DEDICATED TO EMBEDDED SOLUTIONS

DESIGN SAFE FPGA INTERNAL CLOCK DOMAIN CROSSINGS

ESPEN TALLAKSEN
DATA RESPONS

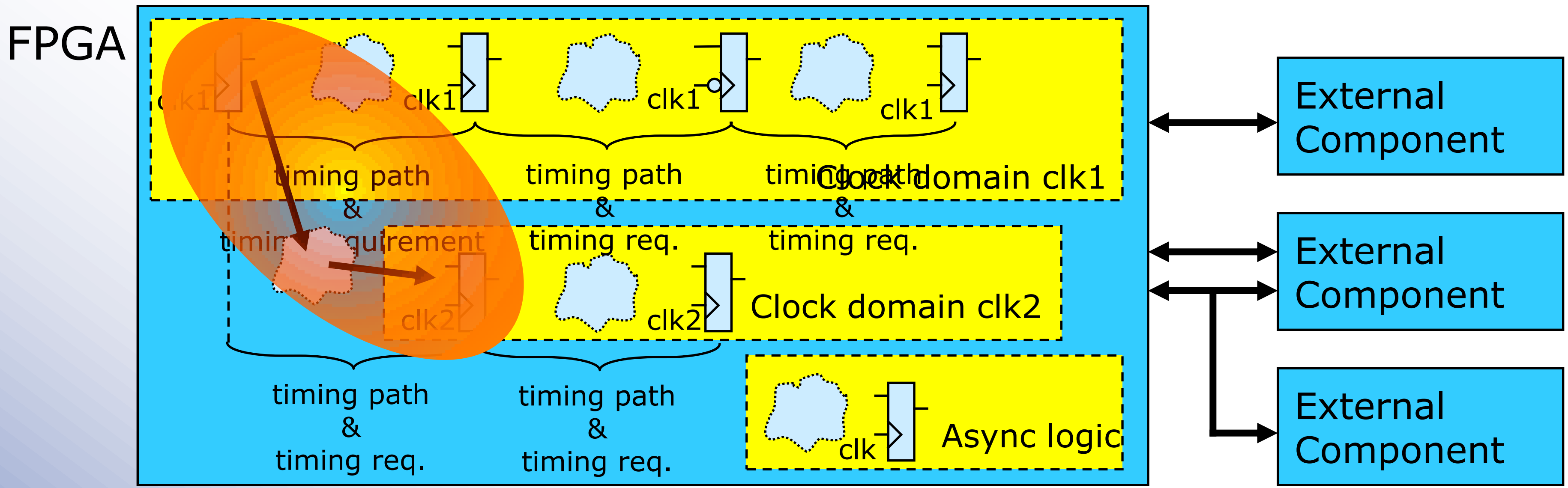
SCOPE

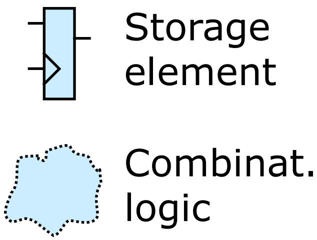
- Clock domain crossings (CDC) is probably the worst source for serious FPGA-bugs that can make your final product fail in fatal and mysterious ways.
  - ✓ This presentation shows why this is a problem and how to handle the most common CDC scenarios.
- Excerpt from
  "FPGA development Best Practices"
  (A two day Digitas course by Data Respons)
  - ✓ Originally a total of 80 slides
  - ✓ Removed basics, details on Metastability, Glitch generation, FPGA re-convergence glitching, handshaking
  - ✓ Reduced number of explained CDC cases and variations
  - ✓ Added and modified a few slides

The worst kind of FPGA bugs

- An FPGA may fail in many different ways:
  - ✓ Logic functionality
  - ✓ General timing-problems (clock domain internal)
  - ✓ Clock domain crossing (FPGA internal and I/O)
  - ✓ Asynchronous logic
  - ✓ Other: I/O characteristics and interfacing, FPGA configuration, power supply, temperature, etc...
- Logic functionality can be verified by simulation and testing
- Internal synchronous timing can be verified by static timing analysis
- Clock domain crossing and Asynchronous logic:
  - ✓ Cannot be fully verified by simulation
  - ✓ Cannot be fully verified by testing (e.g. lab, system, field)
  - ✓ Can only be fully verified by manual analysis
    - » Sometimes in combination with static timing analysis

