Digital System Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Static Timing Analysis (STA)

Unit-III Lecture 1

Sudeendra kumar K
Department of Electronics and Communication Engineering

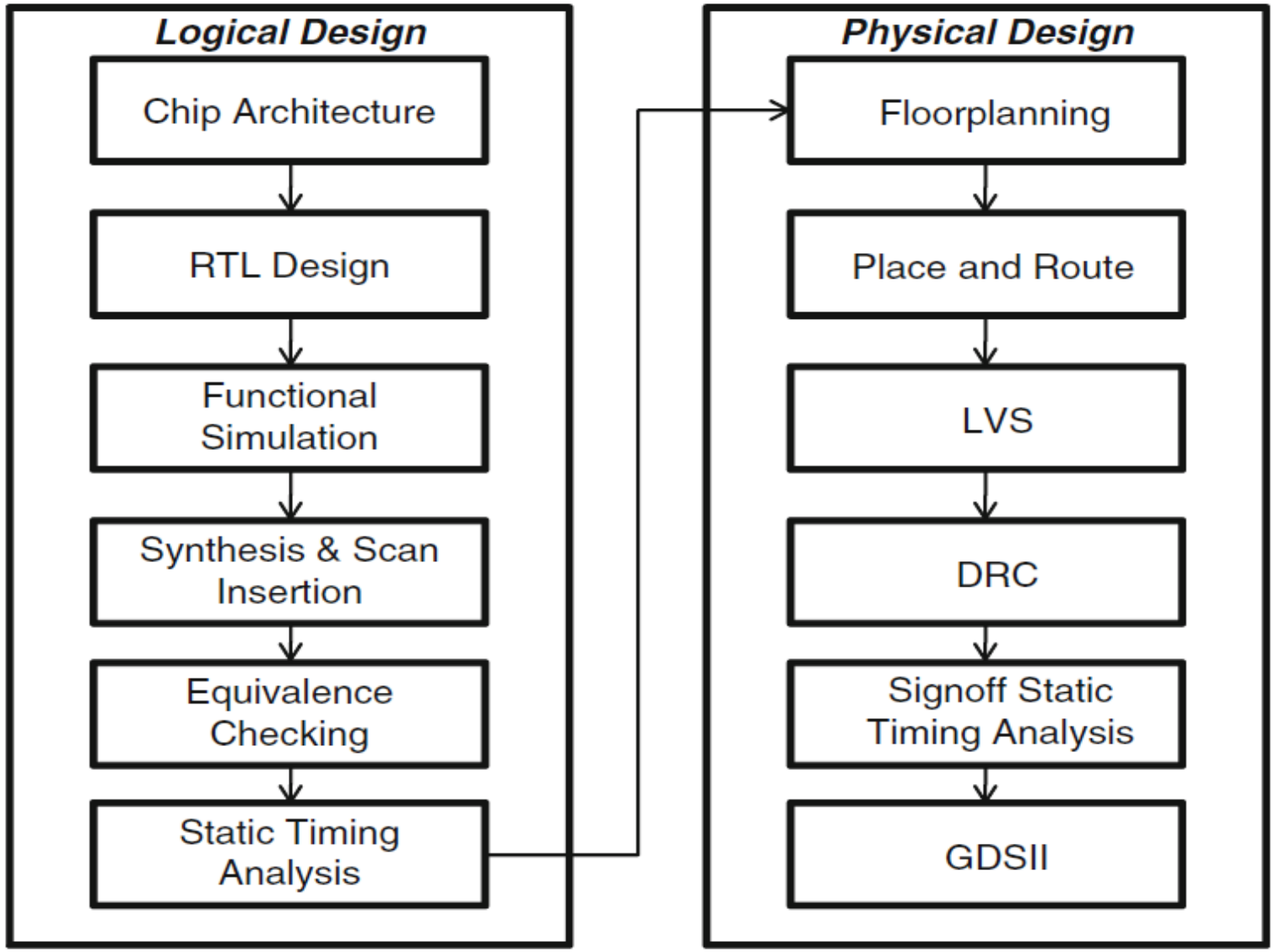
Advanced Digital Design

Contents

- Introduction to Static Timing Analysis
- Importance of STA
- Limitations of STA
- STA Analysis
  - Combinational Circuits
  - Sequential Circuits: Setup and Hold Analysis

Advanced Digital Design

ASIC Design flow



What is Static Timing Analysis (STA)

- STA is one of the important phase in the chip design to verify the timing of a digital design.
- The term timing analysis is used to refer to either of these two methods - static timing analysis, or the timing simulation (Test bench analysis) (dynamic timing simulation).
- The STA is static since the analysis of the design is carried out statically and does not depend upon the data values being applied at the input pins.
- This is in contrast to simulation based timing analysis where a stimulus is applied on input signals, resulting behavior is observed and verified.

What is Static Timing Analysis (STA)

- Given a design with input clock definitions and the definition of the external environment, the purpose of STA is to validate if the design can operate at the rated speed.
- That is, the design can operate safely at the specified frequency of the clocks without any timing violations.
- In STA, entire design is analyzed once and timing checks are performed for all possible paths and scenarios of the design. Thus, STA is a complete and exhaustive method for verifying the timing of a design.

Fmax =
1/T

T = Tsetup + T(clock to Q Delay)

Why STA

- Testbench based timing analysis methods can only verify the portions of the design that get exercised by stimulus.
- To simulate and verify all timing conditions of a design with 10-100 million gates is very slow and the timing cannot be verified completely.
- STA on the other hand provides a faster and simpler way to analyze all the timing paths in a design.
- The noise impact can limit the frequency of Verification based upon logic simulation cannot handle the effects of crosstalk, noise and on-chip variations.

Why STA on Gate-level Netlist

- The STA can be performed on a gate-level netlist depending on:
  - How interconnect is modeled - ideal interconnect, wire-load model, global routes with approximate RCs, or real routes with accurate RCs.
  - How clocks are modeled - whether clocks are ideal (zero delay) or propagated (real delays).
  - Whether the coupling between signals is included – whether any crosstalk noise is analyzed.

Limitations of Static Timing Analysis

- **Reset sequence:** To check if all flip-flops are reset into their required logical values after an asynchronous or synchronous reset. This is something that cannot be checked using STA.

- **X-handling:** The STA techniques only deal with the logical domain of logic-0 and logic-1.

- **PLL settings:** PLL configurations may not be loaded or set properly.

- **Asynchronous clock domain crossings:** STA does not check if the correct clock synchronizers are being used.

- **IO interface timing**: It may not be possible to specify the IO interface requirements in terms of STA constraints only.

References

- This Presentation
- Online Class Lecture Explanation

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

ADVANCED DIGITAL DESIGN

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Setup Check

Unit-III

Sudeendra kumar K
Department of Electronics and Communication Engineering

Contents

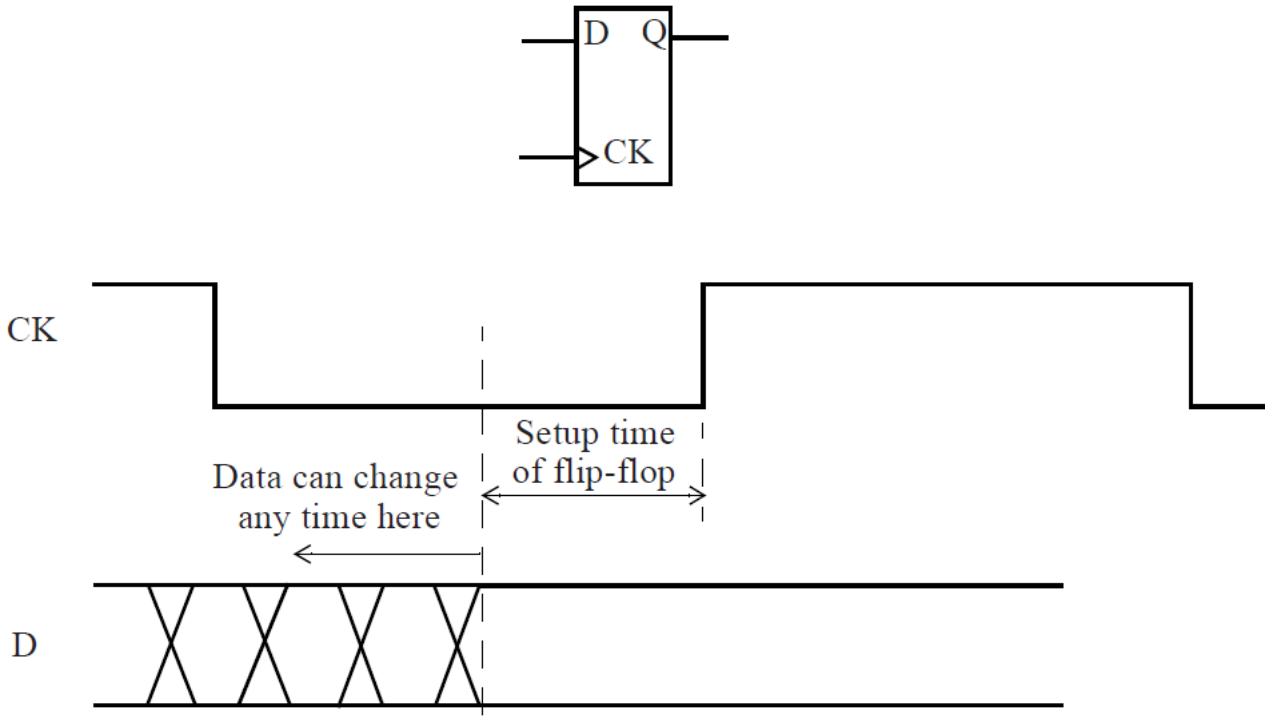
- Setup Check

Setup and Hold Check

- The two primary checks are the setup and hold checks. Once a clock is defined at the clock pin of a flip-flop, setup and hold checks are automatically inferred for the flip-flop.
- The timing checks are generally performed at multiple conditions including the worst-case slow condition and best-case fast condition.
- Typically, the worst-case slow condition is critical for setup checks and best-case fast condition is critical for hold checks - though the hold checks may be performed at the worst-case slow condition also.

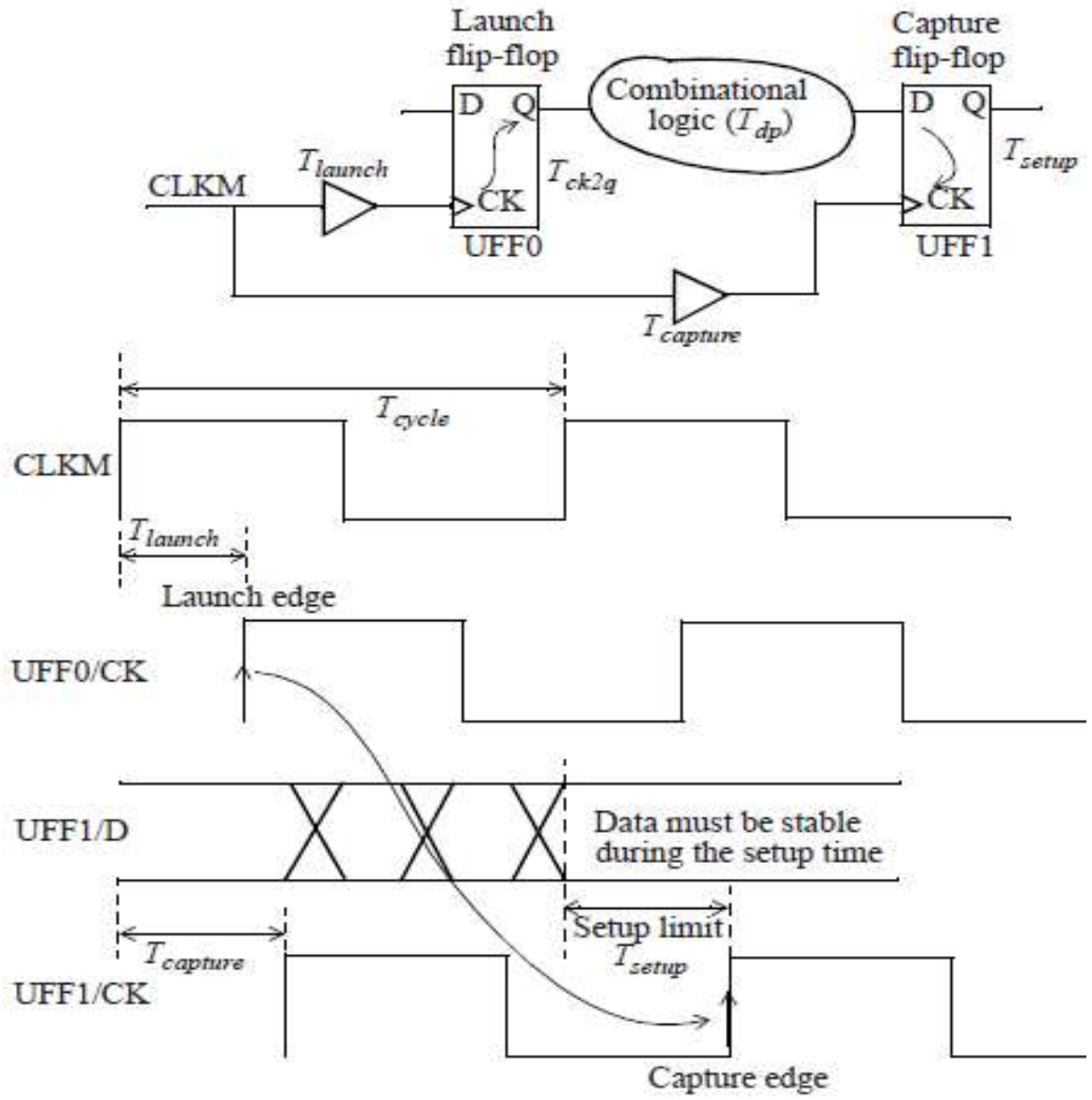
Setup Timing Check

- A setup timing check verifies the timing relationship between the clock and the data pin of a flip-flop so that the setup requirement is met.



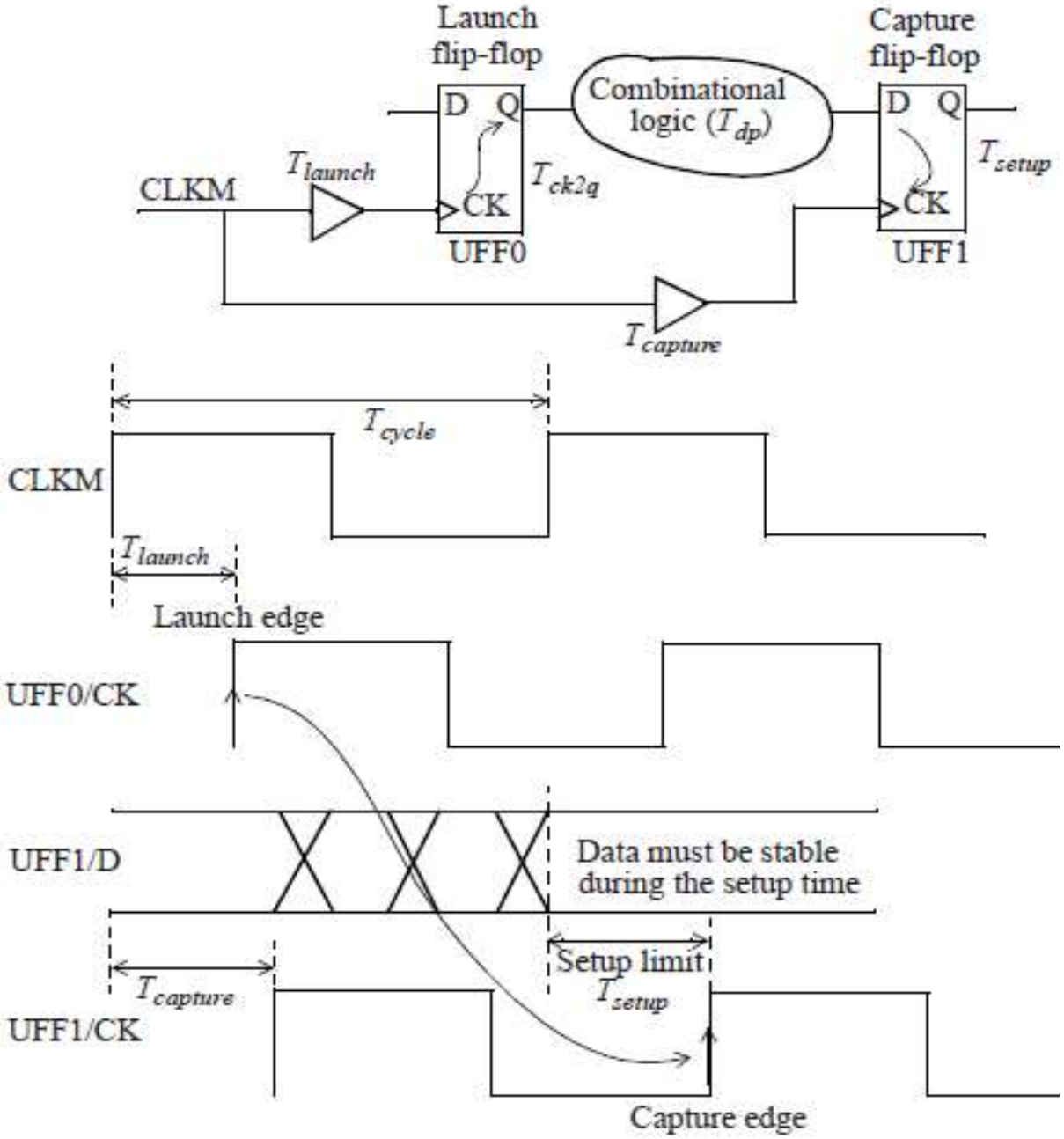
Advanced Digital Design

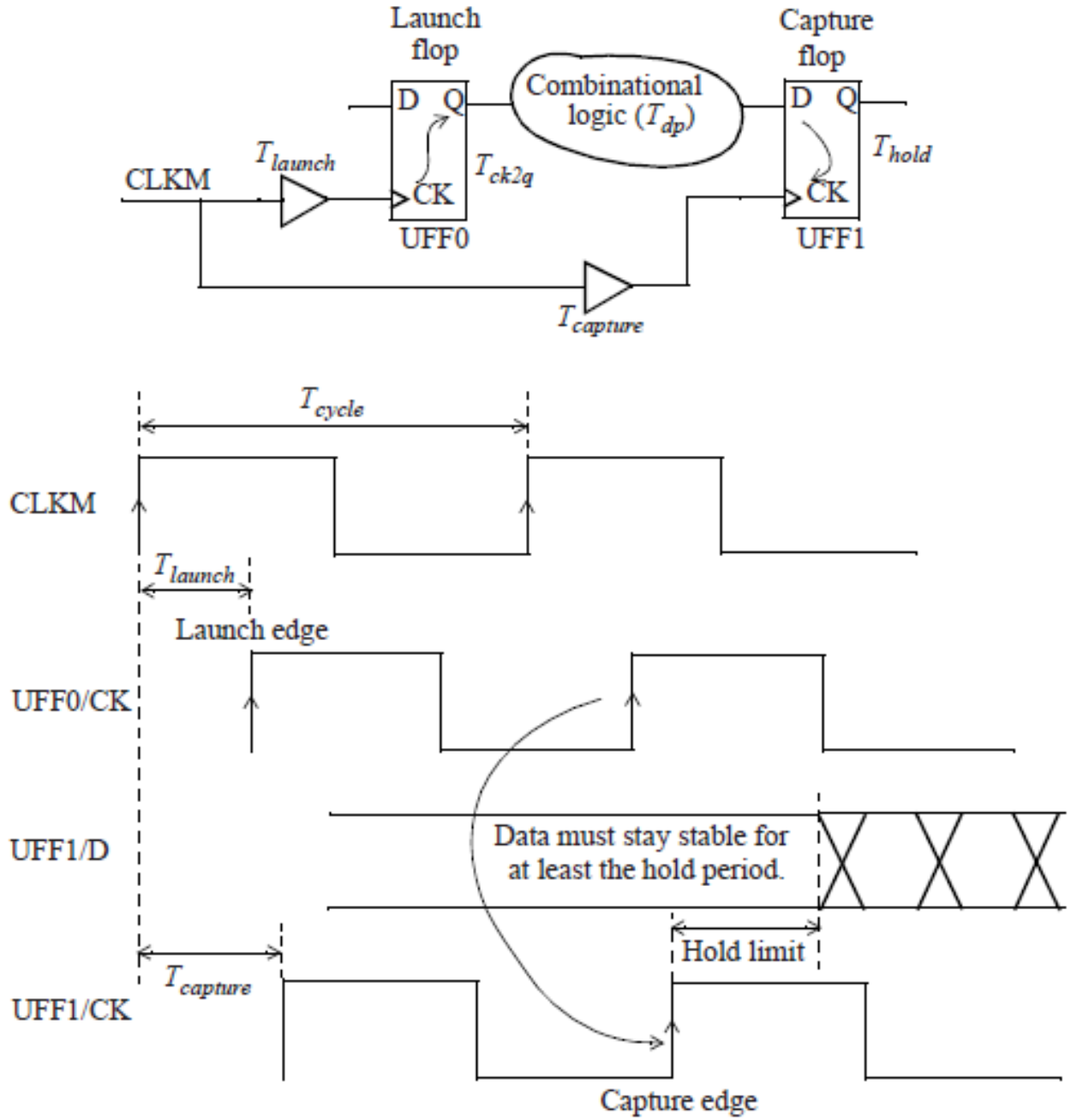
Data and clock signals for setup timing check.



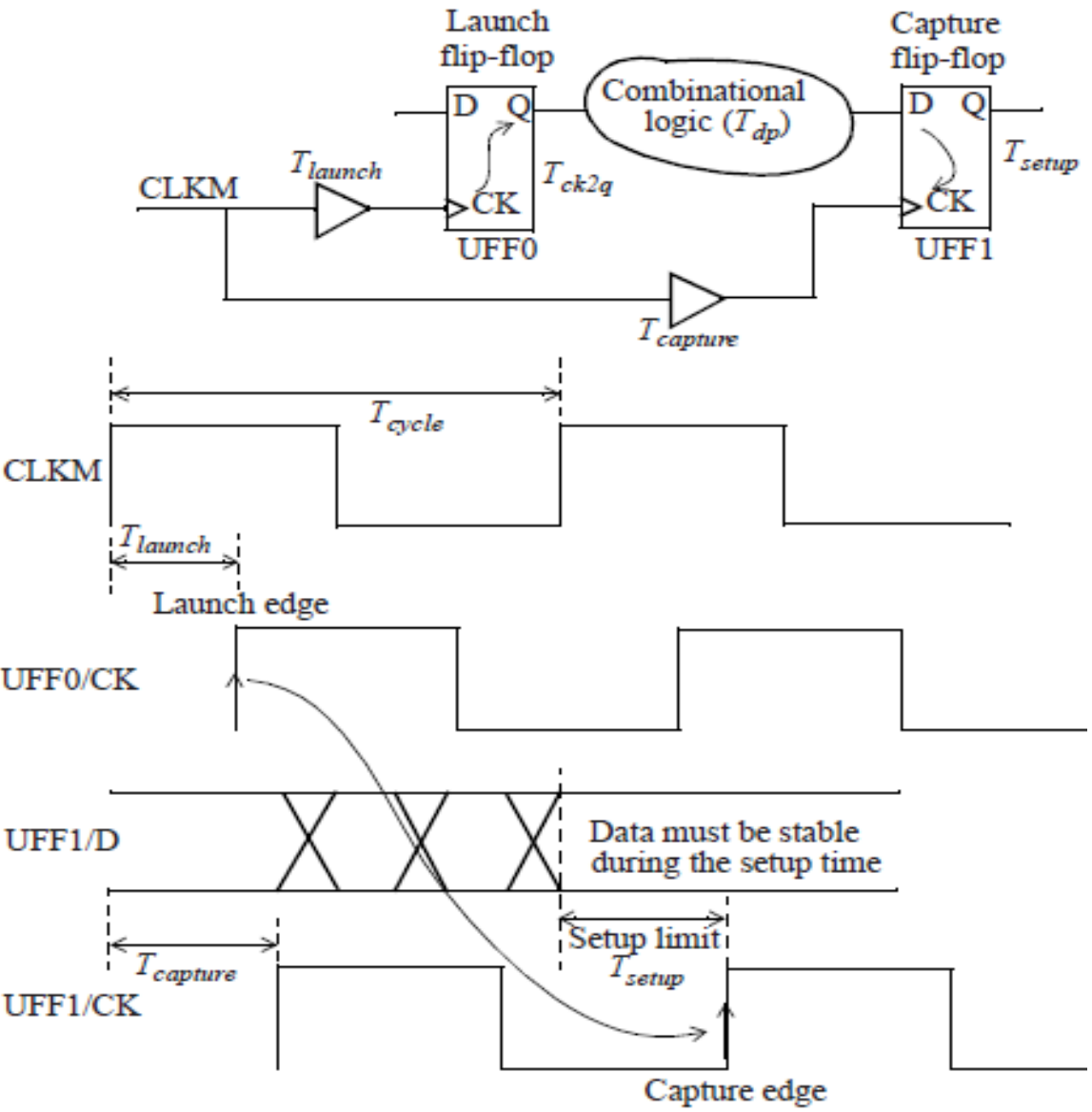
- The first rising edge of clock CLKM appears at time **Tlaunch** at launch flip-flop.
- The data launched by this clock edge appears at time **Tlaunch + Tck2q + Tdp** at the D pin of the flip-flop UFF1.
- The second rising edge of the clock (setup is normally checked after one cycle) appears at time **Tcycle + Tcapture** at the clock pin ofthe capture flip-flop UFF1.
- The difference between these two times must be larger than the setup time of the flip-flop, so that the data can be reliably captured in the flip-flop.

$$T_{launch} + T_{ck2q} + T_{dp} < T_{capture} + T_{cycle} - T_{setup}$$


Launch flip-flop
Capture flip-flop
Combinational logic (T_{dp})
D
Q
CK
UFF0
UFF1
CLKM
T_{launch}
T_{ck2q}
T_{setup}
$T_{capture}$
T_{cycle}
Launch edge
UFF0/CK
UFF1/D
Data must be stable during the setup time
Setup limit
T_{setup}
$T_{capture}$
UFF1/CK
Capture edge


Launch flop
Capture flop
Combinational logic (T_{dp})
D
Q
CK
UFF0
UFF1
CLKM
T_{launch}
T_{ck2q}
T_{hold}
$T_{capture}$
T_{cycle}
Launch edge
UFF0/CK
UFF1/D
Data must stay stable for at least the hold period.
Hold limit
$T_{capture}$
UFF1/CK
Capture edge

Data and clock signals for setup timing check.



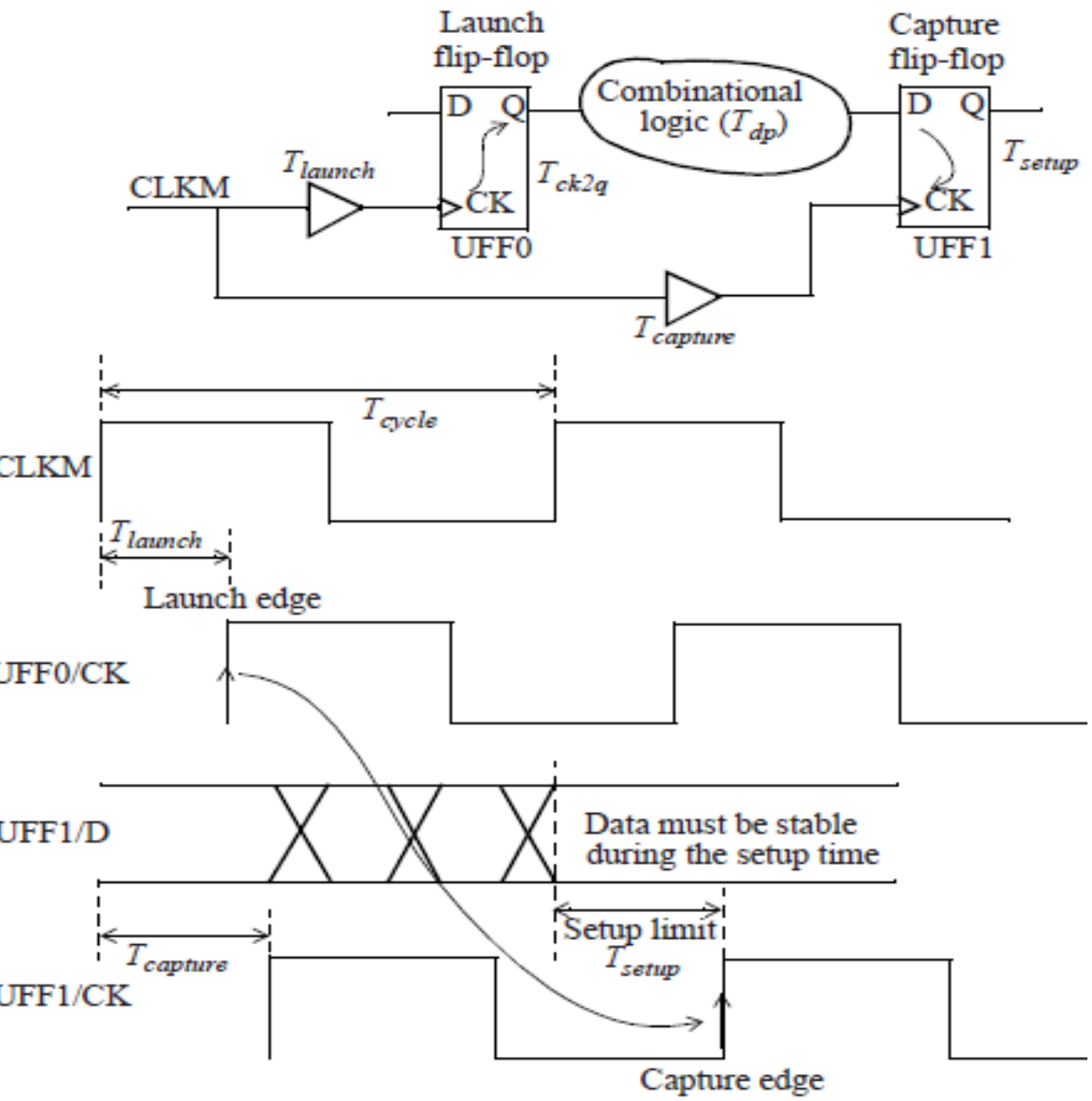
$$T_{launch} + T_{ck2q} + T_{dp} < T_{capture} + T_{cycle} - T_{setup}$$

- Where
- **Tlaunch** is the delay of the clock tree of the launch flip-flop UFF0,
- **Tdp** is the delay of the combinational logic data path and **Tcycle** is the clock period.
- **Tcapture** is the delay of the clock tree for the capture flip-flop UFF1.

Tck2q + Tdp < Tcycle –Tsetup
Tsetup < Tcycle – Tdatapath
Where Tdatapath = Tck2Q + Tdp.
Tcycle –Tdatapath > Tsetup (UFF1)

- The total time it takes for data to arrive at the D pin of the capture flip-flop must be less than the time it takes for the clock to travel to the capture flip-flop plus a clock cycle delay minus the setup time.

Advanced Digital Design
Flip-flop to Flip-flop Path



Startpoint: UFF0 (**rising edge-triggered flip-flop clocked by CLKM**)
Endpoint: UFF1 (**rising edge-triggered flip-flop clocked by CLKM**)
Path Group: CLKM
Path Type: max

Point	Incr	Path
clock CLKM (rise edge)	**0.00**	0.00
clock network delay (ideal)	0.00	0.00
UFF0/CK (DFF)	0.00	0.00 r
UFF0/Q (DFF) <-	0.16	0.16 f
UNOR0/ZN (NR2)	0.04	0.20 r
UBUF4/Z (BUFF)	0.05	0.26 r
UFF1/D (DFF)	0.00	0.26 r
data arrival time		0.26
clock CLKM (rise edge)	**10.00**	10.00
clock network delay (ideal)	0.00	10.00
clock uncertainty	-0.30	9.70
UFF1/CK (DFF)		9.70 r
library setup time	-0.04	9.66
data required time		9.66
data required time		9.66
data arrival time		-0.26
slack (MET)		9.41

Setup Check

```
create_clock -name CLKM -period 10 -waveform {0 5} \
  [get_ports CLKM]
set_clock_uncertainty -setup 0.3 [all_clocks]
set_clock_transition -rise 0.2 [all_clocks]
set_clock_transition -fall 0.15 [all_clocks]
```

- The launch path takes 0.26ns to get to the D pin of flip-flop UFF1 - this is the arrival time at the input of the capture flip-flop.
- The capture edge (which is one cycle away since this is a setup check) is at 10ns. A clock uncertainty of 0.3ns was specified for this clock - thus, the clock period is reduced by the uncertainty margin.
- The clock uncertainty includes the variation in cycle time due to jitter in the clock source and any other timing margin used for analysis. The setup time of the flip-flop 0.04ns (called library setup time), is deducted from the total capture path yielding a required time of 9.66ns.
- Since the arrival time is 0.26ns, there is a positive slack of 9.41ns on this timing path. Note that the difference between the required time and arrival time may appear to be 9.40ns.

Setup Check

- The report shows that the launch flip-flop (specified by Startpoint) has instance name UFF0 and it is triggered by the rising edge of clock CLKM.
- The capture flip-flop (specified by Endpoint) is UFF1 and is also triggered by the rising edge of clock CLKM.
- The Path Type line indicates that the delays shown in this report are all max path delays indicating that this is a setup check. This is because setup checks correspond to the max (or longest path) delays through the logic.
- Note that the hold checks correspond to the min (or shortest path) delays through the logic.
- The ***Incr*** column specifies the incremental cell or net delay for the port or pin indicated. The ***Path*** column shows the cumulative delay for the arrival and the data required paths.

Setup Check: with clock network delay

- What is the clock network delay in the timing report and why is it marked as ideal? This line in the timing report indicates that the clock trees are treated as ideal, that any buffers in the clock path are assumed to have zero delay.
- Once the clock trees are built, the clock network can be marked as propagated which causes the clock paths to show up with real delays, as shown in the next example timing report.
- The 0.11ns delay is the clock network delay on the launch clock and the 0.12ns delay is the clock network delay on the capture flip-flop.

Point	Incr	Path
clock CLKM (rise edge)	0.00	0.00
clock network delay (propagated)	**0.11**	0.11
UFF0/CK (DFF)	0.00	0.11 r
UFF0/Q (DFF) <-	0.14	0.26 f
UNOR0/ZN (NR2)	0.04	0.30 r
UBUF4/Z (BUFF)	0.05	0.35 r
UFF1/D (DFF)	0.00	0.35 r
data arrival time		0.35
clock CLKM (rise edge)	10.00	10.00
clock network delay (propagated)	**0.12**	10.12
clock uncertainty	-0.30	9.82
UFF1/CK (DFF)		9.82 r
library setup time	-0.04	9.78
data required time		9.78
data required time		9.78
data arrival time		-0.35
slack (MET)		9.43

Advanced Digital Design
Setup Check: timing path report with expanded clock paths

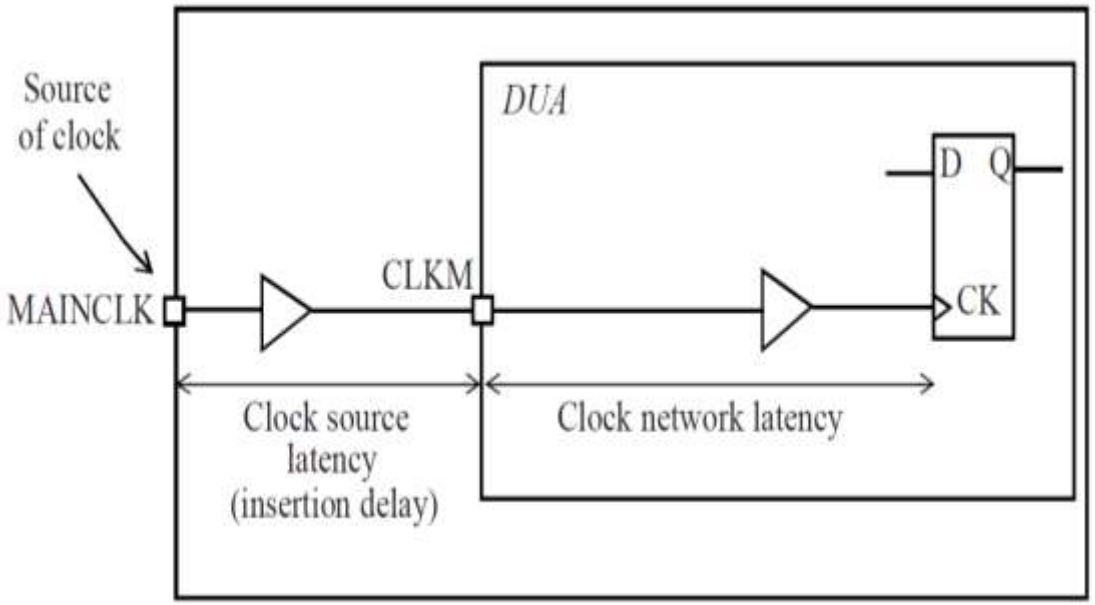
Point	Incr	Path
clock CLKM (rise edge)	0.00	0.00
clock source latency	0.00	0.00
CLKM (in)	**0.00**	**0.00 r**
UCKBUF0/C (CKB)	**0.06**	**0.06 r**
UCKBUF1/C (CKB)	**0.06**	**0.11 r**
UFF0/CK (DFF)	0.00	0.11 r
UFF0/Q (DFF) <-	0.14	0.26 **f**
UNOR0/ZN (NR2)	0.04	0.30 r
UBUF4/Z (BUFF)	0.05	0.35 r
UFF1/D (DFF)	0.00	0.35 **r**
data arrival time		0.35
clock CLKM (rise edge)	10.00	10.00
clock source latency	0.00	10.00
CLKM (in)	**0.00**	**10.00 r**
UCKBUF0/C (CKB)	**0.06**	**10.06 r**
UCKBUF2/C (CKB)	**0.07**	**10.12 r**
UFF1/CK (DFF)	0.00	10.12 r
clock uncertainty	-0.30	9.82
library setup time	-0.04	9.78
data required time		9.78
data required time		9.78
data arrival time		-0.35
slack (MET)		9.43

The cell delay is calculated based on the input transition time and the output capacitance of the cell. Thus, the question is what transition time is used at the input of the first cell in the clock tree. The transition time (or slew) on the input pin of the first clock cell can be explicitly specified using the **set_input_transition** command.

Advanced Digital Design
Clock Source Latency

- What is clock source latency? This is also called insertion delay and is the time it takes for a clock to propagate from its source to the clock definition point of the design under analysis.
- This corresponds to the latency of the clock tree that is outside of the design. For example, if this design were part of a larger block, the clock source latency specifies the delay of the clock tree up to the clock pin of the design under analysis. This latency can be explicitly specified using the **set_clock_latency** command.

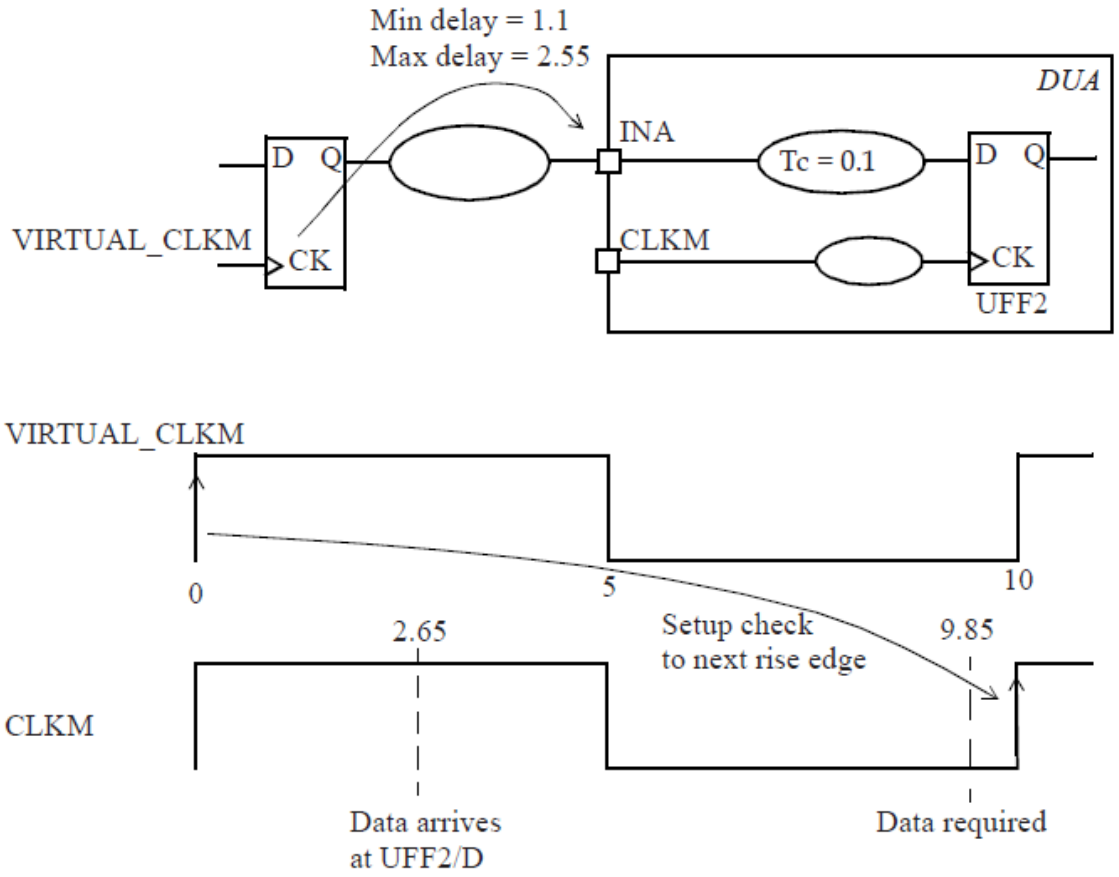
```
set_clock_latency -source -rise 0.7 [get_clocks CLKM]
set_clock_latency -source -fall 0.65 [get_clocks CLKM]
```



Clock Source Latency

- In the absence of such a command, a latency of 0 is assumed. That was the assumption used in earlier path reports.
- Note that the source latency does not affect paths that are internal to the design and have the same launch clock and capture clock.
- This is because the same latency gets added to both the launch clock path and the capture clock path.
- However this latency does impact timing paths that go through the inputs and outputs of the design under analysis.

