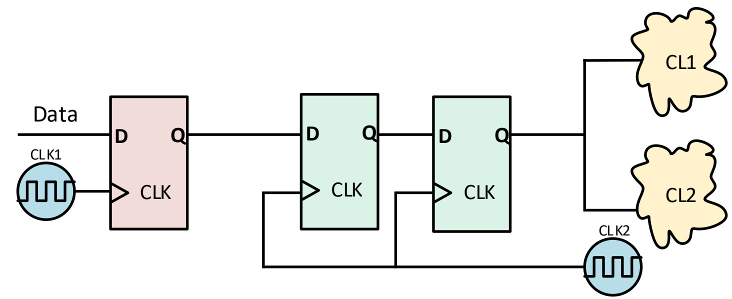
Convergence in CDC path solution



- If reconvergence is detected
  - Move combinational logic into source clock domain and then pass the resulting signal to the destination domain.

Divergence in CDC path => N bit problem





- Output signal from domain 1 is used in two different parts of clock domain 2
  - Using two different brute force synchronizers may cause signal to arrive CL1 and CL2 at different times- which can cause problems later.
  - The output signal from the source clock domain should be synchronized at first (double flopping), then fanned-out to the corresponding destination logics.
  - Ie. For Single signals, synchronize only in one location.
  - For multiple signals, use handshake or FIFO.

Suggested reading

- DHA
  - 15.2 p 331
  - 28.1- 28.3 p580-585
  - 29 p 592-605
- Steve Kilts: Advanced FPGA Design: Architecture, Implementation and Optimization, 2007,
  - chapter 10 (separate pdf).

IN3160

Reset circuits (Recap)

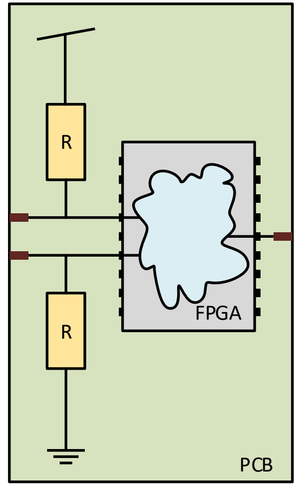
Synchronous or Asynchronous reset?

Source: Steve Kilts: Advanced FPGA Design. 2007

Outline

- Combinational logic and floating pins
- Why reset?
- Asynchronous reset
- Synchronous reset
- Reset circuit

Combinational logic and I/O pins



- No need for reset circuitry for CL
  - No values are stored in CL
  - Setting the input will give the desired output
- However... avoid floating gates
  - Floating gates may cause power surges and noise
  - All input pins should be driven
    - Potentially unconnected inputs should be pulled high or low
      - Pull-up or pull-down on PCB or
      - FPGAs may have internal pull up/down circuitry for IO-pins.

Why reset?

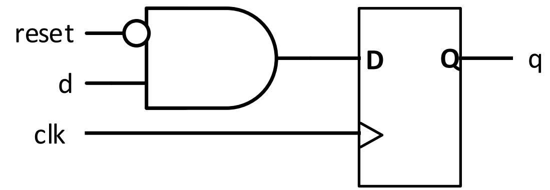
- Avoid unpredictable behavior during startup
  - Metastability creating unpredictable results
    - Both in our system and surrounding systems
  - Random register values may lead to undesired or illegal states
    - Lockup – states with no exit
    - undesired output can have unpleasant consequenses

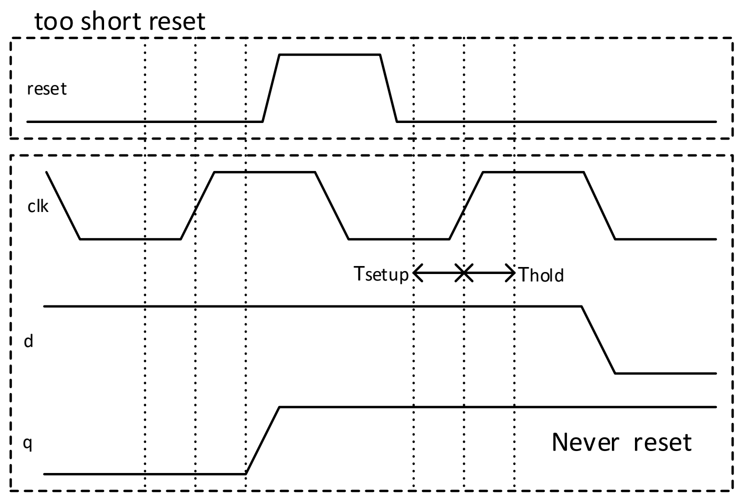
- To get out of illegal states
  - Unpredicted behaviour may lead to illegal states
  - Noise
    - Crosstalk / EMP
    - Radiation - both thermal and radioactive
    - Floating gates

- ... To ensure verifiable predictability ...