FPGA Timing - Basics





Storage elements: Flops, memories, latch, black-box

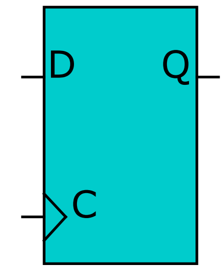
Timing path: From active clock edge on source element to input on destination element plus setup-time

Clock relations, synchronization, handshake, etc…
➔ CDC: Clock Domain Crossing

I/O timing could be pure timing path or additional CDC – with different timing for each external component

Initial effects of a timing violation

- A violation of the setup/hold times may result in:
  1. Small additional delay on Q
  2. Major additional delay on Q
  3. New data is not stored (keeping the old data)
  4. Q temporarily reflects new data, then returns to the old
  5. Q has multiple glitches before settling
  6. Q has undetermined level (voltage) before settling



➔ **Meta-stability** (or undetermined output or delay)

The uncertainty of whether new data is stored is actually as bad as any meta-stability.

Qualified Efficiency

Some results of timing violations

- Wrong data is sampled (e.g. by SW)
- Trigger-signal is missed or multiplied
- State machine enters wrong state
- State machine (or FPGA) enters illegal state
  - ✓ May result in deadlock
- Counters jump to spurious value
- Words are lost or duplicated in a data flow

Consequences of CDC problems - some real examples

- Project delays
  - ✓ 2 months delay due to sporadic errors in the system test
  - ✓ 3-4 months delay due to unstable complex interface
  - ✓ 1 year delay after product was "actually ready"
  - ✓ More than 5 man months to debug a problem that appeared some time after product release
- Product deficiencies – after customer release
  - ✓ Communication switch with lots of bit errors
  - ✓ Industrial system increasingly failing after a few years
  - ✓ Customer's application SW not working

And there are MANY more......

Motivation for proper CDC design and manual timing analysis

- An FPGA with a timing problem

  - Is extremely expensive – increasingly per stage
  - Is very time consuming
  - Is bad for credibility and customer relations

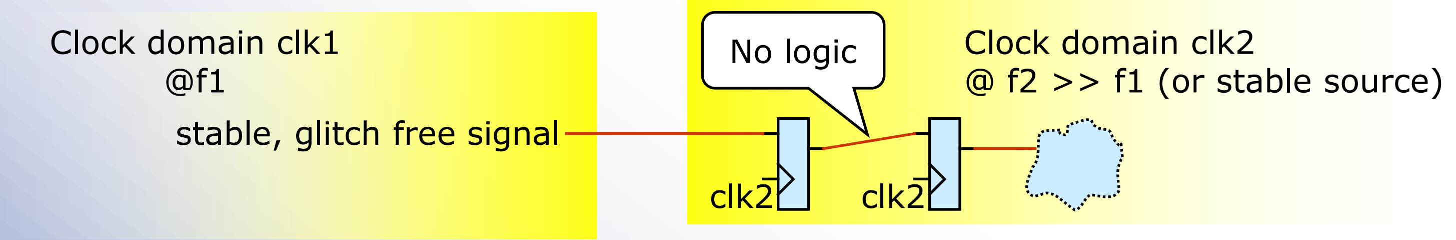
  - ✓ May fail in field operation – for no obvious reason
    - » May happen for new SW, HW, FW
    - » May happen for different temperature, voltage, power
    - » May happen for FPGA #7, or for FPGAs #17 to #3472