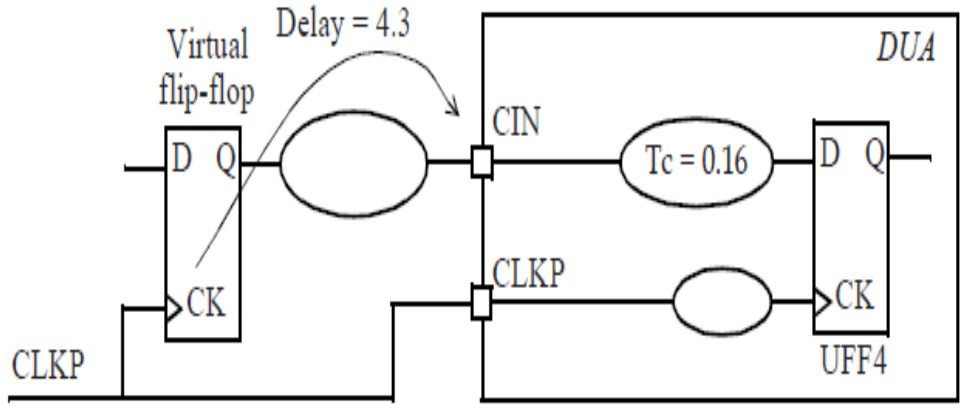
Advanced Digital Design
Input to Flip-flop Path

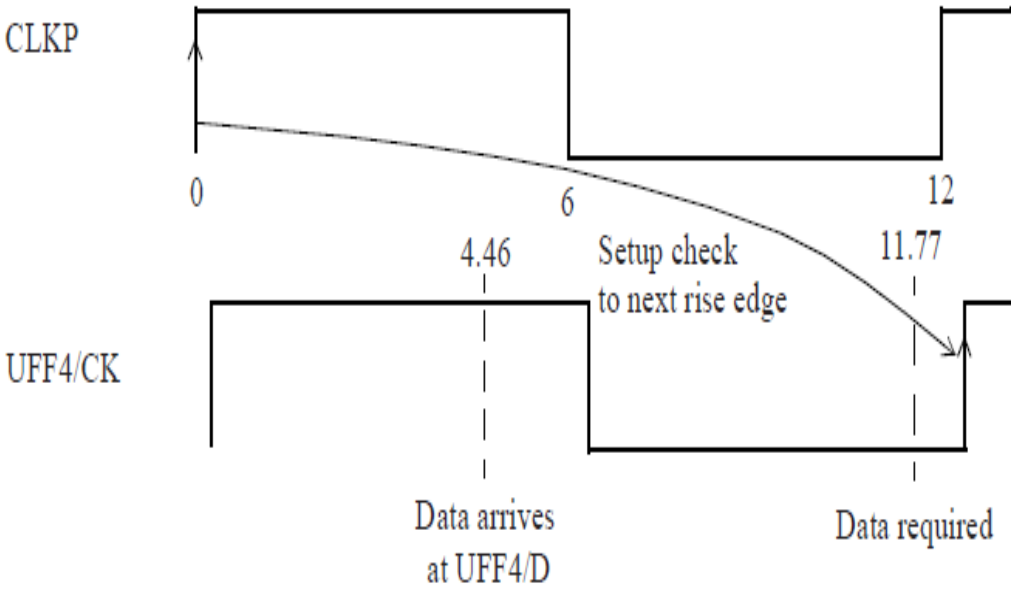



```
create_clock -name VIRTUAL_CLKM -period 10 -waveform {0 5}
set_input_delay -clock VIRTUAL_CLKM \
  -max 2.55 [get_ports INA]
```

```
Startpoint: INA (input port clocked by VIRTUAL_CLKM)
Endpoint: UFF2 (rising edge-triggered flip-flop clocked by CLKM)
Path Group: CLKM
Path Type: max

Point                                              Incr       Path
-----------------------------------------------------------------------
clock VIRTUAL_CLKM (rise edge)                     0.00       0.00
clock network delay (ideal)                        0.00       0.00
input external delay                               2.55       2.55 f

INA (in) <-                                        0.00       2.55 f
UINV1/ZN (INV  )                                   0.02       2.58 r
UAND0/Z (AN2  )                                    0.06       2.63 r
UINV2/ZN (INV  )                                   0.02       2.65 f
UFF2/D (DFF )                                      0.00       2.65 f
data arrival time                                             2.65

clock CLKM (rise edge)                            10.00      10.00
clock source latency                               0.00      10.00
CLKM (in)                                          0.00      10.00 r
UCKBUF0/C (CKB  )                                  0.06      10.06 r
UCKBUF2/C (CKB  )                                  0.07      10.12 r
UCKBUF3/C (CKB  )                                  0.06      10.18 r
UFF2/CK (DFF )                                     0.00      10.18 r
clock uncertainty                                 -0.30       9.88
library setup time                                -0.03       9.85
data required time                                            9.85
-----------------------------------------------------------------------
data required time                                            9.85
data arrival time                                            -2.65
-----------------------------------------------------------------------
slack (MET)                                                   7.20
```

Advanced Digital Design
Input Path with Actual Clock





- Input arrival times can be specified with respect to an actual clock also; these do not necessarily have to be specified with respect to a virtual clock.
- Examples of actual clocks are clocks on internal pins in the design, or on input ports. Figure 8-5 depicts an example where the input constraint on port CIN is specified relative to a clock on input port CLKP. This constraint is specified as:

```
set_input_delay -clock CLKP -max 4.3 [get_ports CIN]
```

Advanced Digital Design

Input Path with Actual Clock

```
Startpoint: CIN (input port clocked by CLKP)
Endpoint: UFF4 (rising edge-triggered flip-flop clocked by CLKP)
Path Group: CLKP
Path Type: max

Point                                          Incr       Path
-----------------------------------------------------------------
clock CLKP (rise edge)                         0.00       0.00
clock network delay (propagated)               0.00       0.00
input external delay                           4.30       4.30 f
CIN (in)                                       0.00       4.30 f
UBUF5/Z (BUFF  )                               0.06       4.36 f
UXOR1/Z (XOR2  )                               0.10       4.46 r
UFF4/D (DFF )                                  0.00       4.46 r
data arrival time                                         4.46

clock CLKP (rise edge)                        12.00      12.00
clock source latency                           0.00      12.00

CLKP (in)                                      0.00      12.00 r
UCKBUF4/C (CKB  )                              0.06      12.06 r
UCKBUF5/C (CKB  )                              0.06      12.12 r
UFF4/CK (DFF )                                 0.00      12.12 r
clock uncertainty                             -0.30      11.82
library setup time                            -0.05      11.77
data required time                                       11.77
-----------------------------------------------------------------
data required time                                       11.77
data arrival time                                        -4.46
-----------------------------------------------------------------
slack (MET)                                               7.31
```

Advanced Digital Design
Flip-flop to output Path

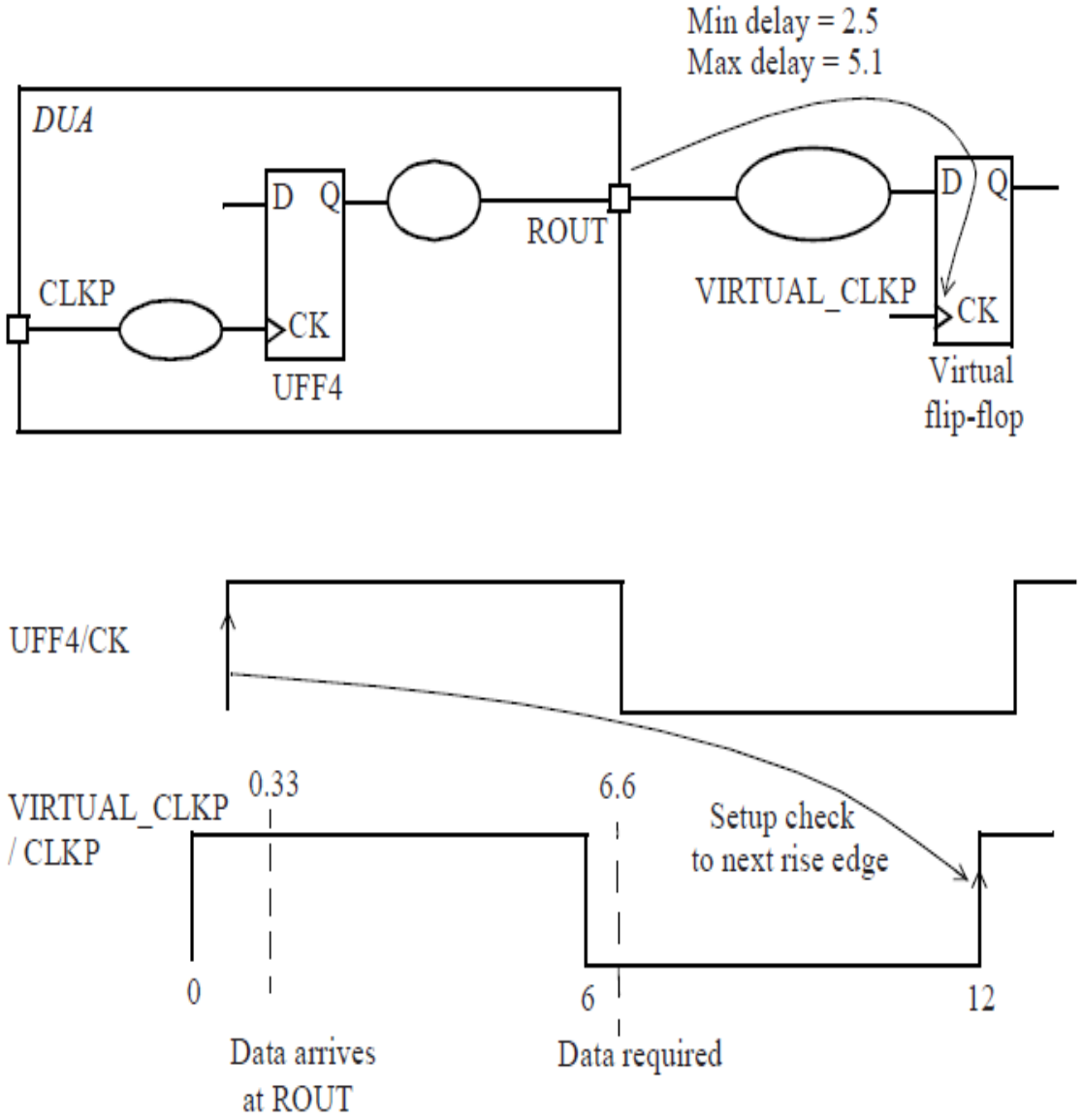
- Output port can be constrained either with respect to a virtual clock, or an internal clock of the design, or an input clock port, or an output clock port. Here is an example that shows the output pin ROUT constrained with respect to a virtual clock. The output constraint is as follows:

```
set_output_delay -clock VIRTUAL_CLKP \
  -max 5.1 [get_ports ROUT]
set_load 0.02 [get_ports ROUT]
```

- To determine the delay of the last cell connected to the output port correctly, one needs to specify the load on this port. The output load is specified above using the **set_load** command.
- Note that the port ROUT may have load contribution internal to the DUA and the **set_load** specification provides the additional load, which is the load contribution from outside the DUA.

Advanced Digital Design

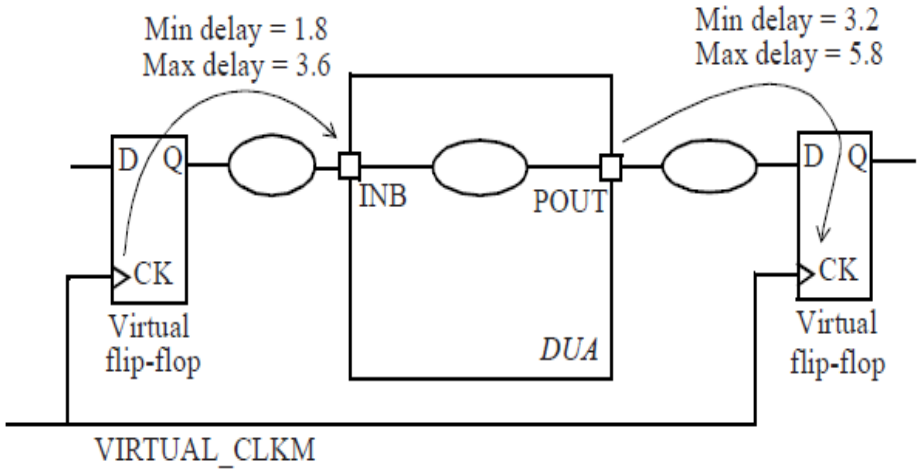
Flip-flop to output Path: Setup check for path through output port

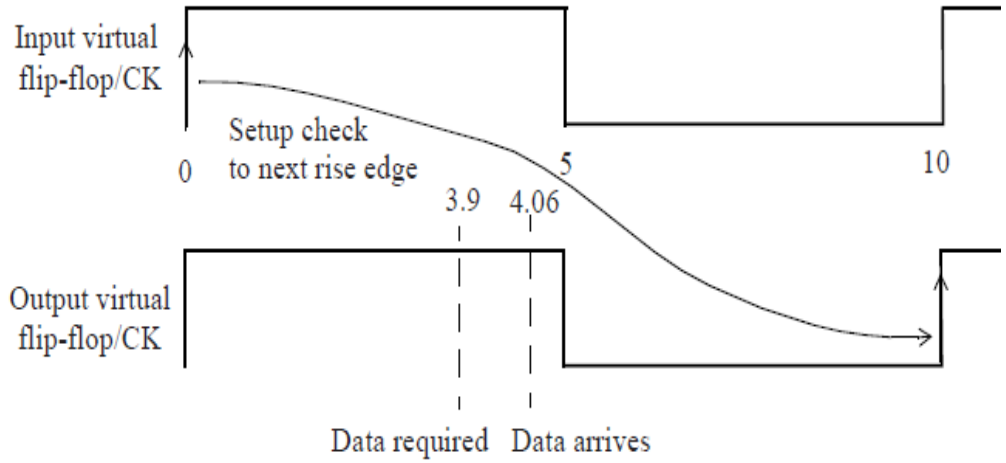



```
Startpoint: UFF4 (rising edge-triggered flip-flop clocked by CLKP)
  Endpoint: ROUT (output port clocked by VIRTUAL_CLKP)
  Path Group: VIRTUAL_CLKP
  Path Type: max

  Point                                              Incr       Path
  -----------------------------------------------------------------------
  clock CLKP (rise edge)                             0.00       0.00
  clock source latency                               0.00       0.00
  CLKP (in)                                          0.00       0.00 r

  UCKBUF4/C (CKB  )                                  0.06       0.06 r
  UCKBUF5/C (CKB  )                                  0.06       0.12 r
  UFF4/CK (DFF )                                     0.00       0.12 r
  UFF4/Q (DFF )                                      0.13       0.25 r
  UBUF3/Z (BUFF  )                                   0.09       0.33 r
  ROUT (out)                                         0.00       0.33 r
  data arrival time                                             0.33

  clock VIRTUAL_CLKP (rise edge)                    12.00      12.00
  clock network delay (ideal)                        0.00      12.00
  clock uncertainty                                 -0.30      11.70
  output external delay                             -5.10       6.60
  data required time                                            6.60
  -----------------------------------------------------------------------
  data required time                                            6.60
  data arrival time                                            -0.33
  -----------------------------------------------------------------------
  slack (MET)                                                   6.27
```

Advanced Digital Design

Input to Output Path: Combinational path from input to output port



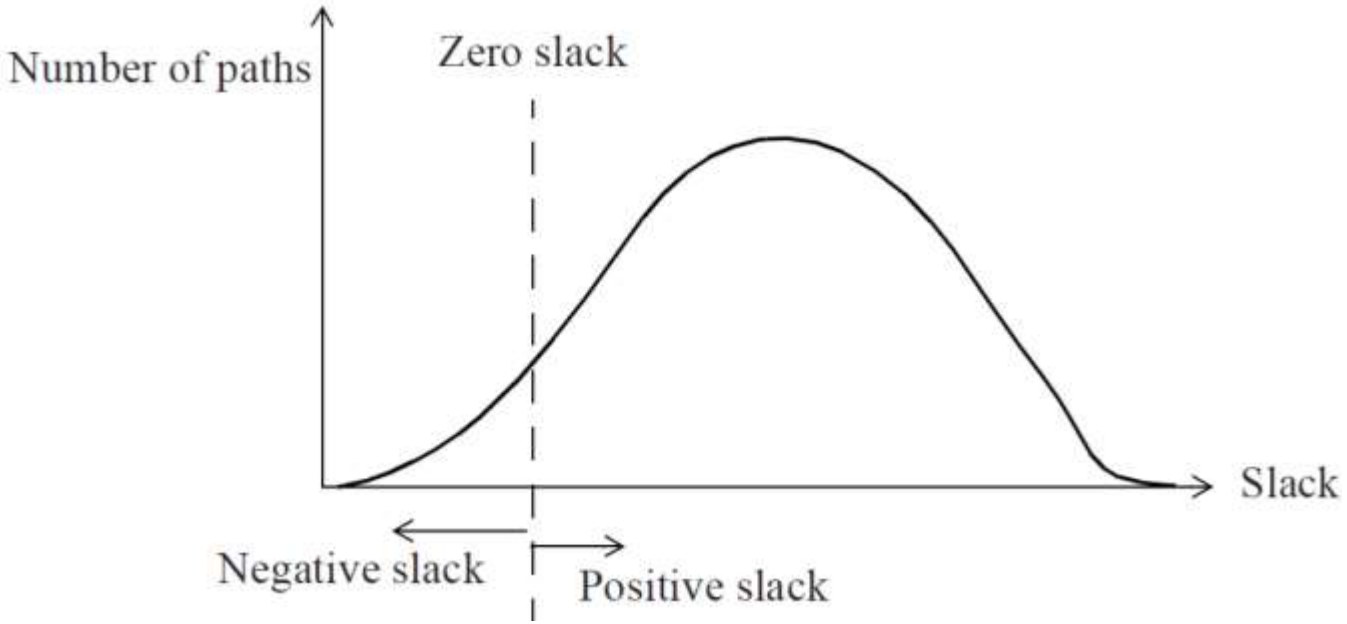



```
set_input_delay -clock VIRTUAL_CLKM \
  -max 3.6 [get_ports INB]
set_output_delay -clock VIRTUAL_CLKM \
  -max 5.8 [get_ports POUT]
```

```
Startpoint: INB (input port clocked by VIRTUAL_CLKM)
Endpoint: POUT (output port clocked by VIRTUAL_CLKM)
Path Group: VIRTUAL_CLKM
Path Type: max

Point                                              Incr       Path
-----------------------------------------------------------------------
clock VIRTUAL_CLKM (rise edge)                     0.00       0.00
clock network delay (ideal)                        0.00       0.00
input external delay                               3.60       3.60 f
INB (in) <-                                        0.00       3.60 f
UBUF0/Z (BUFF  )                                   0.05       3.65 f
UBUF1/Z (BUFF  )                                   0.06       3.72 f
UINV3/ZN (INV  )                                   0.34       4.06 r
POUT (out)                                         0.00       4.06 r
data arrival time                                             4.06

clock VIRTUAL_CLKM (rise edge)                    10.00      10.00
clock network delay (ideal)                        0.00      10.00
clock uncertainty                                 -0.30       9.70
output external delay                             -5.80       3.90
data required time                                            3.90
-----------------------------------------------------------------------
data required time                                            3.90
data arrival time                                            -4.06
-----------------------------------------------------------------------
slack (VIOLATED)                                             -0.16
```

Frequency Histogram

- Frequency histogram of setup slack versus number of paths for a typical design can be plotted.
- Depending upon the state of the design, whether it has been optimized or not, the zero slack line would be more towards the right for an unoptimized design and more towards the left for an optimized design.
- For a design that has zero violations, that is no paths with negative slack, the entire curve would be to the right of the zero slack line.



Frequency Histogram

- Designs that are tough to meet timing would have their hump of the histogram more towards the left, that is, have many paths with slack closer to zero.

- One other observation that can be made by looking at a frequency histogram is on the ability to further optimize the design to achieve zero slack, that is, how difficult it is to close timing.

- If the number of failing paths is small and the negative slack is also small, the design is relatively close to meeting the required timing.

- However, if the number of failing paths is large and the negative slack magnitude is also large, this implies that the design would require a lot of effort to meet the required timing.

Summary

- Setup Time Check
- Setup Check: with clock network delay
- Input Path with Actual Clock
- Flip-flop to output Path: Setup check for path through output port
- Input to Output Path: Combinational path from input to output port
- Frequency Histogram

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

ADVANCED DIGITAL DESIGN

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Hold Time Check

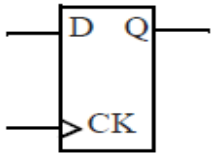
Sudeendra kumar K
Department of Electronics and Communication Engineering

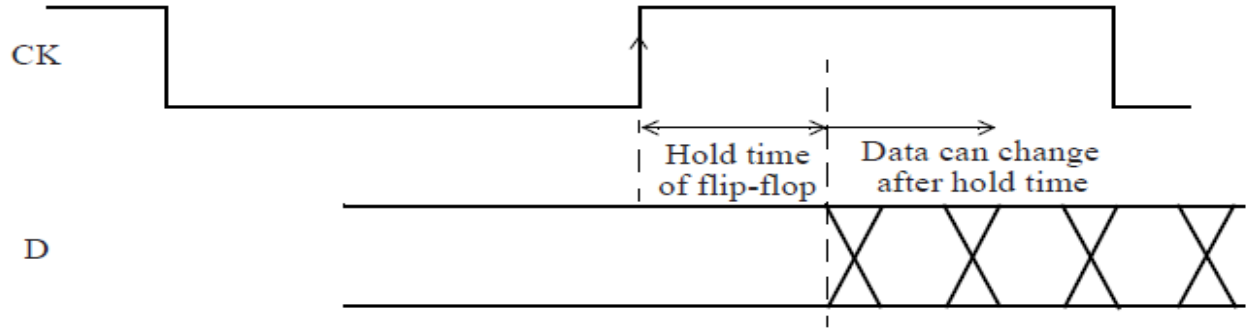
Contents

- Hold Time Check

Hold Check

- A hold timing check ensures that a flip-flop output value that is changing does not pass through to a capture flip-flop and overwrite its output before the flip-flop has had a chance to capture its original value.
- This check is based on the hold requirement of a flip-flop. The hold specification of a flip-flop requires that the data being latched should be held stable for a specified amount of time after the active edge of the clock.

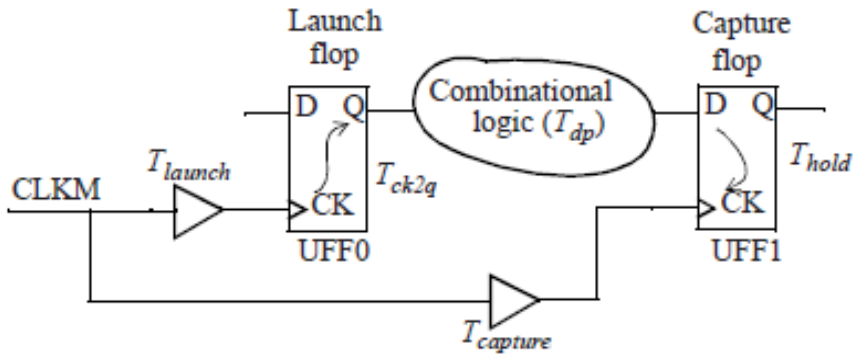


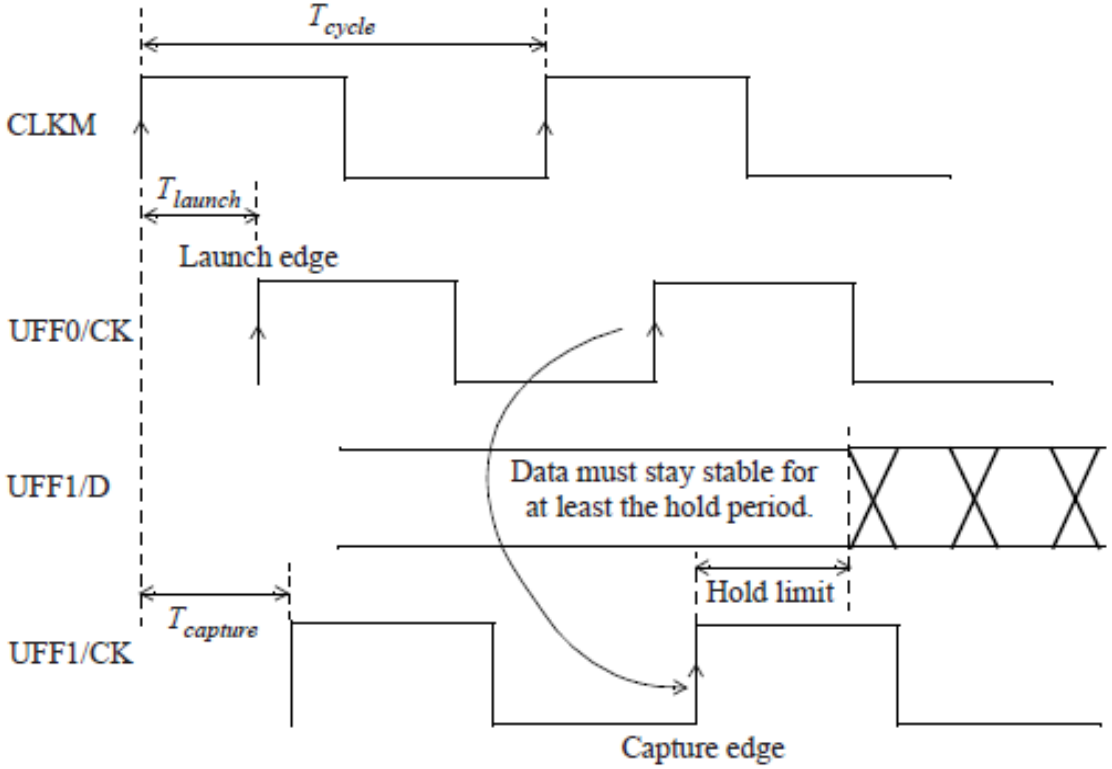


Hold Timing Check

- A hold timing check is between the launch flipflop - the flip-flop that launches the data, and the capture flip-flop – the flip-flop that captures the data and whose hold time must be satisfied.
- The clocks to these two flip-flops can be the same or can be different. The hold check is from one active edge of the clock in the launch flip-flop to the same clock edge at the capture flip-flop.
- Thus, a hold check is independent of the clock period. The hold check is carried out on each active edge of the clock of the capture flip-flop.

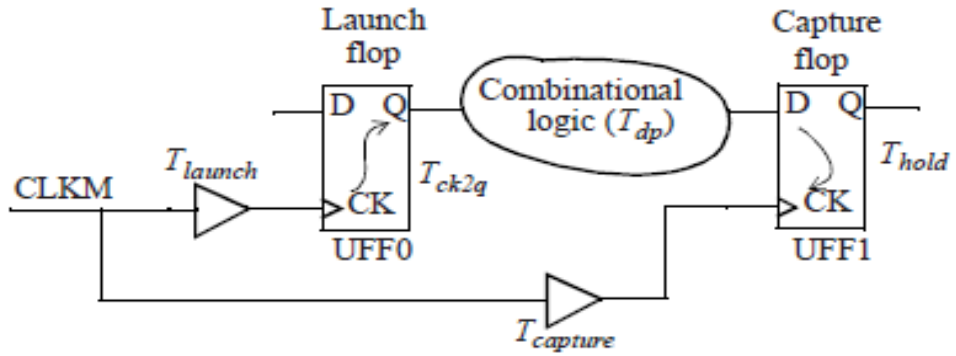
Data and clock signals for Hold timing check.

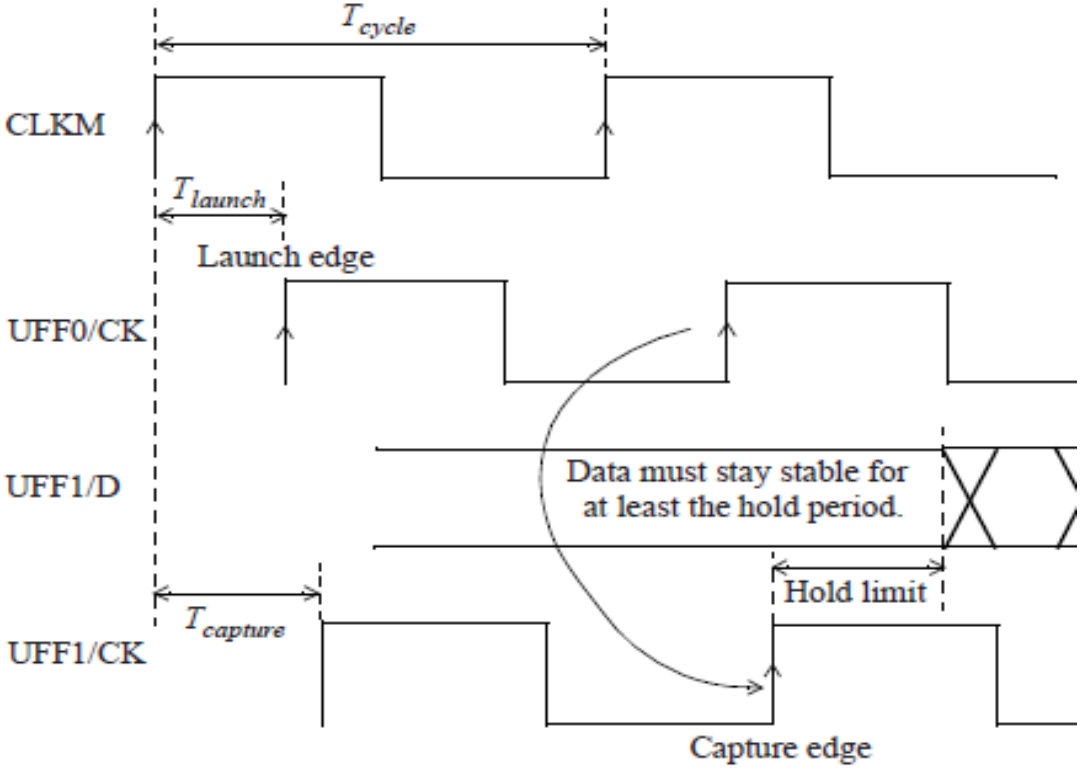




- Consider the second rising edge of clock CLKM.
- The data launched by the rising edge of the clock takes : **Tlaunch + Tck2q + Tdp** time to get to the D pin of the capture flip-flop UFF1.
- The same edge of the clock takes **Tcapture** time to get to the clock pin of the capture flip-flop.
- The intention is for the data from the launch flip-flop to be captured by the capture flip-flop in the next clock cycle.
- If the data is captured in the same clock cycle, the intended data in the capture flip-flop (from the previous clock cycle) is overwritten. The hold time check is to ensure that the intended data in the capture flipflop is not overwritten.

Data and clock signals for Hold timing check.





- The hold time check verifies that the difference between these two times (data arrival time and clock arrival time at capture flip-flop) must be larger than the hold time of the capture flip-flop, so that the previous data on the flip-flop is not overwritten and the data is reliably captured in the flip-flop.

$$T_{launch} + T_{ck2q} + T_{dp} > T_{capture} + T_{hold}$$

- In other words, the total time required for data launched by a clock edge to arrive at the D pin of the capture flipflop must be larger than the time required for the same edge of the clock to travel to the capture flip-flop plus the hold time.
- This ensures that UFF1/D remains stable until the hold time of the flip-flop after the rising edge of the clock on its clock pin UFF1/CK.

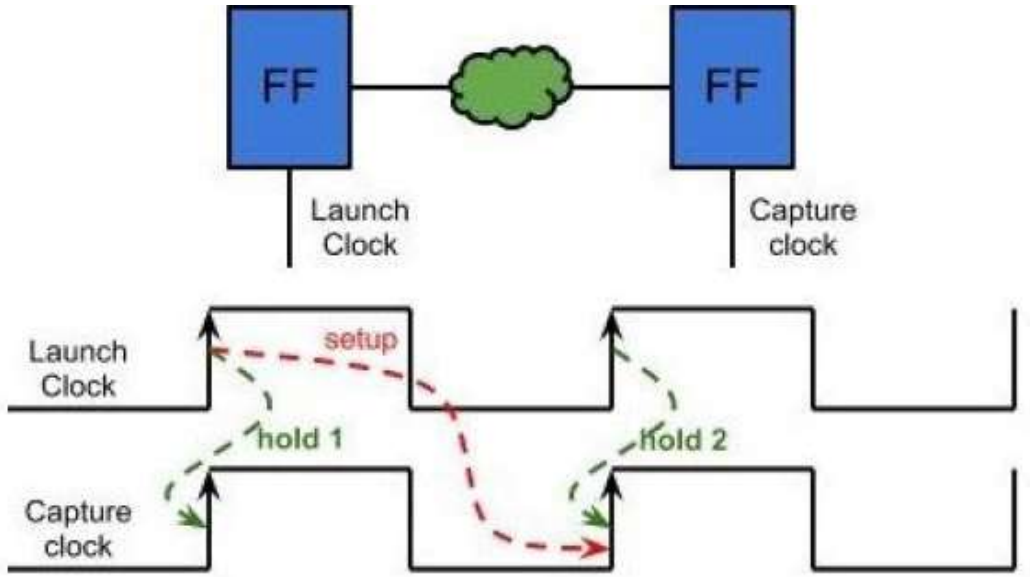
Hold Time Check

- The hold checks impose a lower bound or min constraint for paths to the data pin on the capture flip-flop; the fastest path to the D pin of the capture flip-flop needs to be determined. Thus, the hold checks are typically performed at the fast timing corner.
- To ensure reliable data capture, the clock edge at the capture flip-flop must arrive before the data can change. A hold timing check ensures that : -
  - Data from the subsequent launch edge must not be captured by the setup receiving edge.
  - Data from the setup launch edge must not be captured by the preceding receiving edge.
- These two hold checks are essentially the same if both the launch and capture clock belong to the same clock domain.

Courtesy: STA by Bhasker

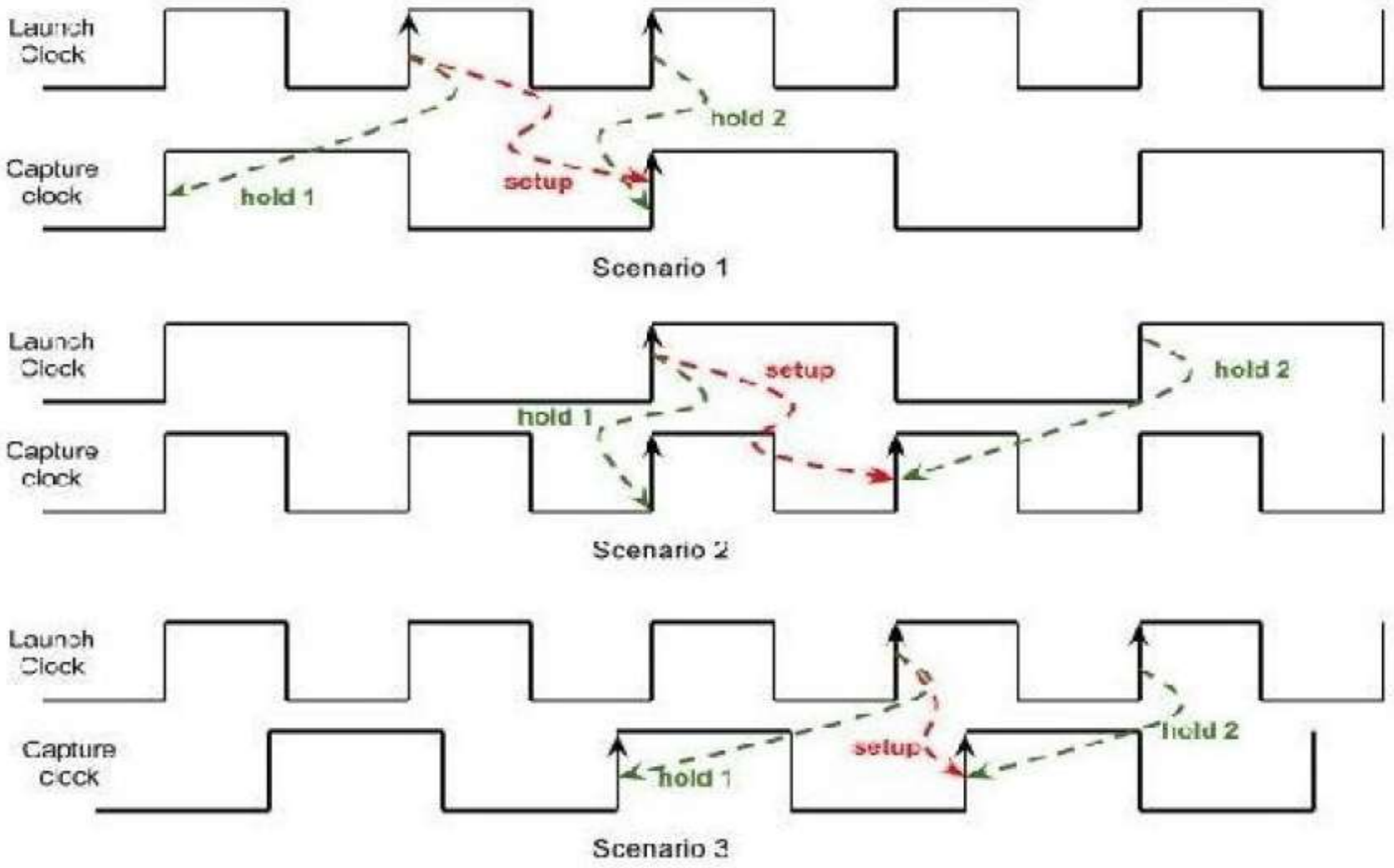
Advanced Digital Design

Waveforms showing Setup and Hold Edges



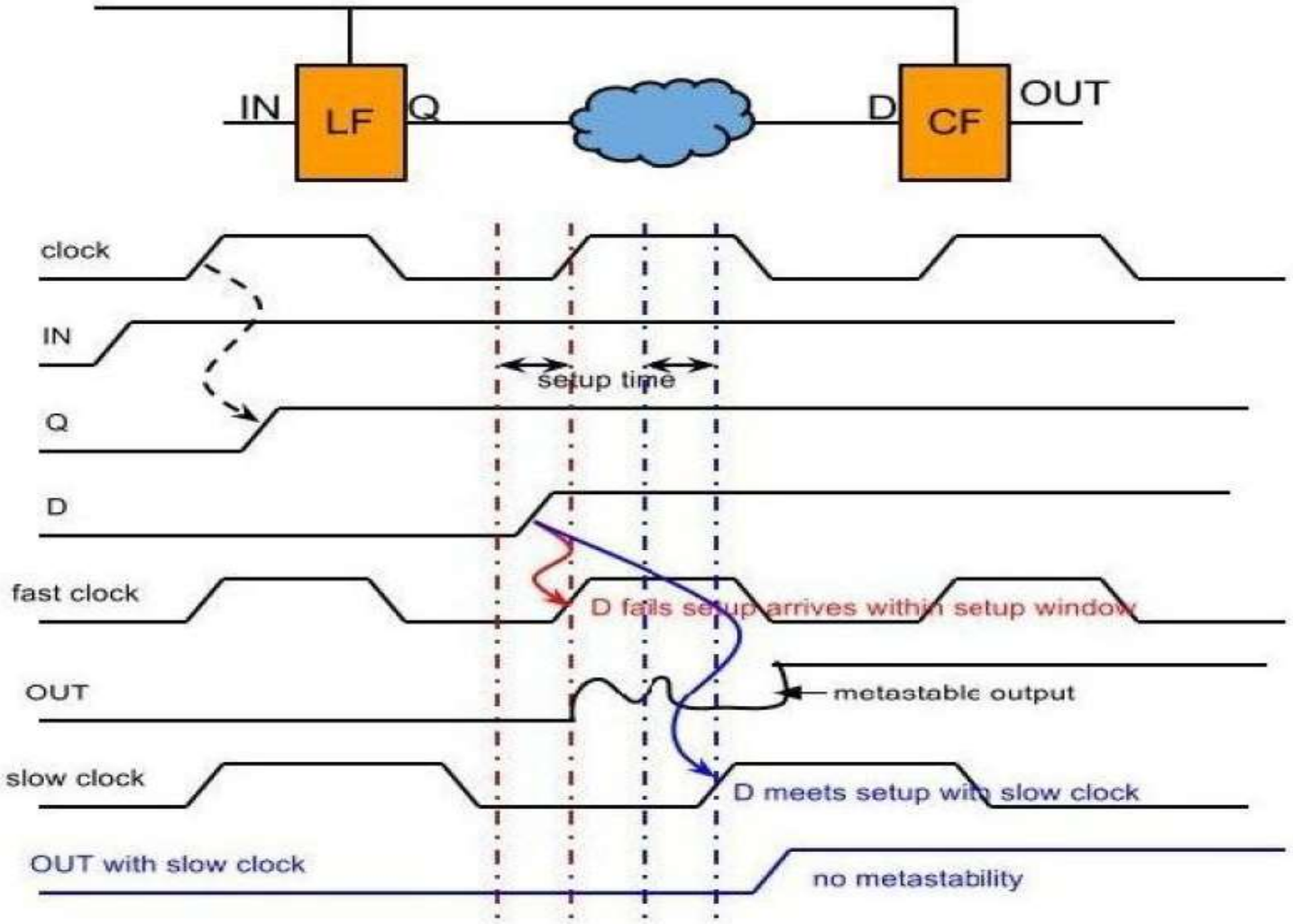
Advanced Digital Design

Waveforms showing Setup and Hold Edges: Different Scenarios



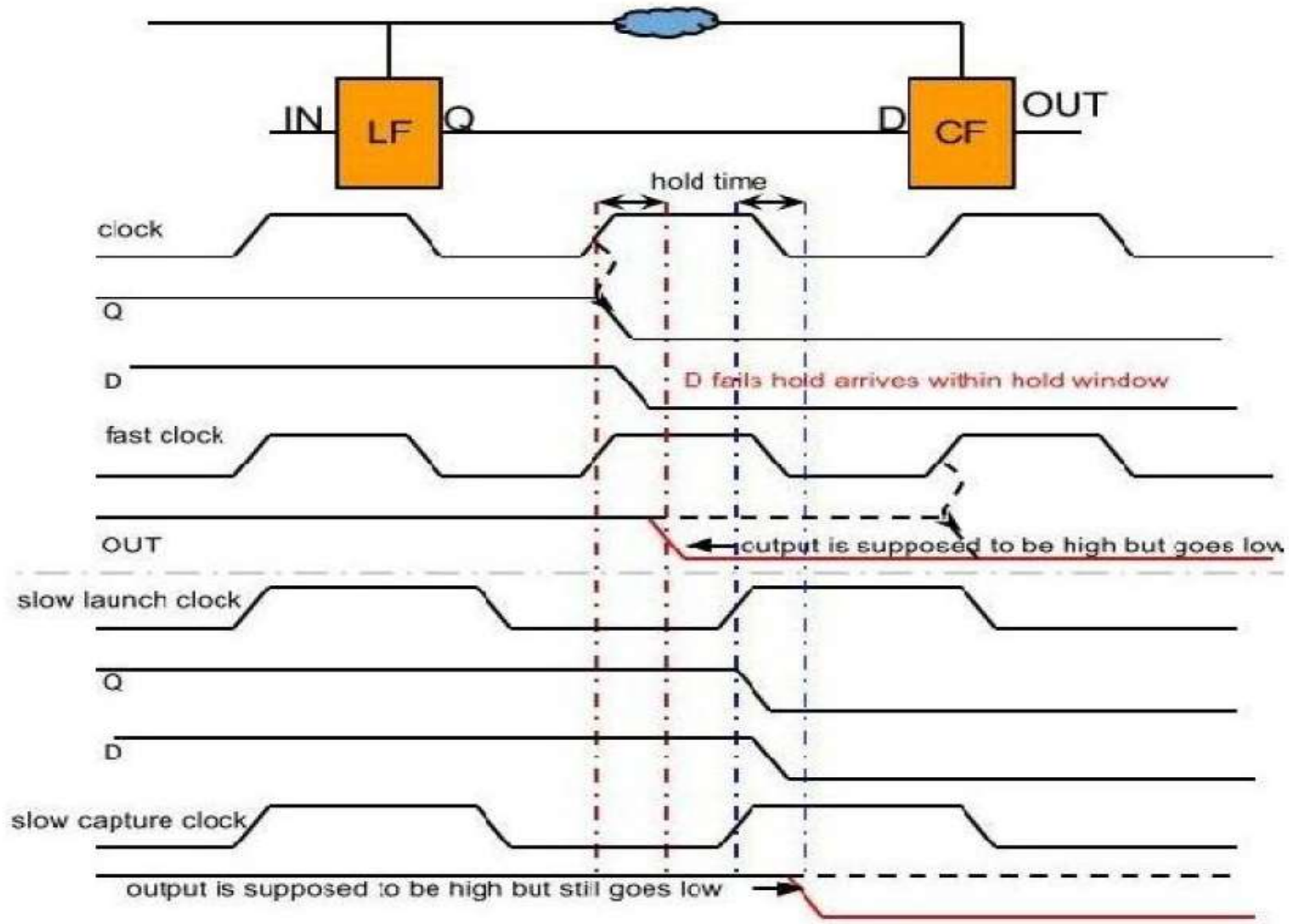
Advanced Digital Design

Example: Waveforms showing Setup Time Failure



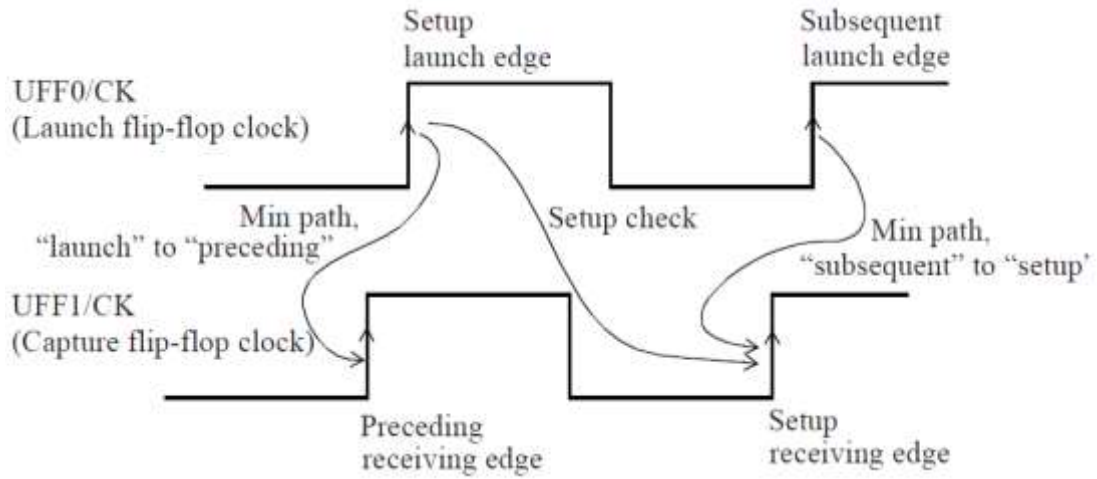
Advanced Digital Design

Example: Waveforms showing Hold Time Failure



Advanced Digital Design

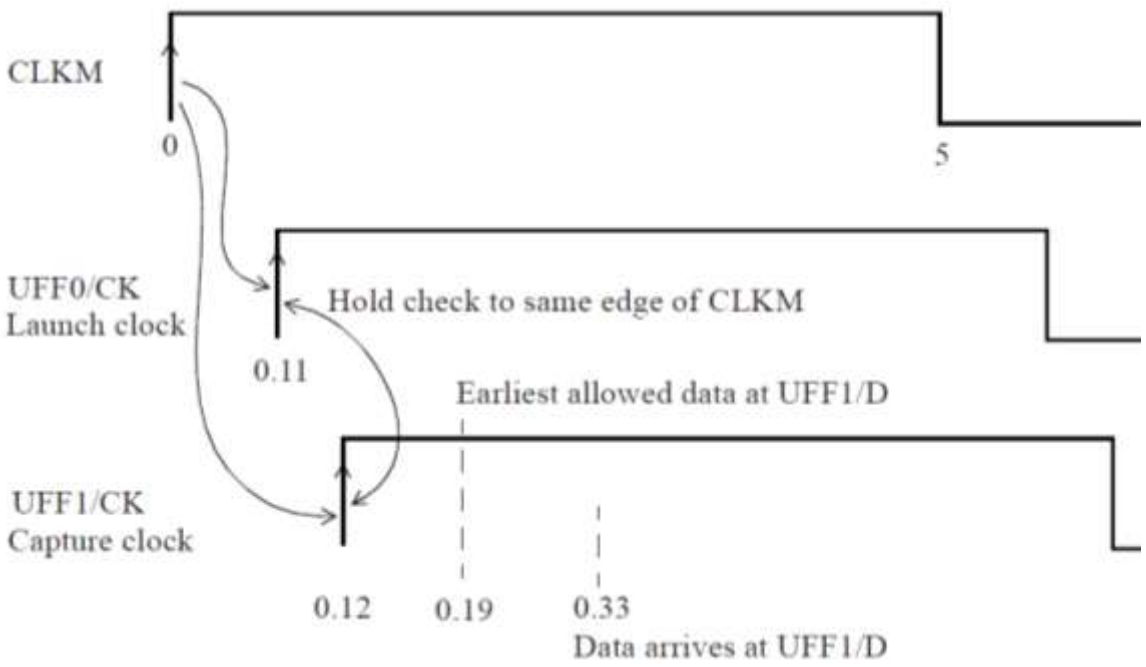
Hold Check

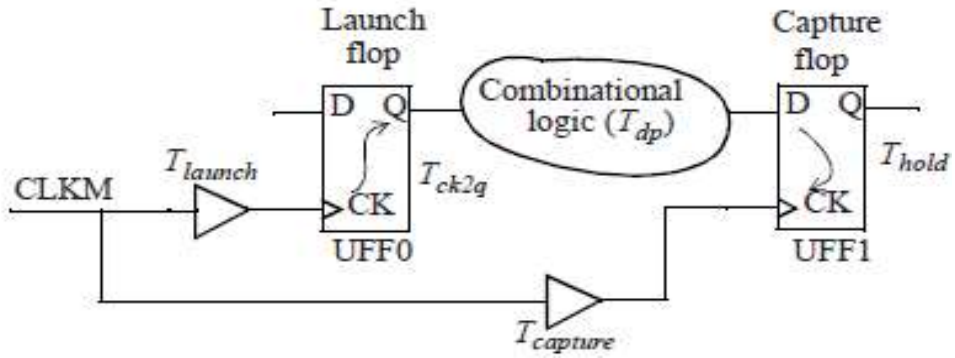


- UFF0 is the launch flip-flop and UFF1 is the capture flip-flop.
- The setup check is between the setup launch edge and the setup receiving edge.
- The subsequent launch edge must not propagate data so fast that the setup receiving edge does not have time to capture its data reliably.
- In addition, the setup launch edge must not propagate data so fast that the preceding receiving edge does not get a chance to capture its data. The worst hold check corresponds to the most restrictive hold check amongst various scenarios.