Synchronous reset

```
next_q <= '0' when reset else d;
q <= next_q when rising_edge(clk);
```

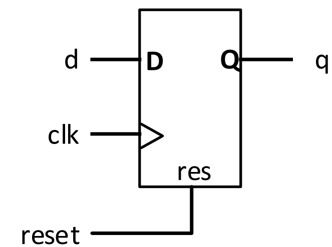




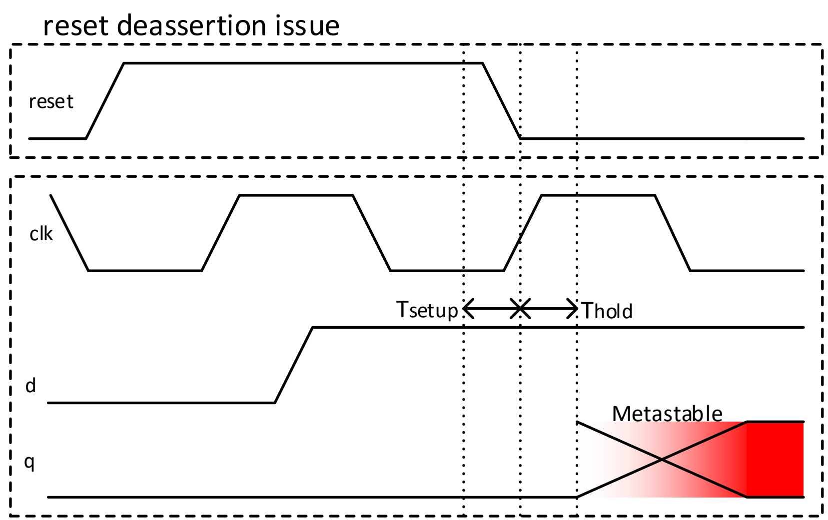
- Externally activated resets are per definition asynchronous
  - Synchronization is needed.
- Synchronous reset ‘and flip-flop input
  - Added logic can add to critical path
  - FPGA primitives may have this option built in.
- Reset pulses coming from faster clock domains may be missed entirely.

Asynchronous reset

```
q <= '0' when reset else d when rising_edge(clk);
```

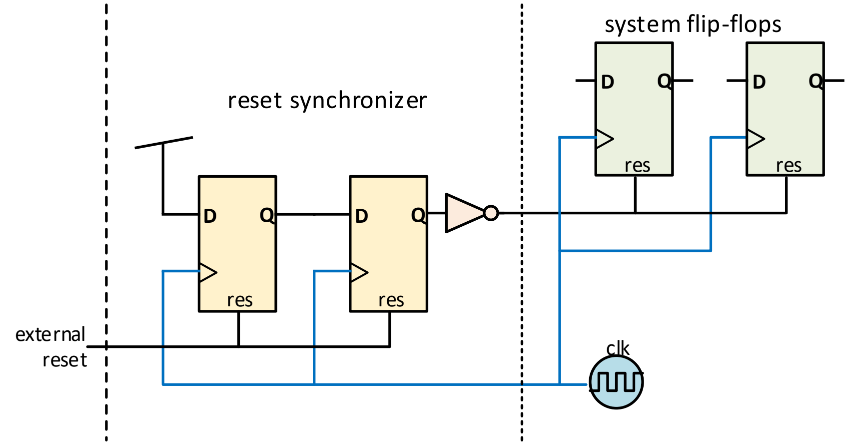


- Asynchronous assertion will always trigger
  - Reset duration must be longer than setup+hold...
- Asynchronous deassertion may cause metastability
  - Deassertion during setup/hold period

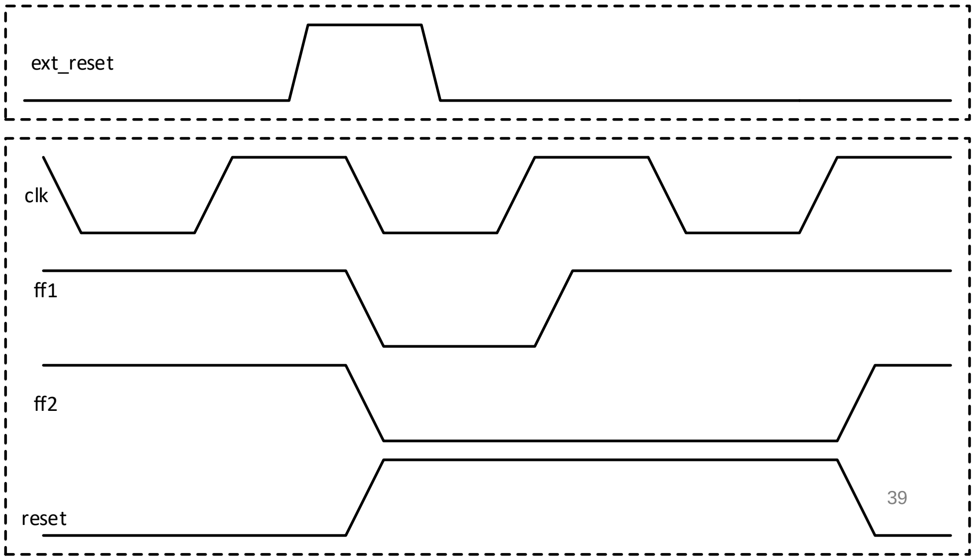


Reset circuit(s)

- Asynchronous assertion, Synchronous deassertion
  - Short reset pulses will trigger
  - 2FF mitigates metastability




```
ff1 <= '0' when ext_reset else '1' when rising_edge(clk);
ff2 <= '0' when ext_reset else ff1 when rising_edge(clk);
reset <= not ff2;
```



- Pitfalls?
  - Hazards => random reset
  - This circuit should not be used unless all CL are hazard-free.
- Multiple sources/conditional reset?
  - Ensure resetpulses are long enough
  - Use Synchronous reset
    - 2FF when crossing domains