Types of CDC

- **Single signal CDC**
- **Multi-signal CDC**
  - ✓ **Vector CDC**
  - ✓ **Complex signal transfer. (E.g. a bus system)**
- Source and destination relations
  - ✓ Frequency relations
  - ✓ Transfer intervals
  - → Handshake variations (two vs four-phase, Boolean- vs toggle-based)
- Possible CDC exceptions – to be treated in a simpler way
  - ✓ Rising + falling (derived from rising)
  - ✓ Aligned clock (several clocks generated from the same source)
  - ✓ **Derived clocks (generated from another clock)**
  - ✓ clock selection (mux'ing between multiple sources)
  - ✓ clock enabling (gated with enable-signal before clock input)

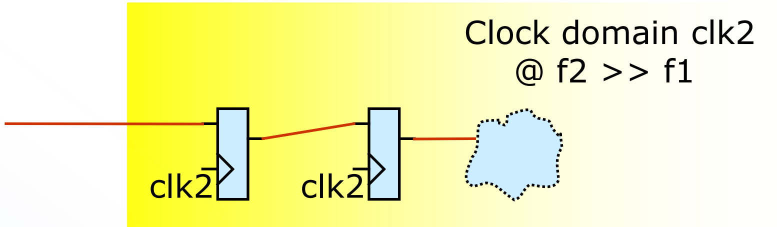
Single signal synchronization
- with faster destination

- Always:
  - ✓ Ensure stable and glitch free signal out of source domain
- For input to a faster domain:
  - ✓ Two synchronization flip-flops normally recommended
  - ✓ High frequencies or tight timing may require more
  - ✓ May utilise both edges to reduce latency



- For input to a slower domain (or unknown frequency):
  - ✓ Need handshake

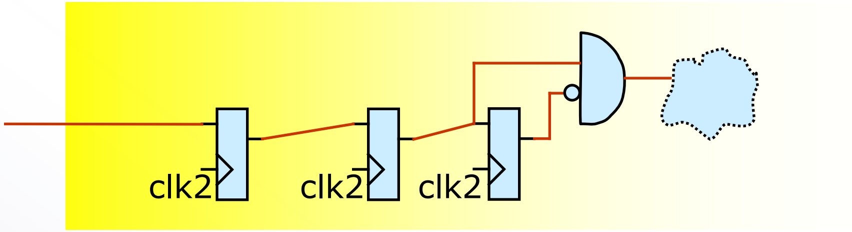
Why two flip-flops?



- Two flops – the rule of thumb for several decades…
- But, meta-stability characteristics has significantly improved - So why isn't a single flop sufficient?
  - ✓ In fact in many cases it would be…, but
    - » Would require tightened timing requirements out of flop 1
    - » Needs tighter follow-up throughout design-phase
    - » Does still have a higher risk of failure – for critical applications
    - » Why save a flop?

➔ Still a good rule: **Use two flip-flops for synchronization**

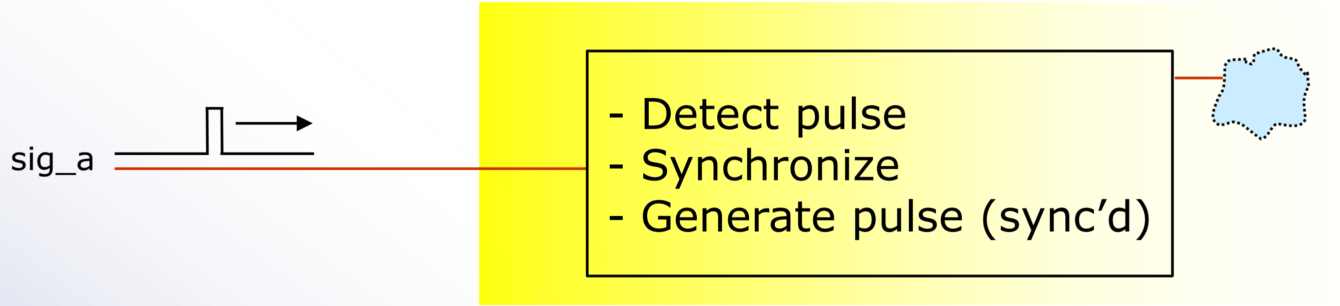
Pulse detection in destination domain



- Required when synchronized signal used as enable/trigger
  - ✓ To assure single enable/trigger
- Position after 2nd synch-flop