Advanced Digital Design
Flipflop to Flipflop Path

```
Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)
  Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)
  Path Group: CLKM
  Path Type: min

  Point                                              Incr       Path
  -----------------------------------------------------------------------
  clock CLKM (rise edge)                             0.00       0.00
  clock source latency                               0.00       0.00
  CLKM (in)                                          0.00       0.00 r
  UCKBUF0/C (CKB  )                                  0.06       0.06 r
  UCKBUF1/C (CKB  )                                  0.06       0.11 r
  UFF0/CK (DFF )                                     0.00       0.11 r
  UFF0/Q (DFF ) <-                                   0.14       0.26 r
  UNOR0/ZN (NR2  )                                   0.02       0.28 f
  UBUF4/Z (BUFF  )                                   0.06       0.33 f
  UFF1/D (DFF )                                      0.00       0.33 f
  data arrival time                                             0.33

  clock CLKM (rise edge)                             0.00       0.00
  clock source latency                               0.00       0.00
  CLKM (in)                                          0.00       0.00 r
  UCKBUF0/C (CKB  )                                  0.06       0.06 r
  UCKBUF2/C (CKB  )                                  0.07       0.12 r
  UFF1/CK (DFF )                                     0.00       0.12 r
  clock uncertainty                                  0.05       0.17
  library hold time                                  0.01       0.19
  data required time                                            0.19
  -----------------------------------------------------------------------
  data required time                                            0.19
  data arrival time                                            -0.33
  -----------------------------------------------------------------------
  slack (MET)                                                   0.14
```





- The timing report shows that the earliest time the new data can arrive at UFF1 while enabling the previous data to be safely captured is 0.19ns. Since the new data arrives at 0.33ns, the report shows a positive hold slack of 0.14ns.

Hold Slack Calculation

- In the setup timing reports, the arrival time and the required time are computed and the slack is computed to be the required time minus arrival time.
- In the hold timing reports, when we compute required time minus arrival time, a negative result translates into a positive slack (means hold constraint is satisfied), while a positive result translates into a negative slack (means hold constraint is not satisfied).

Advanced Digital Design
Hold Slack Calculation

Startpoint: UFF0 (**rising edge-triggered flip-flop clocked by CLKM**)
Endpoint: UFF1 (**rising edge-triggered flip-flop clocked by CLKM**)
Path Group: CLKM
Path Type: max

Point	Incr	Path
clock CLKM (rise edge)	**0.00**	0.00
clock network delay (ideal)	0.00	0.00
UFF0/CK (DFF)	0.00	0.00 r
UFF0/Q (DFF) <-	0.16	0.16 f
UNOR0/ZN (NR2)	0.04	0.20 r
UBUF4/Z (BUFF)	0.05	0.26 r
UFF1/D (DFF)	0.00	0.26 r
data arrival time		0.26
clock CLKM (rise edge)	**10.00**	10.00
clock network delay (ideal)	0.00	10.00
clock uncertainty	-0.30	9.70
UFF1/CK (DFF)		9.70 r
library setup time	-0.04	9.66
data required time		9.66
data required time		9.66
data arrival time		-0.26
slack (MET)		9.41

Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)
Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)
Path Group: CLKM
Path Type: min

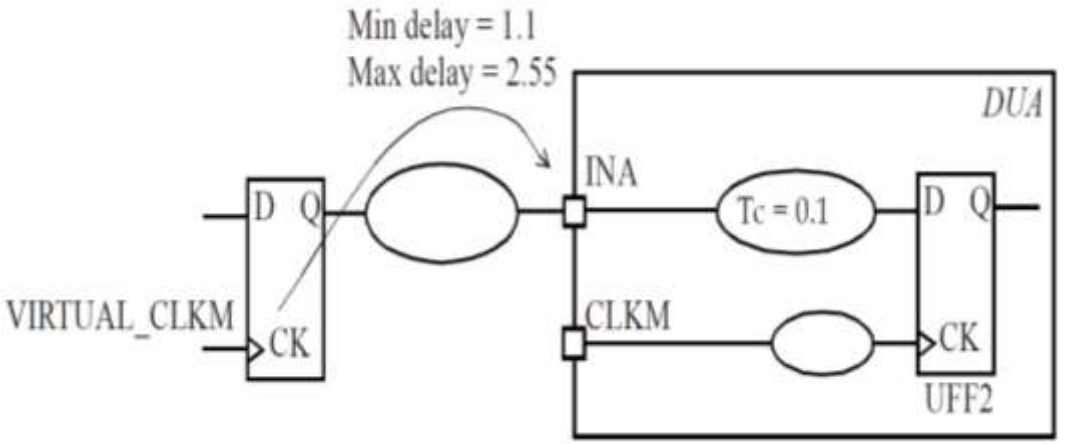
Point	Incr	Path
clock CLKM (rise edge)	**0.00**	0.00
clock source latency	0.00	0.00
CLKM (in)	0.00	0.00 r
UCKBUF0/C (CKB)	0.06	0.06 r
UCKBUF1/C (CKB)	0.06	0.11 r
UFF0/CK (DFF)	0.00	0.11 r
UFF0/Q (DFF) <-	0.14	0.26 r
UNOR0/ZN (NR2)	0.02	0.28 f
UBUF4/Z (BUFF)	0.06	0.33 f
UFF1/D (DFF)	0.00	0.33 f
data arrival time		0.33
clock CLKM (rise edge)	**0.00**	0.00
clock source latency	0.00	0.00
CLKM (in)	0.00	0.00 r
UCKBUF0/C (CKB)	0.06	0.06 r
UCKBUF2/C (CKB)	0.07	0.12 r
UFF1/CK (DFF)	0.00	0.12 r
clock uncertainty	0.05	0.17
library hold time	**0.01**	0.19
data required time		0.19
data required time		0.19
data arrival time		-0.33
slack (MET)		0.14

Advanced Digital Design
Input to Flip-flop Path

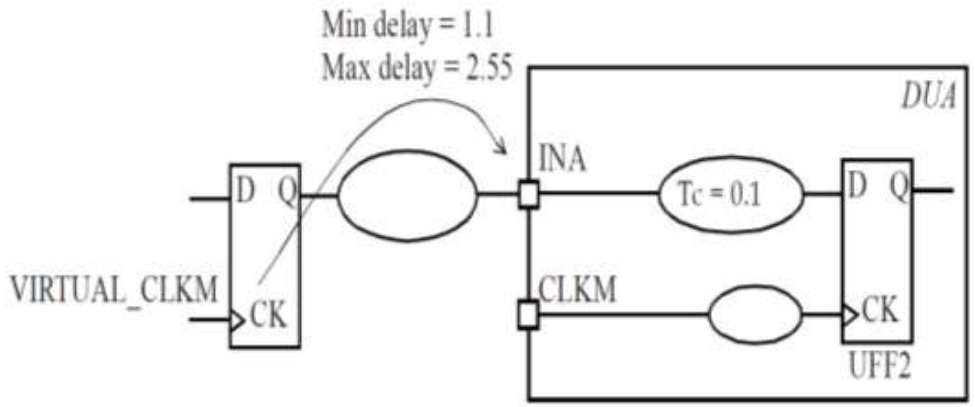
```
Startpoint: INA (input port clocked by VIRTUAL_CLKM)
Endpoint: UFF2 (rising edge-triggered flip-flop clocked by CLKM)

Path Group: CLKM
Path Type: min

Point                                              Incr       Path
-----------------------------------------------------------------------
clock VIRTUAL_CLKM (rise edge)                     0.00       0.00
clock network delay (ideal)                        0.00       0.00
input external delay                               1.10       1.10 f
INA (in) <-                                        0.00       1.10 f
UINV1/ZN (INV  )                                   0.02       1.13 r
UAND0/Z (AN2  )                                    0.06       1.18 r
UINV2/ZN (INV  )                                   0.02       1.20 f
UFF2/D (DFF )                                      0.00       1.20 f
data arrival time                                             1.20

clock CLKM (rise edge)                             0.00       0.00
clock source latency                               0.00       0.00
CLKM (in)                                          0.00       0.00 r
UCKBUF0/C (CKB  )                                  0.06       0.06 r
UCKBUF2/C (CKB  )                                  0.07       0.12 r
UCKBUF3/C (CKB  )                                  0.06       0.18 r
UFF2/CK (DFF )                                     0.00       0.18 r
clock uncertainty                                  0.05       0.23
library hold time                                  0.01       0.25
data required time                                            0.25
-----------------------------------------------------------------------
data required time                                            0.25
data arrival time                                            -1.20
-----------------------------------------------------------------------
slack (MET)                                                   0.95
```

- The min delay on the input port is specified using a virtual clock as:

```
set_input_delay -clock VIRTUAL_CLKM \
  -min 1.1 [get_ports INA]
```

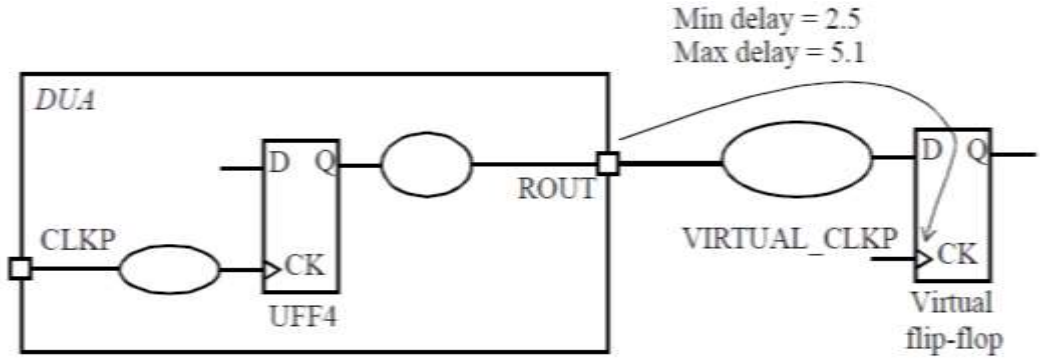


Advanced Digital Design
Input to Flip-flop Path



- The **set_input_delay** appears as input external delay. The hold check is done at time 0 between rising edge of VIRTUAL_CLKM and rising edge of CLKM.
- The required arrival time for data to be captured by UFF2 without violating its hold time is 0.25ns - this indicates that the data should arrive after 0.25ns. Since the data can arrive only at 1.2ns, this shows a positive slack of 0.95ns.

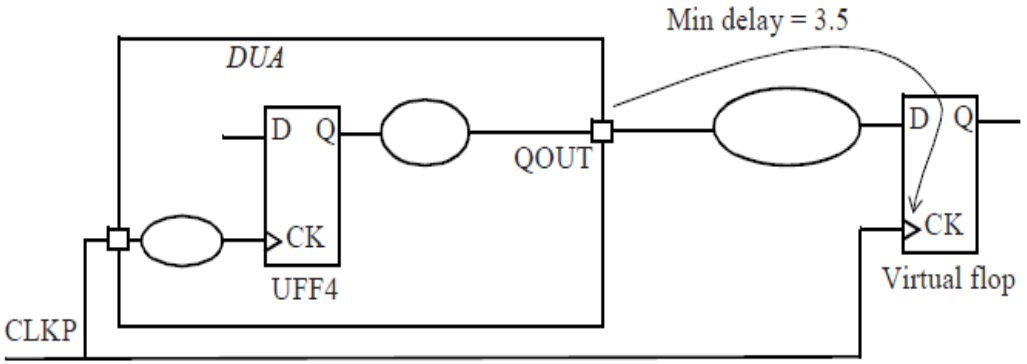
Advanced Digital Design
Flip-flop to Output Path

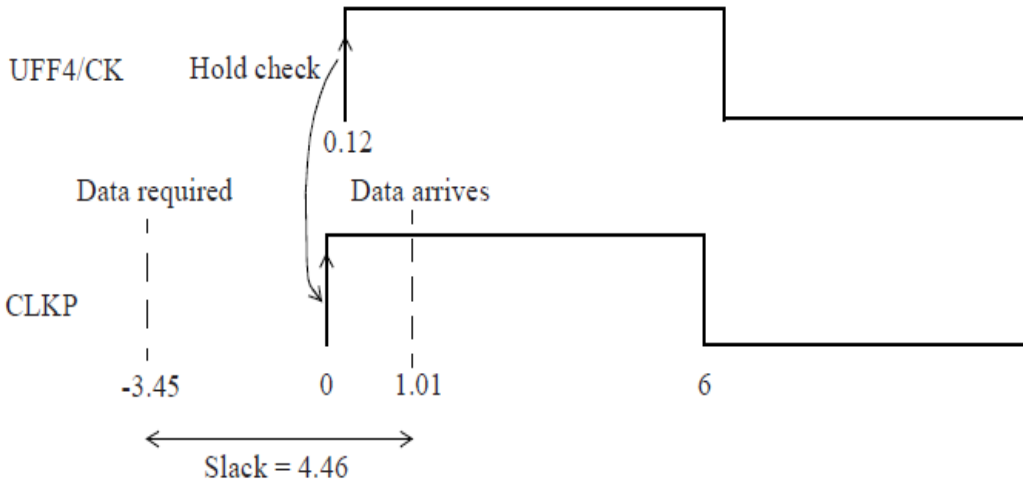



```
set_output_delay -clock VIRTUAL_CLKP \
  -min 2.5 [get_ports ROUT]
```

```
Startpoint: UFF4 (rising edge-triggered flip-flop clocked by CLKP)
  Endpoint: ROUT (output port clocked by VIRTUAL_CLKP)
  Path Group: VIRTUAL_CLKP
  Path Type: min

  Point                                              Incr       Path
  -----------------------------------------------------------------------
  clock CLKP (rise edge)                             0.00       0.00
  clock source latency                               0.00       0.00
  CLKP (in)                                          0.00       0.00 r
  UCKBUF4/C (CKB  )                                  0.06       0.06 r
  UCKBUF5/C (CKB  )                                  0.06       0.12 r
  UFF4/CK (DFF )                                     0.00       0.12 r
  UFF4/Q (DFF )                                      0.13       0.25 f
  UBUF3/Z (BUFF  )                                   0.08       0.33 f
  ROUT (out)                                         0.00       0.33 f
  data arrival time                                             0.33

  clock VIRTUAL_CLKP (rise edge)                     0.00       0.00
  clock network delay (ideal)                        0.00       0.00
  clock uncertainty                                  0.05       0.05
  output external delay                             -2.50      -2.45
  data required time                                           -2.45
  -----------------------------------------------------------------------
  data required time                                           -2.45
  data arrival time                                            -0.33
  -----------------------------------------------------------------------
  slack (MET)                                                   2.78
```

Advanced Digital Design
Flip-flop to Output Path with Actual Clock






```
set_output_delay -clock CLKP -min 3.5 [get_ports QOUT]
set_load 0.55 [get_ports QOUT]
```

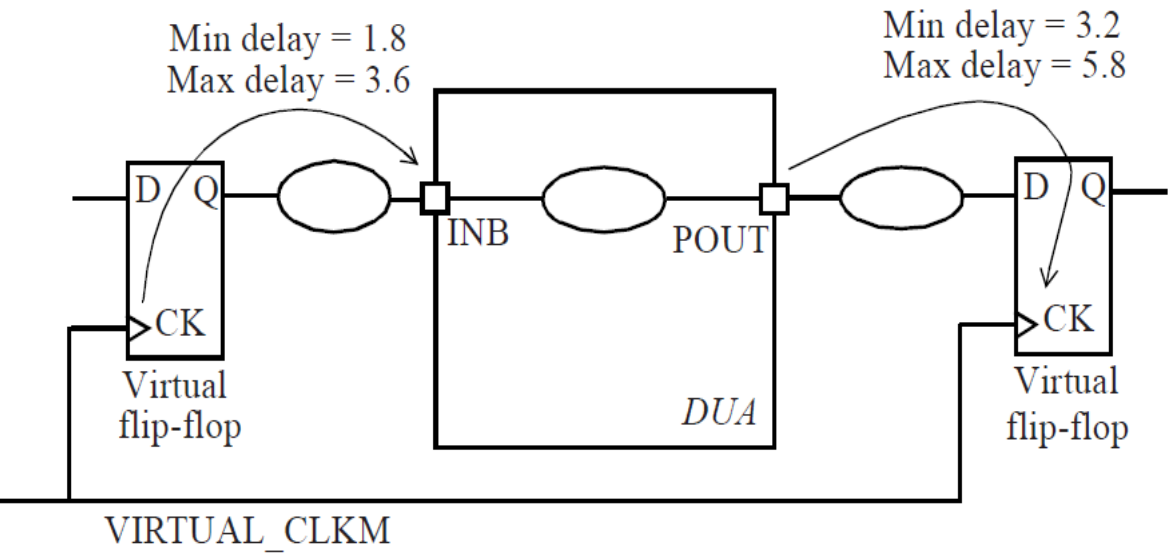
```
Startpoint: UFF4 (rising edge-triggered flip-flop clocked by CLKP)
  Endpoint: QOUT (output port clocked by CLKP)
  Path Group: CLKP
  Path Type: min


  Point                                              Incr       Path
  -----------------------------------------------------------------------
  clock CLKP (rise edge)                             0.00       0.00
  clock source latency                               0.00       0.00
  CLKP (in)                                          0.00       0.00 r
  UCKBUF4/C (CKB  )                                  0.06       0.06 r
  UCKBUF5/C (CKB  )                                  0.06       0.12 r
  UFF4/CK (DFF )                                     0.00       0.12 r
  UFF4/Q (DFF )                                      0.14       0.26 r
  UINV4/ZN (INV  )                                   0.75       1.01 f
  QOUT (out)                                         0.00       1.01 f
  data arrival time                                             1.01

  clock CLKP (rise edge)                             0.00       0.00
  clock network delay (propagated)                   0.00       0.00
  clock uncertainty                                  0.05       0.05
  output external delay                             -3.50      -3.45
  data required time                                            -3.45
  -----------------------------------------------------------------------
  data required time                                            -3.45
  data arrival time                                             -1.01
  -----------------------------------------------------------------------
  slack (MET)                                                    4.46
```

Flip-flop to Output Path with Actual Clock

- The hold timing check is performed on the rising edge (active edge of flipflop) clock CLKP.
- The above report indicates that the flip-flop to output has a positive slack of 4.46ns for the hold time.

Advanced Digital Design

Input to Output Path




```
set_load -pin_load 0.15 [get_ports POUT]
set_output_delay -clock VIRTUAL_CLKM \
  -min 3.2 [get_ports POUT]
set_input_delay -clock VIRTUAL_CLKM \
  -min 1.8 [get_ports INB]
set_input_transition 0.8 [get_ports INB]
```

```
Startpoint: INB (input port clocked by VIRTUAL_CLKM)
Endpoint: POUT (output port clocked by VIRTUAL_CLKM)
Path Group: VIRTUAL_CLKM
Path Type: min

Point                                          Incr       Path
-----------------------------------------------------------------
clock VIRTUAL_CLKM (rise edge)                 0.00       0.00
clock network delay (ideal)                    0.00       0.00
input external delay                           1.80       1.80 r
INB (in) <-                                    0.00       1.80 r
UBUF0/Z (BUFF  )                               0.04       1.84 r
UBUF1/Z (BUFF  )                               0.06       1.90 r
UINV3/ZN (INV  )                               0.22       2.12 f
POUT (out)                                     0.00       2.12 f
data arrival time                                         2.12

clock VIRTUAL_CLKM (rise edge)                 0.00       0.00
clock network delay (ideal)                    0.00       0.00
clock uncertainty                              0.05       0.05
output external delay                         -3.20      -3.15
data required time                                       -3.15
-----------------------------------------------------------------
data required time                                       -3.15
data arrival time                                        -2.12
-----------------------------------------------------------------
slack (MET)                                               5.27
```

Advanced Digital Design

Summary

- Hold Time Check
- Flipflop to Flipflop Path
- Input to Flip-flop Path
- Flip-flop to Output Path
- Input to Output Path

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

ADVANCED DIGITAL DESIGN

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

False Paths, Half-Cycle Path, Removal and Recovery Checks

Unit-III

Sudeendra kumar K
Department of Electronics and Communication Engineering

Contents

- False Paths
- Half Cycle Path
- Recovery Time Check
- Removal Time Check

False Path

- It is possible that certain timing paths are not real in the actual functional operation of the design. Such paths can be turned off during STA by setting these as false paths. A false path is ignored by the STA for analysis.

- Examples of false paths could be from one clock domain to another clock domain, from a clock pin of a flip-flop to the input of another flip-flop, through a pin of a cell, through pins of multiple cells, or a combination of these.

- When a false path is specified through a pin of a cell, all paths that go through that pin are ignored for timing analysis.

- The advantage of identifying the false paths is that the analysis space is reduced, thereby allowing the analysis to focus only on the real paths.

Advanced Digital Design
Specifying the false path

```
set_false_path -from [get_clocks SCAN_CLK] \
  -to [get_clocks CORE_CLK]
# Any path starting from the SCAN_CLK domain to the
# CORE_CLK domain is a false path.

set_false_path -through [get_pins UMUX0/S]
# Any path going through this pin is false.

set_false_path \
  -through [get_pins SAD_CORE/RSTN]]
# The false path specifications can also be specified to,
# through, or from a module pin instance.


set_false_path -to [get_ports TEST_REG*]
# All paths that end in port named TEST_REG* are false paths


set_false_path -through UINV/Z -through UAND0/Z
# Any path that goes through both of these pins
# in this order is false.
```

Specifying False Path during Clock domains

```
set_false_path -from [get_clocks clockA] \
  -to [get_clocks clockB]
```

instead of:

```
set_false_path -from [get_pins {regA_*}/CP] \
  -to [get_pins {regB_*}/D]
```

- Another recommendation is to minimize the usage of ***-through*** options, as it adds unnecessary runtime complexity.
- The ***-through*** option should only be used where it is absolutely necessary and there is no alternate way to specify the false path.

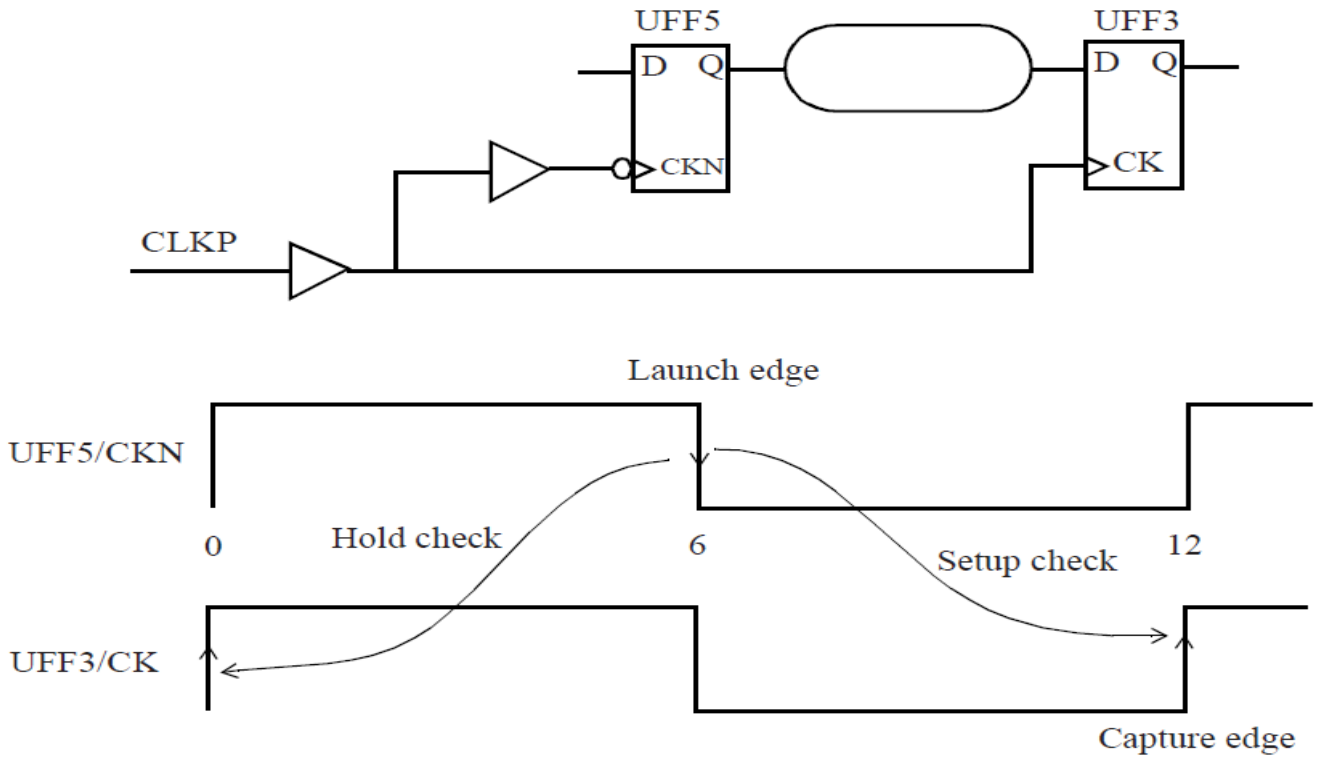
Half Cycle Paths

- If a design has both negative-edge triggered flip-flops (active clock edge is falling edge) and positive-edge triggered flip-flops (active clock edge is rising edge), it is likely that half-cycle paths exist in the design.
- A half-cycle path could be from a rising edge flip-flop to a falling edge flip-flop, or vice versa.
- Figure shows an example when the launch is on the falling edge of the clock of flip-flop UFF5, and the capture is on the rising edge of the clock of flip-flop UFF3.

Courtesy: Static Timing Analysis by Bhasker

Half cycle path

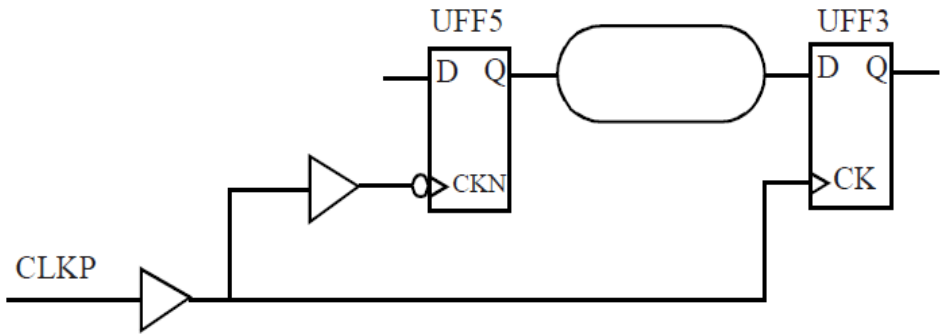
- Figure shows an example when the launch is on the falling edge of the clock of flip-flop UFF5, and the capture is on the rising edge of the clock of flip-flop UFF3.

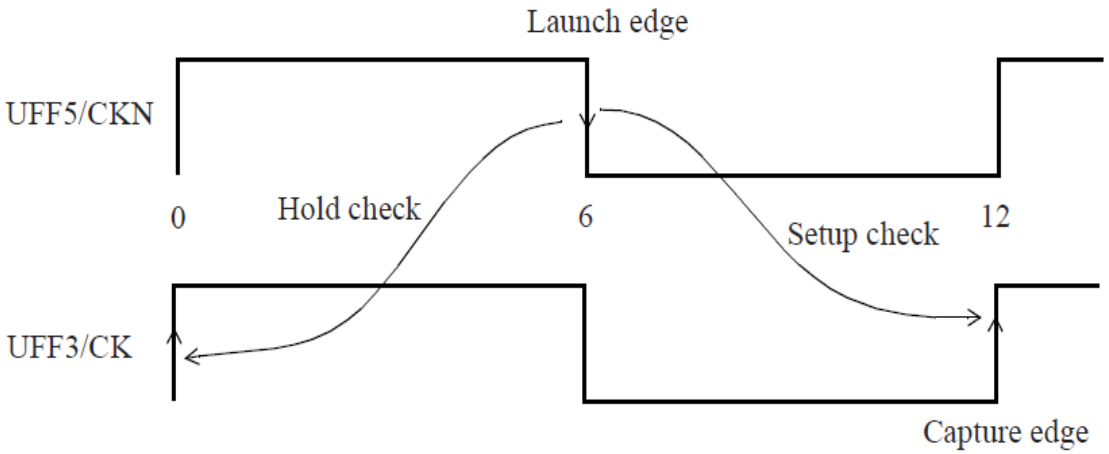


Courtesy: STA by Bhasker

Advanced Digital Design

Half Cycle Path



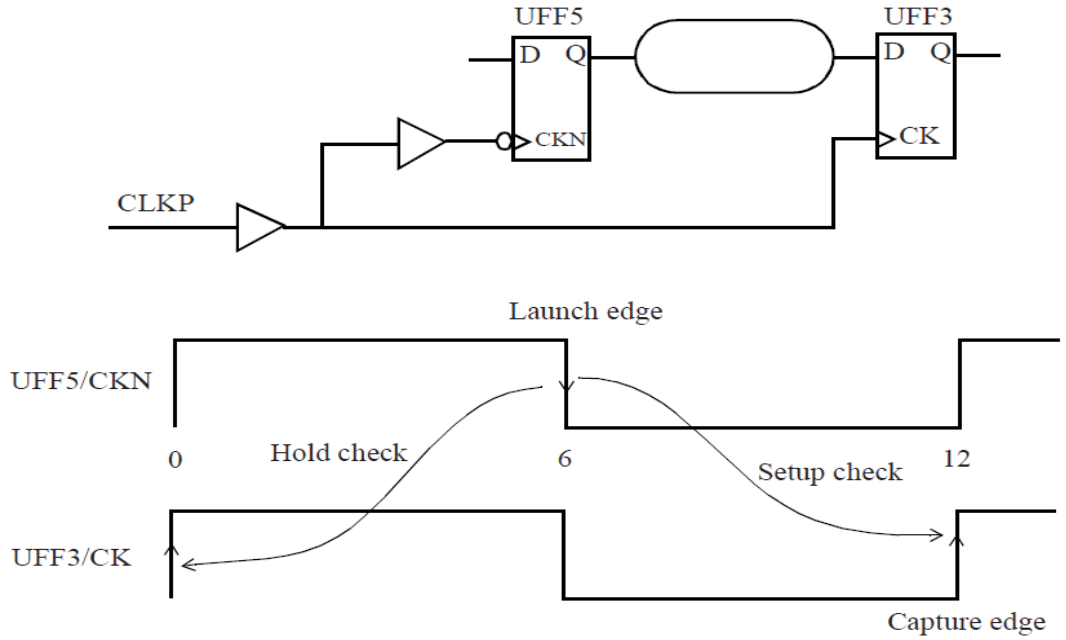



```
Startpoint: UFF5 (falling edge-triggered flip-flop clocked by CLKP)
  Endpoint: UFF3 (rising edge-triggered flip-flop clocked by CLKP)
  Path Group: CLKP
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock CLKP (fall edge)                   6.00       6.00
  clock source latency                     0.00       6.00
  CLKP (in)                                0.00       6.00 f
  UCKBUF4/C (CKB  )                        0.06       6.06 f
  UCKBUF6/C (CKB  )                        0.06       6.12 f
  UFF5/CKN (DFN  )                         0.00       6.12 f
  UFF5/Q (DFN  ) <-                        0.16       6.28 r
  UNAND0/ZN (ND2  )                        0.03       6.31 f
  UFF3/D (DFF )                            0.00       6.31 f
  data arrival time                                   6.31

  clock CLKP (rise edge)                  12.00      12.00
  clock source latency                     0.00      12.00
  CLKP (in)                                0.00      12.00 r
  UCKBUF4/C (CKB  )                        0.07      12.07 r
  UFF3/CK (DFF )                           0.00      12.07 r
  clock uncertainty                       -0.30      11.77
  library setup time                      -0.03      11.74
  data required time                                 11.74
  -----------------------------------------------------------
  data required time                                 11.74
  data arrival time                                  -6.31
  -----------------------------------------------------------
  slack (MET)                                         5.43
```

Note the edge specification in the Startpoint and Endpoint. The falling edge occurs at 6ns and the rising edge occurs at 12ns. Thus, the data gets only a half-cycle, which is 6ns, to propagate to the capture flip-flop.

Half Cycle Path

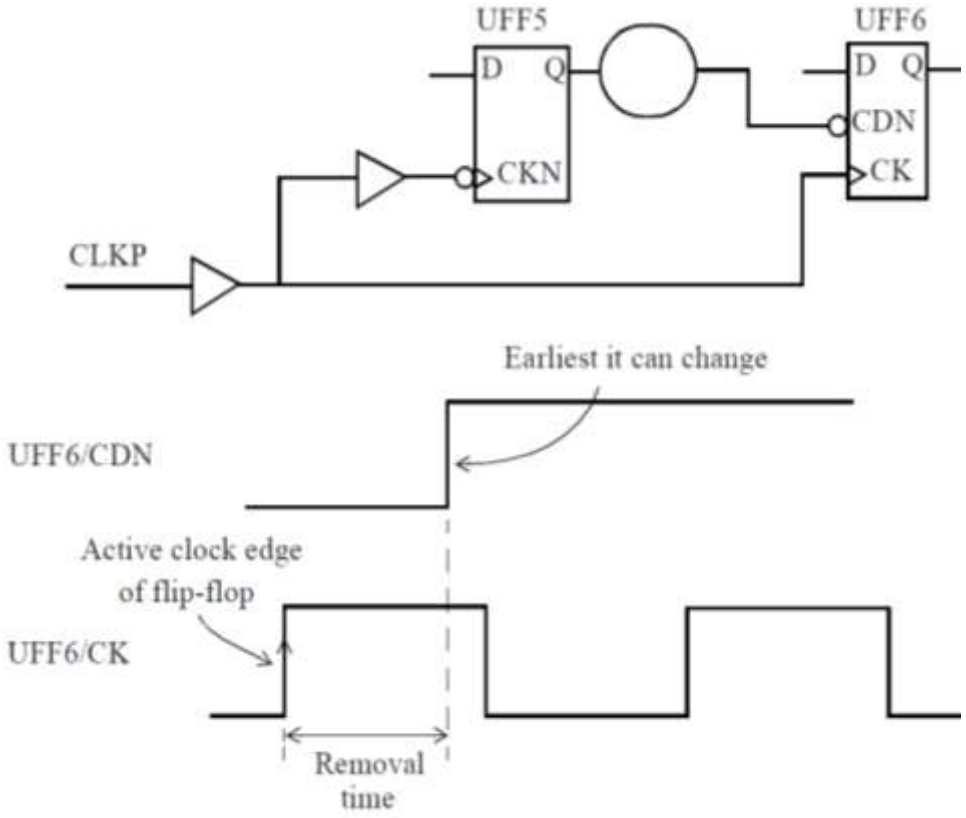
- While the data path gets only half-cycle for setup check, an extra half-cycle is available for the hold timing check.
- The hold check always occurs one cycle prior to the capture edge. Since the capture edge occurs at 12ns, the previous capture edge is at 0ns, and hence the hold gets checked at 0ns. This effectively adds a half-cycle margin for hold checking and thus results in a large positive slack on hold.




```
Startpoint: UFF5 (falling edge-triggered flip-flop clocked by CLKP)
  Endpoint: UFF3 (rising edge-triggered flip-flop clocked by CLKP)
  Path Group: CLKP
  Path Type: min

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock CLKP (fall edge)                   6.00       6.00
  clock source latency                     0.00       6.00
  CLKP (in)                                0.00       6.00 f
  UCKBUF4/C (CKB  )                        0.06       6.06 f
  UCKBUF6/C (CKB  )                        0.06       6.12 f
  UFF5/CKN (DFN  )                         0.00       6.12 f
  UFF5/Q (DFN  ) <-                        0.16       6.28 r
  UNAND0/ZN (ND2  )                        0.03       6.31 f
  UFF3/D (DFF )                            0.00       6.31 f
  data arrival time                                   6.31

  clock CLKP (rise edge)                   0.00       0.00
  clock source latency                     0.00       0.00
  CLKP (in)                                0.00       0.00 r
  UCKBUF4/C (CKB  )                        0.07       0.07 r
  UFF3/CK (DFF )                           0.00       0.07 r
  clock uncertainty                        0.05       0.12
  library hold time                        0.02       0.13
  data required time                                  0.13
  -----------------------------------------------------------
  data required time                                  0.13
  data arrival time                                  -6.31
  -----------------------------------------------------------
  slack (MET)                                         6.18
```

Removal Time Check

- A removal timing check ensures that there is adequate time between an active clock edge and the release of an asynchronous control signal.
- The check ensures that the active clock edge has no effect because the asynchronous control signal remains active until removal time after the active clock edge.
- In other words, the asynchronous control signal is released (becomes inactive) well after the active clock edge so that the clock edge can have no effect.



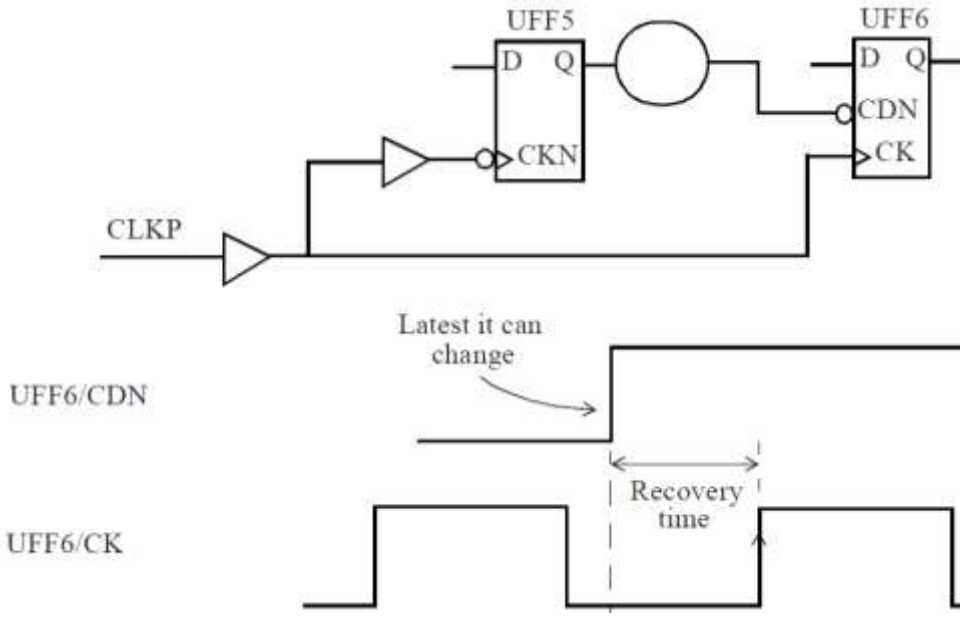
Advanced Digital Design
Removal Time Check

- Like a hold check, it is a min path check except that it is on an asynchronous pin of a flip-flop.
- The Endpoint shows that it is removal check. It is on the asynchronous pin CDN of flip-flop UFF6.
- The removal time for this flip-flop is listed as library removal time with a value of 0.19ns.
- All asynchronous timing checks are assigned to the async default path group.

```
Startpoint: UFF5(falling edge-triggered flip-flop clocked by CLKP)
  Endpoint: UFF6 (removal check against rising-edge clock CLKP)
  Path Group: **async_default**
  Path Type: min

  Point                                                  Incr       Path
  -----------------------------------------------------------------------
  clock CLKP (fall edge)                                 6.00       6.00
  clock source latency                                   0.00       6.00
  CLKP (in)                                              0.00       6.00 f
  UCKBUF4/C (CKB  )                                      0.06       6.06 f
  UCKBUF6/C (CKB  )                                      0.07       6.13 f
  UFF5/CKN (DFN  )                                       0.00       6.13 f
  UFF5/Q (DFN  )                                         0.15       6.28 f
  UINV8/ZN (INV  )                                       0.03       6.31 r
  UFF6/CDN (DFCN  )                                      0.00       6.31 r
  data arrival time                                                 6.31

  clock CLKP (rise edge)                                 0.00       0.00
  clock source latency                                   0.00       0.00
  CLKP (in)                                              0.00       0.00 r
  UCKBUF4/C (CKB  )                                      0.07       0.07 r
  UCKBUF6/C (CKB  )                                      0.07       0.14 r
  UCKBUF7/C (CKB  )                                      0.05       0.19 r
  UFF6/CK (DFCN  )                                       0.00       0.19 r
  clock uncertainty                                      0.05       0.24
  library removal time                                   0.19       0.43
  data required time                                                0.43
  -----------------------------------------------------------------------
  data required time                                                0.43
  data arrival time                                                -6.31
  -----------------------------------------------------------------------
  slack (MET)                                                       5.88
```

Advanced Digital Design
Recovery Timing Check

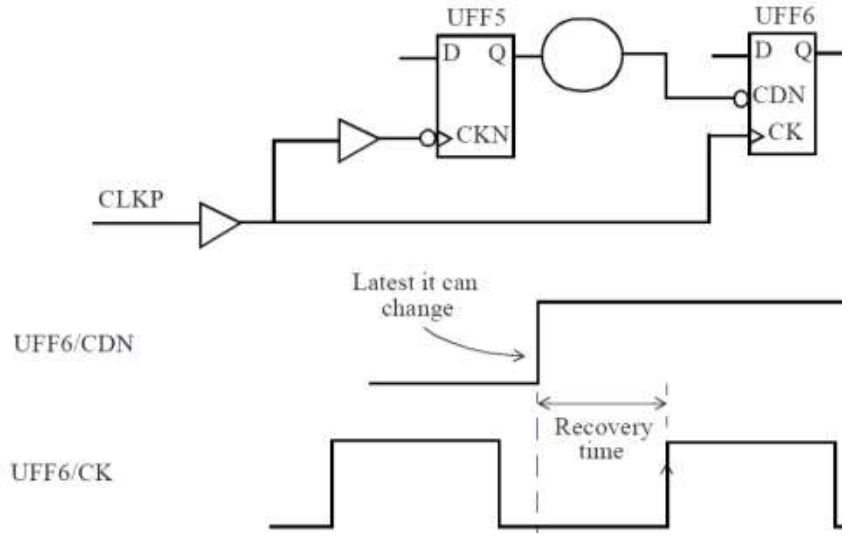
- A recovery timing check ensures that there is a minimum amount of time between the asynchronous signal becoming inactive and the next active clock edge.
- In other words, this check ensures that after the asynchronous signal becomes inactive, there is adequate time to recover so that the next active clock edge can be effective.
- For example, consider the time between an asynchronous reset becoming inactive and the clock active edge of a flip-flop. If the active clock edge occurs too soon after the release of reset, the state of the flip-flop may be unknown.



Advanced Digital Design
Recovery Time Check

- Like a setup check, this is a max path check except that it is on an asynchronous signal.

```
Startpoint: UFF5 (falling edge-triggered flip-flop clocked by CLKP)
  Endpoint: UFF6 (recovery check against rising-edge clock CLKP)
  Path Group: **async_default**
  Path Type: max

  Point                                              Incr       Path
  -----------------------------------------------------------------
  clock CLKP (fall edge)                             6.00       6.00
  clock source latency                               0.00       6.00
  CLKP (in)                                          0.00       6.00 f
  UCKBUF4/C (CKB  )                                  0.06       6.06 f
  UCKBUF6/C (CKB  )                                  0.07       6.13 f
  UFF5/CKN (DFN  )                                   0.00       6.13 f
  UFF5/Q (DFN  )                                     0.15       6.28 f
  UINV8/ZN (INV  )                                   0.03       6.31 r
  UFF6/CDN (DFCN  )                                  0.00       6.31 r
  data arrival time                                             6.31

  clock CLKP (rise edge)                            12.00      12.00
  clock source latency                               0.00      12.00
  CLKP (in)                                          0.00      12.00 r
  UCKBUF4/C (CKB  )                                  0.07      12.07 r
  UCKBUF6/C (CKB  )                                  0.07      12.14 r
  UCKBUF7/C (CKB  )                                  0.05      12.19 r
  UFF6/CK (DFCN  )                                   0.00      12.19 r
  clock uncertainty                                 -0.30      11.89
  library recovery time                              0.09      11.98
  data required time                                           11.98
  -----------------------------------------------------------------
  data required time                                           11.98
  data arrival time                                            -6.31
  -----------------------------------------------------------------
  slack (MET)                                                   5.67
```



- The Endpoint shows that it is recovery check. The recovery time for the UFF6 flip-flop is listed as the library recovery time with a value of 0.09ns. Recovery checks also belong to the async default path group.

Advanced Digital Design

Summary

- False Paths
- Half Cycle Path
- Recovery Time Check
- Removal Time Check

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Digital System Design

Sudeendra kumar K

Department of Electronics and Communication Engineering

DIGITAL SYSTEM DESIGN

Crosstalk and Noise-I

Sudeendra kumar K
Department of Electronics and Communication Engineering

Contents

- Crosstalk and Noise in Static Timing Analysis (STA)
- Crosstalk and Glitch Analysis
- Types of Glitches

Crosstalk and Glitch in Static Timing Analysis (STA)

- Noise refers to undesired or unintentional effects affecting the proper operation of the chip.

- In nanometer technologies, the noise can impact in terms of functionality or in terms of timing of the devices.

Why Crosstalk and Noise Analysis in Static Timing Analysis (STA)

- Increasing number of metal layers: For example, a 0.25mm or 0.3mm process has four or five metal layers and it increases to ten or higher metal layers in the 65nm and 45nm process geometries.
- **Vertically dominant metal aspect ratio:** This means that the wires are thin and tall unlike the wide and thin in the earlier process geometries.
- Thus, a greater proportion of the capacitance is comprised of sidewall coupling capacitance which maps into wire to wire capacitance between neighboring wires.
- Higher routing density due to finer geometry: Thus, more metal wires are packed in close physical proximity.