Single asynchronous sampling

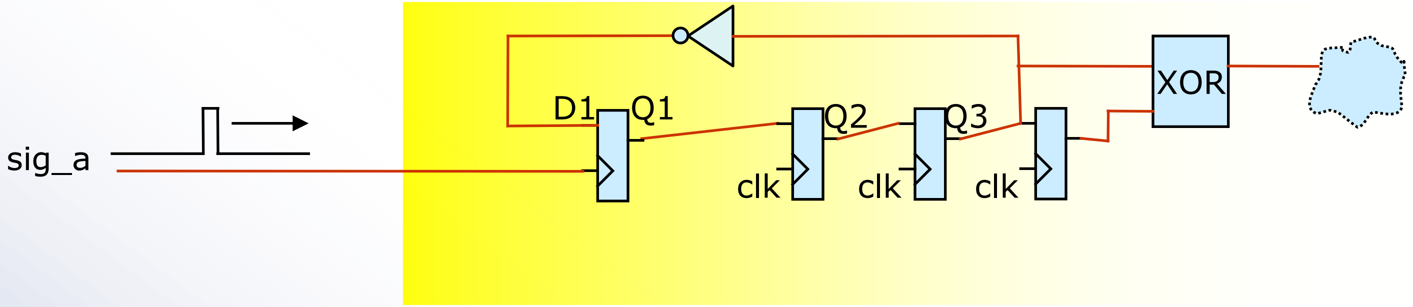
For input to (pot.) slower domain when source cannot be held



- Two main categories – depending on application
  - ✓ Counting multiple fast pulses (faster than available clock periods)
    - ➔ Need to count in a separate sig_a clock domain
    - ➔ Handle as normal CDC between domains
  - ✓ Detection of single pulse – with sufficient time between pulses,
    - » or multiple pulses where only first pulse is of interest
    - ➔ May synchronize "immediately" – after detection
    - ➔ Various solutions observed
      - Some applied solutions are definitely error prone

Sampling a single, fast pulse

For input to (pot.) slower domain when source cannot be held



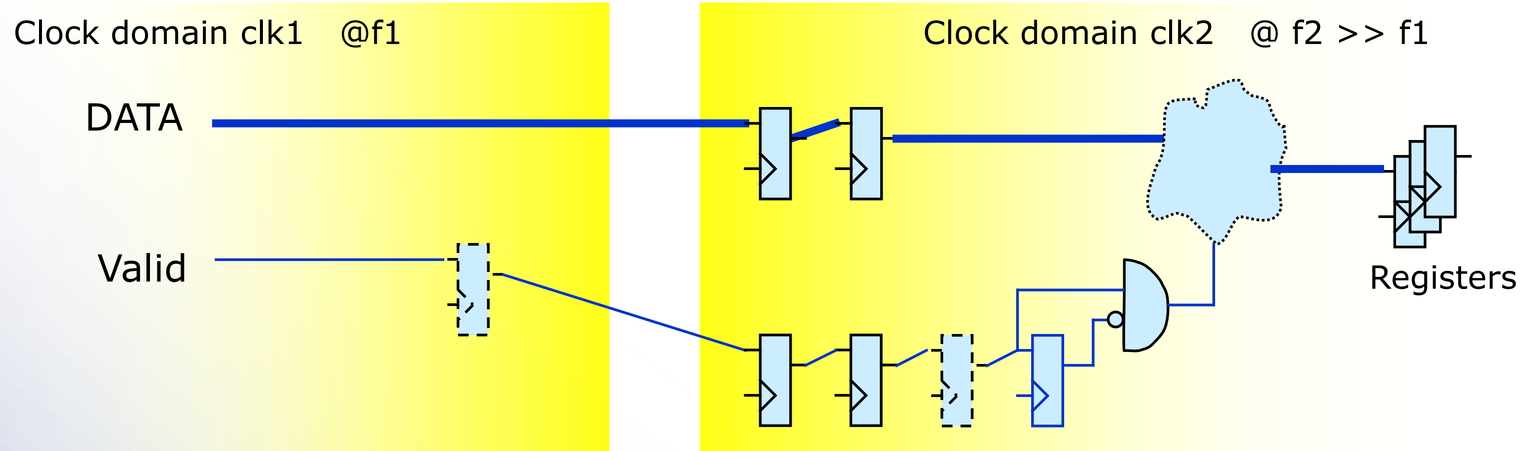
- Recommended solution
  - ✓ Uses no asynchronous set/reset
  - ✓ Will only detect one out of multiple pulses within 3 T (clk period)
    - » Multiple fast pulses will require a separate clock)
  - ✓ Dual edge triggering requires dual implementation

Note:

The feedback to D1 must come from Q3.
If coming from Q1, input pulses may be lost if pulsing twice (or any even number) between rising edges of clk.

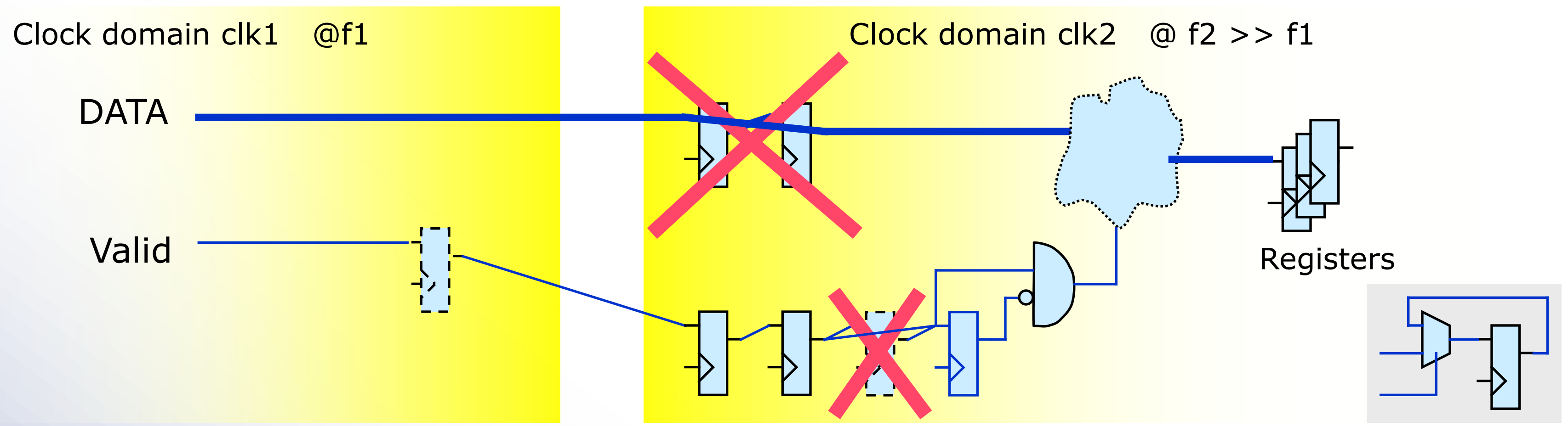
digitas Qualified Efficiency

Vector CDC – Default solution



- Synchronize data and trigger
- Use extra delay flop in trigger path if needed for timing balancing
- Register data out of source if required for stability

Vector CDC – Optimized solution



- Small combinatorial logic and fan in (3 inputs – 4 incl. synch. reset)
- No sharing of intermediate terms