Why Crosstalk and Noise Analysis in Static Timing Analysis (STA)

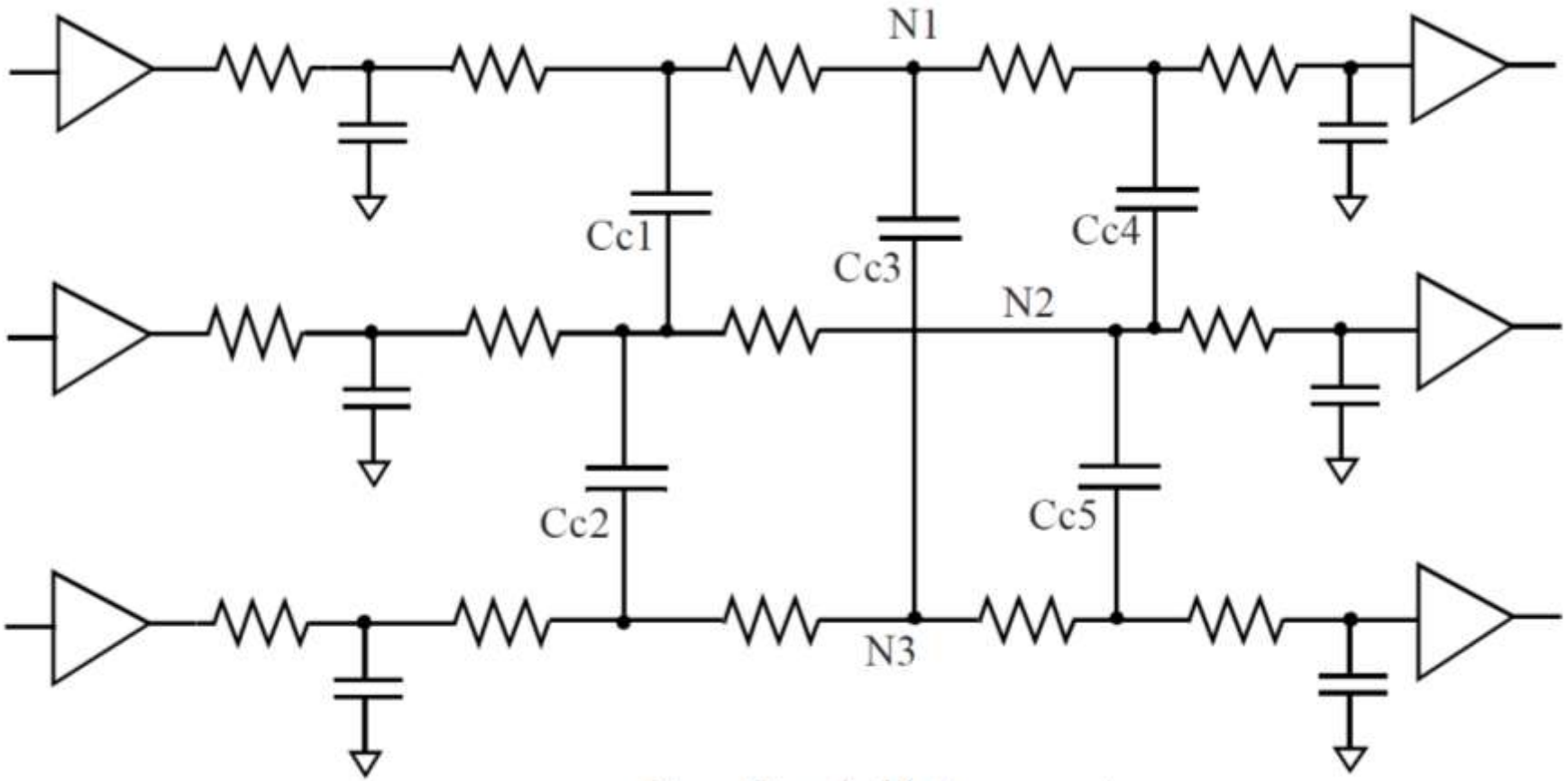
- **Larger number of interacting devices and interconnects:** Thus, greater number of active standard cells and signal traces are packed in the same silicon area causing a lot more interactions.
- **Faster waveforms due to higher frequencies:** Fast edge rates cause more current spikes as well as greater coupling impact on the neighbouring traces and cells.
- **Lower supply voltage:** The supply voltage reduction leaves little margin for noise.

Courtesy: Static Timing Analysis by Bhasker

Why Crosstalk and Noise Analysis in Static Timing Analysis (STA)

- The crosstalk noise refers to unintentional coupling of activity between two or more signals.
- The crosstalk noise is caused by the capacitive coupling between neighboring signals on the die.
- This results in switching activity on a net to cause unintentional effects on the coupled signals.

Example of Coupled Interconnect



Cc = Coupled interconnect

Crosstalk

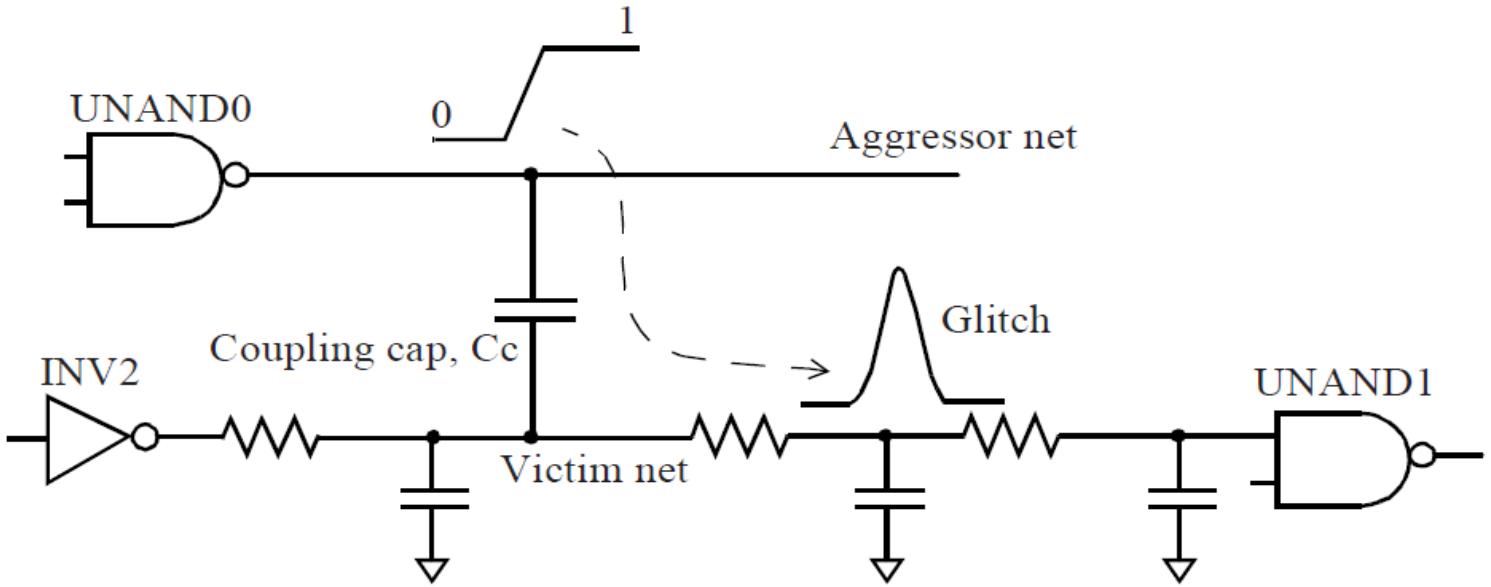
- The crosstalk noise refers to unintentional coupling of activity between two or more signals.
- The crosstalk noise is caused by the capacitive coupling between neighboring signals on the die.
- This results in switching activity on a net to cause unintentional effects on the coupled signals.
- The affected signal is called the victim, and the affecting signals are termed as aggressors.
- Note that two coupled nets can affect each other, and often a net can be a victim as well as an aggressor.

Crosstalk

- Broadly, there are two types of noise effects caused by crosstalk –
- Glitch, which refers to noise caused on a steady victim signal due to coupling of switching activity of neighboring aggressors, and
- Change in timing (crosstalk delta delay), caused by coupling of switching activity of the victim with the switching activity of the aggressors.

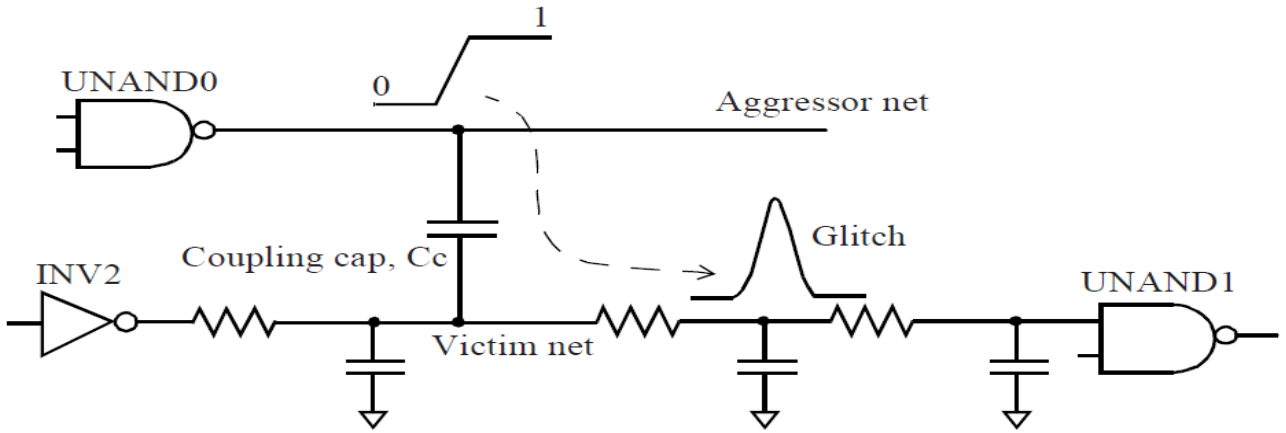
Crosstalk Glitch Analysis

- A steady signal net can have a glitch (positive or negative) due to charge transferred by the switching aggressors through the coupling capacitances.
- The coupling capacitance between the two nets is depicted as one lumped capacitance Cc instead of distributed coupling, this is to simplify the explanation below without any loss of generality.



Crosstalk Glitch Analysis

- In this example, the NAND2 cell UNAND0 switches and charges its output net (labeled Aggressor).
- Some of the charge is also transferred to the victim net through the coupling capacitance Cc and results in the positive glitch.
- The amount of charge transferred is directly related to the coupling capacitance, Cc, between the aggressor and the victim net.
- The charge transferred on the grounded capacitances of the victim net causes the glitch at that net.
- The steady value on the victim net (in this case, 0 or low) is restored because the transferred charge is dissipated through the pull-down stage of the driving cell INV2.



Magnitude of Glitch depends on: -

Coupling capacitance between the aggressor net and victim: The greater the coupling capacitance, the larger the magnitude of the glitch.

Slew of the aggressor net: The faster the slew at the aggressor net, the larger the magnitude of glitch. In general, faster slew is because of higher output drive strength for the cell driving the aggressor net.

Victim net grounded capacitance: The smaller the grounded capacitance on the victim net, the larger the magnitude of the glitch.

Victim net driving strength: The smaller the output drive strength of the cell driving the victim net, the larger the magnitude of the glitch.

Glitch can affect the functionality of the circuit

- The glitch magnitude may be large enough to be seen as a different logic value by the fanout cells (e.g. a victim at logic-0 may appear as logic-1 for the fanout cells). This is especially critical for the sequential cells (flip-flops, latches) or memories, where a glitch on the clock or asynchronous set/reset can be catastrophic to the functionality of the design. Similarly, a glitch on the data signal at the latch input can cause incorrect data to be latched which can also be catastrophic if the glitch occurs when the data is being clocked in.
- Even if the victim net does not drive a sequential cell, a wide glitch may be propagated through the fanouts of the victim net and reach a sequential cell input with catastrophic consequences for the design.

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Digital System Design

Sudeendra kumar K

Department of Electronics and Communication Engineering

DIGITAL SYSTEM DESIGN

Crosstalk and Noise-II

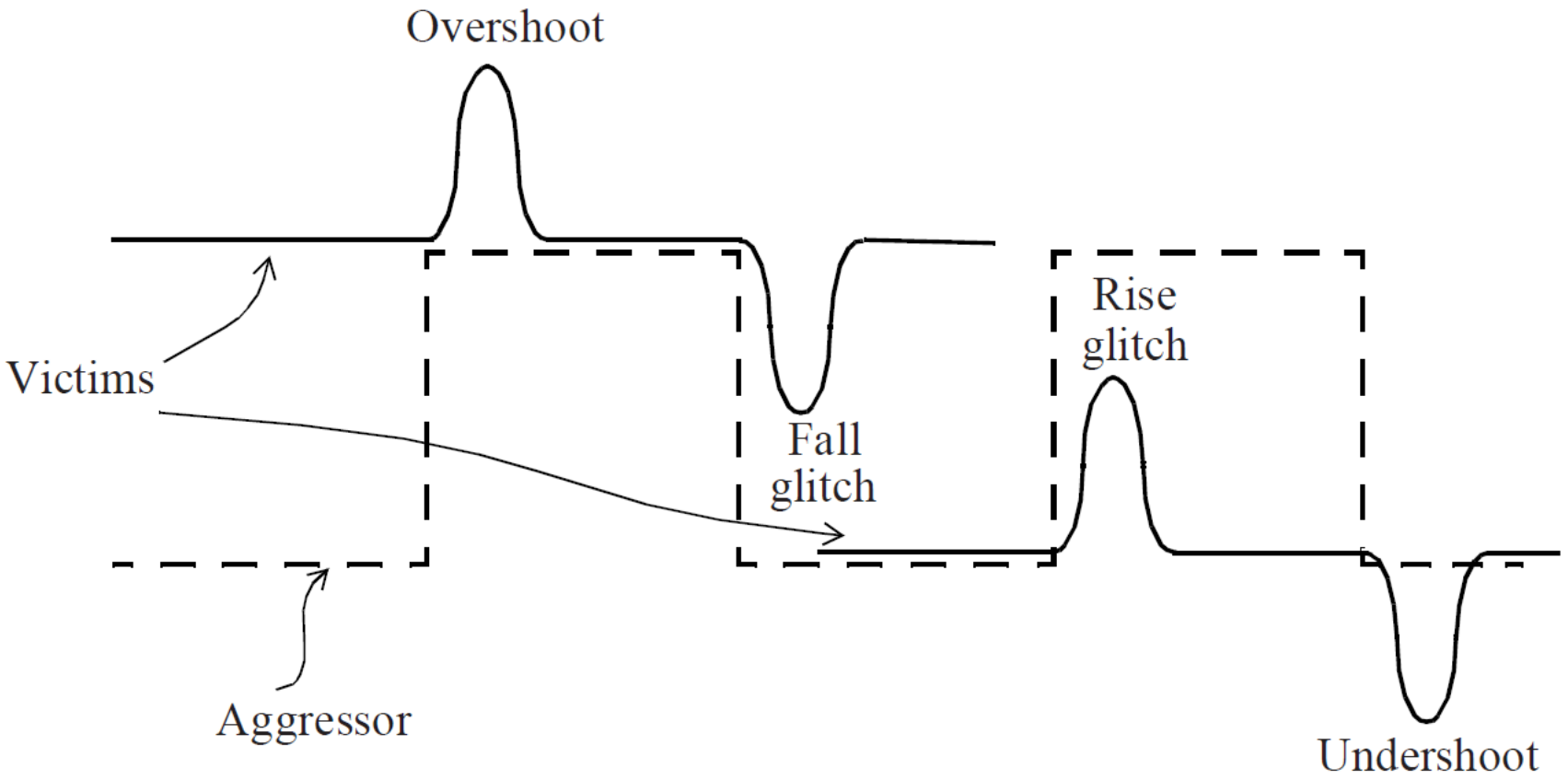
Sudeendra kumar K
Department of Electronics and Communication Engineering

Contents

- Types of Glitches

Types of Glitches

Rise and Fall Glitches:



Types of Glitches

- **Rise and Fall Glitch: -** A positive or rise glitch on a victim net which is steady low. An analogous case is of a negative glitch on a steady high signal. A falling aggressor net induces a fall glitch on a steady high signal.
- **Overshoot and Undershoot Glitch:** - There is still a glitch which takes the victim net voltage above its steady high value. Such a glitch is called an overshoot glitch. Similarly, a falling aggressor when coupled to a steady low victim net causes an undershoot glitch on the victim net.

Review of Glitch

- The glitch is governed by the coupling capacitance, aggressor slew and the drive strength of the victim net.
- The glitch computation is based upon the amount of current injected by the switching aggressor, the RC interconnect for the victim net, and the output impedance of the cell driving the victim net.
- The detailed glitch calculation is based upon the library models; the related noise models for the calculation are part of the standard cell library models

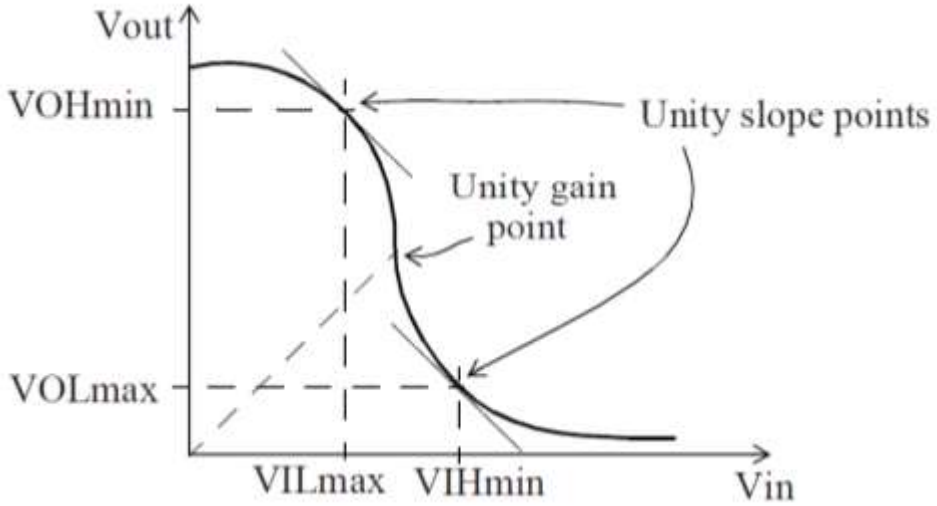
Courtesy: Static Timing Analysis by Bhasker

Glitch Thresholds and Propagation

- A glitch caused by coupling from a switching aggressor can propagate through the fanout cell depending upon the fanout cell and glitch attributes such as glitch height and glitch width.
- This analysis can be based upon DC or AC noise thresholds.
- The DC noise analysis only examines the glitch magnitude and is conservative.
- The AC noise analysis examines other attributes such as glitch width and fanout cell output load.

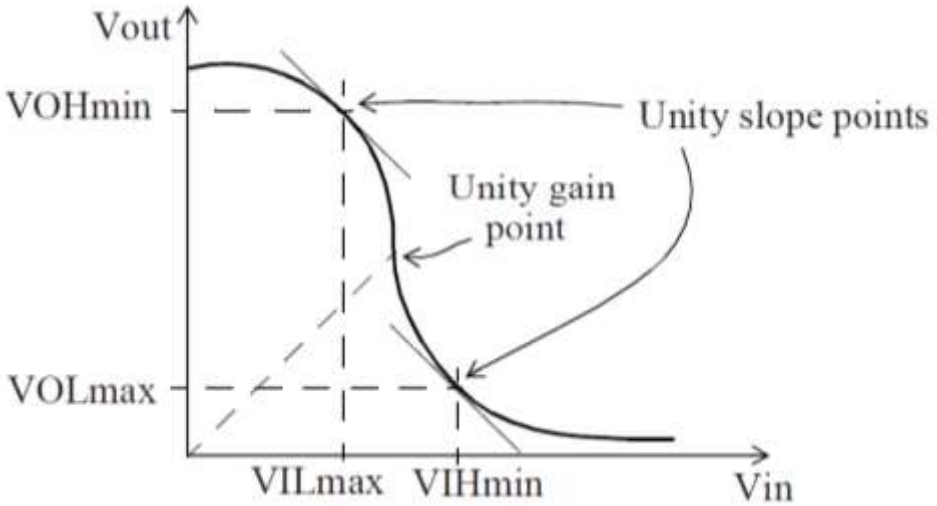
DC Thresholds

- VOH is the range of output voltage that is considered as a logic-one or high.
- VIL is the range of input voltage that is considered a logic-zero or low.
- VIH is the range of input voltage that is considered as a logic-one.
- VOL is the range of output voltage that is considered as a logic-zero.



DC Thresholds

- The DC noise margin is a check used for glitch magnitude and refers to the DC noise limits on the input of a cell while ensuring proper logic functionality.
- For example, the output of an inverter cell may be high (that is, stay above the minimum value of VOH) as long as the input stays below the max value of VIL for the cell.
- Similarly, the output of the inverter cell may be low (that is, stay below VOL maximum value) as long as the input stays above the VIH minimum value.

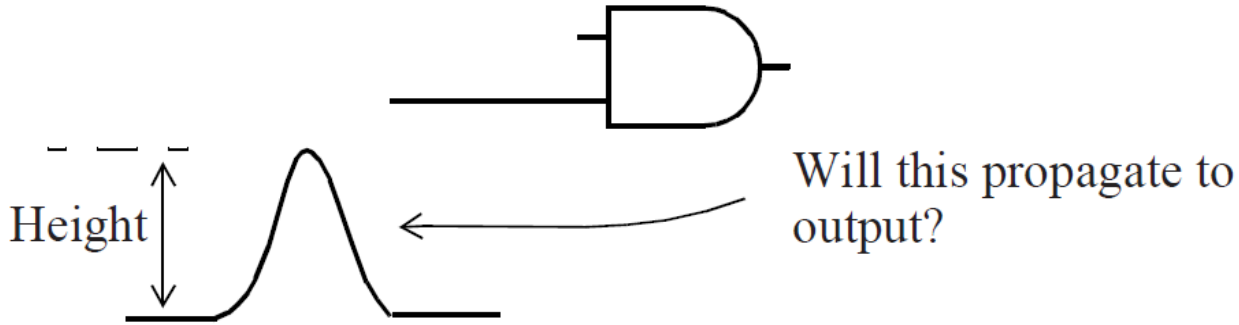


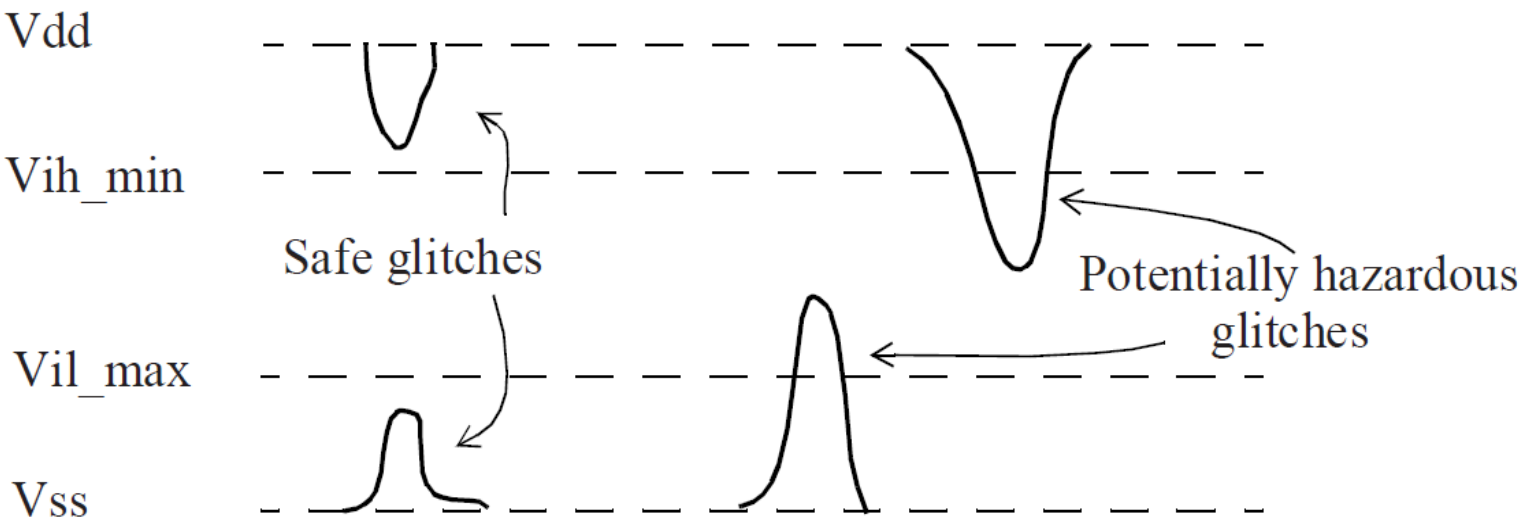
DC Thresholds

- The VILmax and VIHmin limits are also referred to as DC margin limits.
- The DC margin limits are separate for rise_glitch (input low) and fall_glitch (input high).
- A glitch below the DC margin limit (for example, a rise glitch below the VILmax of the fanout pins) cannot be propagated through the fanout irrespective of the width of the glitch.
- Thus, a conservative glitch analysis checks that the peak voltage level (for all glitches) meets the VIL and the VIH levels of the fanout cells.
- As long as all nets meet the VIL and VIH levels for the fanout cells in spite of any glitches, it can be concluded that the glitches have no impact on the functionality of the design (since the glitches cannot cause the output to change).

TIMING ANALYSIS OF DIGITAL CIRCUITS

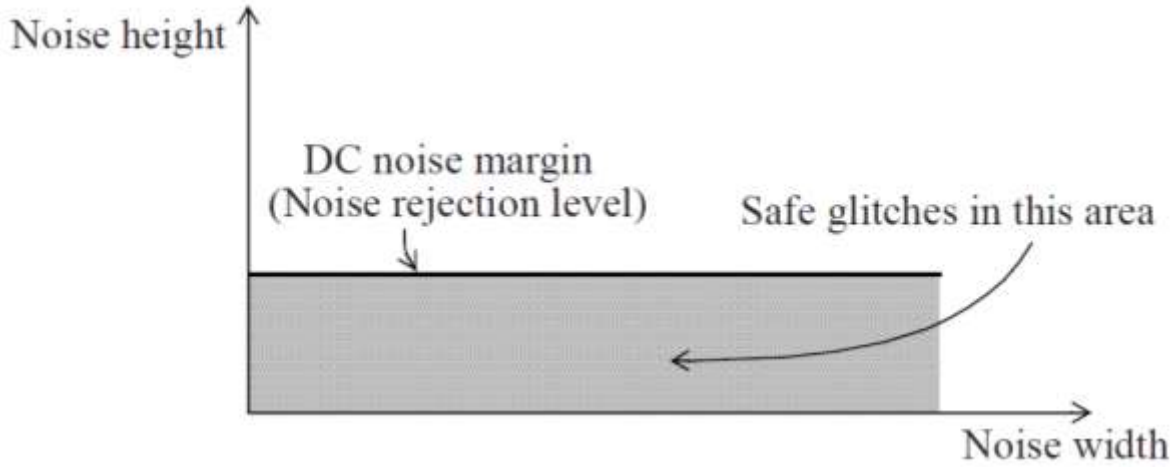
Glitch and DC Noise Margin





TIMING ANALYSIS OF DIGITAL CIRCUITS
Glitch and DC Noise Margin

- Not all glitches with magnitude larger than the DC noise margin can change the output of a cell.
- The width of the glitch is also an important consideration in determining whether the glitch will propagate to the output or not.
- A narrow glitch at a cell input will normally not cause any impact at the output of a cell.
- DC noise margin uses only a constant worst case value irrespective of the signal noise width.
- This provides a noise rejection level that is a very conservative estimate of the noise tolerance of a cell.



AC Thresholds

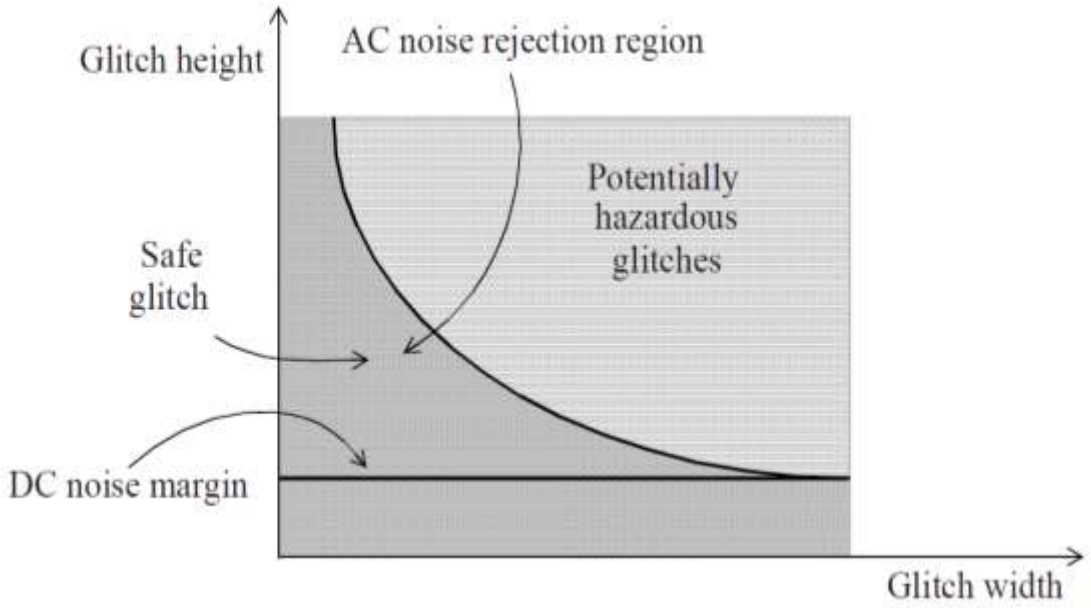
- The DC margin limits for glitch analysis are conservative since these analyze the design under the worst-case conditions.
- The DC margin limits verify that even if the glitch is arbitrarily wide, it will not affect the proper operation of the design.
- In DC analysis, it becomes imperative to verify the impact of glitches with respect to the glitch width and the output load of the cell.
- In general, if the glitch is narrow or if the fanout cell has a large output capacitance, the glitch would have no effect on the proper functional operation.
- In general, a single stage cell will stop any input glitch which is much narrower than the delay through the cell.

AC Thresholds

- This is because with a narrow glitch, the glitch is over before the fanout cell can respond to it. Thus, a very narrow glitch does not have any effect on the cell.
- Since the output load increases the delay through the cell, increasing the output load has the effect of minimizing the impact of glitch at the input - though it has the adverse effect of increasing the cell delay.

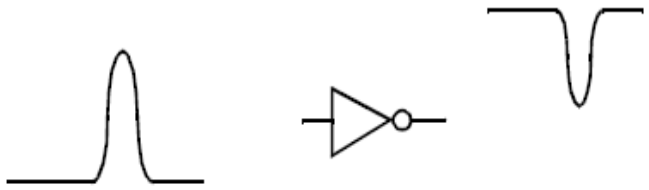
AC Thresholds

- The dark shaded region represents good or acceptable glitches since these are either too narrow or too short, or both, and thus have no effect on the functional behavior of the cell. The lightly shaded region represents bad or unacceptable glitches since these are too wide or too tall, or both, and thus such a glitch at the cell input affects the output of the cell. In the limiting case of very wide glitches, the glitch threshold corresponds to the DC noise margin.

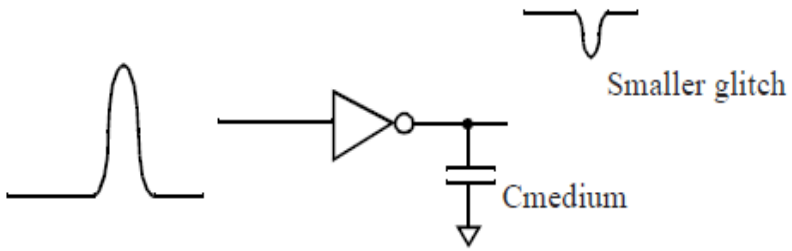


TIMING ANALYSIS OF DIGITAL CIRCUITS

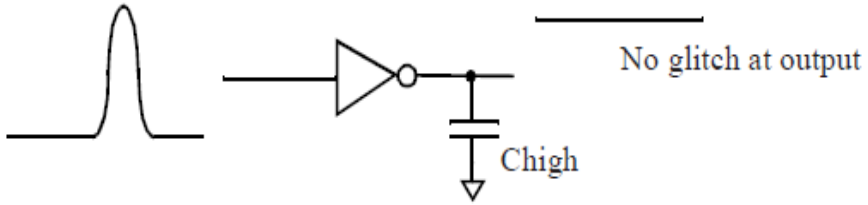
AC Glitches



(a) No output load.



(b) Medium load capacitance.



(c) High load capacitance.

- The glitches below the AC threshold can be ignored or the fanout cell can be considered to be immune from such a glitch.
- The AC threshold (or noise immunity) region depends upon the output load and the glitch width.

What happens if the glitches are larger than the AC threshold

- In such a case where the glitch magnitude exceeds the AC threshold, the glitch at the cell input produces another glitch at the output of the cell.
- The output glitch height and width is a function of input glitch width and height as well as the output load.
- This information is characterized in the cell library which contains detailed tables or functions for the output glitch magnitude and width as a function of the input pin glitch magnitude, glitch width and the load at the output pin.
- The glitch propagation is governed by *propagated_noise models* which are included in the library cell description

AC Thresholds and Glitches

- Rise glitch (*modeled by propagated_noise_high; or noise_immunity_high in earlier models)*,
- fall glitch (*modeled by propagated_noise_low; or noise_immunity_low in earlier models*),
- overshoot glitch (*modeled by noise_immunity_above_high*)
- undershoot glitch (*modeled by noise_immunity_below_low*)

AC Thresholds and Glitches

- To summarize, different inputs of a cell have different limits on the glitch threshold which is a function of glitch width and output capacitance.
- These limits are separate for input high (low transition glitch) and for input low (high transition glitch).
- The noise analysis examines the peak as well as the width of the glitch and analyzes whether it can be neglected or whether it can propagate to fanouts.

Summary

- DC Noise
- DC Threshold
- AC Threshold
- AC Glitches

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

TIMING ANALYSIS OF DIGITAL CIRCUITS

Sudeendra kumar K

Department of Electronics and Communication Engineering

TIMING ANALYSIS OF DIGITAL CIRCUITS

Crosstalk and Noise-III and Analysis

Unit-IV: Session III

Sudeendra kumar K
Department of Electronics and Communication Engineering

Contents

- Noise Accumulation with Multiple Aggressors
- Aggressor Timing Correlation
- Aggressor Functional Correlation
- Positive and Negative Glitch

Noise Accumulation with Multiple Aggressors

- In general, a victim net may be capacitive-coupled to many nets. When multiple nets switch concurrently, the crosstalk coupling noise effect on the victim is compounded due to multiple aggressors.
- Most analyses for coupling due to multiple aggressors add the glitch effect due to each aggressor and compute the cumulative effect on the victim.
- This may appear conservative, however it does indicate the worst-case glitch on the victim.
- An alternate approach is the use of RMS (root-mean squared) approach.
- When using the RMS option, the magnitude of the glitch on the victim is computed by taking the root-mean-square of the glitches caused by individual aggressors.

Aggressor Timing Correlation

- For crosstalk glitch due to multiple aggressors, the analysis must include the timing correlation of the aggressor nets and determine whether the multiple aggressors can switch concurrently.
- The STA obtains this information from the timing windows of the aggressor nets.
- During timing analysis, the earliest and the latest switching times of the nets are obtained.
- These times represent the timing windows during which a net may switch within a clock cycle.
- The switching windows (rising and falling) provide the necessary information on whether the aggressor nets can switch together.

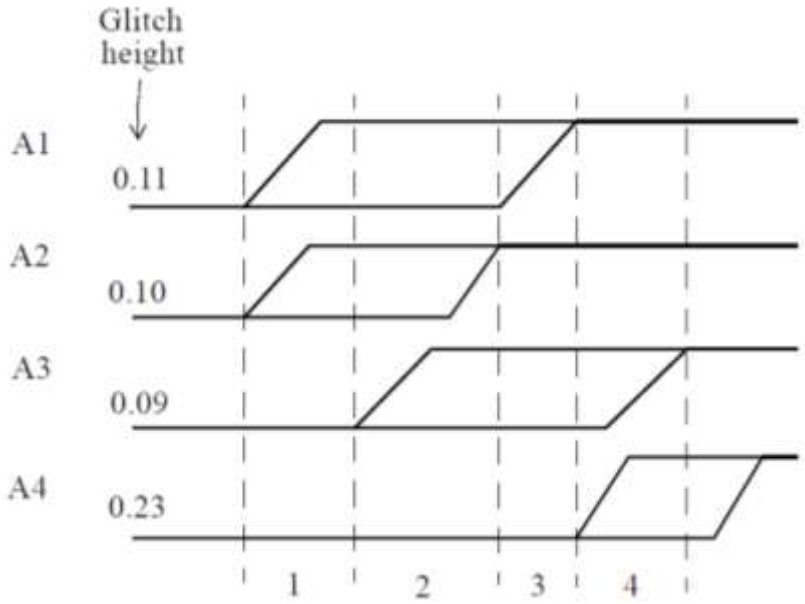
Aggressor Timing Correlation

- The glitch analysis computes the four types of glitches (rise, fall, undershoot, and overshoot) separately for each potential aggressor.
- The next step combines the glitch contributions from the various individual aggressors.
- The multiple aggressors can combine separately for each type of glitch.

Courtesy: Static Timing Analysis by Bhasker

TIMING ANALYSIS OF DIGITAL CIRCUITS
Aggressor Timing Correlation

- Bin 1 has A1 and A2 switching which can result in a glitch magnitude of 0.21 (= 0.11 + 0.10). Bin 2 has A1, A2, and A3 switching which can result in a glitch magnitude of 0.30 (= 0.11 + 0.10 + 0.09).
- Bin 3 has A1 and A3 switching which can result in a glitch magnitude of 0.20 (= 0.11 + 0.09). Bin 4 has A3 and A4 switching which can result in a glitch magnitude of 0.32 (= 0.09 + 0.23). Bin 4 has the worst possible glitch magnitude of 0.32.
- Note that an analysis without using timing windows will predict a combined glitch magnitude of 0.53 (= 0.11 + 0.10 + 0.09 + 0.23) which can be overly pessimistic.



Aggressor Functional Correlation

- In addition, another factor to be considered is the functional correlation between various signals.

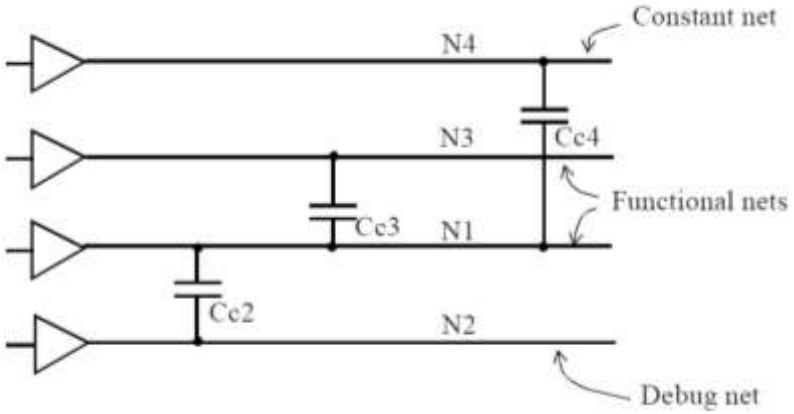
- Ex: - The scan control signals only switch during the scan mode and are steady during functional or mission mode of the design. Thus, the scan control signals cannot cause a glitch on other signals during the functional mode.

- The logic controlled by test clocks and the logic controlled by functional clocks create two disjoint sets of aggressors.

    - For such cases, both signal and its complement cannot be switching in the same direction for crosstalk noise computation.

Aggressor Functional Correlation

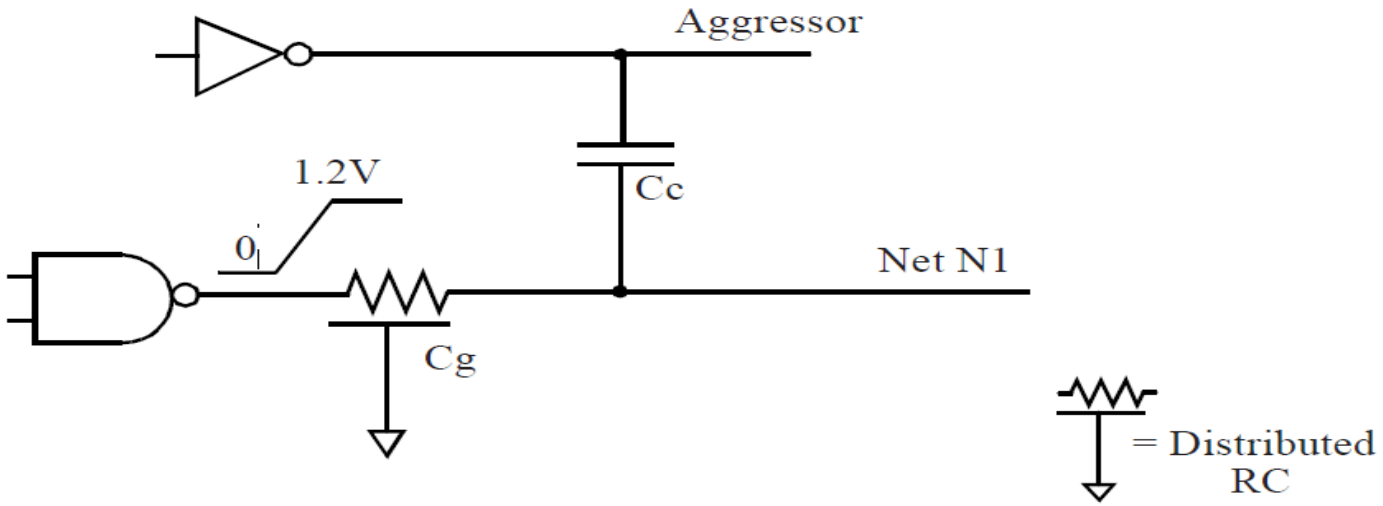
- Figure shows an example of net N1 having coupling with three other nets N2, N3 and N4. In functional correlation, the functionality of the nets needs to be considered.
- Assume net N4 is a constant and thus cannot be an aggressor on net N1, in spite of its coupling. Assume N2 is a net that is part of debug bus, but is in steady state in functional mode.
- Thus, net N2 cannot be an aggressor for net N1. Assuming net N3 carries functional data, only net N3 can be considered as a potential aggressor for net N1.



Crosstalk Delay Analysis

- Capacitances are considered as part of the total net capacitance during the basic delay calculation (without considering any crosstalk).
- When the neighboring nets are steady (or not switching), the inter-signal capacitances can be treated as grounded.
- When a neighboring net is switching, the charging current through the coupling capacitance impacts the timing of the net.
- The equivalent capacitance seen from a net can be larger or smaller based upon the direction of the aggressor net switching.

Crosstalk delay Analysis



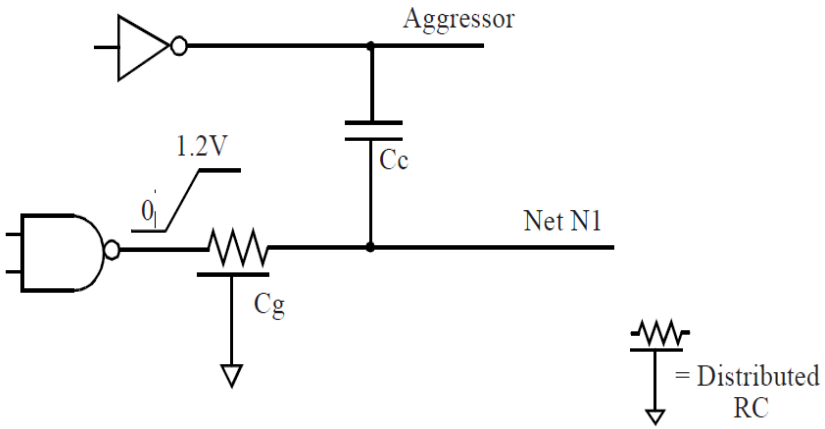
- Figure shows net N1 which has a coupling capacitance Cc to a neighboring net (labeled Aggressor) and a capacitance Cg to ground.
- This example assumes that the net N1 has a rising transition at the output and considers different scenarios depending on whether or not the aggressor net is switching at the same time.

Crosstalk delay Analysis

- The capacitive charge required from the driving cell in various scenarios can be different as described below: -
  - Aggressor net steady
  - Aggressor switching in same direction
  - Aggressor switching in opposite direction

Aggressor net steady

- In this scenario, the driving cell for the net N1 provides the charge for Cg and Cc to be charged to Vdd. The total charge provided by the driving cell of this net is thus (Cg + Cc) * Vdd. The base delay calculation obtains delays for this scenario where no crosstalk is considered from the aggressor nets.

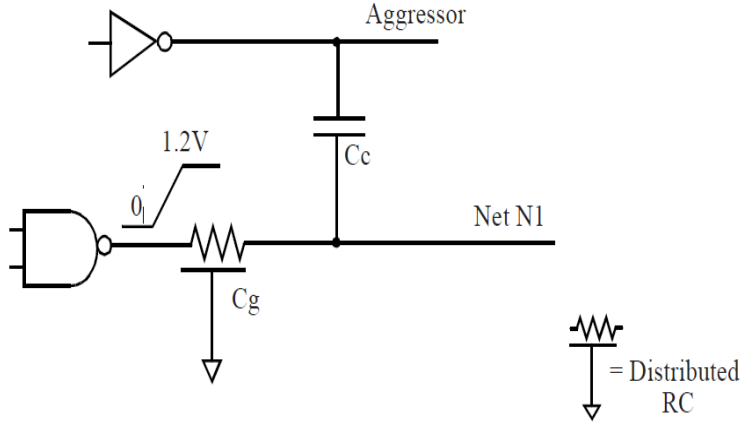


Capacitance		*Before rising transition at net N1*	*After rising transition at net N1*
Grounded Cap, Cg		V(Cg) = 0	V(Cg) = Vdd
Coupling Cap, Cc	Aggressor net steady LOW	V(Cc) = 0	V(Cc) = Vdd
	Aggressor net steady HIGH	V(Cc) = - Vdd	V(Cc) = 0

No Crosstalk

Aggressor switching in same direction

- The driving cell is aided by the aggressor switching in the same direction. If the aggressor transitions at the same time with the same slew (identical transition time), the total charge provided by the driving cell is only (Cg * Vdd).
- If the slew of the aggressor net is faster than that of N1, the actual charge required can be even smaller than (Cg * Vdd) since the aggressor net can provide charging current for Cg also.

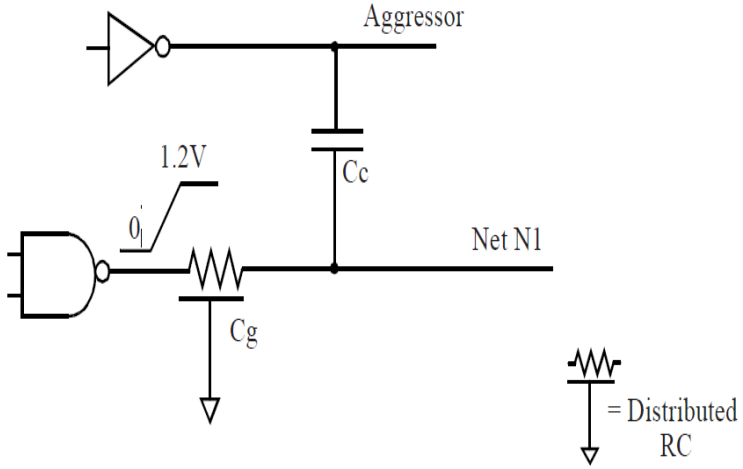


Capacitance	*Before rising transition at net N1 and aggressor net*	*After rising transition at net N1 and aggressor net*
Grounded Cap, Cg	V(Cg) = 0	V(Cg) = Vdd
Coupling Cap, Cc	V(Cc) = 0	V(Cc) = 0

Aggressor switching in same direction (negative crosstalk)

Aggressor switching in opposite direction

- In this scenario, the coupling capacitance is charged from -Vdd to Vdd. Thus, the charge on coupling capacitance changes by (2 * Cc * Vdd) before and after the transitions. This additional charge is provided by both the driving cell of net N1 as well as the aggressor net. This scenario results in a larger delay for the switching net N1; the increase in delay is labeled as positive crosstalk delay.