➔ Complete combinatorial logic can be handled in a single LUT

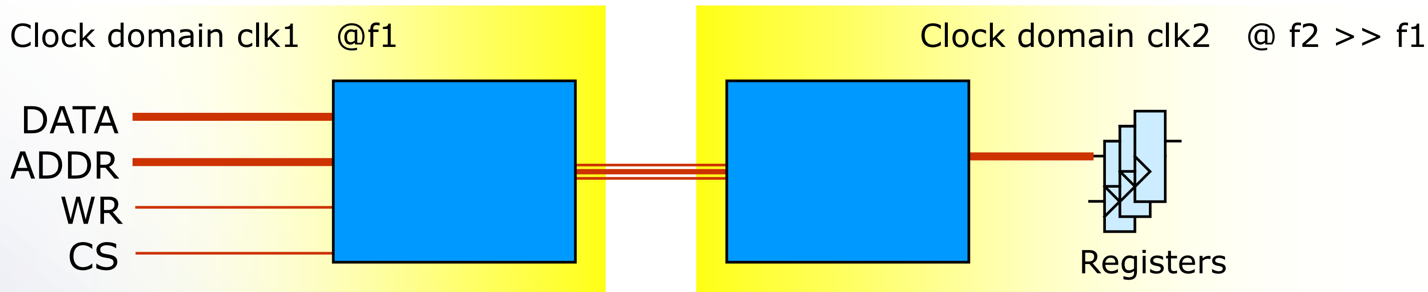
➔ No need to synchronize data

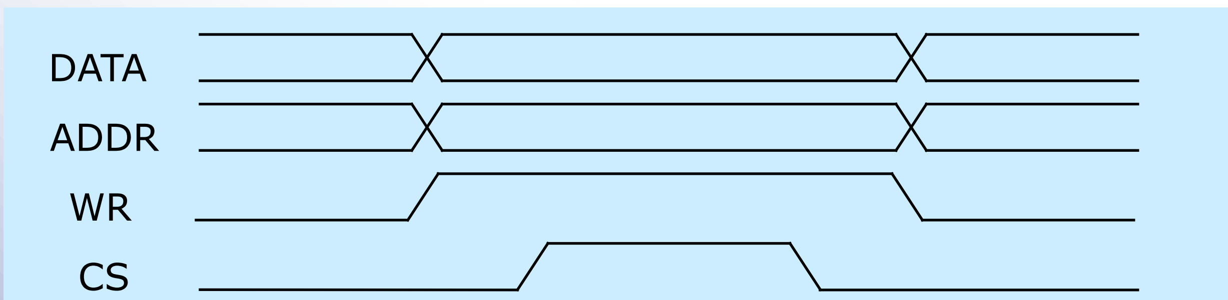
But - what if control signals are added later – or functional update of registers (e.g. a loadable counter)?

Flip-flops are cheap in FPGAs ➔ Always synchronize all signals

digitas Qualified Efficiency

Complex CDC – with **stable** data

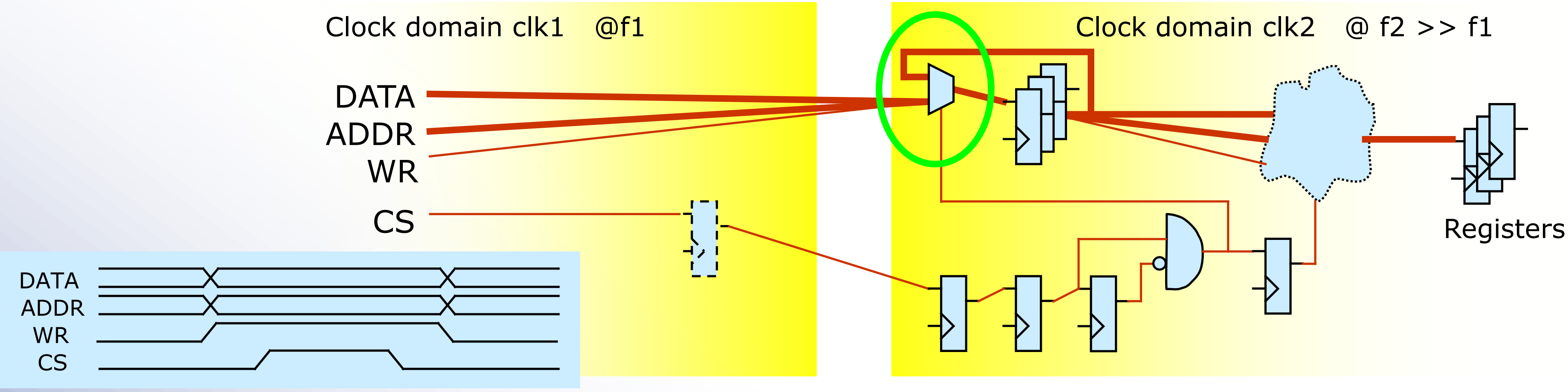




- Assume stable bus and known trigger condition
  - ✓ Data and control valid for a sufficient time around trigger (e.g. a slow bus writing to registers in a fast clock domain)

digitas Qualified Efficiency

Complex CDC
– with **stable** data - for write access



- Dual flop sync is not required for data/addr/ctr in this scenario
- Derive single trigger signal
  - ✓ Might have to combine in source domain
  - ✓ Assure glitch-less trigger signal from source domain (e.g. directly from flop)
- Synchronize trigger signal (as for single signal synchronization)
  - ✓ Use edge trigger to load synchronized data into register.

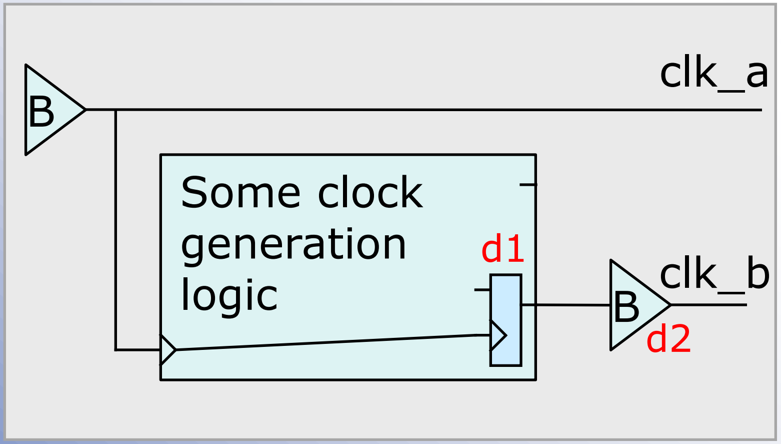
digitas Qualified Efficiency

Possible CDC exceptions

- Related clocks may often be handled in a synchronous or quasi synchronous manner
- Worst case they must be handled as asynchronous
- Typical related clock scenarios are:
  - ✓ Using both rising and falling edges of the same clock
  - ✓ "Aligned" clocks – e.g. 4 and 8 MHz, both derived from 32 MHz
  - ✓ **Derived clock - and its source**
  - ✓ Clock selection – selecting "same" or derived clock
  - ✓ Clock enabling

Derived clock

- Logic clocked by a source clock and a derived clock must sometimes be treated as two separate clock domains
- Implementation in device will depend on lots of issues



- Must consider architecture and coding
- Must consider FPGA technology
- Must consider synthesis + P&R tool
- Must consider constraints
- Must be ensure correct implementation
- Must document properly
- Must review

➔ **May** be dead simple
➔ **May** require semi asynchronous handling

digitas Qualified Efficiency

Potential CDC bug secondary effects

- Power consumption may increase
  - ✓ if more toggling/glitches
  - ✓ if unintended state is reached (e.g. bit-rate, clock-control, memory-outputs, …)
  - ✓ if illegal combinations occur (e.g. enabling 2 external chip selects for read)
- An FPGA deadlock may occur
  - ✓ e.g. if entering an undefined state
- External HW may be damaged
  - ✓ Temporarily or permanent

Conclusion on CDC

- Bugs sometimes result in serious product malfunction
- Bugs often result in major project delays
- Manual analysis and reviews may be time consuming
  - ✓ If so – spend that time
- Documenting CDC may be time consuming
  - ✓ The better reason to do it…
- Lots of designers/companies do not handle CDC properly
  - ✓ They often lose magnitudes of time compared to what they "save"

WWW.DATARESPONS.COM