Capacitance	*Before transition at net N1 and aggressor net (net N1 is low; aggressor net is high)*	*After transition (net N1 is high; and aggressor net is low)*
Grounded Cap, Cg	V(Cg) = 0	V(Cg) = Vdd
Coupling Cap, Cc	V(Cc) = - Vdd	V(Cc) = Vdd

Aggressor switching in opposite direction (positive crosstalk)

Positive Glitch

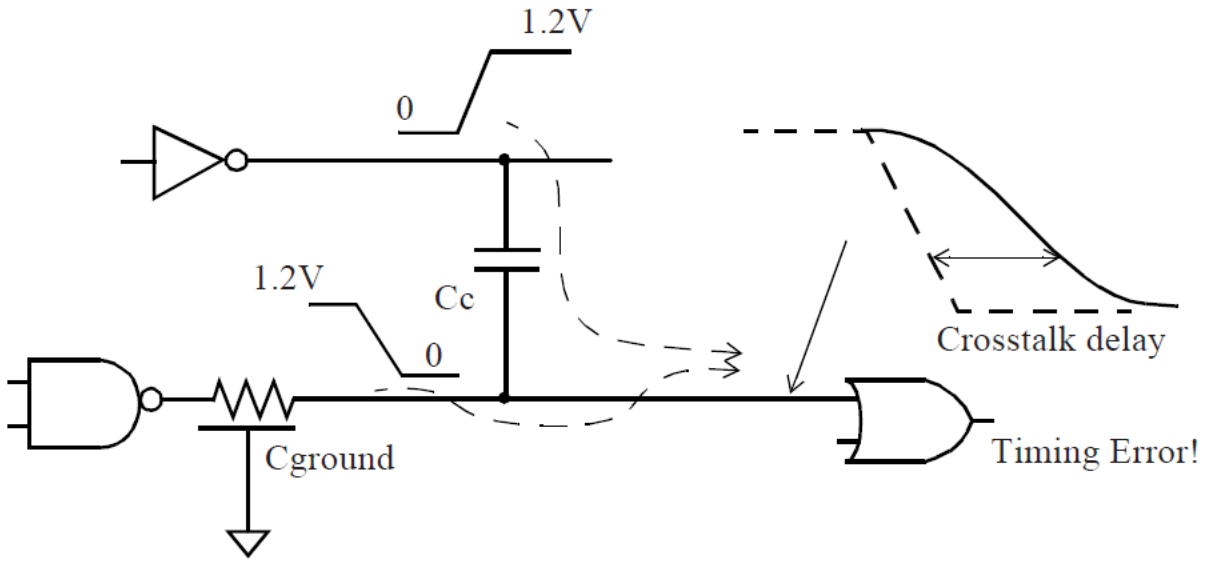
- The base delay calculation (without any crosstalk) assumes that the driving cell provides all the necessary charge for rail-to-rail transition of the total capacitance of a net, Ctotal (= Cground + Cc).
- The charge required for the coupling capacitance Cc is larger when the coupled (aggressor) net and victim net are switching in the opposite directions.
- The aggressor switching in the opposite direction increases the amount of charge required from the driving cell of the victim net and increases the delays for the driving cell and the interconnect for the victim net.

Positive Glitch and Negative Glitch

- Similarly, when the coupled (aggressor) net and the victim net are switching in the same direction, the charge on Cc remains the same before and after the transitions of the victim and aggressor.
- This reduces the charge required from the driving cell of the victim net. The delays for the driving cell and the interconnect for the victim net are reduced.
- Concurrent switching of victim and aggressor affects the timing of the victim transition.
- Depending upon the switching direction of the aggressor, the crosstalk delay effect can be positive (slow down the victim transition) or negative (speed up the victim transition).

TIMING ANALYSIS OF DIGITAL CIRCUITS

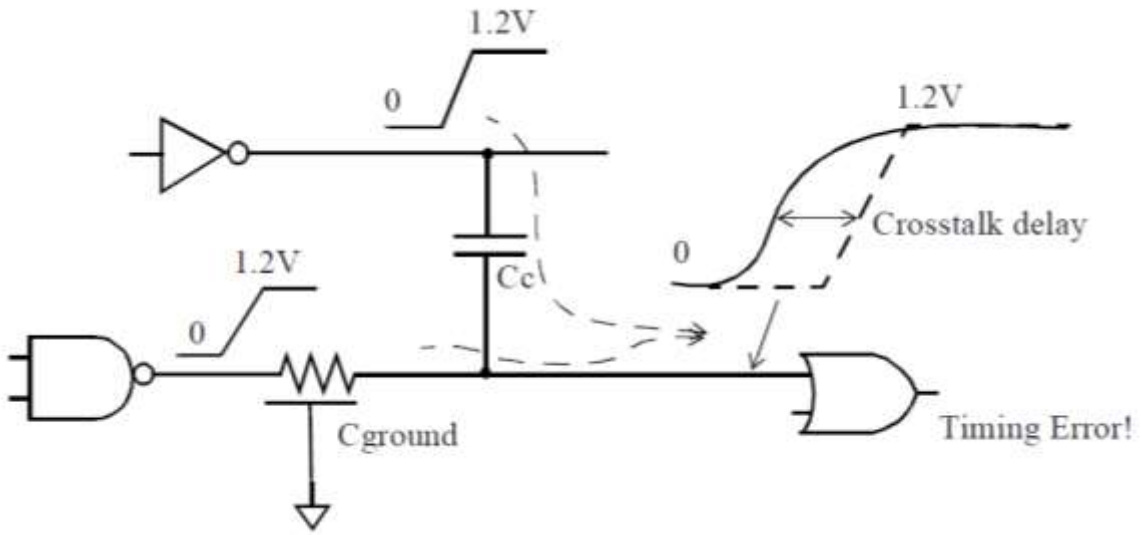
Positive Crosstalk Delay



- Aggressor net is rising at the same time when the victim net has a falling transition.
- The aggressor net switching in opposite direction increases the delay for the victim net.
- The positive crosstalk impacts the driving cell as well as the interconnect - the delay for both of these gets increased.

Negative Glitch delay

- The aggressor net is rising at the same time as the victim net.
- The aggressor net switching in the same direction as the victim reduces the delay of the victim net.
- As before, the negative crosstalk affects the timing of the driving cell as well as the interconnect - the delay for both of these is reduced.



Summary

- Noise Accumulation with Multiple Aggressors
- Aggressor Timing Correlation
- Aggressor Functional Correlation
- Positive and Negative Glitch

TIMING ANALYSIS OF DIGITAL CIRCUITS

Next Session

- Setup Check
- Hold check

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

TIMING ANALYSIS OF DIGITAL CIRCUITS

Sudeendra kumar K

Department of Electronics and Communication Engineering

TIMING ANALYSIS OF DIGITAL CIRCUITS

Timing Verification in the presence of Crosstalk Delay

Unit-IV: Session IV

Sudeendra kumar K

Department of Electronics and Communication Engineering

Contents

- Setup Check
- Hold Check
- Computational Complexity
- Noise Avoidance Techniques

Types of Delay Measurements

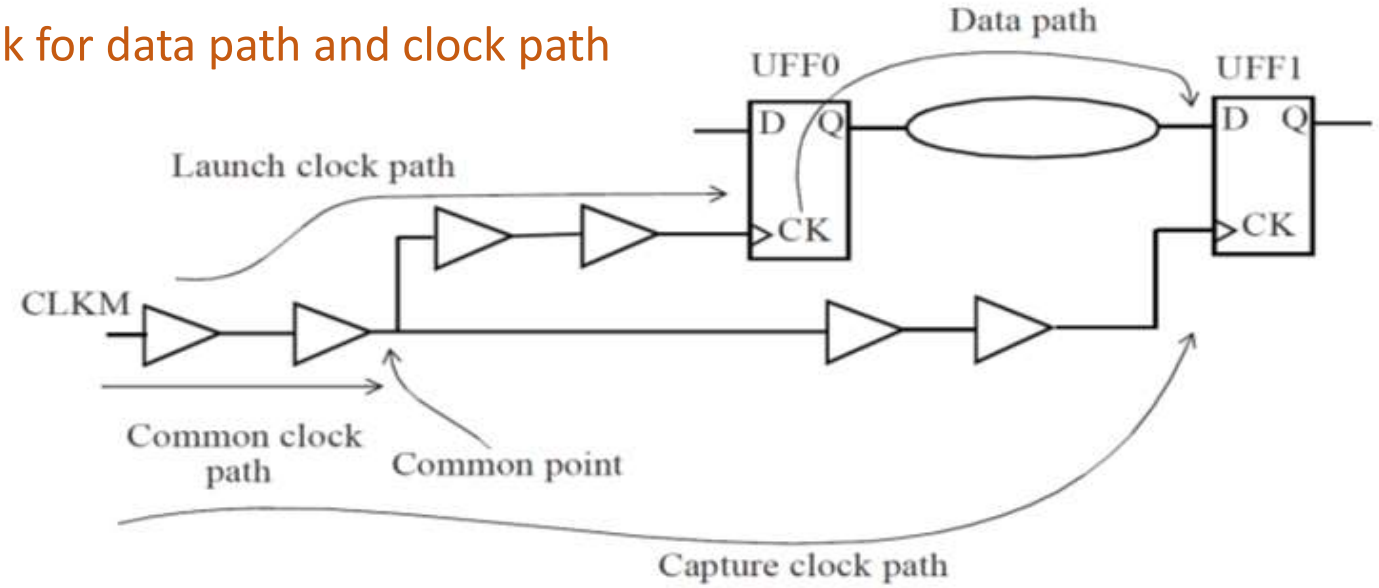
- The following four types of crosstalk delay contributions are computed for every cell and interconnect in the design:
  - Positive rise delay (rise edge moves forward in time)
  - Negative rise delay (rise edge moves backward in time)
  - Positive fall delay (fall edge moves forward in time)
  - Negative fall delay (fall edge moves backward in time)

- The crosstalk delay contributions are then utilized during timing analysis for the verification of the max and min paths (setup and hold checks). The clock path for the launch and capture flip-flops are handled differently.

Setup Analysis

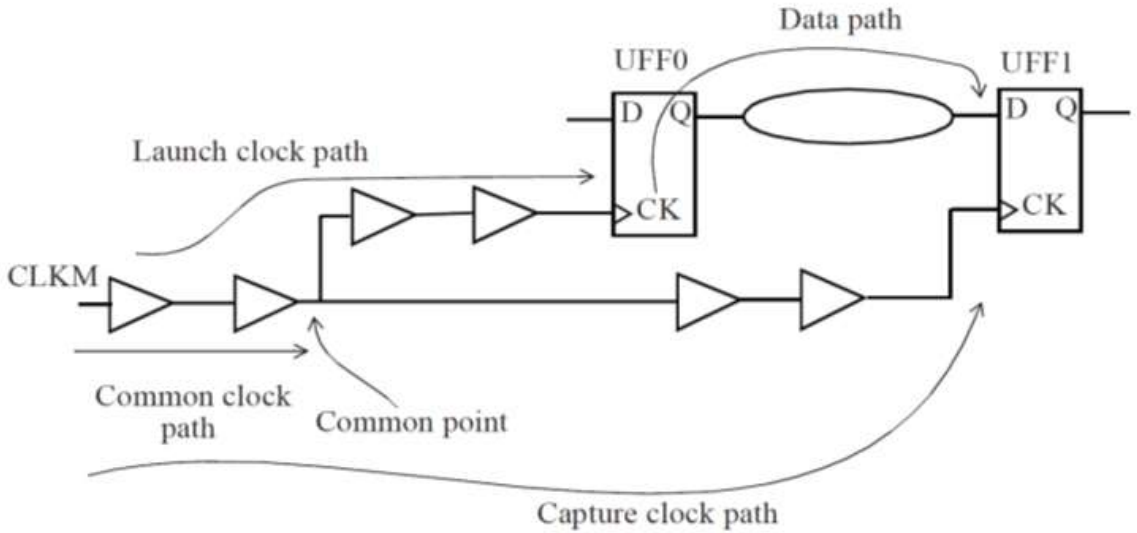
- The STA with crosstalk analysis verifies the design with the worst-case crosstalk delays for the data path and the clock paths.
- The worst condition for setup check is when both the launch clock path and the data path have positive crosstalk and the capture clock path has negative crosstalk.
- The positive crosstalk contributions on launch clock path and data path delay the arrival of data at the capture flip-flop.
- In addition, the negative crosstalk on capture clock path results in capture flip-flop being clocked early.

Crosstalk for data path and clock path



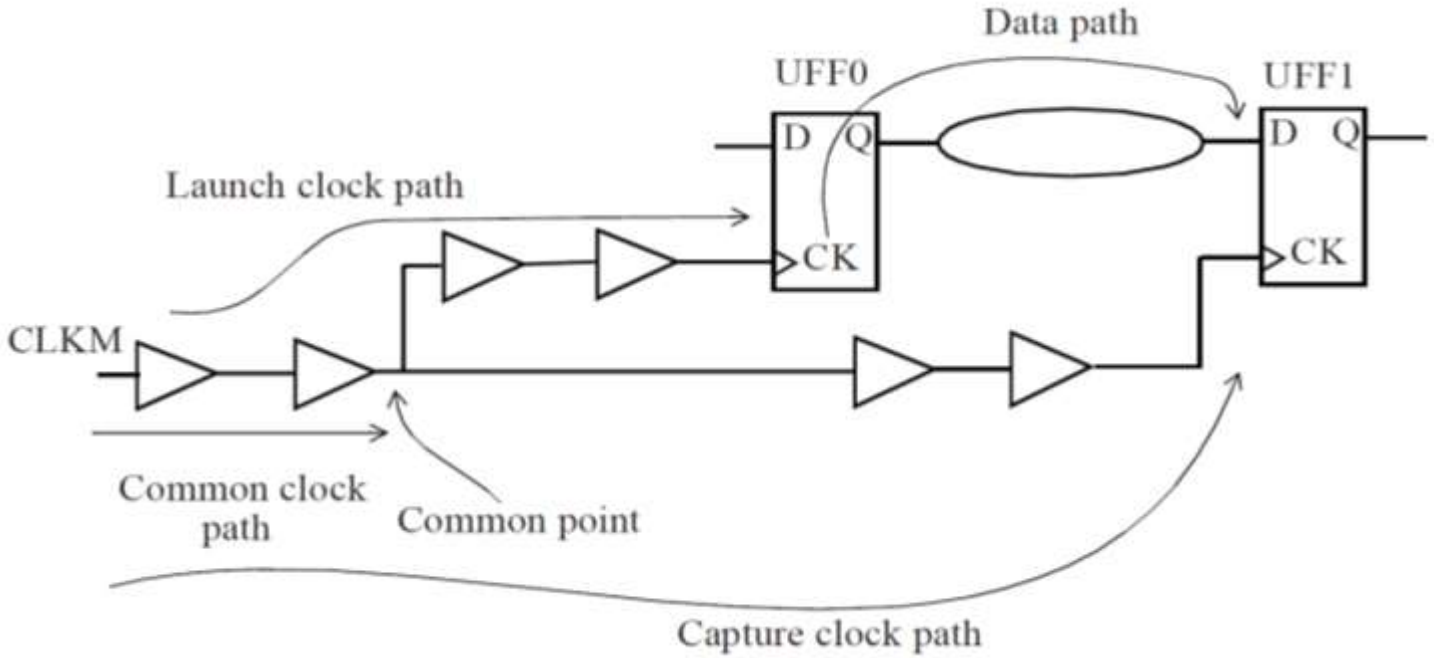
Setup Analysis

- The setup (or max path) analysis assumes that:
  - Launch clock path sees positive crosstalk delay so that the data is launched late.
  - Data path sees positive crosstalk delay so that it takes longer for the data to reach the destination.
  - Capture clock path sees negative crosstalk delay so that the data is captured by the capture flip-flop early.
- Since the launch and capture clock edges for a setup check are different (normally one clock cycle apart), the common clock path can have different crosstalk contributions for the launch and capture clock edges.



Hold Analysis

- The worst condition for hold check occurs when both the launch clock path and the data path have negative crosstalk and the capture clock path has positive crosstalk.
- The negative crosstalk contributions on launch clock path and data path result in early arrival of the data at the capture flip-flop. In addition, the positive crosstalk on capture clock path results in capture flip-flop being clocked late.

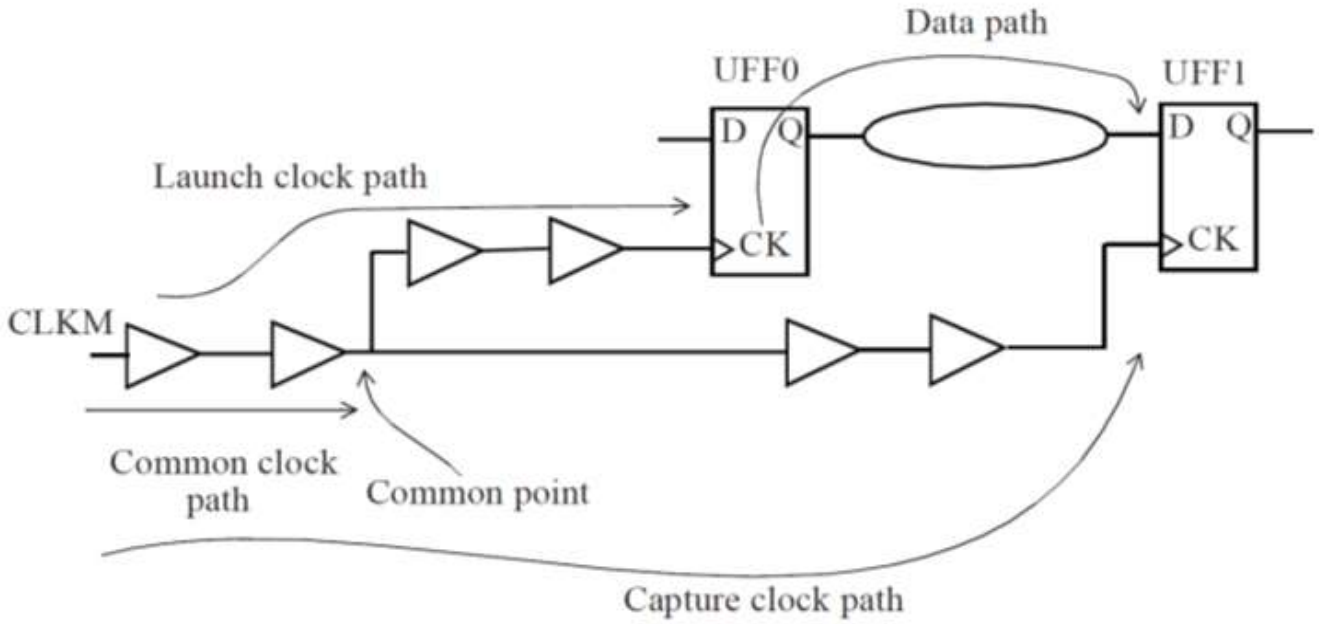


Hold Analysis

- There is one important difference between the hold and setup analyses related to crosstalk on the common portion of the clock path.
- The launch and capture clock edge are normally the same edge for the hold analysis. The clock edge through the common clock portion cannot have different crosstalk contributions for the launch clock path and the capture clock path.
- Therefore, the worst-case hold analysis removes the crosstalk contribution from the common clock path.

Hold Analysis

- The worst-case hold (or min path) analysis for STA with crosstalk assumes:
  - Launch clock (not including the common path) sees negative crosstalk delay so that the data is launched early.
  - Data path sees negative crosstalk delay so that it reaches the destination early.
  - Capture clock (not including the common path) sees positive crosstalk delay so that the data is captured by the capture flipflop late



Crosstalk, Setup and Hold

- The crosstalk impact on the common portion of the clock tree is not considered for the hold analysis.
- The positive crosstalk contribution of the launch clock and negative crosstalk contribution of the capture clock are only computed for the non-common portions of the clock tree.
- In STA reports for hold analysis, the common clock path may show different crosstalk contributions for the launch clock path and the capture clock path.
- However, the crosstalk contributions from the common clock path are removed as a separate line item labeled as common path pessimism removal.

Computational Complexity

- A large nanometer design is generally too complex to allow for every coupling capacitance to be analyzed with reasonable turnaround time.
- The parasitic extraction of a typical net contains coupling capacitances to many neighboring signals. A large design will normally require appropriate settings for the parasitic extraction and crosstalk delay and glitch analyses.
- These settings are selected to provide acceptable accuracy for the analyses while ensuring that the CPU requirements remain feasible.

Crosstalk delay Analysis: Computational Complexity

- For a large design, it is normally not practical to obtain parasitic extraction in one run. The parasitics for each hierarchical block can be extracted separately.

- This in turn requires that a hierarchical design methodology be used for the design implementation. This implies that there be no coupling between signals inside the hierarchical block and signals outside the block.

- This can be achieved either with no routing over the block or by adding a shield layer over the block.

- In addition, signal nets should not be routed close to the boundary of the block and any nets routed close to the boundary of the block should be shielded. This avoids any coupling with the nets from other blocks.

Computational Complexity: Filtering of Coupling Capacitances

- Even for a medium sized block, the parasitics will normally include a large number of very small coupling capacitances. The small coupling capacitances can be filtered during extraction or during the analysis procedures.

Computational Complexity: Filtering of Coupling Capacitances

- This filtering can be based upon the following criteria: -
  - **Small value:** During extraction, the small couplings (1fF) can be treated as grounded capacitances.
  - **Coupling ratio:** Aggressor nets with a small coupling ratio, for example, below 0.001, can be excluded from crosstalk delay or glitch analyses.
  - **Lumping small aggressors together:** Multiple aggressors with very small contributions can be mapped to one larger virtual aggressor. This can be pessimistic but can simplify the analysis. Some of the possible pessimism can be mitigated by switching a subset of the aggressors. The exact subset of switching aggressors can be determined by statistical methods.

Noise Avoidance Techniques

- **Shielding:** This method requires that shield wires are placed on either side of the critical signals. The shields are connected to power or ground rails.
- The shielding of critical signals ensures that there are no active aggressors for the critical signals since the nearest neighbors in the same metal layer are shield traces at a fixed potential. While there can be some coupling from routes in the different metal layers, most of the coupling capacitances are due to the capacitive coupling in the same layer.
- Since the immediate metal layers (above and below) would normally be routed orthogonally, the capacitive coupling across layers is minimized. Thus, placing shield wires in the same metal layer ensures that there is minimal coupling for the critical signals.

Noise Avoidance Techniques

- **Wire spacing:** This reduces the coupling to the neighboring nets.
- **Fast slew rate:** A fast slew rate on the net implies that the net is less susceptible to crosstalk and is inherently immune to crosstalk effects.
- **Maintain good stable supply:** This is important not for crosstalk but for minimizing jitter due to power supply variations. Significant noise can be introduced on the clock signals due to noise on the power supply. Adequate decoupling capacitances should be added to minimize noise on the power supply.

Noise Avoidance Techniques

- **Guard ring:** A guard ring (or double guard ring) in the substrate helps in shielding the critical analog circuitry from digital noise.
- **Deep n-well:** This is similar to the above as having deep n-well for the analog portions helps prevent noise from coupling to the digital portions.
- **Isolating a block:** In a hierarchical design flow, routing halos can be added to the boundary of the blocks; furthermore, isolation buffers could be added to each of the IO of the block.

Summary

- Noise Accumulation with Multiple Aggressors
- Aggressor Timing Correlation
- Aggressor Functional Correlation
- Positive and Negative Glitch

Next Session

Creating STA Environment

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Digital System Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

DIGITAL SYSTEM DESIGN

ON-Chip Variations

Sudeendra kumar K
Department of Electronics and Communication Engineering

On-Chip Variations

- In general, the process and environmental parameters may not be uniform across different portions of the die. Due to process variations, identical MOS transistors in different portions of the die may not have similar characteristics.
- These differences are due to process variations within the die. Note that the process parameter variations across multiple manufactured lots can cover the entire span of process models from slow to fast

On-Chip Variations

- The analysis of the process variations possible on one die (called local process variations) which are much smaller than the variations across multiple manufacturing lots (called global process variations).

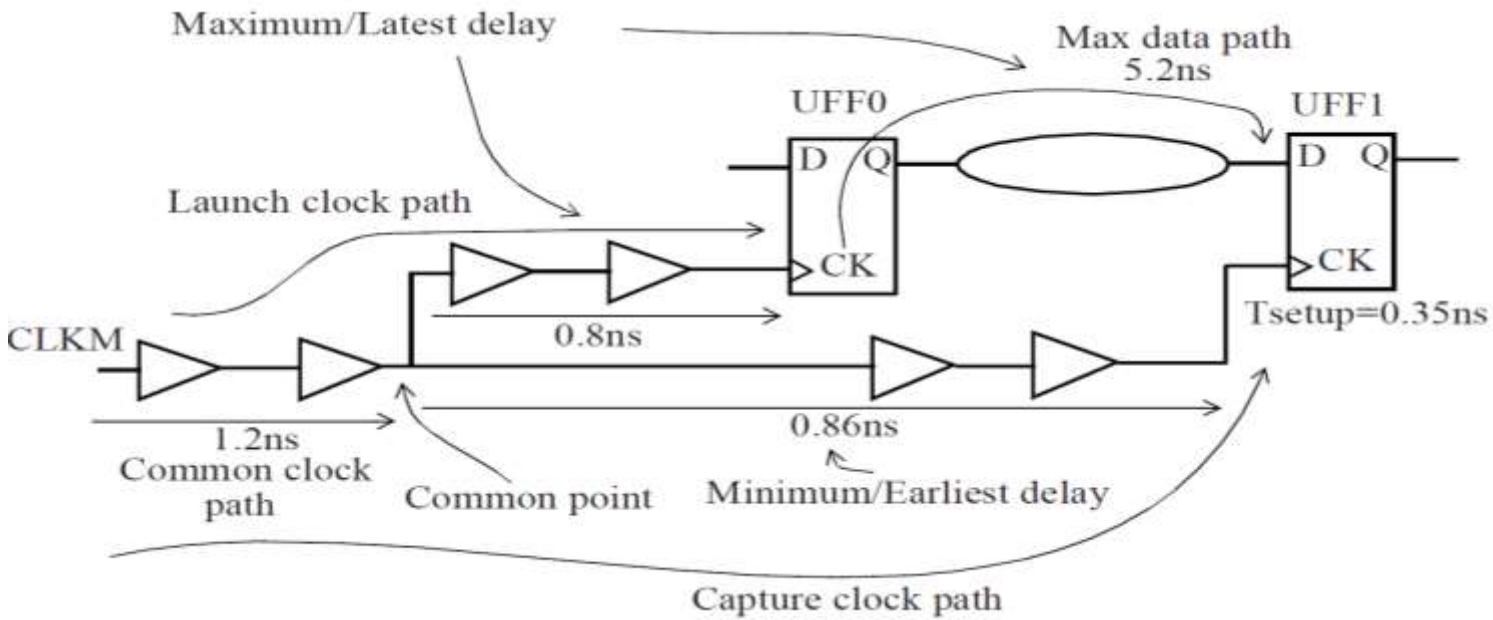
On-Chip Variations

- Besides the variations in the process parameters, different portions of the design may also see different power supply voltage and temperature. It is therefore possible that two regions of the same chip are not at identical PVT conditions. These differences can arise due to many factors, including:
  - IR drop variation along the die area affecting the local power supply.
  - Voltage threshold variation of the PMOS or the NMOS device.
  - Channel length variation of the PMOS or the NMOS device.
  - Temperature variations due to local hot spots.
  - Interconnect metal etch or thickness variations impacting the interconnect resistance or capacitance.

On-Chip Variations

- The PVT variations described above are referred to as On-Chip Variations (OCV) and these variations can affect the wire delays and cell delays in different portions of the chip.
- The OCV effect is typically more pronounced on clock paths as they travel longer distances in a chip. One way to account for the local PVT variations is to incorporate the OCV analysis during STA.
- Since the clock and data paths can be affected differently by the OCV, the timing verification can model the OCV effect by making the PVT conditions for the launch and capture paths to be slightly different.
- The STA can include the OCV effect by derating the delays of specific paths, that is, by making those paths faster or slower and then validating the behavior of the design with these variations.
- The cell delays or wire delays or both can be derated to model the effect of OCV.

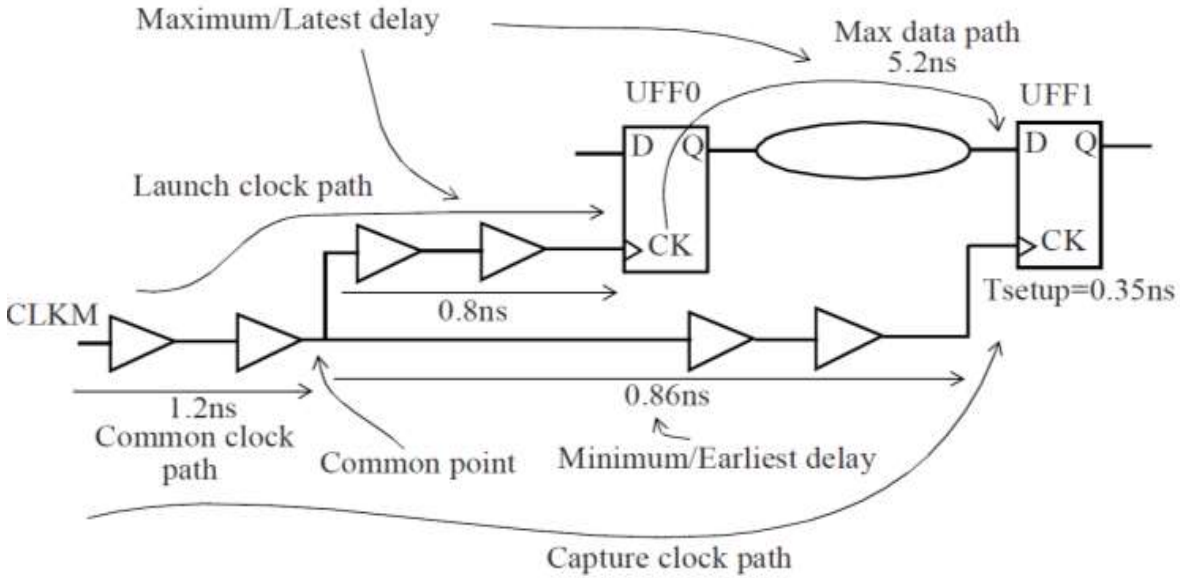
Derating setup timing check for OCV



- The worst condition for setup check occurs when the launch clock path and the data path have the OCV conditions which result in the largest delays, while the capture clock path has the OCV conditions which result in the smallest delays.
- Note that the smallest and largest here are due to local PVT variations on a die.

- The setup timing check condition; this does not include any OCV setting for derating delays.

```
LaunchClockPath + MaxDataPath <= ClockPeriod +
 CaptureClockPath - Tsetup_UFF1

This implies that the minimum clock period = LaunchClockPath +
 MaxDataPath - CaptureClockPath + Tsetup_UFF1

From the figure,
  LaunchClockPath = 1.2 + 0.8 = 2.0
  MaxDataPath = 5.2
  CaptureClockPath = 1.2 + 0.86 = 2.06
  Tsetup_UFF1 = 0.35

This results in a minimum clock period of:
  2.0 + 5.2 - 2.06 + 0.35 = 5.49ns
```



- The above path delays correspond to the delay values without any OCV derating. Cell and net delays can be derated using the **set_timing_derate** specification. For example, the commands:

```
set_timing_derate -early 0.8
set_timing_derate -late 1.1
```

- derate the minimum/shortest/early paths by -20% and derate the maximum/ longest/latest paths by +10%.
- Long path delays (for example, data paths and launch clock path for setup checks or capture clock paths for hold checks) are multiplied by the derate value specified using the -late option, and short path delays (for example, capture clock paths for setup checks or data paths and launch clock paths for hold checks) are multiplied by the derate values specified using the -early option. If no derating factors are specified, a value of 1.0 is assumed.

On-Chip Variations

- The derating factors apply uniformly to all net delays and cell delays. If an application scenario warrants different derating factors for cells and nets, the **-cell_delay** and the **-net_delay** options can be used in the **set_timing_derate** specification.

```
# Derate only the cell delays - early paths by -10%, and
# no derate on the late paths:
set_timing_derate -cell_delay -early 0.9
set_timing_derate -cell_delay -late 1.0


# Derate only the net delays - no derate on the early paths
# and derate the late paths by +20%:
set_timing_derate -net_delay -early 1.0
set_timing_derate -net_delay -late 1.2
```

On-Chip Variations

- Cell check delays, such as setup and hold of a cell, can be derated by using the **-cell_check** option.
- With this option, any output delay specified using **set_output_delay** is also derated as this specification is part of the setup requirement of that output.
- However, no such implicit derating is applied for the input delays specified using the **set_input_delay** specification.

```
# Derate the cell timing check values:
set_timing_derate -early 0.8 -cell_check
set_timing_derate -late 1.1 -cell_check

# Derate the early clock paths:
set_timing_derate -early 0.95 -clock
# Derate the late data paths:
set_timing_derate -late 1.05 -data
```

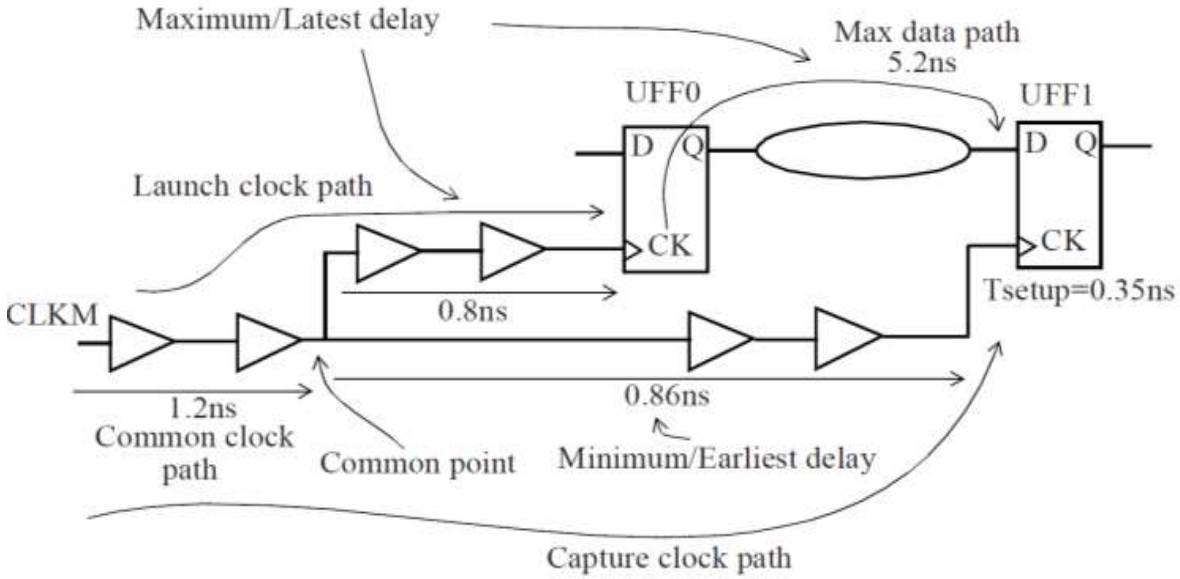
The *-clock* option applies derating only to clock paths. Similarly, the *-data* option applies derating only to data paths.

```
set_timing_derate -early 0.9
set_timing_derate -late 1.2
set_timing_derate -late 1.1 -cell_check
```

With these derating values, we get the following for setup check:

LaunchClockPath = 2.0 * 1.2 = 2.4
MaxDataPath = 5.2 * 1.2 = 6.24
CaptureClockPath = 2.06 * 0.9 = 1.854
Tsetup_UFF1 = 0.35 * 1.1 = 0.385

This results in a minimum clock period of:
2.4 + 6.24 - 1.854 + 0.385 = 7.171ns



On-Chip Variations: CPPR

- In the setup check above, there is a discrepancy since the common clock path of the clock tree, with a delay of 1.2ns, is being derated differently for the launch clock and for the capture clock.
- This part of the clock tree is common to both the launch clock and the capture clock and should not be derated differently.
- Applying different derating for the launch and capture clock is overly pessimistic as in reality this part of the clock tree will really be at only one PVT condition, either as a maximum path or as a minimum path but never both at the same time.
- The pessimism caused by different derating factors applied on the common part of the clock tree is called Common Path Pessimism (CPP) which should be removed during the analysis.
- CPPR (Common Path Pessimism Removal), is listed as a separate item in a path report.

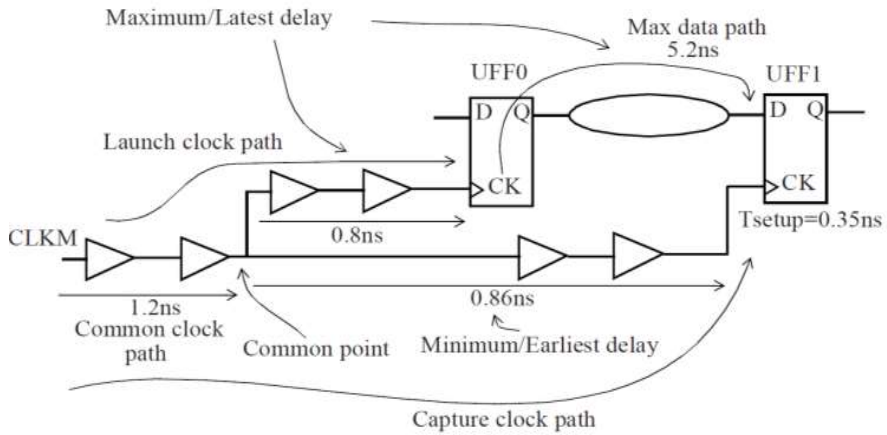
On-Chip Variations

- CPPR is the removal of artificially induced pessimism between the launch clock path and the capture clock path in timing analysis.
- If the same clock drives both the capture and the launch flip-flops, then the clock tree will likely share a common portion before branching.
- CPP itself is the delay difference along this common portion of the clock tree due to different deratings for launch and capture clock paths.
- The difference between the minimum and the maximum arrival times of the clock signal at the common point is the CPP.
- The common point is defined as the output pin of the last cell in the common portion of the clock tree.

```
CPP = LatestArrivalTime@CommonPoint -
  EarliestArrivalTime@CommonPoint
```

- The Latest and Earliest times in the above analysis are in reference to the OCV derating at a specific timing corner - for example worst-case slow or best-case fast.

```
LatestArrivalTime@CommonPoint = 1.2 * 1.2 = 1.44
EarliestArrivalTime@CommonPoint = 1.2 * 0.9 = 1.08

This implies a CPP of: 1.44 - 1.08 = 0.36ns

With the CPP correction, this results in a
minimum clock period of: 7.171 - 0.36 = 6.811ns
```



- Applying the OCV derating has increased the minimum clock period from 5.49ns to 6.811ns for this example design.
- This illustrates that the OCV variations modeled by these derating factors can reduce the maximum frequency of operation of the design.

Analysis with OCV at Worst PVT Condition

- If the setup timing check is being performed at the worst-case PVT condition, no derating is necessary on the late paths as they are already the worst possible.
- However, derating can be applied to the early paths by making those paths faster by using a specific derating, for example, speeding up the early paths by 10%.
- A derate specification at the worst-case slow corner may be something like:

```
set_timing_derate -early 0.9
set_timing_derate -late 1.0
# Don't derate the late paths as they are already the slowest,
# but derate the early paths to make these faster by 10%.
```

- The above derate settings are for max path (or setup) checks at the worst case slow corner; thus the late path OCV derate setting is kept at 1.0 so as not to slow it beyond the worst-case slow corner.

Analysis with OCV at Worst PVT Condition

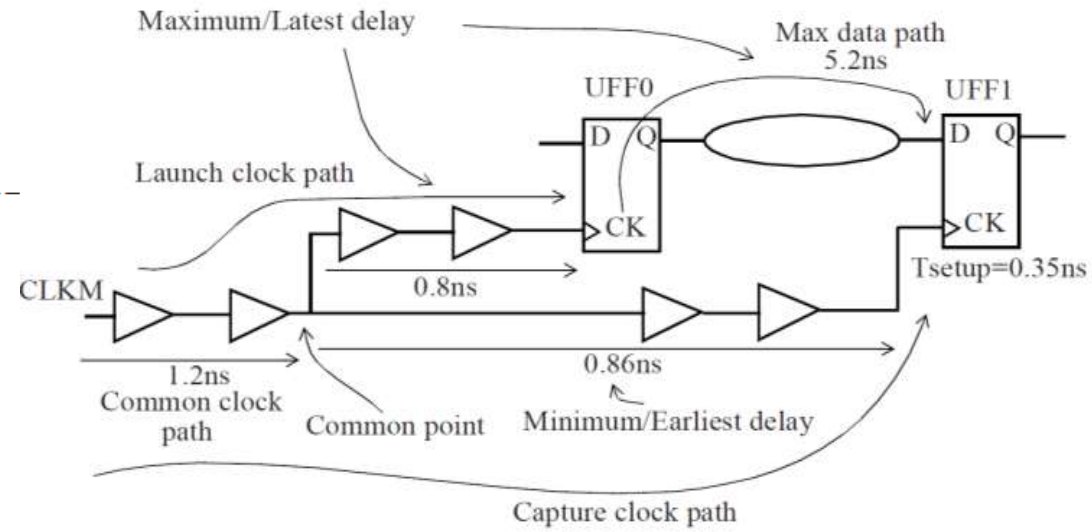
- The derate specification is specified for the capture clock path below:

```
# Derate the early clock paths:
set_timing_derate -early 0.8 -clock
```

- The derating used by the late paths are reported as **Max Data Paths Derating Factor** and as **Max Clock Paths Derating Factor**. The derating used for the early paths is reported as **Min Clock Paths Derating Factor.**

Digital System Design
Analysis with OCV at Worst PVT Condition

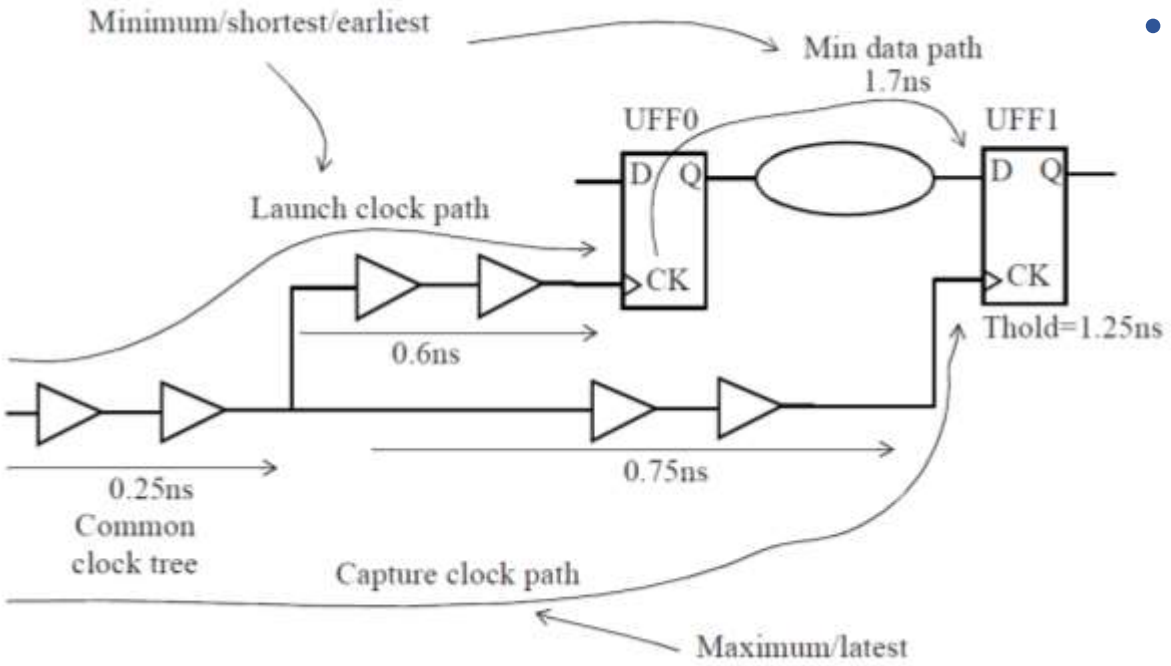
```
Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)
  Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)
  Path Group: CLKM
  Path Type: max
  Max Data Paths Derating Factor   : 1.000
  Min Clock Paths Derating Factor : 0.800
  Max Clock Paths Derating Factor : 1.000

  Point                                        Incr       Path
  ------------------------------------------------------------
  clock CLKM (rise edge)                      0.000      0.000
  clock source latency                        0.000      0.000
  CLKM (in)                                   0.000      0.000 r
  UCKBUF0/C (CKB   )                          0.056      0.056 r
  UCKBUF1/C (CKB   )                          0.058      0.114 r
  UFF0/CK (DF   )                             0.000      0.114 r
  UFF0/Q (DF   ) <-                           0.143      0.258 f
  UNOR0/ZN (NR2   )                           0.043      0.301 r
  UBUF4/Z (BUFF   )                           0.052      0.352 r
  UFF1/D (DF   )                              0.000      0.352 r
  data arrival time                                      0.352

  clock CLKM (rise edge)                     10.000     10.000
  clock source latency                        0.000     10.000
  CLKM (in)                                   0.000     10.000 r
  UCKBUF0/C (CKB   )                          0.045     10.045 r
  UCKBUF2/C (CKB   )                          0.054     10.098 r
  UFF1/CK (DF   )                             0.000     10.098 r
  clock reconvergence pessimism               0.011     10.110
  clock uncertainty                          -0.300      9.810
  library setup time                         -0.044      9.765
  data required time                                     9.765
  ------------------------------------------------------------
  data required time                                     9.765
  data arrival time                                     -0.352
  ------------------------------------------------------------
  slack (MET)                                            9.413
```



Analysis with OCV at Worst PVT Condition

- Notice that the capture clock path is derated by 20%. See cell UCKBUF0 in the timing report. In the launch path, it has a delay of 56ps, while it has a derated delay of 45ps in the capture path.
- The cell UCKBUF0 is on the common clock path, that is, on both the capture clock path and the launch clock path.
- Since the common clock path cannot have a different derating, the difference in timing for this common path, 56ps - 45ps = 11ps, is corrected separately. This appears as the line clock reconvergence pessimism in the report.
- In summary, if one were to compare the reports of this path, with and without derating, one would notice that only the cell and net delays for the capture clock path have been derated.

OCV for Hold Checks



- If the PVT conditions are different along the chip, the worst condition for hold check occurs when the launch clock path and the data path have OCV conditions which result in the smallest delays, that is, when we have the earliest launch clock, and the capture clock path has the OCV conditions which result in the largest delays, that is, has the latest capture clock.

(without applying any derating):

```
LaunchClockPath + MinDataPath - CaptureClockPath -
  Thold_UFF1 >= 0
```

```
LaunchClockPath = 0.25 + 0.6 = 0.85
MinDataPath = 1.7
CaptureClockPath = 0.25 + 0.75 = 1.00
Thold_UFF1 = 1.25
```

This implies that the condition is:

```
0.85 + 1.7 - 1.00 - 1.25 = 0.3n >=0
```

which is true, and thus no hold violation exists.

Digital System Design
OCV for Hold Checks

Applying the following derate specification:

```
set_timing_derate -early 0.9
set_timing_derate -late 1.2
set_timing_derate -early 0.95 -cell_check
```

we get:

```
LaunchClockPath = 0.85 * 0.9 = 0.765
MinDataPath = 1.7 * 0.9 = 1.53

CaptureClockPath = 1.00 * 1.2 = 1.2
Thold_UFF1 = 1.25 * 0.95 = 1.1875

Common clock path pessimism: 0.25 * (1.2 - 0.9) = 0.075
```

- Common clock path pessimism created by applying derating on the common clock tree for both launch and capture clock paths is also removed for hold timing checks. The hold check condition then becomes:

$$0.765 + 1.53 - 1.2 - 1.1875 + 0.075 = -0.0175\text{ns}$$

Digital System Design
OCV Hold Time Checks

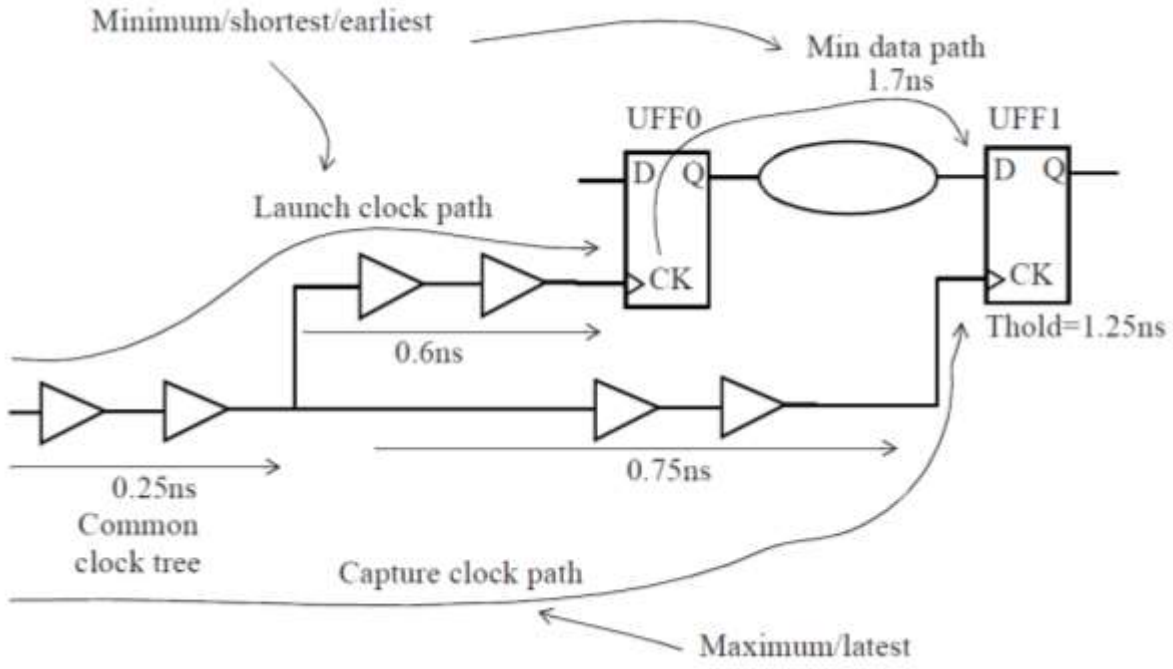
In general, the hold timing check is performed at the best-case fast PVT corner. In such a scenario, no derating is necessary on the early paths, as those paths are already the earliest possible. However, derating can be applied on the late paths by making these slower by a specific derating factor, for example, slowing the late paths by 20%. A derate specification at this corner would be something like:

```
set_timing_derate -early 1.0
set_timing_derate -late 1.2
# Don't derate the early paths as they are already the
# fastest, but derate the late paths slower by 20%.
```

```
LatestArrivalTime@CommonPoint = 0.25 * 1.2 = 0.30
EarliestArrivalTime@CommonPoint = 0.25 * 1.0 = 0.25
```

This implies a common path pessimism of:

```
0.30 - 0.25 = 0.05ns
```



Digital System Design
OCV Hold Time Checks

```
Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)
  Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)
  Path Group: CLKM
  Path Type: min
  Min Data Paths Derating Factor   : 1.000
  Min Clock Paths Derating Factor : 1.000
  Max Clock Paths Derating Factor : 1.200

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock CLKM (rise edge)                  0.000      0.000
  clock source latency                    0.000      0.000
  CLKM (in)                               0.000      0.000 r
  UCKBUF0/C (CKB   )                      0.056      0.056 r
  UCKBUF1/C (CKB   )                      0.058      0.114 r
  UFF0/CK (DF   )                         0.000      0.114 r
  UFF0/Q (DF   ) <-                       0.144      0.258 r
  UNOR0/ZN (NR2   )                       0.021      0.279 f
  UBUF4/Z (BUFF   )                       0.055      0.334 f
  UFF1/D (DF   )                          0.000      0.334 f
  data arrival time                                  0.334

  clock CLKM (rise edge)                  0.000      0.000
  clock source latency                    0.000      0.000
  CLKM (in)                               0.000      0.000 r
  UCKBUF0/C (CKB   )                      0.067      0.067 r
  UCKBUF2/C (CKB   )                      0.080      0.148 r
  UFF1/CK (DF   )                         0.000      0.148 r
  clock reconvergence pessimism          -0.011      0.136
  clock uncertainty                       0.050      0.186
  library hold time                       0.015      0.201
  data required time                                 0.201
  -----------------------------------------------------------
  data required time                                 0.201
  data arrival time                                 -0.334
  -----------------------------------------------------------
  slack (MET)                                        0.133
```

- Notice that the late paths are derated by +20% while the early paths are not derated. See cell UCKBUF0. Its delay on the launch path is 56ps while the delay on the capture path is 67ps - derated by +20%. UCKBUF0 is the cell on the common clock tree and thus the pessimism introduced due to different derating on this common clock tree is, 67ps - 56ps = 11ps, which is accounted for separately on the line clock reconvergence pessimism.

Multi Vt Cells

- The multi Vt cells are used to tradeoff speed with leakage. The high Vt cells have less leakage, though these are slower than the standard Vt cells which are faster but have higher leakage.
- Similarly, the low Vt cells are faster than standard Vt cells but the leakage is also correspondingly higher.
- Leakage can be a significant component of the total power, implementing a design with only high Vt cells to reduce leakage can increase the total power consumption even though the leakage contribution may be reduced.
- However, there are scenarios where the leakage is a dominant component of the total power; in such cases, a design with high Vt cells can result in reduction of the total power.
- The above trade-off between cells with different Vt in terms of their speed and leakage needs to be utilized suitably since it is dependent on the design and its switching activity profile.

High Performance Block with High Activity

- This scenario is of a high performance block with high switching activity and the power is dominated by the active power. For such blocks, focusing only on reducing leakage power can cause the total power to increase even though the leakage contribution may be minimized.
- In such cases, the initial design implementation should use standard Vt (or low Vt) cells to meet the desired performance. After the required timing is achieved, the cells along the paths which have positive timing slack can be changed into high Vt cells so that the leakage contribution is reduced while still meeting the timing requirement.
- Thus, in the final implementation, the standard Vt (or low Vt) cells are used only along the critical or hard to achieve timing

High Performance Block with Low Activity

- This scenario is of a high performance block with very low switching activity so that the leakage power is a significant component of the total power.
- Since the block has low activity, the active power is not a major component for the total power of the design. For such blocks, the initial implementation attempts to use only high Vt cells in the combinational logic and flipflops.
- An exception is the clock tree which is always active and therefore is built with standard Vt (or low Vt) cells.
- After the initial implementation with only high Vt cells, there may be some timing paths where the required timing cannot be achieved.
- The cells along such paths are then replaced with standard Vt (or low Vt) cells to achieve the required timing performance.

Where to look for problems: Large Delays and Transition Times

- One key item is to check for unusually large values for the delays or transition times along the data path. Some of these can be due to:
  - High-fanout nets: Nets which are not buffered properly.
  - Long nets: Nets which need buffer insertion in between.
  - Low strength cells: Cells which may not have been replaced because these are labeled as don't touch in the design.
  - Memory paths: Paths that typically fail due to large setup times on memory inputs and large output delays on memory outputs.

What if Timing Still Cannot be Met

- One can utilize **useful skew** to help close the timing. Useful skew is where one purposely imbalances the clock trees, especially the launch and capture clock paths of a failing path so that the timing passes on that path.
- It typically means that the capture clock can be delayed so that the clock at the capture flip-flop arrives later when the data is ready. This does assume that there is enough slack on the succeeding data paths, that is, the data path for the next stage of flip-flop to flip-flop paths.
- The reverse can also be attempted, that is, the launch clock path can be made shorter so that the data from the launch flip-flop is launched earlier to help meet the setup timing.
- Useful skew techniques can be used to fix both setup and hold violations. One disadvantage of this technique is that if the design has multiple modes of operation, then useful skew can potentially cause a problem in another mode.

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Advanced Digital Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Lecture-1: Synchronization Fundamentals

Unit-IV-Synchronization

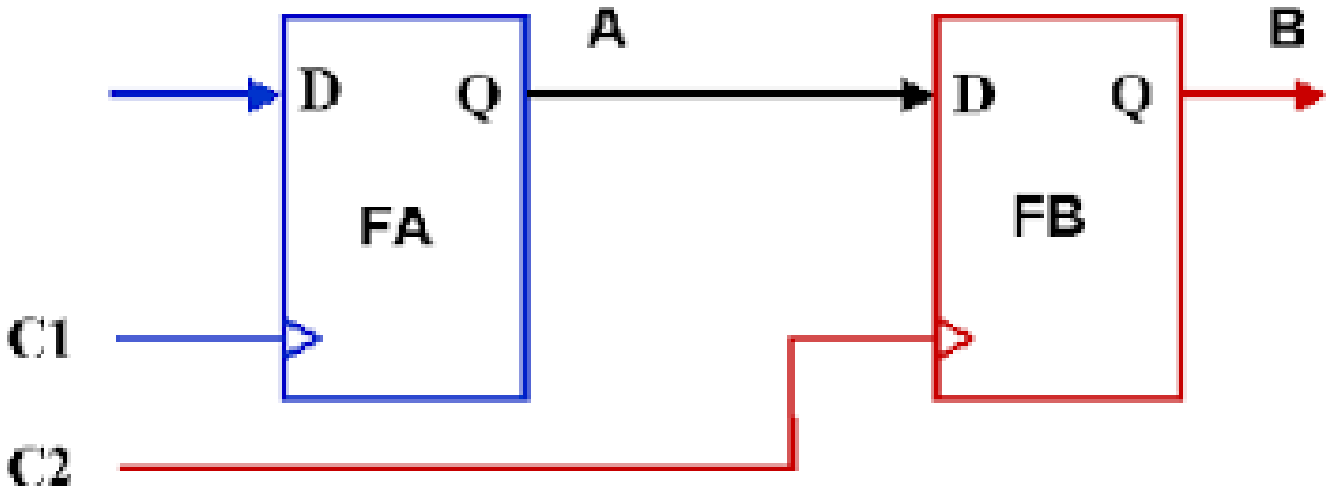
Sudeendra kumar K
Department of Electronics and Communication Engineering

Advanced Digital Design and Verification

Contents

- Synchronization fundamentals

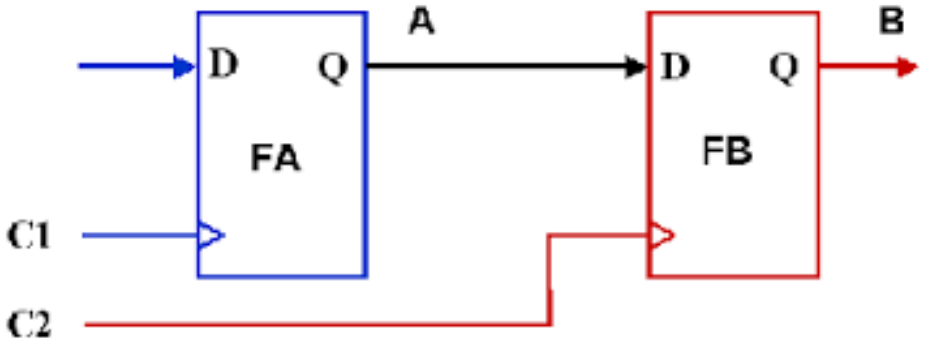
Advanced Digital Design and Verification
Clock Domain Crossing

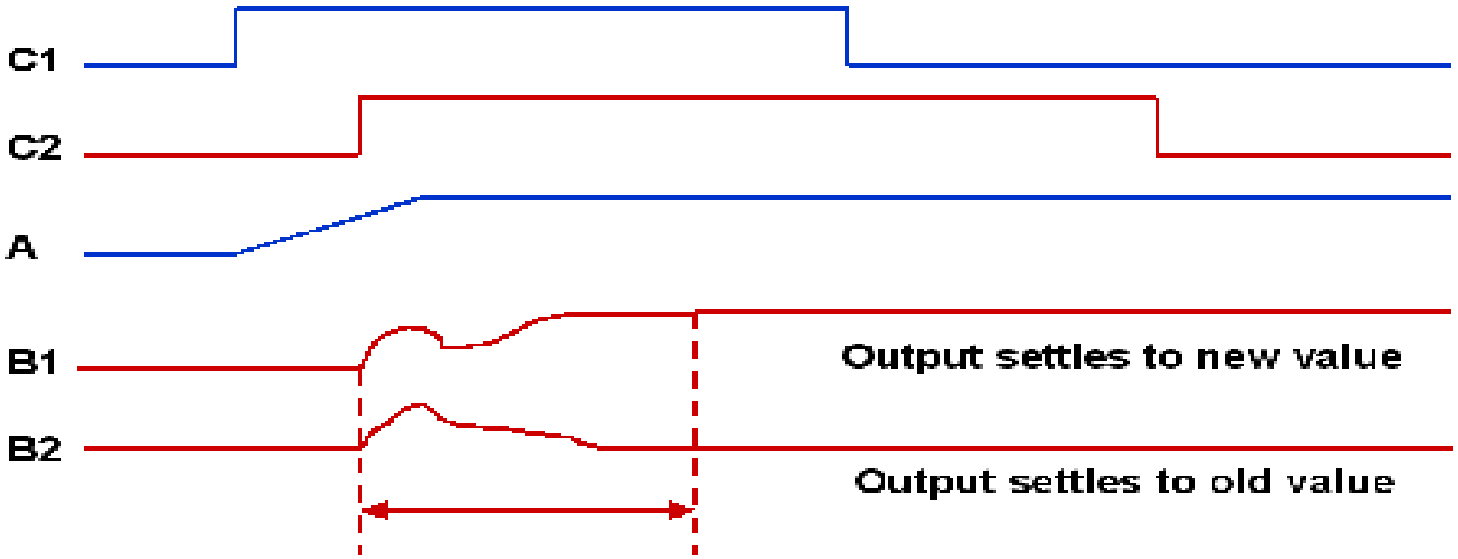


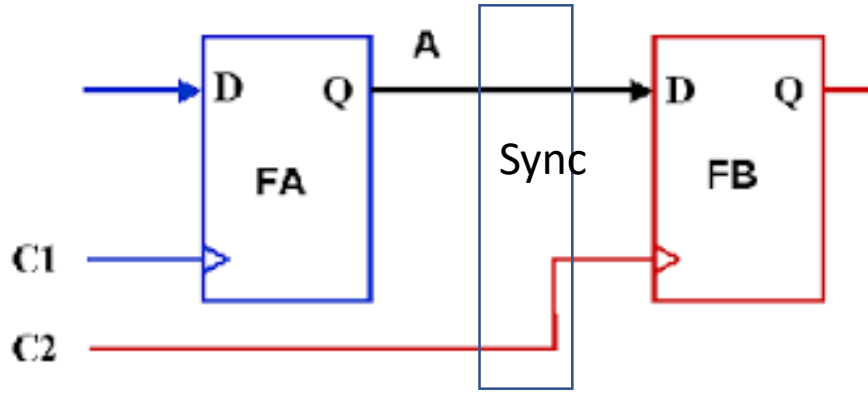
- A clock domain crossing occurs whenever data is transferred from a flop driven by one clock to a flop driven by another clock.
- Signal A is launched by the C1 clock domain and needs to be captured properly by the C2 clock domain.
- Depending on the relationship between the two clocks, there could be different types of problems in transferring data from the source clock to the destination clock. Along with that, the solutions to those problems can also be different.

Clock Domain Crossing: Metastability Problem

- If the transition on signal A happens very close to the active edge of clock C2, it could lead to setup or hold violation at the destination flop "FB". As a result, the output signal B may oscillate for an indefinite amount of time. Thus the output is unstable and may or may not settle down to some stable value before the next clock edge of C2 arrives. This phenomenon is known as **metastability** and the flop "FB" is said to have entered a metastable state.



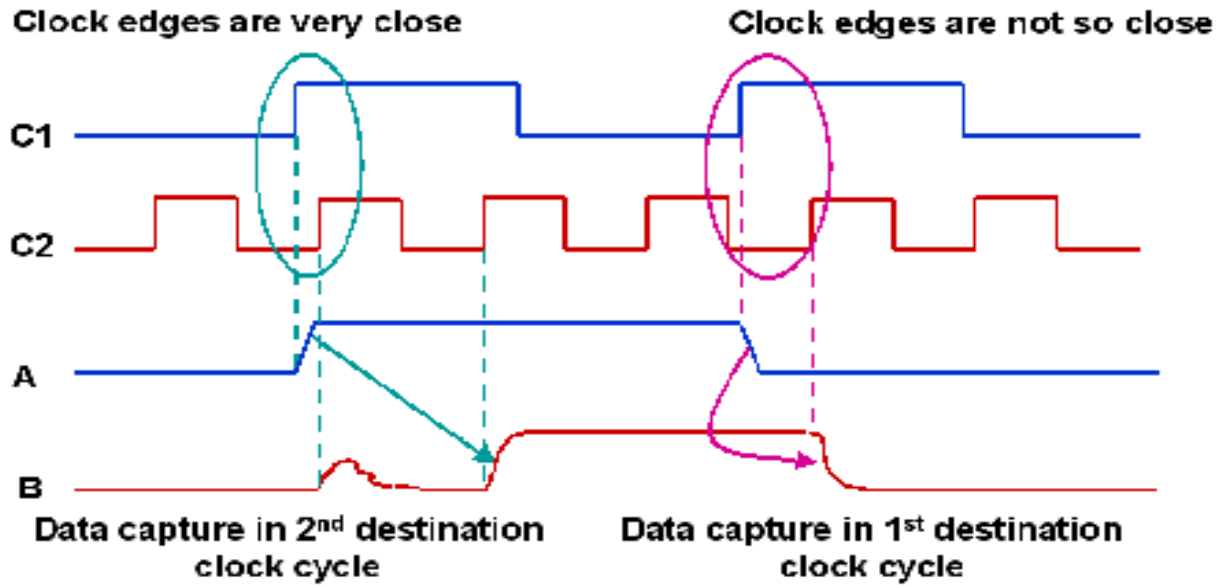


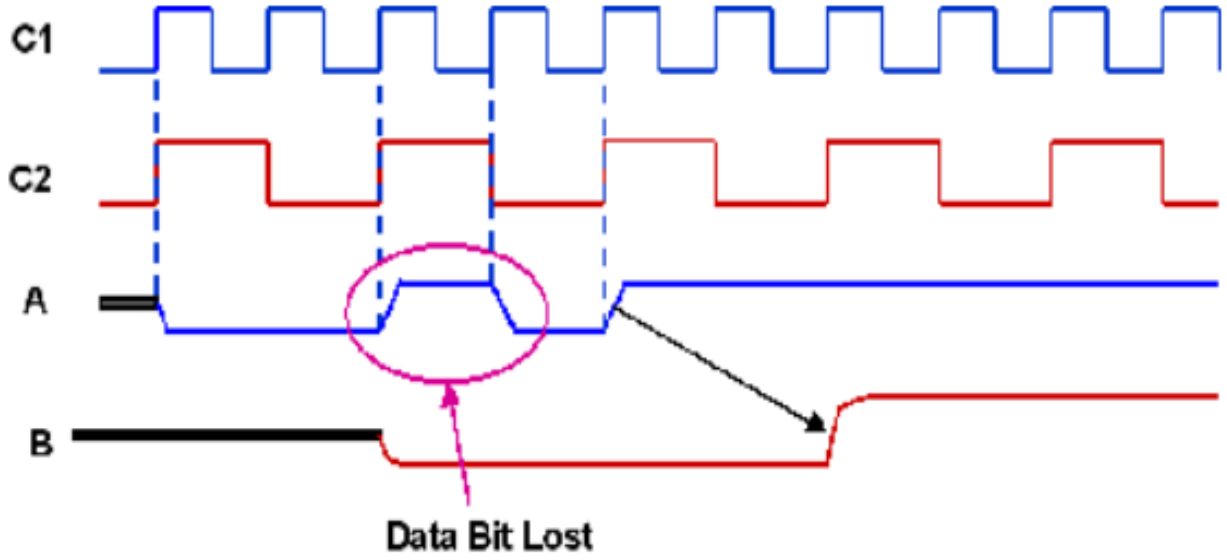


Advanced Digital Design and Verification

Clock Domain Crossing: Data Loss

- As long as each transition on the source signal is captured in the destination domain, data is not lost. In order to ensure this, the source data should remain stable for some minimum time, so that the setup and hold time requirements are met with respect to at least one active edge of destination clock.
- If the active clock edges of C1 and C2 arrive close together, the first clock edge of C2, which comes after the transition on source data A, is not able to capture it. The data finally gets captured by the second edge of clock C2.
- However, if there is sufficient time between the transition on data A and the active edge of clock C2, the data is captured in the destination domain in the first cycle of C2.





Advanced Digital Design and Verification

Synchronous Clock Domain Crossing

- Clocks which have a known phase and frequency relationship between them are known as synchronous clocks. These are essentially the clocks originating from the same clock-root.
- A clock crossing between such clocks is known as a synchronous clock domain crossing. It can be divided into several categories based on the phase and frequency relationship of the source and destination clocks as follows:
  - Clocks with the same frequency and zero phase difference
  - Clocks with the same frequency and constant phase difference
  - Clocks with different frequency and variable phase difference
    - Integer multiple clocks
    - Rational multiple clocks

Synchronization Basics

- Synchronization must be performed in a bounded amount of time, some probability exists of synchronization failure. That is, if the two events happen very close together, it may not be possible for a synchronizer circuit to resolve unambiguously which occurred first in a fixed amount of time.
- A properly designed synchronizer can make the probability of synchronization failure arbitrarily small by waiting arbitrarily long. However, several rather common mistakes can result in very high failure probabilities.
- It is possible to synchronize without any probability of failure, provided one is willing to wait as long as it takes for the synchronizer to signal that it has completed the task.

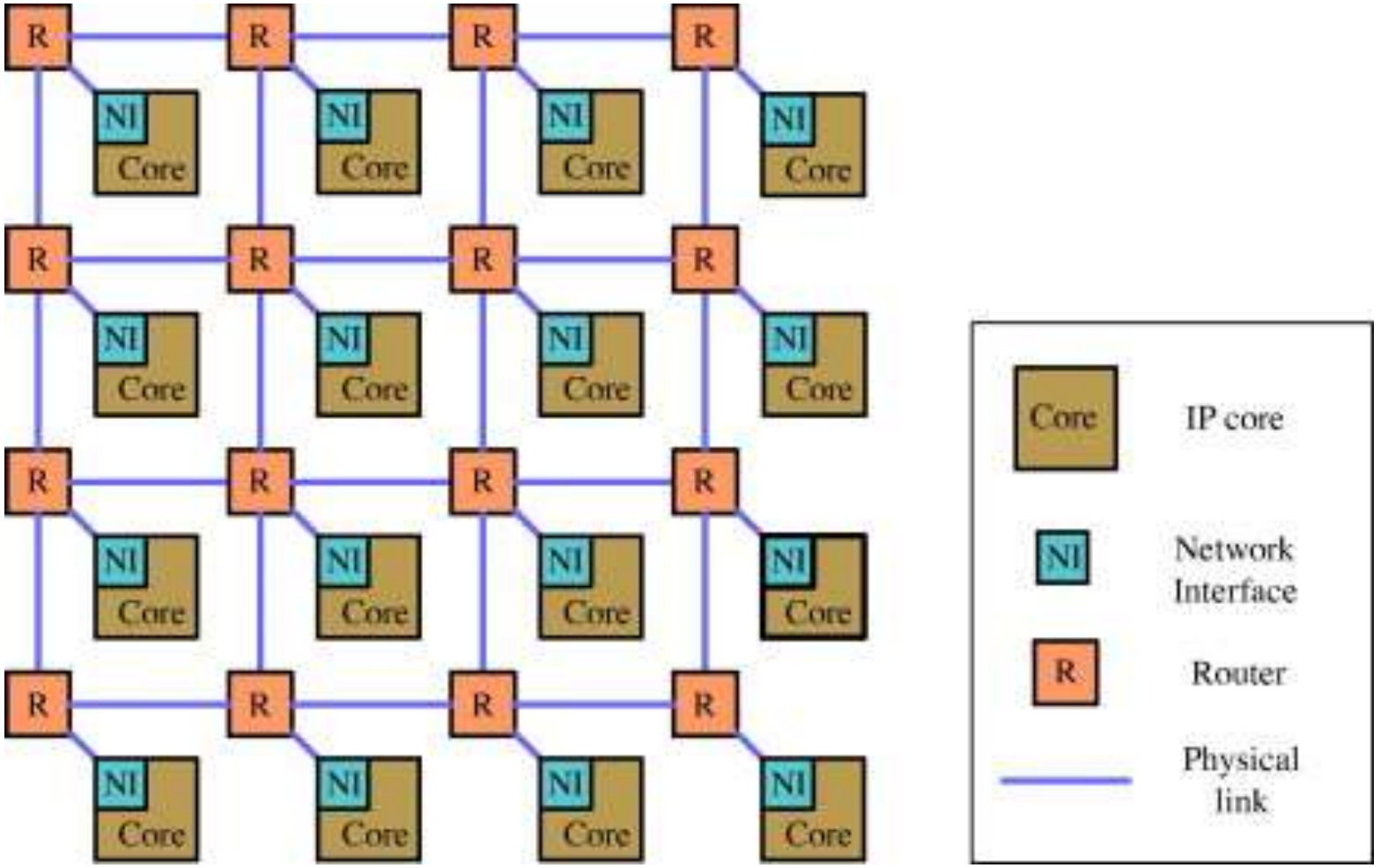
Synchronization Types

- The difficulty of synchronizing a signal with a clock depends on the predictability of events on the signal relative to the clock. The easiest case is when the signal is synchronized to a clock with the same frequency as the sample clock but with an arbitrary phase.
- In this case we say the signal and clock are **Mesochronous,** and a single phase measurement suffices to predict possible transition times arbitrarily far in the future.
- If the signal is generated by a clock with a slightly different frequency, there is a slow drift in the phase, and we say that the signal and clock are **Plesiochronous**.

Synchronous: both clocks are same in all aspects like phase and freq.
Meso: same freq, constant phase difference
Plesio: slight difference in freq and variable phase difference
Async: completely unrelated clocks.

Advanced Digital Design and Verification

Synchronization Types



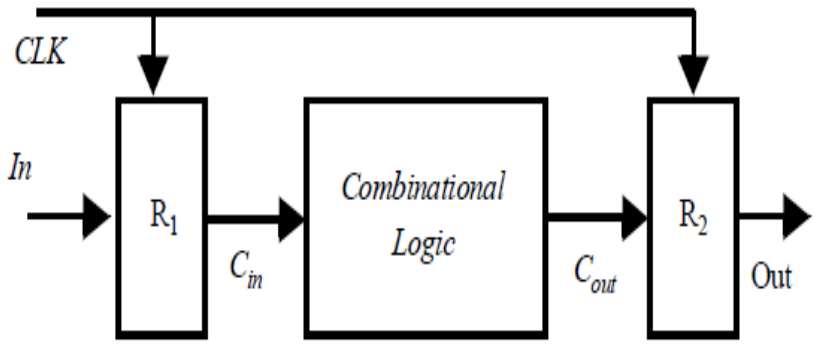
Synchronous: both clocks are same in all aspects like phase and freq.

Meso: same freq, constant phase difference

Plesio: slight difference in freq and variable phase difference

Async: completely unrelated clocks.

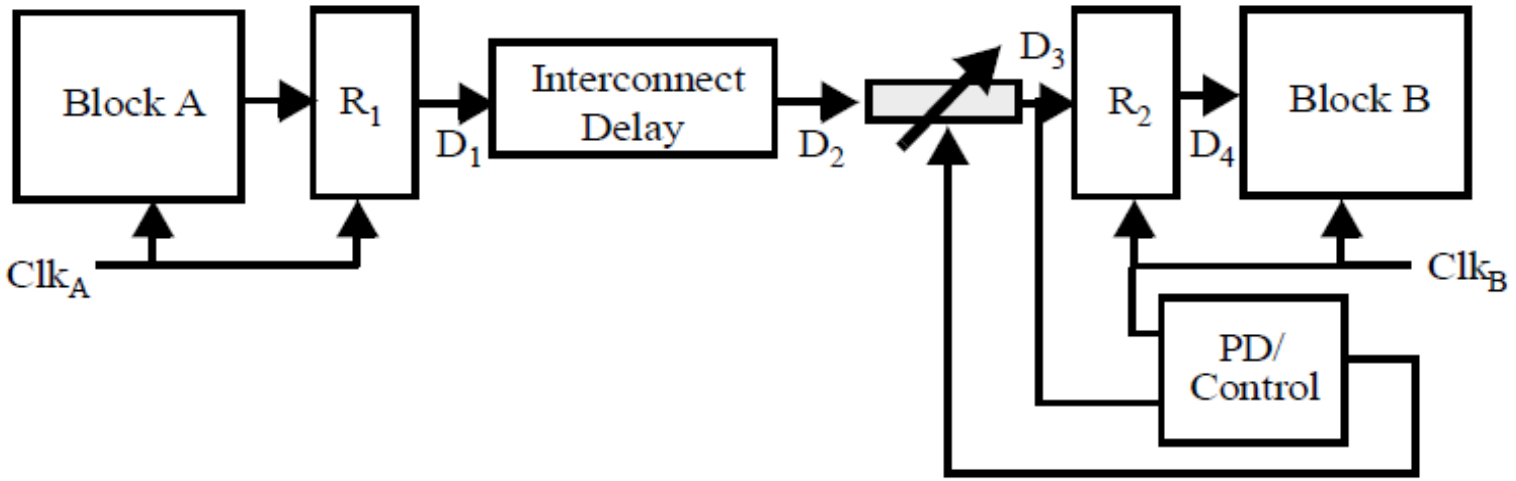
Synchronous Interconnect



- A synchronous signal is one that has the exact same frequency. In such a timing methodology, the signal is "synchronized" with the clock, and the data can be sampled directly without any uncertainty.
- The input data signal In is sampled with register **R1** to give signal **Cin**, which is synchronous with the system clock and then passed along to the combinational logic block. After a suitable setting period, the output **Cout** becomes valid and can be sampled by **R2** which synchronizes the output with the clock. In a sense, the "certainty period" of signal Cout, or the period where data is valid is synchronized with the system clock, which allows register R2 to sample the data with complete confidence. The length of the "uncertainty period," or the period where data is not valid, places an upper bound on how fast a synchronous interconnect system can be clocked.

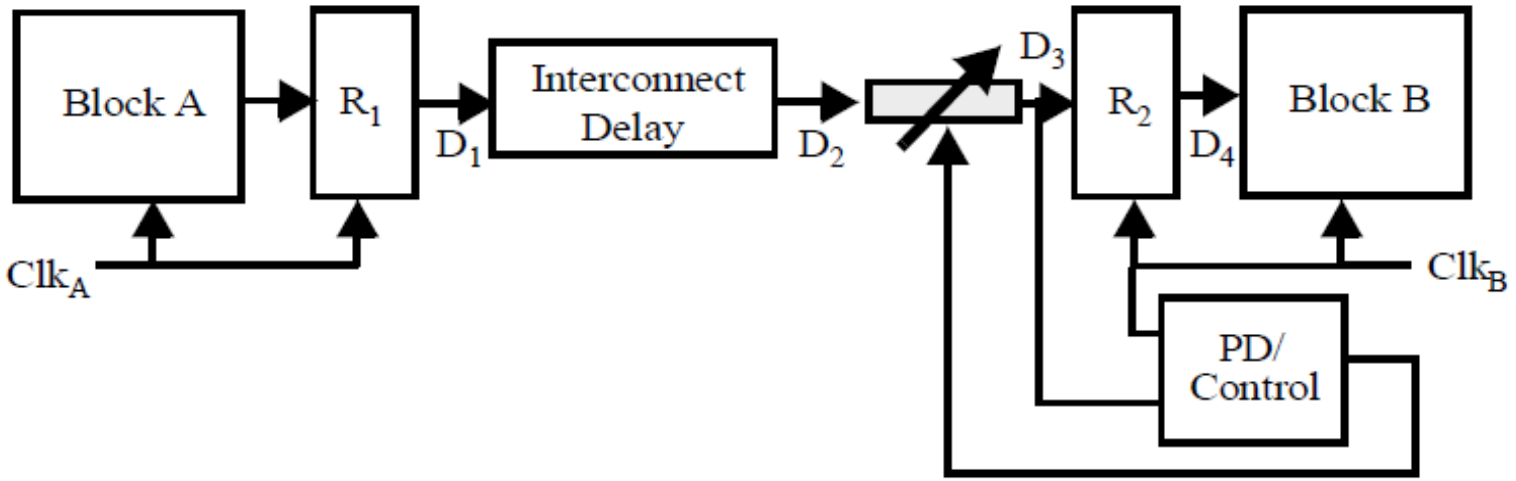
Mesochronous Interconnect

- A mesochronous signal is one that has the same frequency but an unknown phase offset with respect to the local clock ("meso" from Greek is middle).
- For example, if data is being passed between two different clock domains, then the data signal transmitted from the first module can have an unknown phase relationship to the clock of the receiving module.
- In such a system, it is not possible to directly sample the output at the receiving module because of the uncertainty in the phase offset.
- A (mesochronous) synchronizer can be used to synchronize the data signal with the receiving clock. The synchronizer serves to adjust the phase of the received signal to ensure proper sampling.



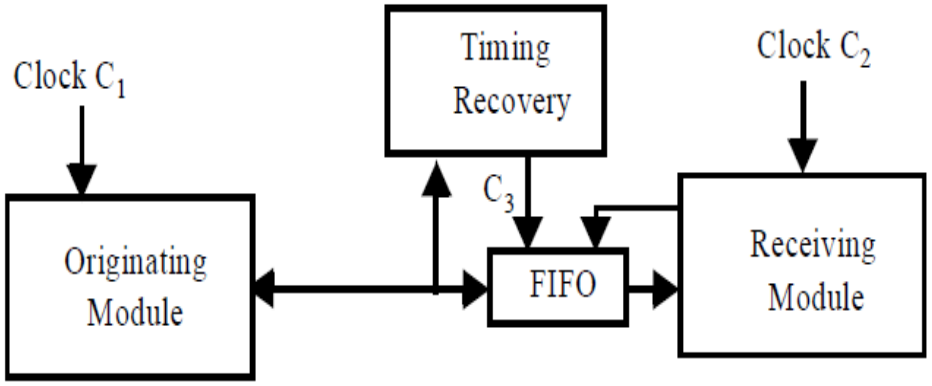
Mesochronous Interconnect

- In signal D1 is synchronous with respect to ClkA. However, D1 and D2 are mesochronous with ClkB because of the unknown phase difference between ClkA and ClkB and the unknown interconnect delay in the path between Block A and Block B.
- The role of the synchronizer is to adjust the variable delay line such that the data signal D3 (a delayed version of D2) is aligned properly with the system clock of block B.
- In this example, the variable delay element is adjusted by measuring the phase difference between the received signal and the local clock. After register R2 samples the incoming data during the certainty period, then signal D4 becomes synchronous with ClkB.



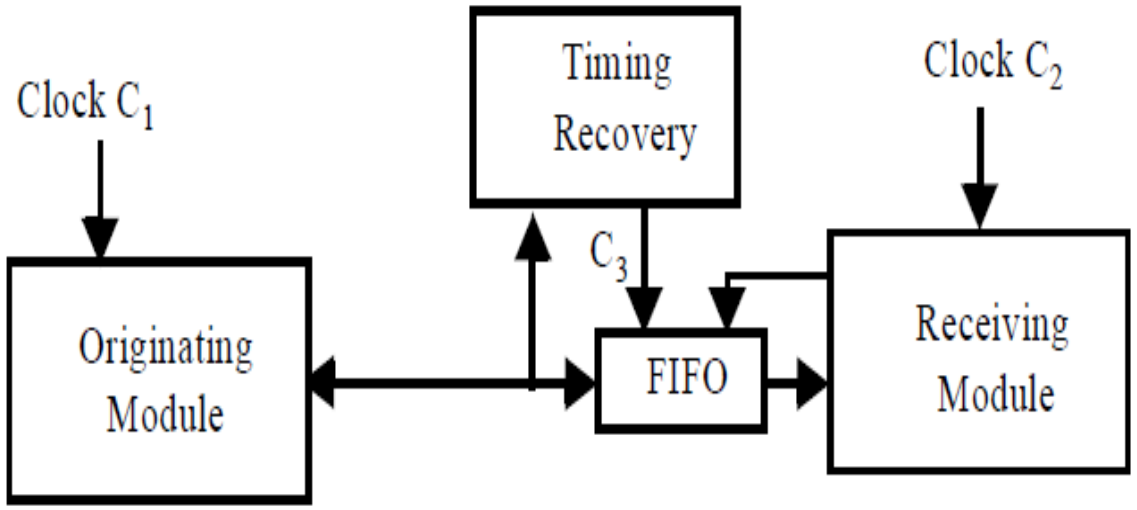
Plesiochronous Interconnect

- A plesiochronous signal is one that has nominally the same, but slightly different frequency as the local clock ("plesio" from Greek is near). In effect, the phase difference drifts in time.
- This scenario can easily arise when two interacting modules have independent clocks generated from separate crystal oscillators. Since the transmitted signal can arrive at the receiving module at a different rate than the local clock, one needs to utilize a buffering scheme to ensure all data is received.



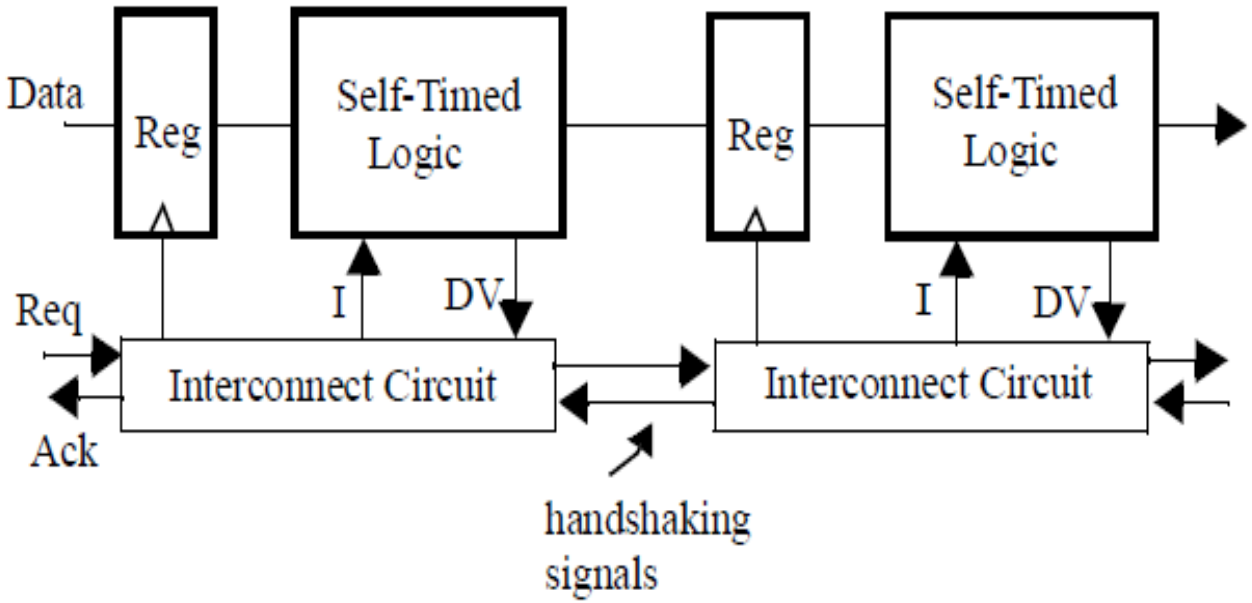
Plesiochronous Interconnect

- In this digital communications framework, the originating module issues data at some unknown rate characterized by C1, which is plesiochronous with respect to C2.
- The timing recovery unit is responsible for deriving clock C3 from the data sequence, and buffering the data in a FIFO. As a result, C3 will be synchronous with the data at the input of the FIFO and will be mesochronous with C1.
- Since the clock frequencies from the originating and receiving modules are mismatched, data might have to be dropped if the transmit frequency is faster, and data can be duplicated if the transmit frequency is slower than the receive frequency.
- However, by making the FIFO large enough, and periodically resetting the system whenever an overflow condition occurs, robust communication can be achieved.



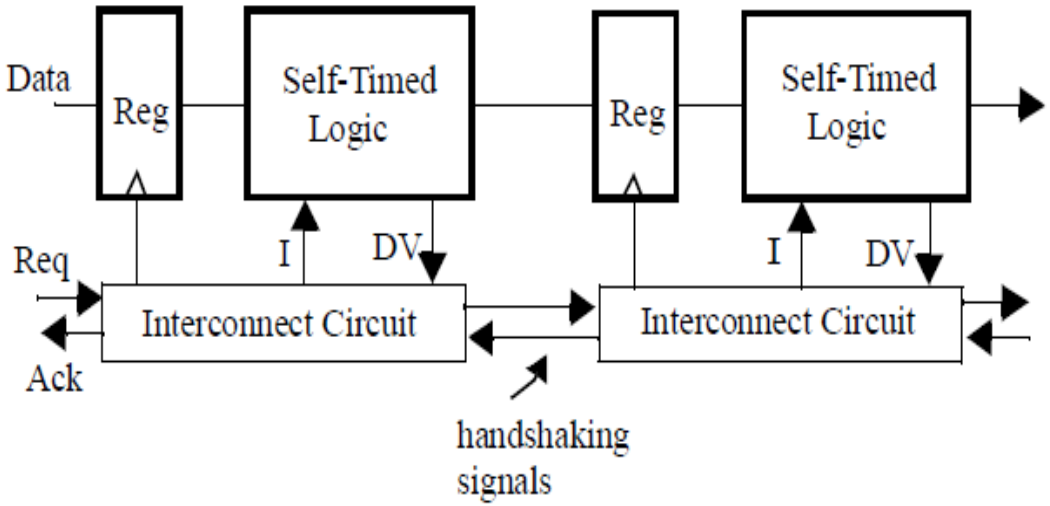
Asynchronous Interconnect

- Asynchronous signals can transition at any arbitrary time, and are not slaved to any local clock. As a result, it is not straightforward to map these arbitrary transitions into a synchronized data stream.
- Although it is possible to synchronize asynchronous signals by detecting events and introducing latencies into a data stream synchronized to a local clock, a more natural way to handle asynchronous signals is to simply eliminate the use of local clocks and utilize a self-timed asynchronous design approach.
- In such an approach, communication between modules is controlled through a handshaking protocol to perform the proper ordering of commands.



Asynchronous Interconnect

- When a logic block completes an operation, it will generate a completion signal DV to indicate that output data is valid. The handshaking signals then initiate a data transfer to the next block, which latches in the new data and begins a new computation by asserting the initialization signal **I**.
- Asynchronous designs are advantageous because computations are performed at the native speed of the logic, where block computations occur whenever data becomes available.
- There is no need to manage clock skew, and the design methodology leads to a very modular approach where interaction between blocks simply occur through a handshaking procedure.
- However, these handshaking protocols result in increased complexity and overhead in communication that can reduce performance.



Uses of Synchronization

- There are three common uses of synchronization: -
  - Arbitering between asynchronous requests,
  - Sampling asynchronous signals with a clock, and
  - Transmitting synchronous signals between two clock domains.

References

- William J Dally, John W Poulton, “Digital Systems Engineering”, Cambridge University Press.
- Digital Integrated Circuits, by John Rabaey. (This is also useful for Digital VLSI subject also).

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Advanced Digital Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Synchronizer Design

Unit4: Lecture-1

Sudeendra kumar K
Department of Electronics and Communication Engineering

Advanced Digital Design

Contents

- Delay line Synchronizer
- Two Register Synchronizer
- Two FF Synchronizer

Mesochronous Synchronizers

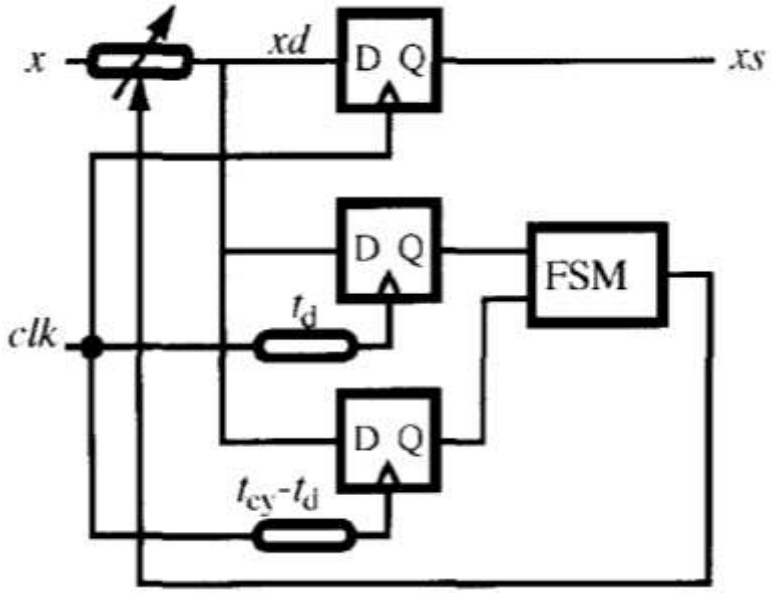
- A mesochronous signal, strictly speaking, is synchronized with the local clock. Its events occur at the local clock frequency with an arbitrary fixed phase shift. All that is required of a mesochronous synchronizer is to adjust the phase shift, if needed, to keep the signal transitions away from the unsafe regions of the local clock.

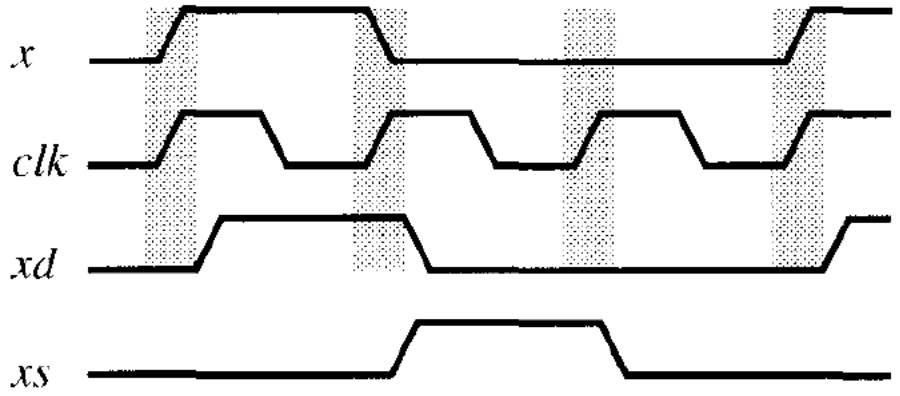
  - Delay Line Synchronizers
  - Two Register Synchronizers (with phase comparator)
  - FIFO Based Synchronizers
  - Brute Force Synchronizers (without phase comparator in double flopping mode)
  - **Application of Two register sync/Brute force**: - (Double flopping can also be used in CDC between slower transmission clock to higher receiving clock)

Advanced Digital Design

Delay Line Synchronizers

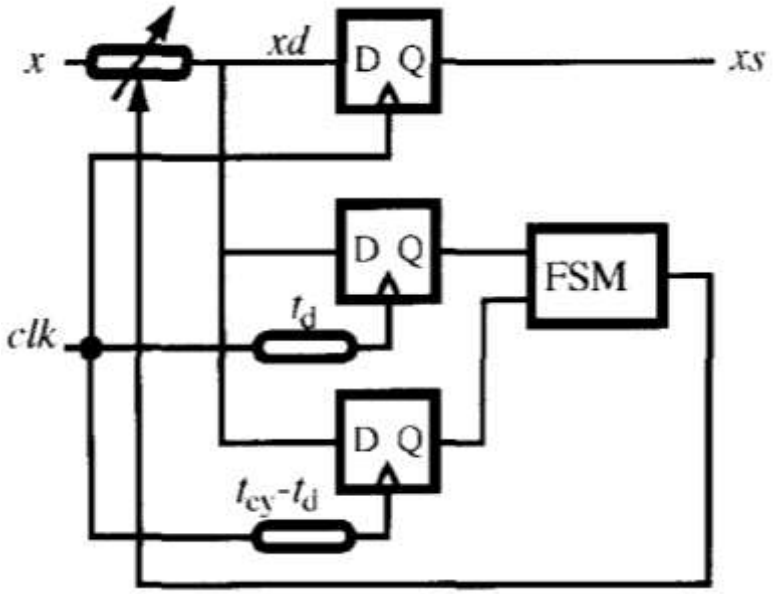
- The synchronizer consists of a variable delay line with an adjustment range of at least a clock period and a flip-flop. The shaded areas denote the clock keep-out regions.
- Signal **x** has transitions during the keep-out regions, and thus it cannot be sampled directly.
- The delay line is adjusted so that signal **xd** has its transitions after the keep-out region and is safe to sample. The flip-flop samples **xd** and generates the synchronized signal **xs.**

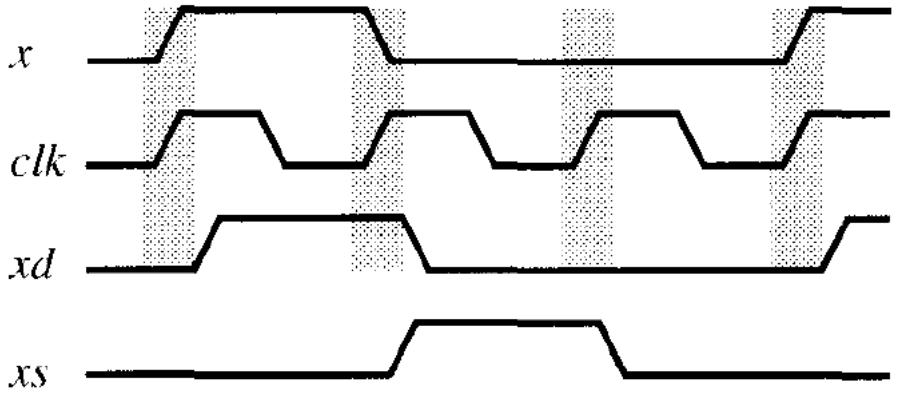




Advanced Digital Design
Delay Line Synchronizer

- The variable delay line is adjusted by measuring the relative phase of signal **xd.** The circuit in the figure performs this measurement using two flip-flops clocked with delayed versions of the clock that sample the signal at either end of the keep-out region, in this case with width **2 * td** centered on the clock.
- When signal **xd** makes its transitions in a safe region, these two flip-flops will sample the change during the same cycle. If a transition is made during the keep-out region, the lower flip-flop will sample the change of a cycle before the upper flip-flop.
- The FSM monitors the outputs of these flip-flops and sets the variable delay to the minimum amount that keeps the transitions of **xd** in the safe region. Because these two flip-flops may enter metastable states during normal operation, the FSM must resample their outputs after waiting sufficient time for any metastable states to decay with high probability.





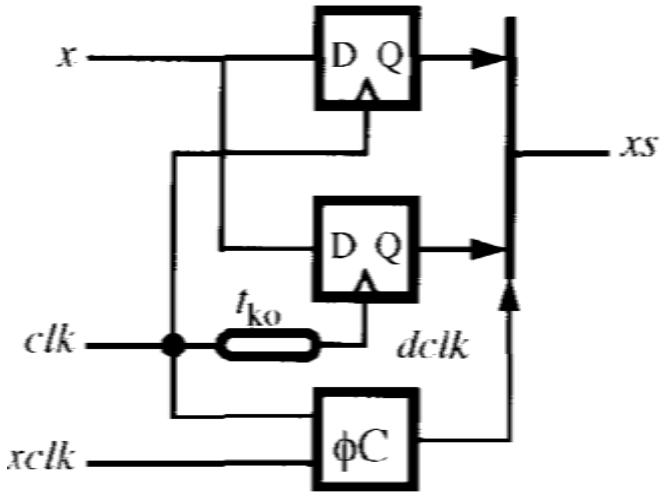
Advanced Digital Design
Delay Line Synchronizer

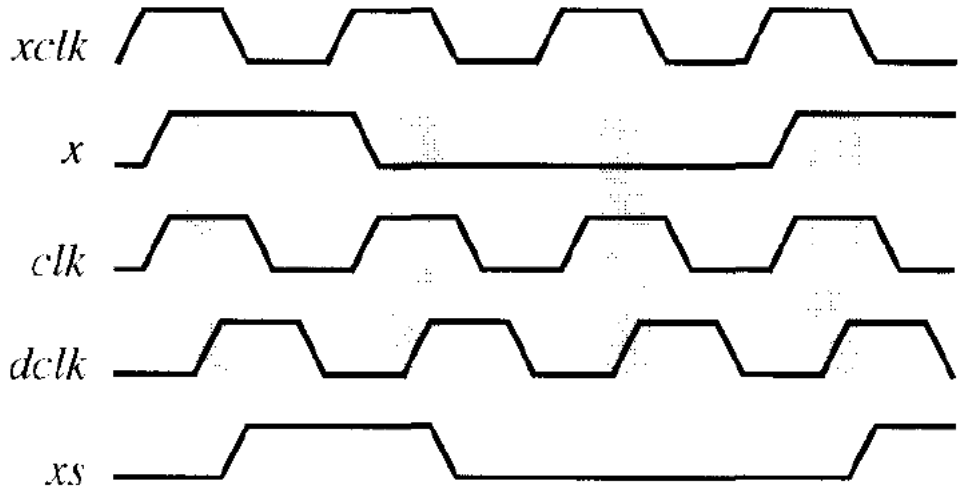
- Alternatively the synchronizer itself can be used to measure the phase of the input signal during an initial or periodic training sequence.
- With this approach the delay of the variable line is swept while sampling a known signal to detect the position of the signal transition. Using this phase information, the variable line is then set to the minimum value that keeps transitions on **xd** out of the keep-out region.
- The delay-line synchronizer is an expensive way to synchronize a wide, multibit signal, for it requires a variable delay line for each data bit.
- Also, it requires that the signal be delayed before sampling and thus cannot be used with a clocked receive amplifier.
- For this reasons, other methods are generally preferred for synchronizing wide signals.

Advanced Digital Design

Two register Synchronizer

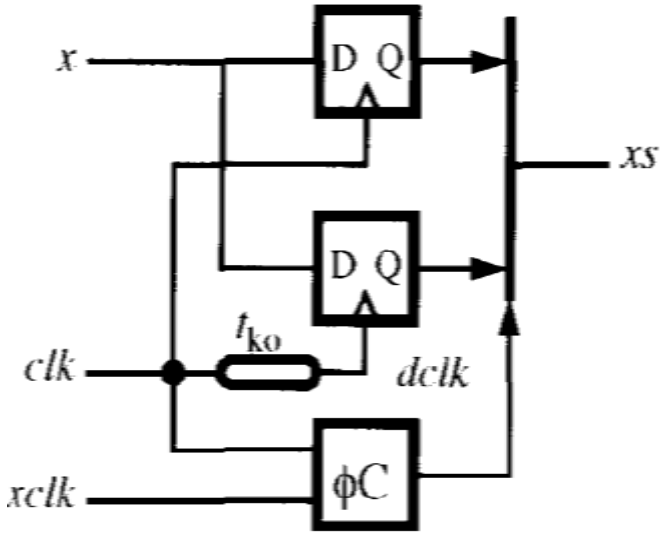
- For wide data paths the two-register synchronizer avoids the cost of a delay line per data bit by delaying the clock rather than the data.
- This synchronizer consists of a pair of flip-flops (registers for a multibit datapath).
- The upper flip-flop samples the input on positive edges of the local clock, clk.
- A delayed clock, dclk, that lags clk by the width of the keep-out region samples the input into the lower flip-flop. As long as the keep-out region of the sampling flip-flop is less than half a clock period, the keep-out regions of the two flip-flops do not overlap, and the input signal will be safely sampled by one of the two flip-flops. The output of the safe flip-flop is selected by a multiplexer and output as synchronized signal, **xs.**

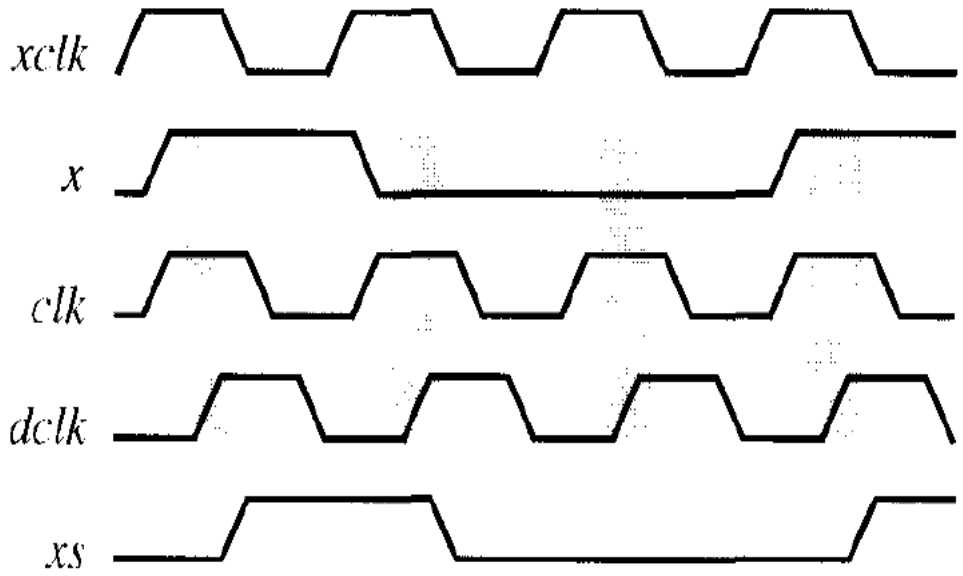




Advanced Digital Design
Two register Synchronizer

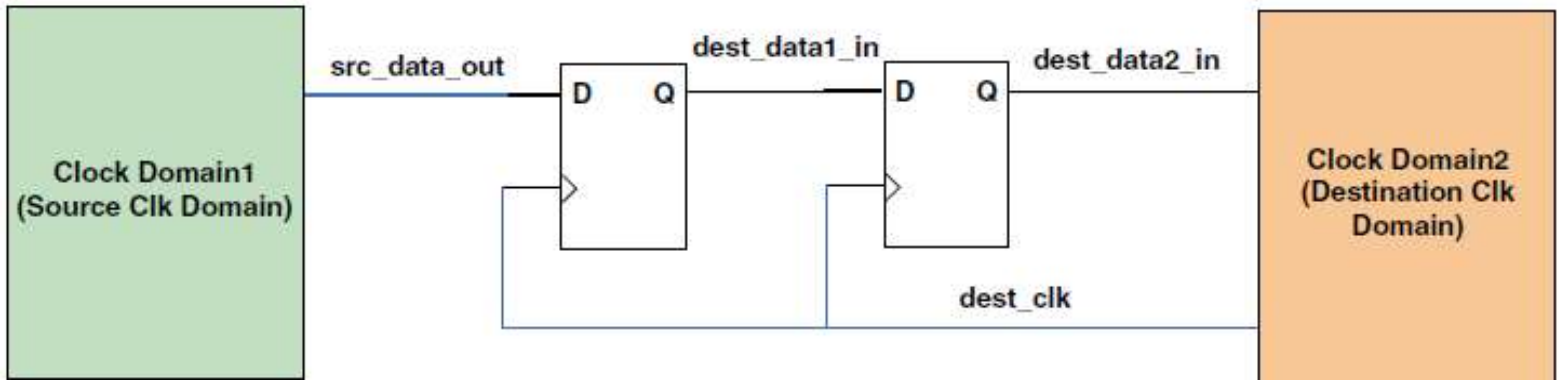
- The multiplexer control is a constant because the phase of x does not change with respect to the local clock. During system configuration, a phase comparator measures the relative phase between **xclk**, the clock from which **x** is derived, and the local clock, **clk**.
- The phase comparator selects the upper flip-flop unless transitions of x occur during the keep-out region of elk, in which case it selects the lower flip-flop.
- The waveforms at the right of Figure illustrate this latter case. In the figure, **xclk** slightly leads **clk,** causing transitions of x to occur in the center of the keep-out region of **clk.** The phase comparator detects this condition during startup and selects the lower flip-flop to produce output **xs.**

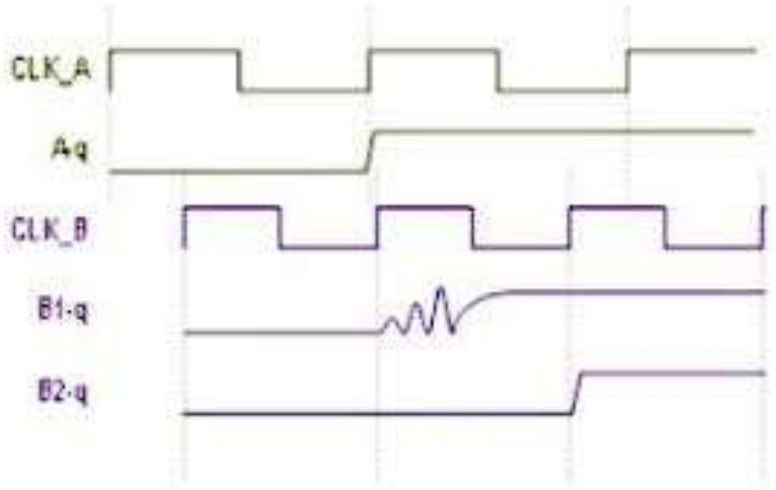


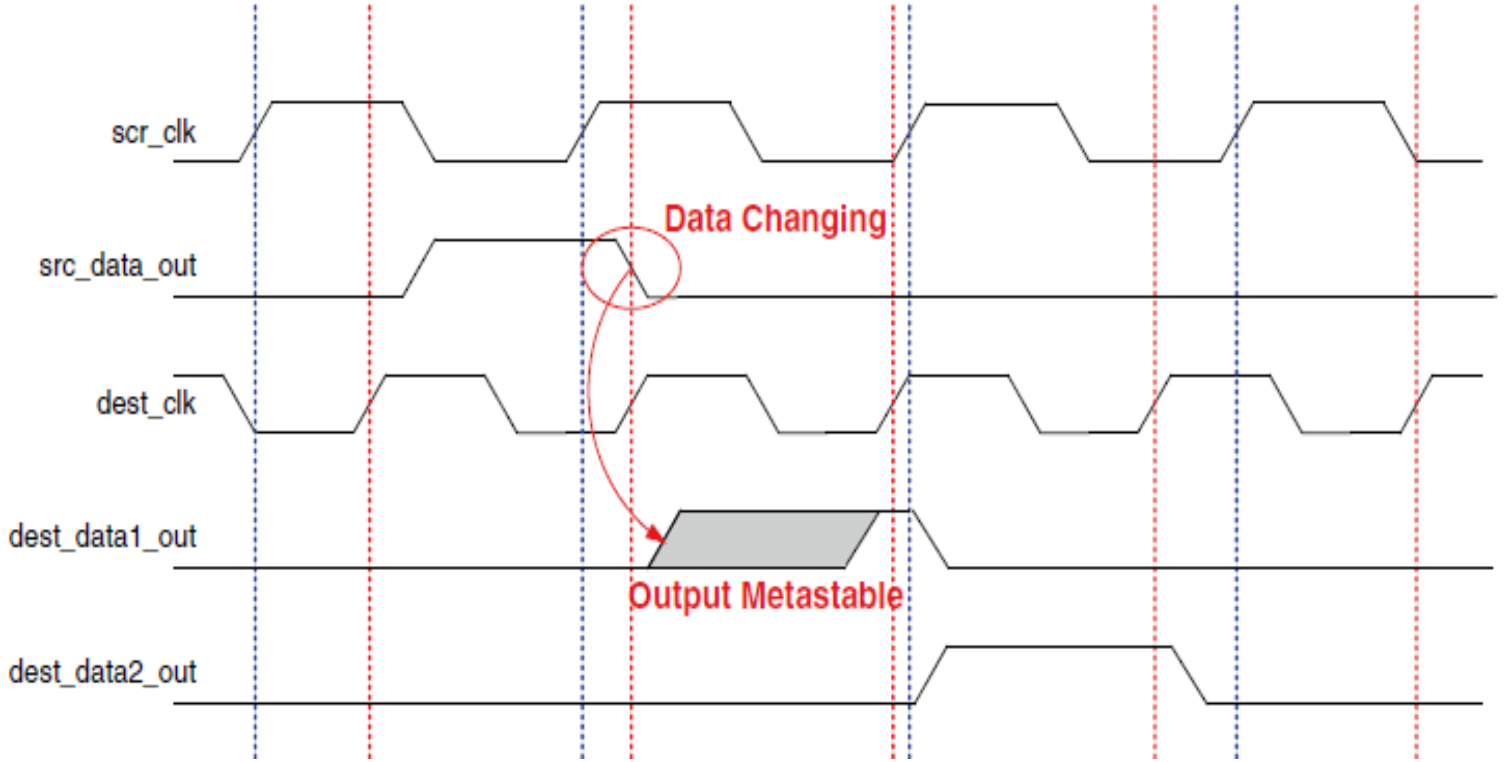


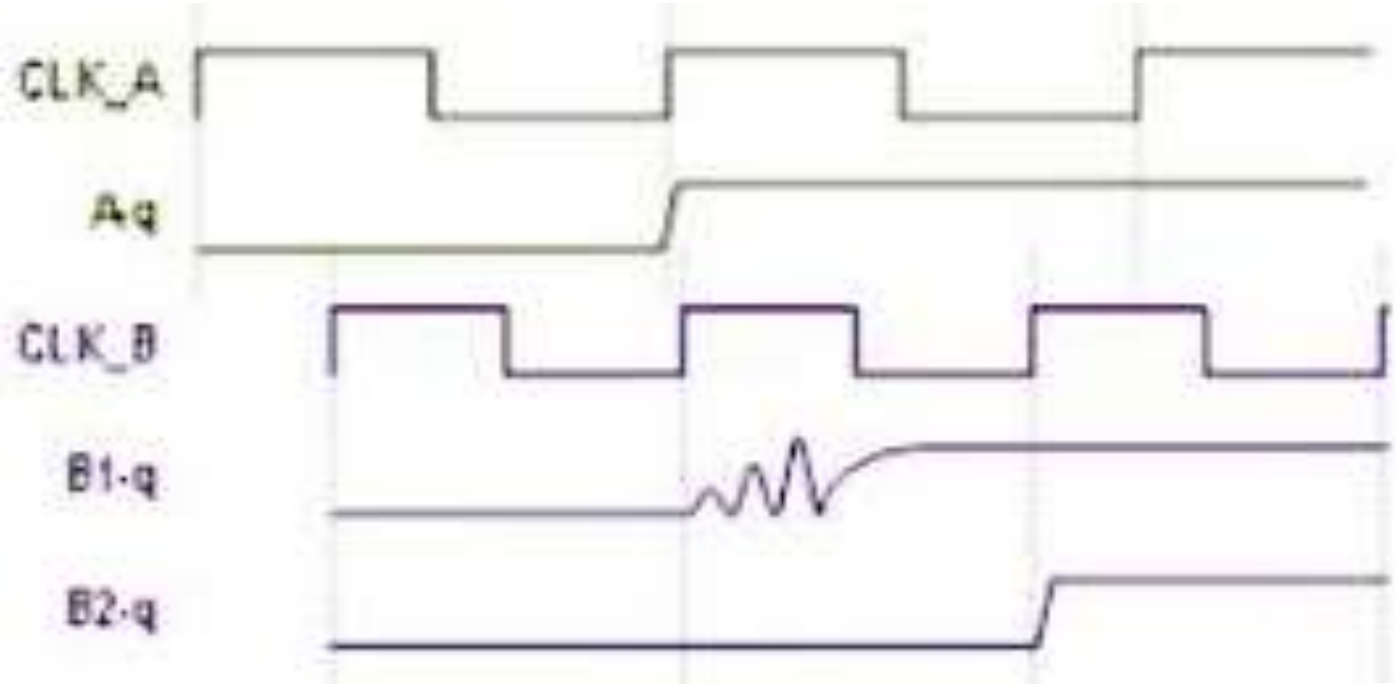
Advanced Digital Design

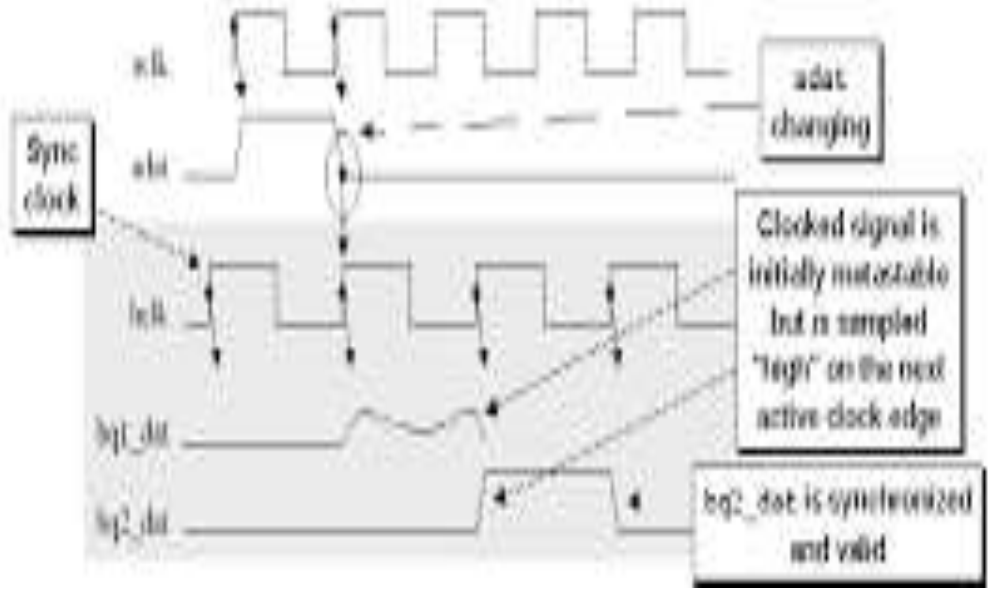
Two Register Synchronization

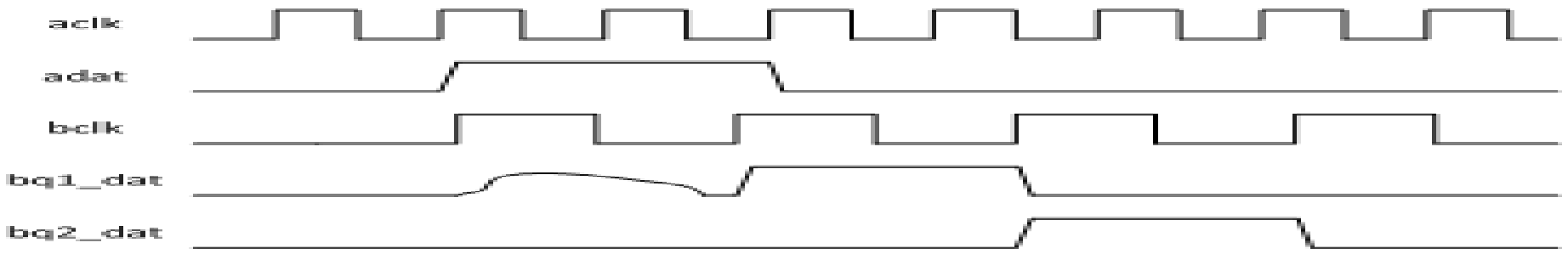






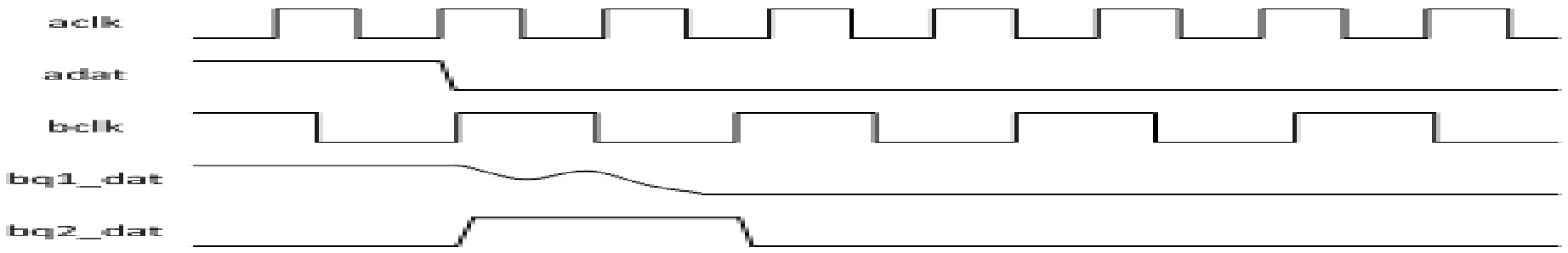

CLK_A
A-q
CLK_B
B1-q
B2-q


Sync clock
Clocked signal is initially metastable but is sampled "high" on the next active clock edge
bq2_dat is synchronized and valid

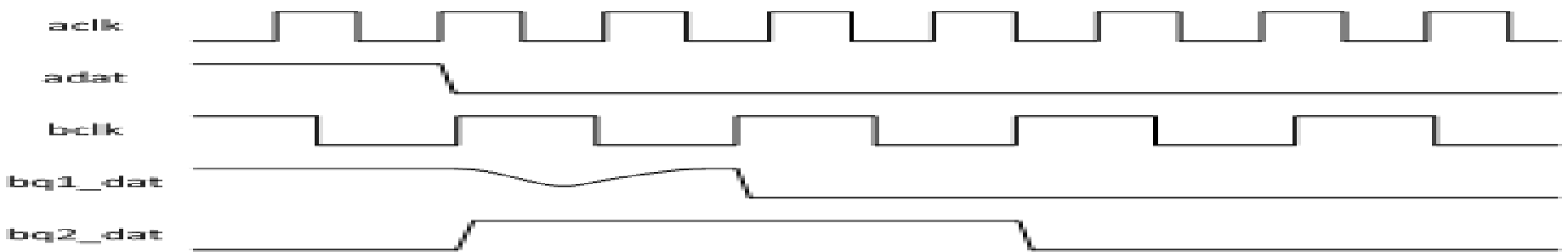


(a) bq1_dat goes into metastable state when adat is being asserted and settles down at '0'.

(b) bq1dat goes into metastable state when adat is being asserted and settles down at '1'.

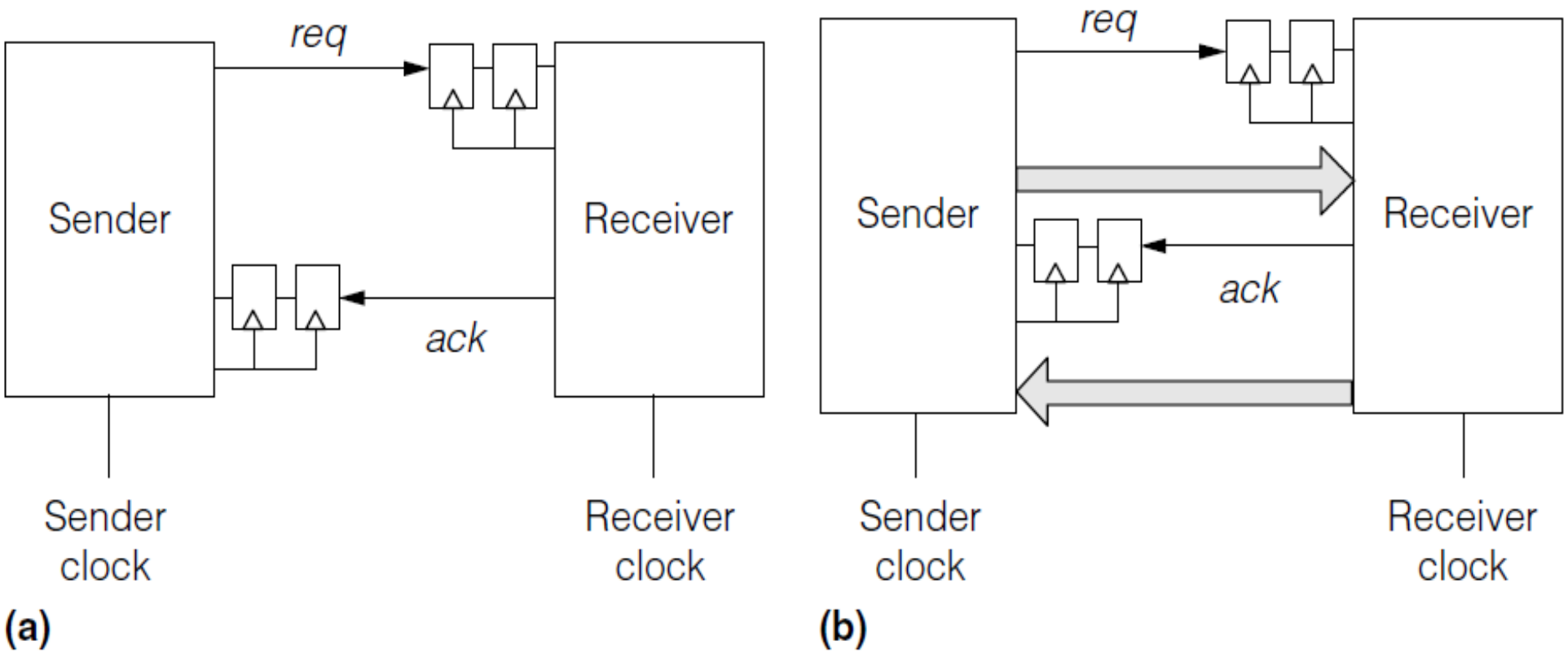


(c) bq1dat goes into metastable state when adat is being deasserted and settles down at '0'.



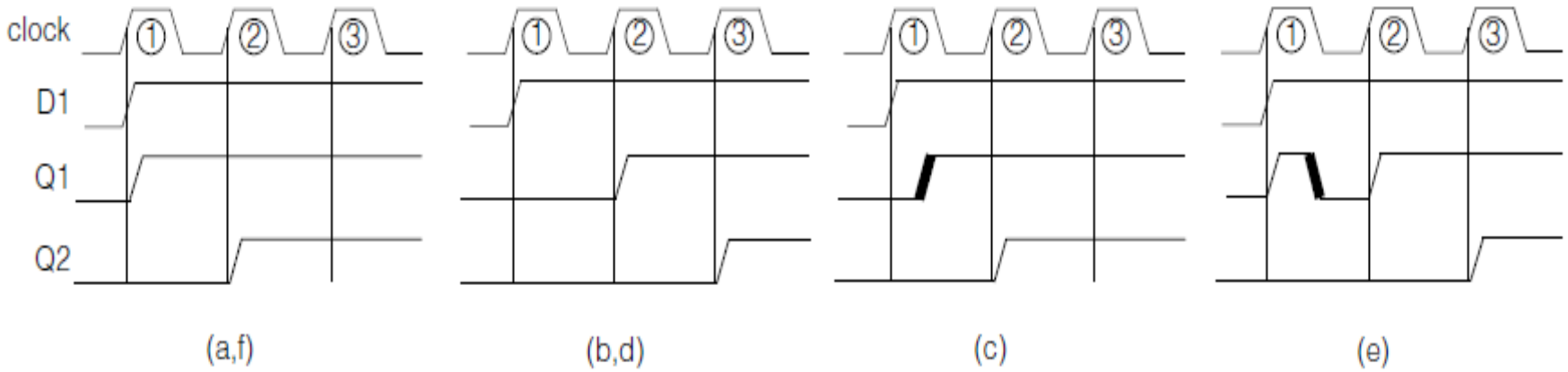
Advanced Digital Design

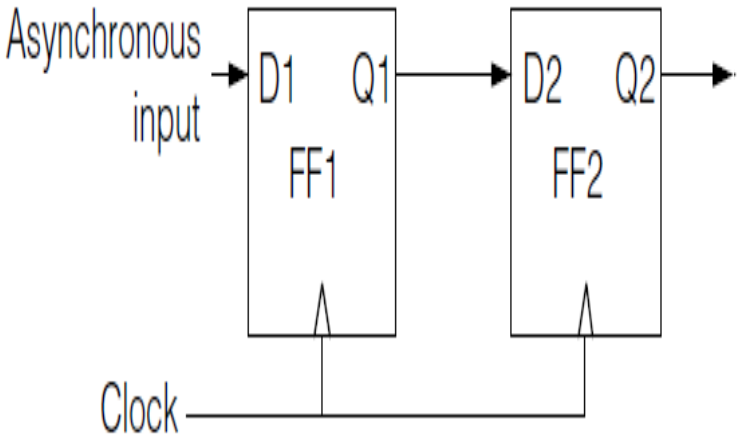
Two Flop Synchronizer



- Complete two-way control synchronizer
- Complete two-way data synchronizer

Advanced Digital Design
Two Flop Synchronizer





(a) Q1 could switch at the beginning of clock cycle 1 and Q2 will copy that on clock cycle 2.

(b) Q1 could completely miss D1. It will surely rise on cycle 2, and Q2 will rise one cycle later.

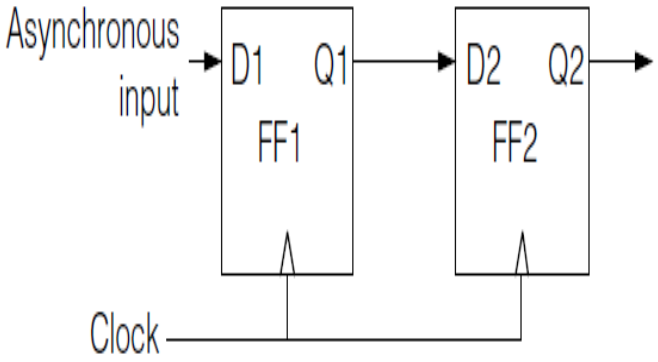
(c) FF1 could become metastable, but its output stays low. It later resolves so that Q1 rises (the bold rising edge). This will happen before the end of the cycle (except, maybe, once every MTBF years). Then Q2 rises in cycle 2.

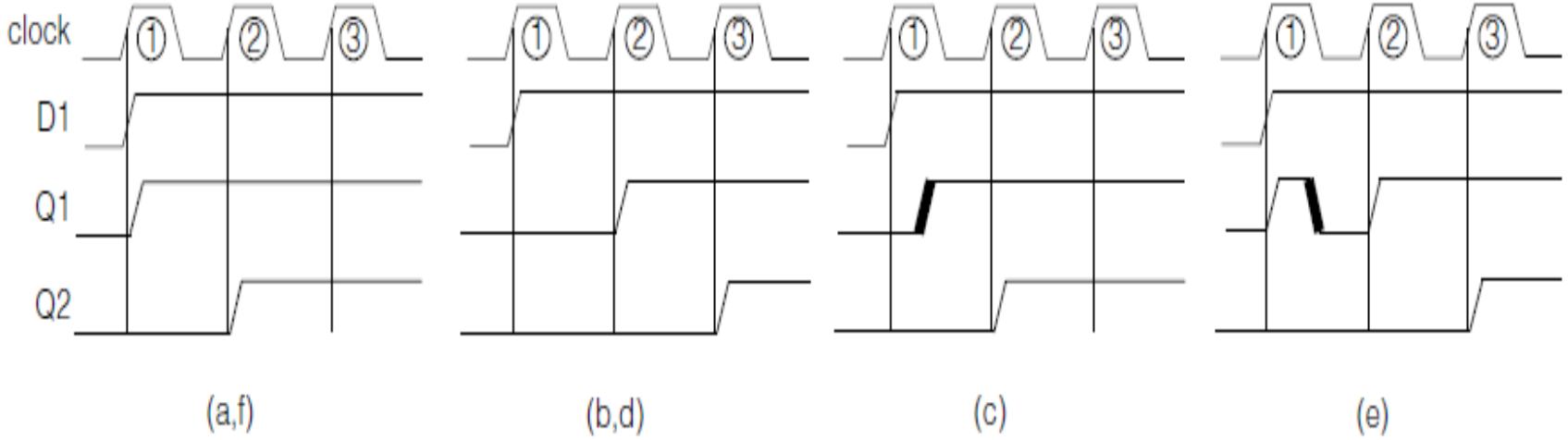
(d) (d) FF1 could become metastable, its output stays low, and when it resolves, the output still stays low. This appears the same as case (b). Q1 is forced to rise in cycle 2, and Q2 rises in cycle 3.

Two Flop Synchronizer

(e) FF1 goes metastable, and its output goes high. Later, it resolves to low (we see a glitch on Q1). By the end of cycle 1, Q1 is low. It rises in cycle 2, and Q2 rises in cycle 3.

(f) FF1 goes metastable, its output goes high, and it later resolves to high. Q1 appears the same as case (a). Q2 rises in cycle 2.





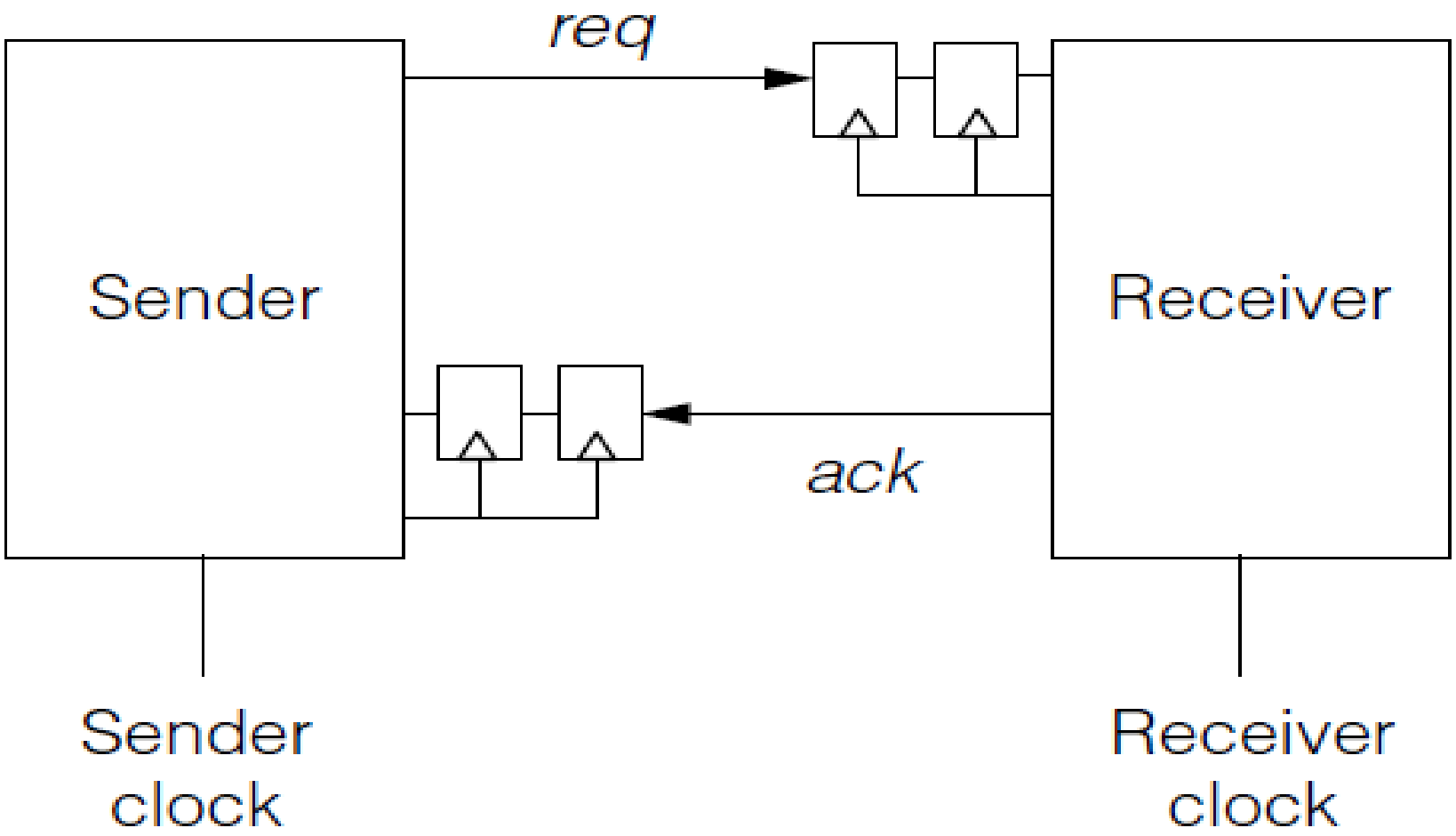
Advanced Digital Design

Two Flop Synchronizer

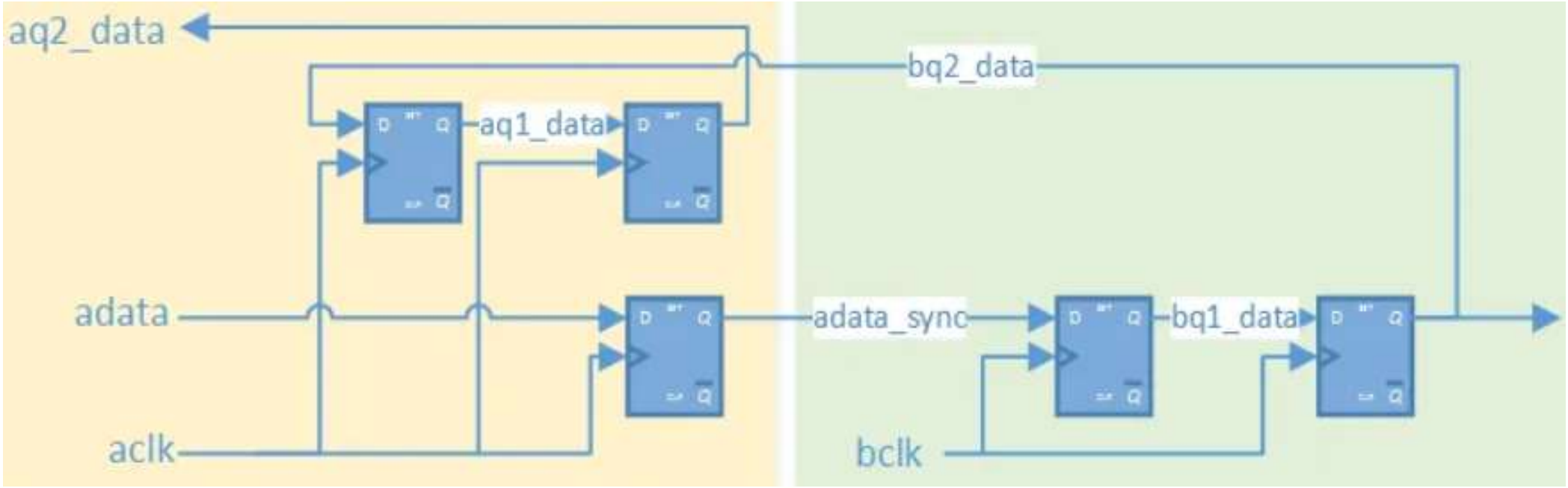
- The synchronization circuit exchanges the ‘‘analog’’ uncertainty of metastability (continuous voltage levels changing over continuous time) for a simpler ‘‘digital’’ uncertainty (discrete voltage levels switching only at uncertain discrete time points) of whether the output switches one or two cycles later. Other than this uncertainty, the output signal is a solid, legal digital signal.
- What does happen when it really fails? Well, once every MTBF years, FF1 becomes metastable and resolves exactly one clock cycle later. Q1 might then switch exactly when FF2 samples it, possibly making FF2 metastable. Is this as unrealistic as it sounds?
- No. Run your clocks sufficiently fast, and watch for meltdown!
- A word of caution: the two flip-flops should be placed near each other, or else the wire delay between them would detract from the resolution time. Missing this seemingly minor detail has made quite a few synchronizers fail unexpectedly.

Two Flop Synchronizer

- To assure correct operation, D1 stays high for at least two cycles (in cases b, d, e) so that FF1 is guaranteed to sample logic1 at its input on the rising clock of cycle 2.
- How would the sender know how long D1 must be kept high? We have no idea how fast the sender clock is ticking, so we can't simply count cycles.
- To solve that, the receiver must send back an acknowledgment signal.
- The sender sends **req** (also known as **request, strobe, ready, or valid**), **req** is synchronized by the top synchronization circuits, the receiver sends **ack (or acknowledgment, or stall), ack** is synchronized by the sender, and only then is the sender allowed to change req again. This roundtrip handshake is the key to correct synchronization.


req
Sender
Receiver
ack
Sender
clock
Receiver
clock

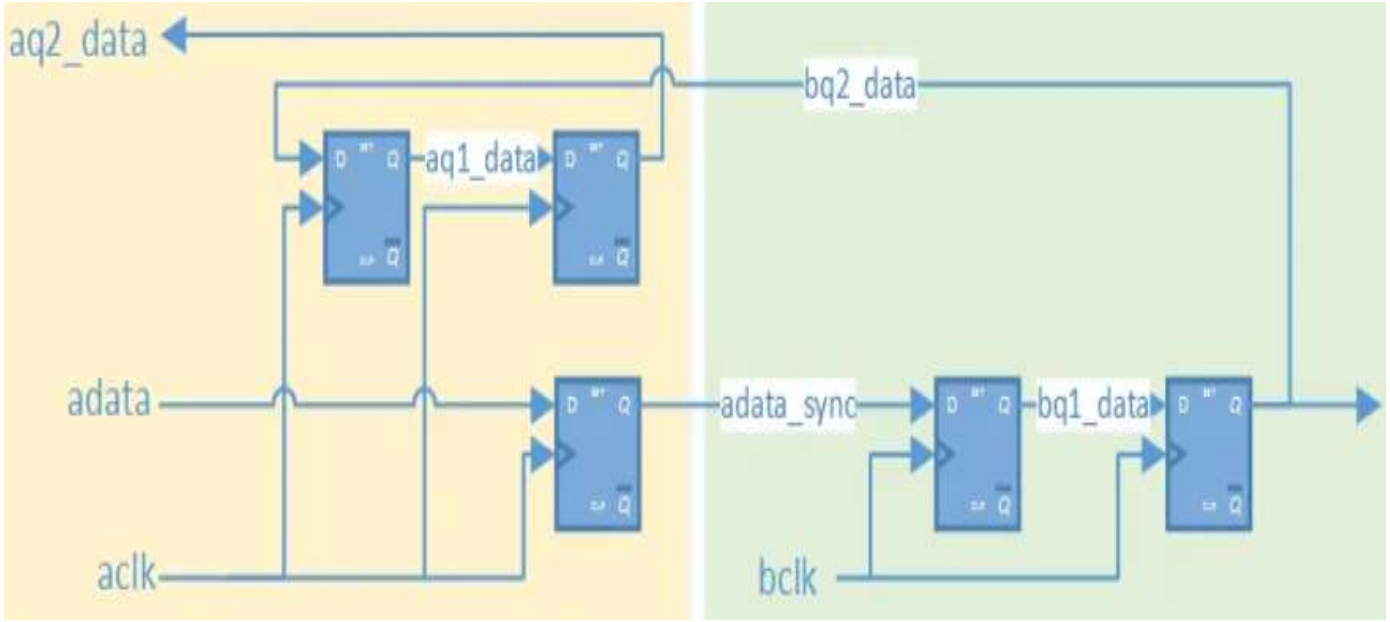
Single bit — synchronizer with feedback acknowledge



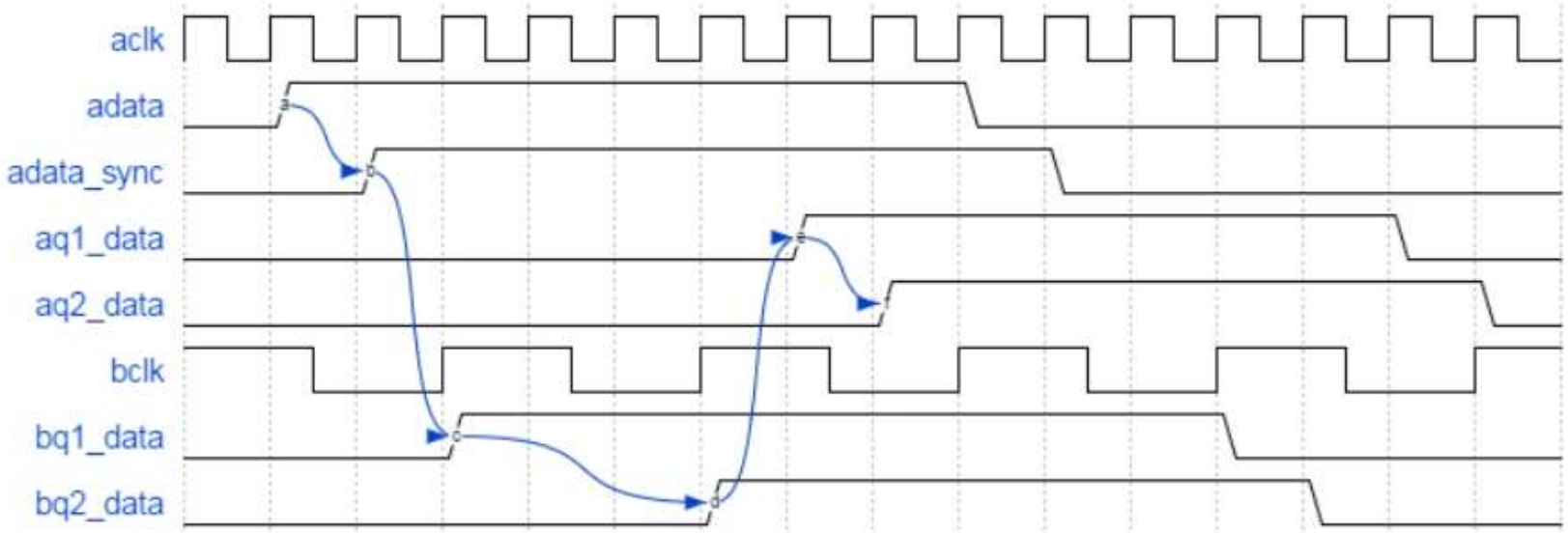
- The source domain sends the signal to the destination clock domain through a two flip-flop synchronizer, then passes the synchronized signal back to the source clock domain through another two flip-flop synchronizer as a feedback acknowledgement.

Advanced Digital Design

Single bit — synchronizer with feedback acknowledge



- This solution is very safe, but it does come at a cost of increased delay due to synchronizing in both directions before allowing the signal to change again. This solution would work in my memory controller design to handle the varying clock frequency relationship.



References

- William J Dally, John W Poulton, “Digital Systems Engineering”, Cambridge University Press.

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Advanced Digital Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Multi-Cycle Path Synchronizer

Sudeendra kumar K
Department of Electronics and Communication Engineering

Advanced Digital Design

Contents

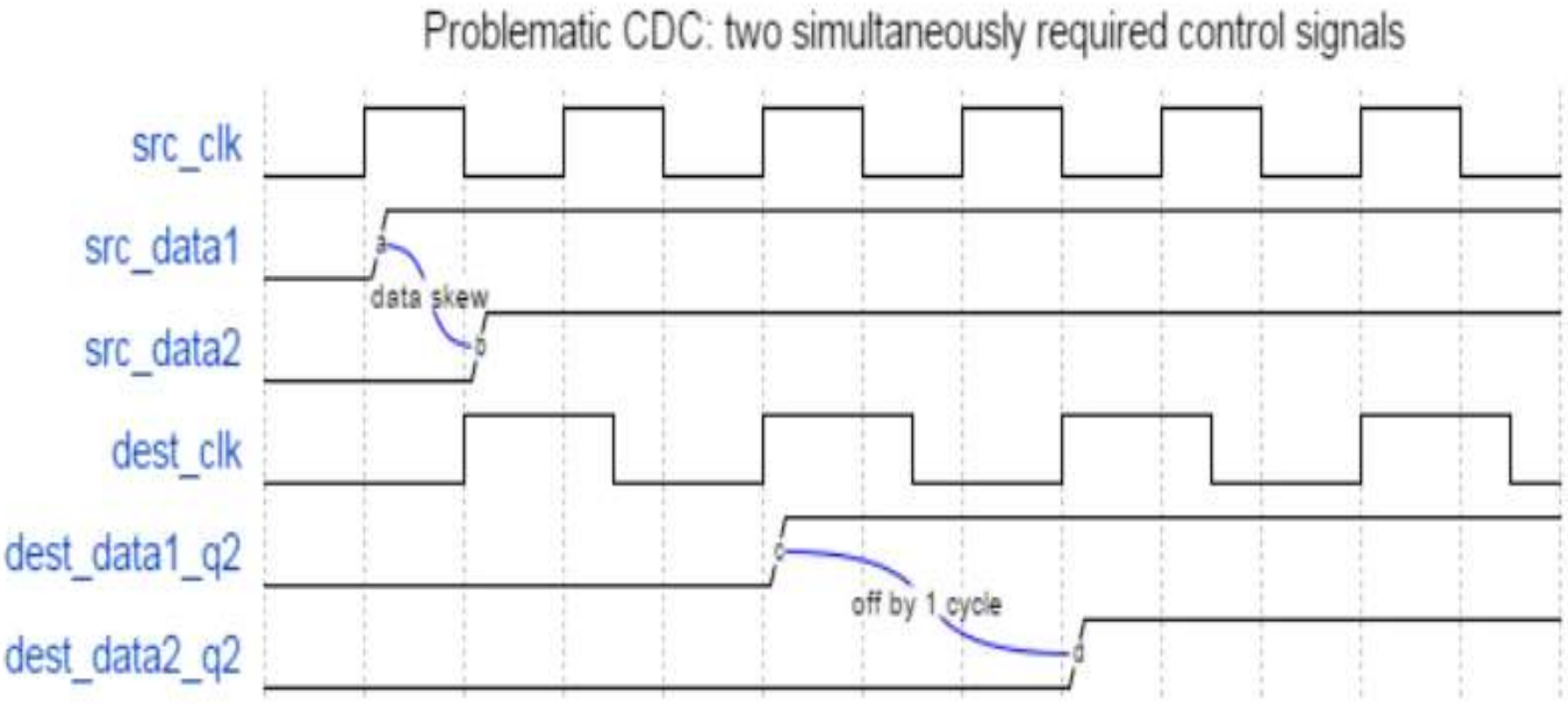
- Multi-Cycle Path Synchronizer

Problems With Passing Multiple Control Signals Across CDC

- The fundamental problem with passing multiple bits is if they are synchronized individually, they cannot be guaranteed to arrive in the destination clock domain on the same clock edge.
- When the individual bits are launched from the source clock domain, they may be skewed relative to each other due to trace length, process variation, etc.
- Since in an asynchronous clock domain crossing (CDC) the destination clock can have every possible alignment relative to the source clock (and relative to the skewed data bits), the destination clock can (and will) sample at a time when not all the bits are at their stable final values.

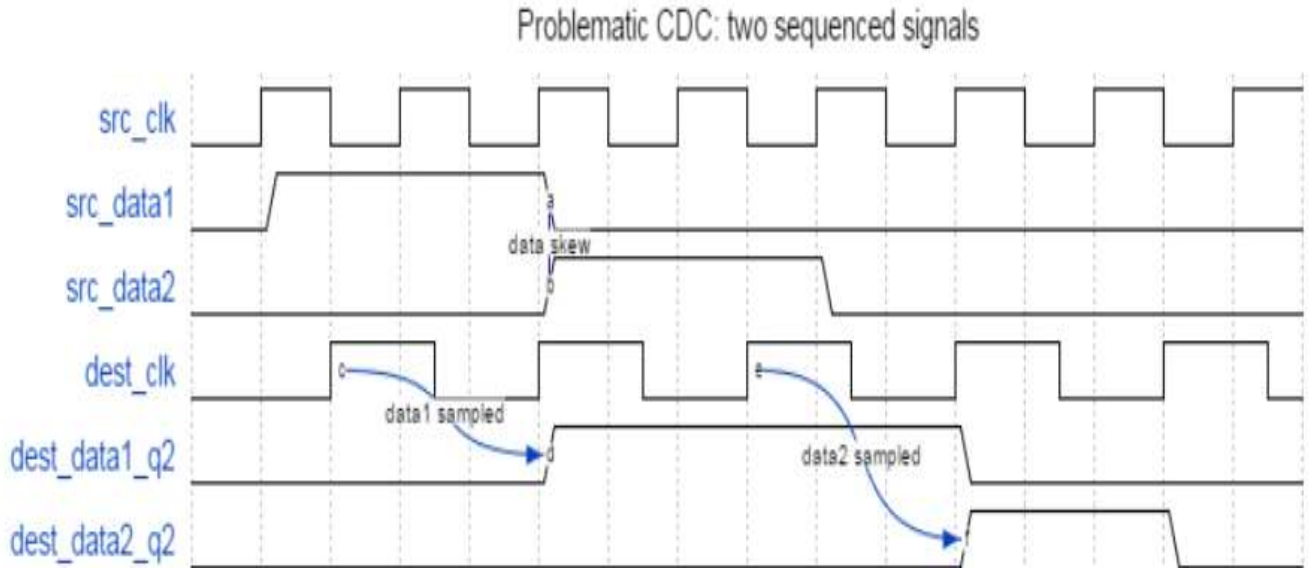
Advanced Digital Design
Two simultaneously required signals

- The waveform below shows how data skew from the source clock domain can cause two signals to arrive in different clock cycles in the destination domain, if they are synchronized individually using two flip-flop synchronizers. Don't do this!



Advanced Digital Design
Two sequenced signals

- Individually synchronizing two signals that require precise sequencing across an asynchronous clock domain crossing (CDC) is a recipe for disaster.
- In fact a recent ASIC project at work had a problem like this that resulted in a chip that only booted 50% of the time, months of debug, and finally a re-spin (we make mistakes too).
- The waveform below shows how two separate signals that are intended to arrive 1 cycle apart in the destination domain, can actually arrive either 1 or 2 cycles apart depending on data skew.
- It's difficult to even analyze the frequency difference from the source to destination clock domain and come up with a potential sequence that may work… Just don't do this. There are better ways!



Advanced Digital Design

Solutions For Passing Multiple Signals Across CDC

- Multi-bit signal consolidation
- Multi-cycle path (MCP) formulation without feedback
- Multi-cycle path (MCP) formulation with feedback acknowledge
- Dual-Clock Asynchronous FIFO.

Solutions For Passing Multiple Signals Across CDC

- Multi-cycle path (MCP) formulation is a particularly interesting and widely applicable solution. It refers to sending unsynchronized data from the source clock domain to the destination clock domain, paired with a synchronized control (e.g. a load enable) signal.
- The data and control signals are sent simultaneously from the source clock domain. The data signals do not go through any synchronization, but go straight into a multi-bit flip-flop in the destination clock domain.
- The control signal is synchronized through a two-flip-flop synchronizer, then used to load the unsynchronized data into the flip-flops in the destination clock domain.
- This allows the data signals to settle (while the control signal is being synchronized), and captured together on a single clock edge.

Advanced Digital Design

Multi-bit Signal Consolidation

- Consolidating multiple bits across clock domain crossing (CDC) into one is more of a best practice, than a technique. It's always good to reduce as much as possible the number of signals that need to cross a clock domain crossing (CDC).
- However, this can be applied directly to the problem of sequencing two signals into the destination clock domain.
- A single signal can be synchronized across the clock domain crossing (CDC), and the two sequenced signals can be recreated in the destination clock domain once the synchronizing signal is received.

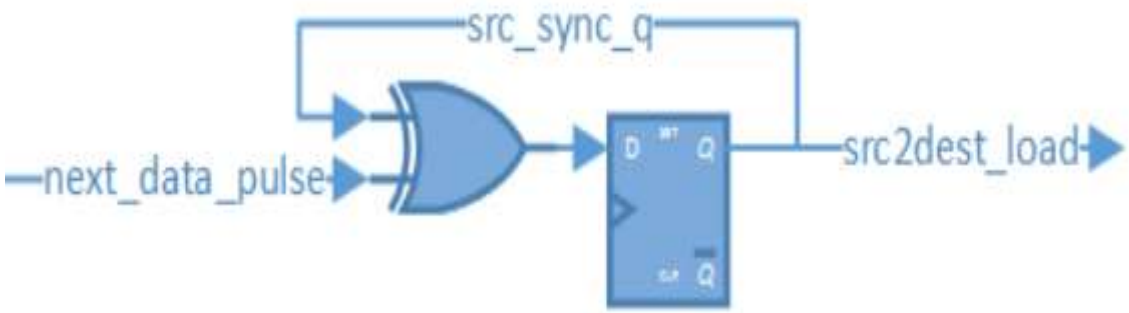
Advanced Digital Design

Multi-cycle path (MCP) formulation without feedback

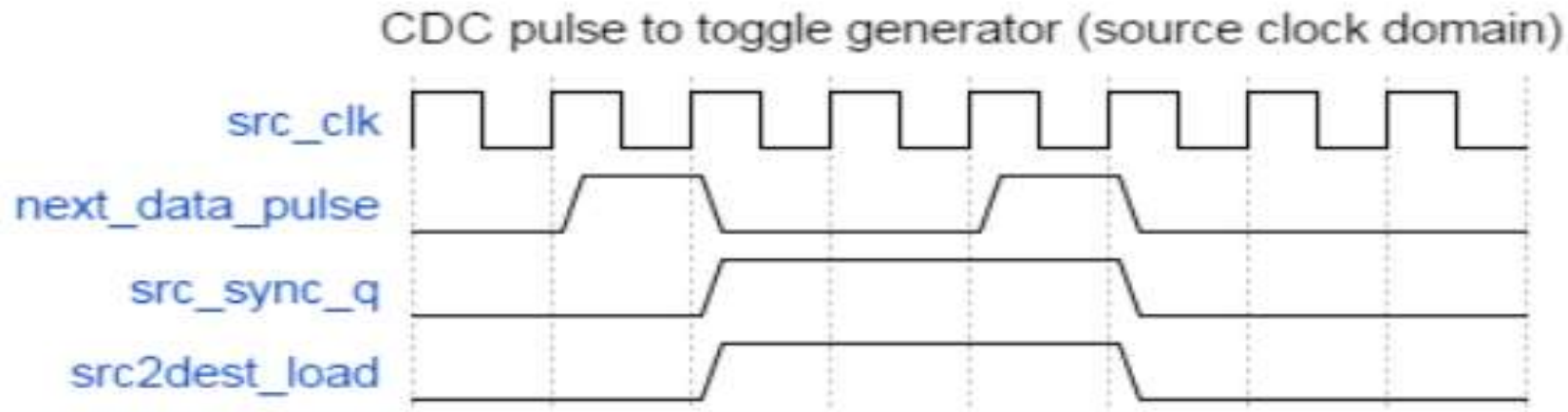
- The multi-cycle path (MCP) synchronizer is comprised of several components:
  - Logic that converts a synchronization event from source clock domain to a toggle to pass across the clock domain crossing (CDC)
  - Logic that converts the toggle into a load pulse in the destination domain
  - Flip-flops to capture the unsynchronized data bits
- One key idea in this design is that the synchronization event (a pulse) is converted into a single toggle (either low to high, or high to low) before being synchronized into the destination clock domain.
- Each toggle represents one event. You need to be careful when resetting the synchronizer such that no unintended events are generated (i.e. if the source domain is reset on its own, and the toggle signal goes from high to low due to reset).

Advanced Digital Design
Multi-cycle path (MCP) formulation without feedback

- Source clock domain event to toggle generator



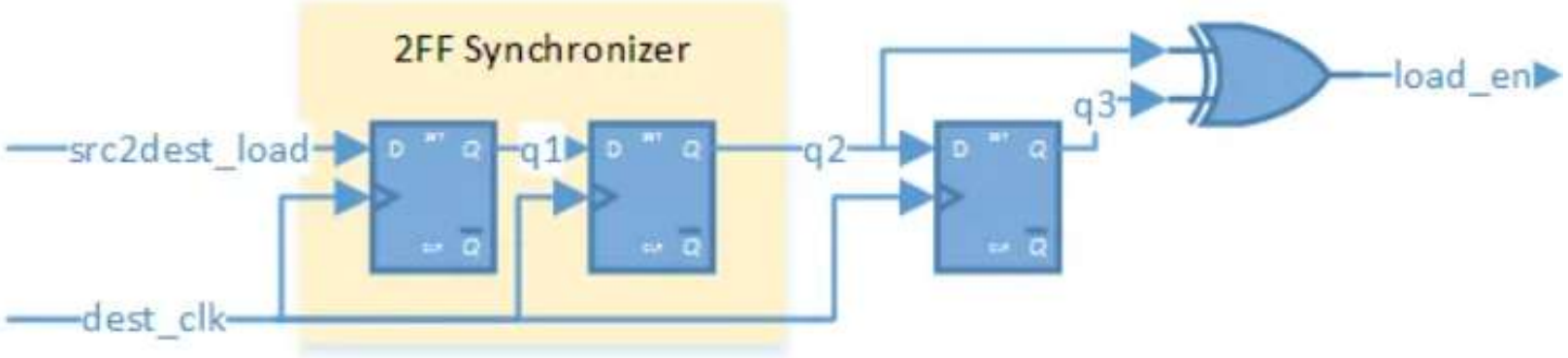
- The following circuit resides in the source clock domain, and converts an event that needs to traverse the clock domain crossing (CDC) into a toggle, which cannot be missed due to sampling in the destination clock domain.



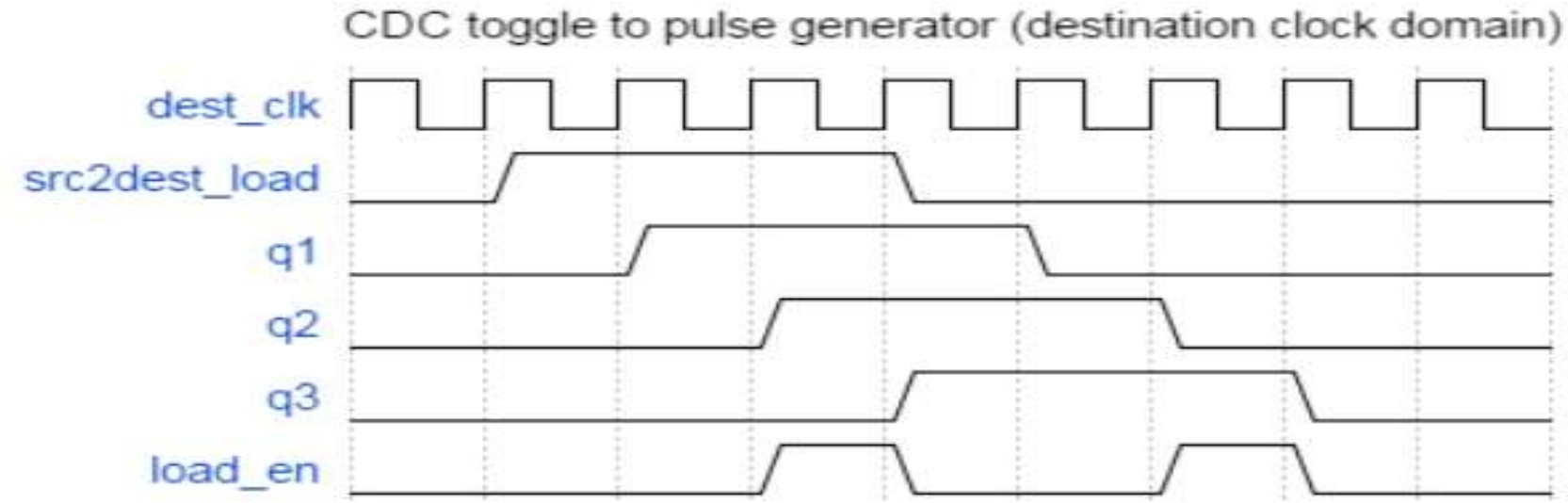
Advanced Digital Design

Multi-cycle path (MCP) formulation without feedback

- Destination clock domain toggle to load pulse generator

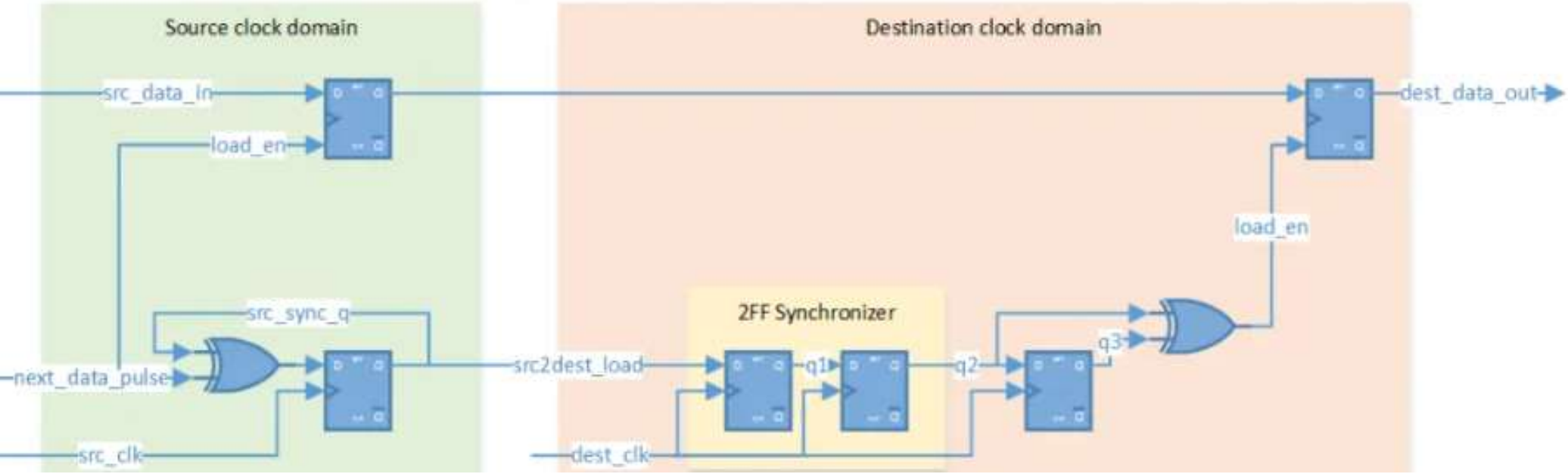


- Next, we need a circuit in the destination clock domain to convert the toggle back into a pulse to capture the multi-bit signal.



Advanced Digital Design
Multi-cycle path (MCP) formulation without feedback

- Finally, putting the entire synchronizer circuit together, we get the following.



- Notice the multi-bit data signal passes straight from source (clock) flip-flop to destination (clock) flip-flop to avoid problems with synchronizing multiple bits. A single control signal is synchronized to allow time for the multi-bit data to settle from possible metastable state.
- The load pulse from the source clock domain first gets converted into a toggle. The toggle is synchronized across the clock domain crossing (CDC), then gets converted back to a load pulse in the destination clock domain.
- Finally that load pulse is used to load the multi-bit data signal into flip-flops in the destination clock domain.

Advanced Digital Design
Multi-cycle path (MCP) formulation without feedback

- There is a limitation of the rate of how often data can be synchronized across the synchronizer. If you look at the complete circuit, the input data has to be held until the synchronization pulse loads the data in the destination clock domain.
- The whole process takes at least two destination clocks. Therefore to use this circuit, you must be certain that the input data only needs to be synchronized not more than once every three destination clock cycles.
- If you are unsure, then a more advanced synchronization circuit like the synchronizer with feedback acknowledgement or Dual-Clock Asynchronous FIFO should be used.

Advanced Digital Design
Multi-cycle path (MCP) formulation without feedback

- **Conclusion: -**Passing multiple signals across an asynchronous clock domain crossing (CDC) can become a recipe for disaster if not done properly.
- How does logic in the source clock domain know when it is safe to send another piece of data?

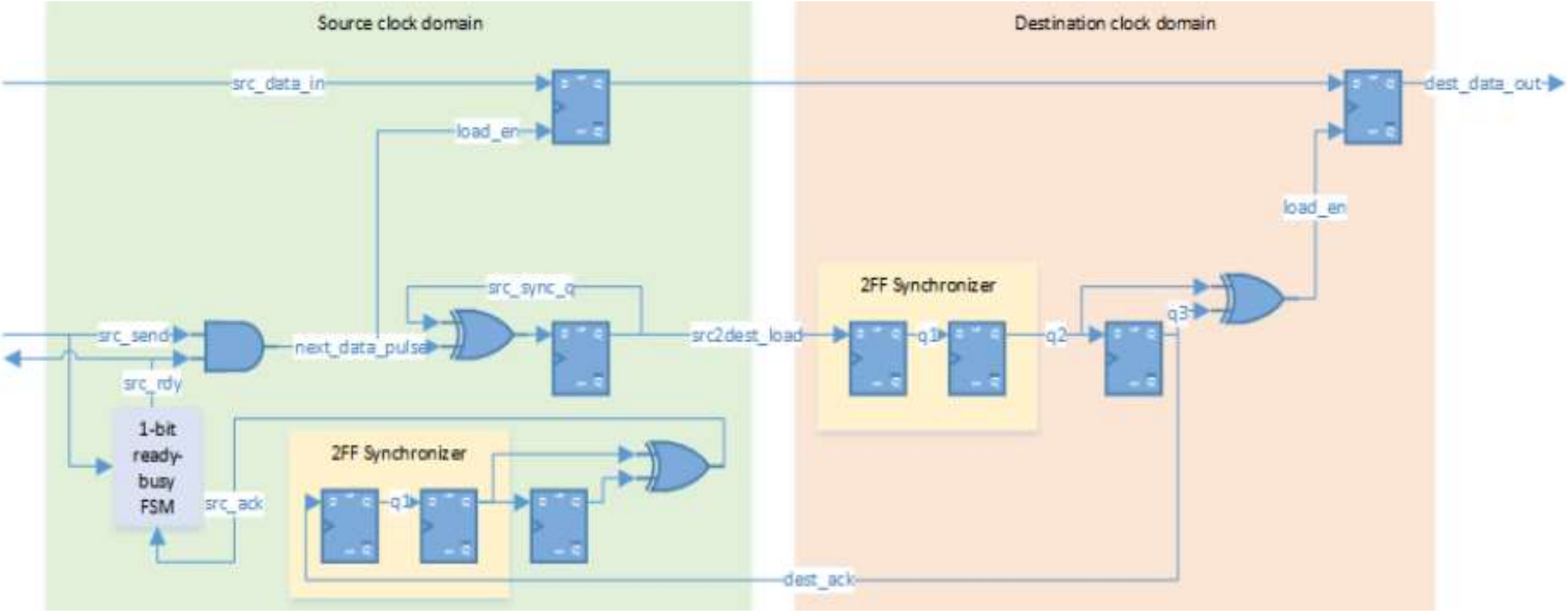
Multi-cycle path (MCP) formulation with feedback

- How can the source domain logic know when it is safe to send the next piece of data to the synchronizer? It can wait a fixed number of cycles, which can be determined from the synchronizer circuit.
- But a better way is to have logic in the synchronizer to indicate this to the source domain.

Advanced Digital Design

Multi-cycle path (MCP) formulation with feedback

- The following figure illustrates how this can be done. Compared to the MCP without feedback circuit, it adds a 1-bit finite state machine (FSM) to indicate to the source domain whether the synchronizer is ready to accept a new piece of data.



Advanced Digital Design

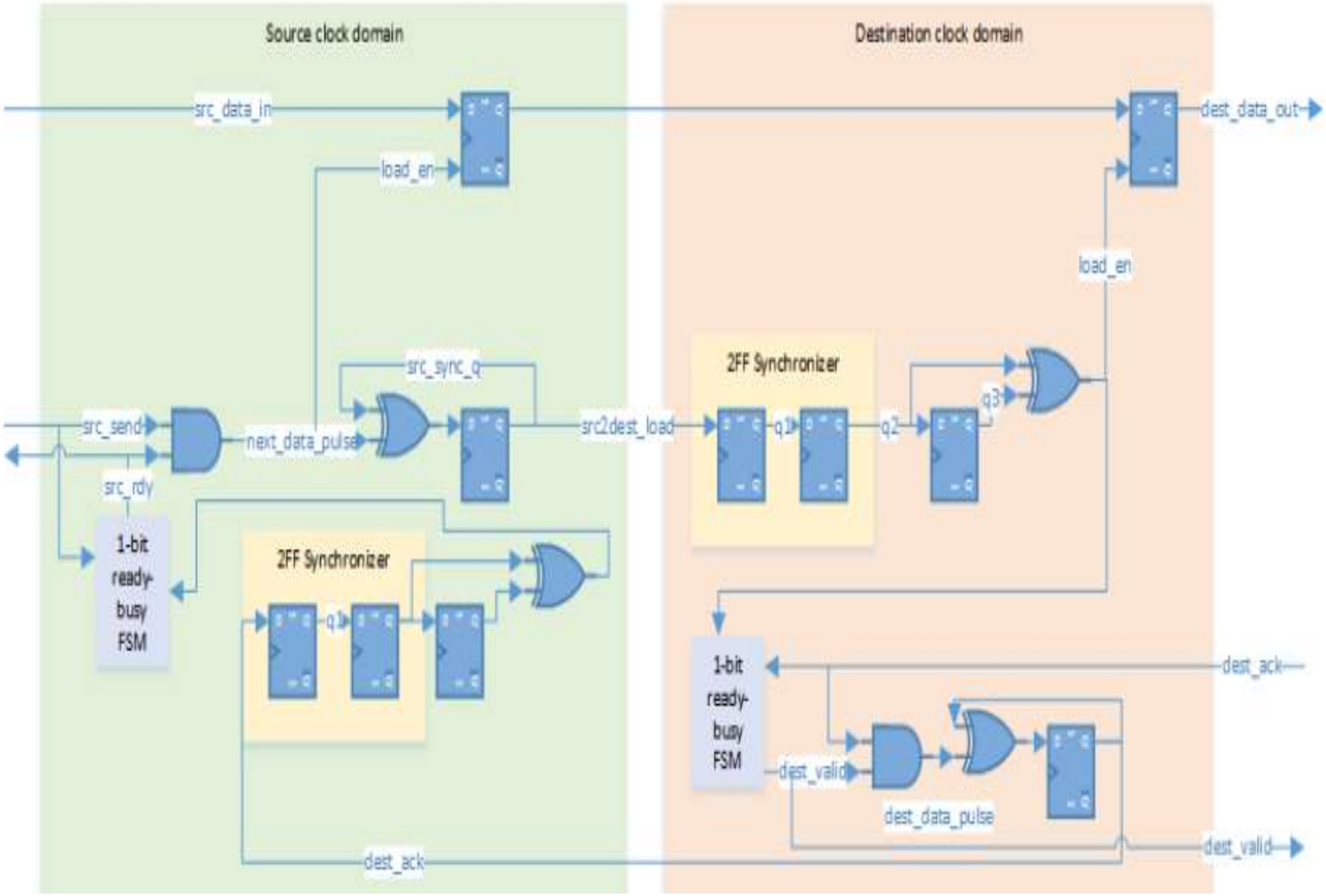
Multi-cycle path (MCP) formulation with feedback

- The 1-bit FSM has 2 states, 2 inputs, and 1 output (besides clock and reset).
  - States: **Idle, Busy**
  - Inputs**: src_send, src_ack**
  - Output: **src_rdy**
- **src_send** causes the FSM to transition to **Busy, src_ack** causes the FSM to transition back to Idle. **src_rdy** output is **1** when **Idle**, and **0** when **Busy**. The user logic outside the synchronizer needs to monitor these signals to determine when it is safe to send a new piece of data.

Advanced Digital Design

Multi-cycle path (MCP) formulation with feedback

- What if the destination domain further needs to back-pressure data from the source domain? To do so, the destination domain will need control over when to send feedback acknowledgement to the source domain.
- This can be accomplished by adding a similar 1-bit FSM in the destination side of the synchronizer.
- This addition allows the destination clock domain to throttle the source domain to not send any data until the destination domain logic is ready. The following figure illustrates this design.



Recommendations

- **Recommended 1-bit CDC techniques**
  - Register the signal in the source domain to remove combinational settling (glitches)
  - Synchronize the signal into the destination domain
- **Recommended multi-bit CDC techniques**
  - Consolidate multiple signals into a 1-bit representation, then synchronize the 1-bit signal
  - Use multi-cycle path (MCP) formulation to pass multi-bit data buses
  - Use asynchronous FIFOs to pass multi-bit data or control buses
  - Use gray code counters

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Advanced Digital Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Unit4: Synchronizer Design

Unit4: Lecture-2

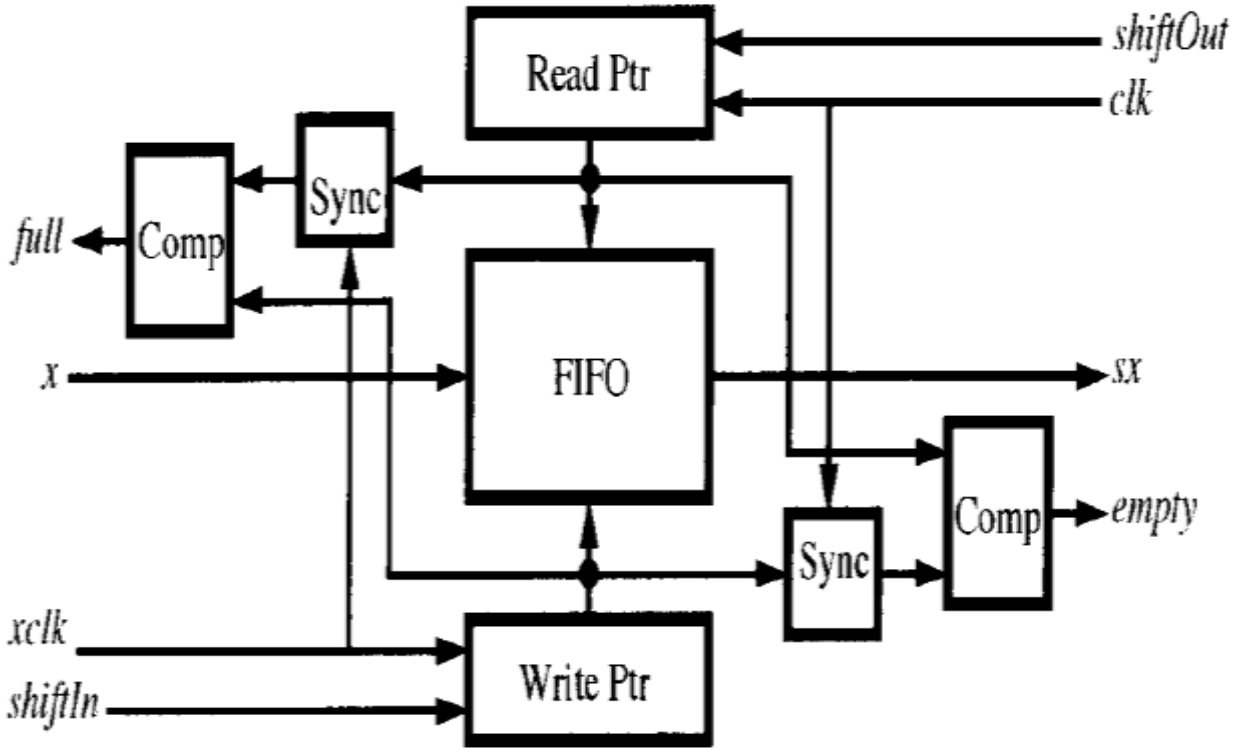
Sudeendra kumar K
Department of Electronics and Communication Engineering

Contents

- Asynchronous Synchronizer (FIFO based)

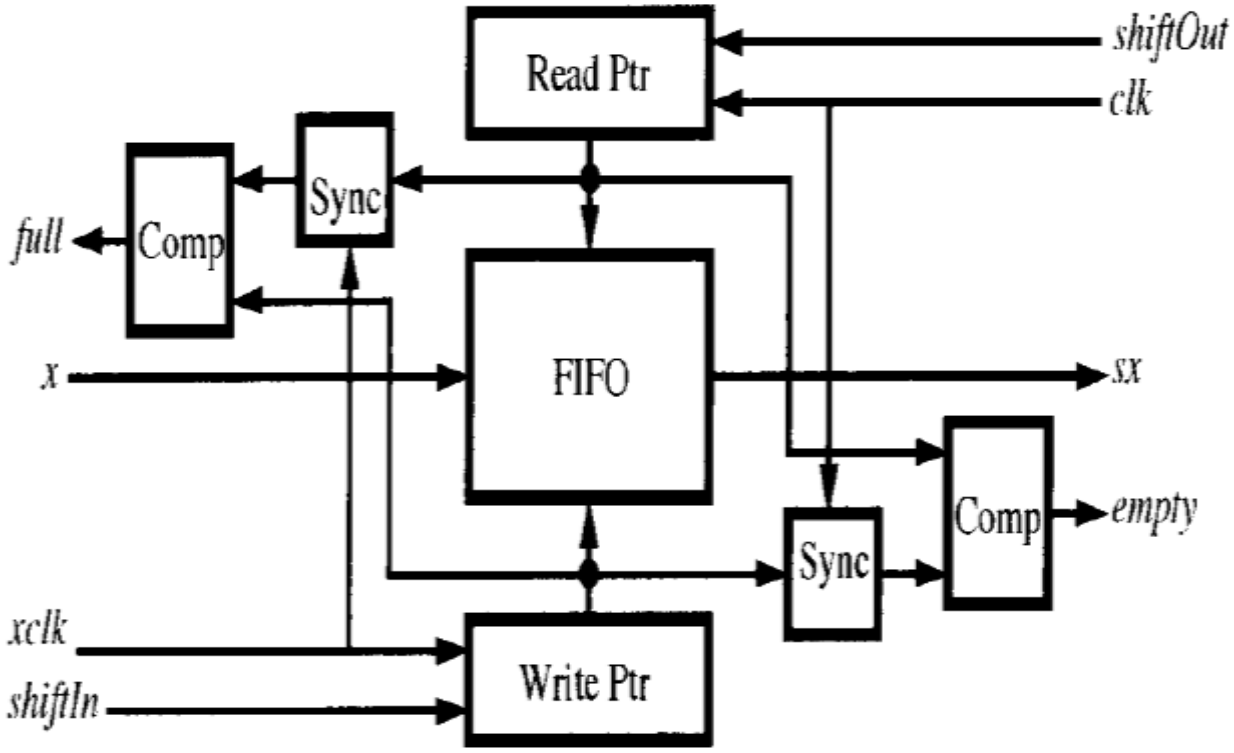
Advanced Digital Design
Asynchronous FIFO Synchronizers

- Using a FIFO to synchronize an asynchronous data stream moves the synchronization out of the data path and facilitates flow control.
- As illustrated in Figure, this method involves shifting data into a FIFO using the transmit clock and shifting data out of the FIFO using the local clock. Both clocks may be aperiodic.
- Synchronization is performed on the transmit and receive pointers to generate an "empty" signal in the local clock domain and a "full" signal in the transmit clock domain.



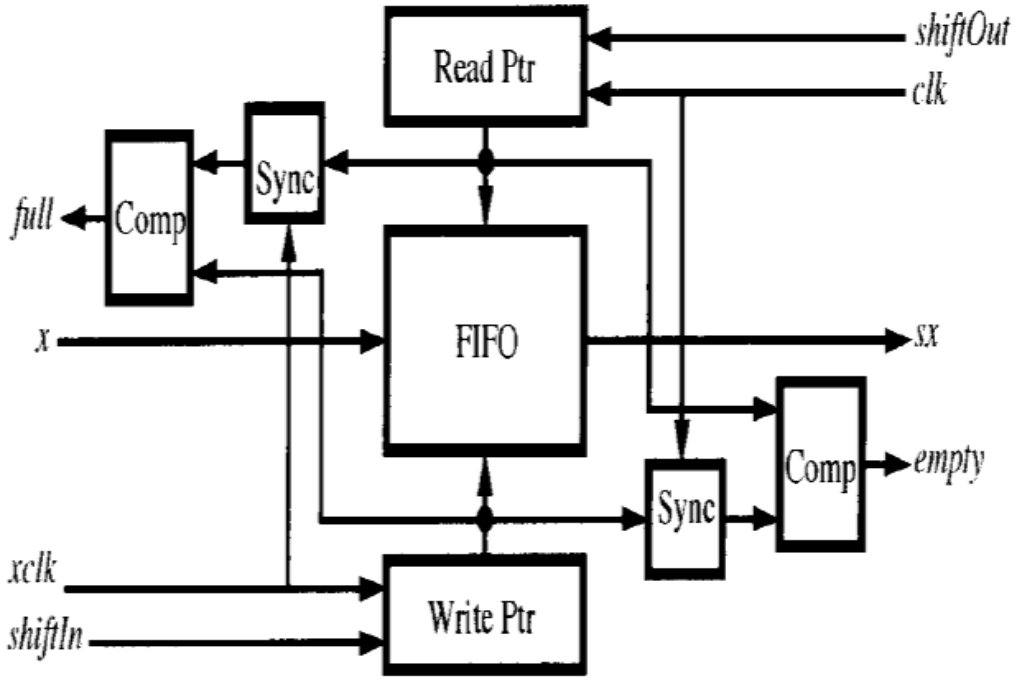
Advanced Digital Design
Asynchronous FIFO Synchronizers

- The FIFO in the center of Figure can be realized as a set of flip-flops with clock enables and an output multiplexer, as in Figure below.
- In practice, a register file or transparent latches are often used to save area. The read and write pointers are ring counters that sequence data into and out of the FIFO.
- The write pointer is advanced synchronously with the transmit clock, **xclk**, when the **shiftIn** signal is asserted. Similarly the read pointer is advanced synchronously with **clk** when **shiftOut** is asserted.
- Data are shifted into the FIFO in the transmit clock domain and shifted out of the FIFO in the receive clock domain.
- No synchronization is required unless the receiver attempts to read past the last word in the FIFO or the transmitter attempts to write beyond the FIFO capacity.



Advanced Digital Design
Asynchronous FIFO Synchronizers

- Synchronization is required to detect the "full" and "empty" conditions of the FIFO because this requires comparing the two pointers that are in different clock domains.
- In the figure, the read (write) pointer is synchronized with the transmit (local) clock, and the comparison to check for full (empty) is performed in the transmit (local) clock domain.
- This method is preferred to the alternative of performing the comparison directly on the two pointers without synchronization and then synchronizing the resulting asynchronous full and empty signals because the alternative delays the response of the empty (full) signal to the **shiftOut (shiftIn)** signal. The method illustrated in Figure on the other hand, immediately sets empty (full) when the last word is shifted out (in), avoiding underruns (overruns).
- The synchronization delay is only incurred when information must be passed between the two clock domains. For example, when the transmitter shifts the first word into an empty FIFO, a synchronizer delay is required before the receiver detects empty as low.



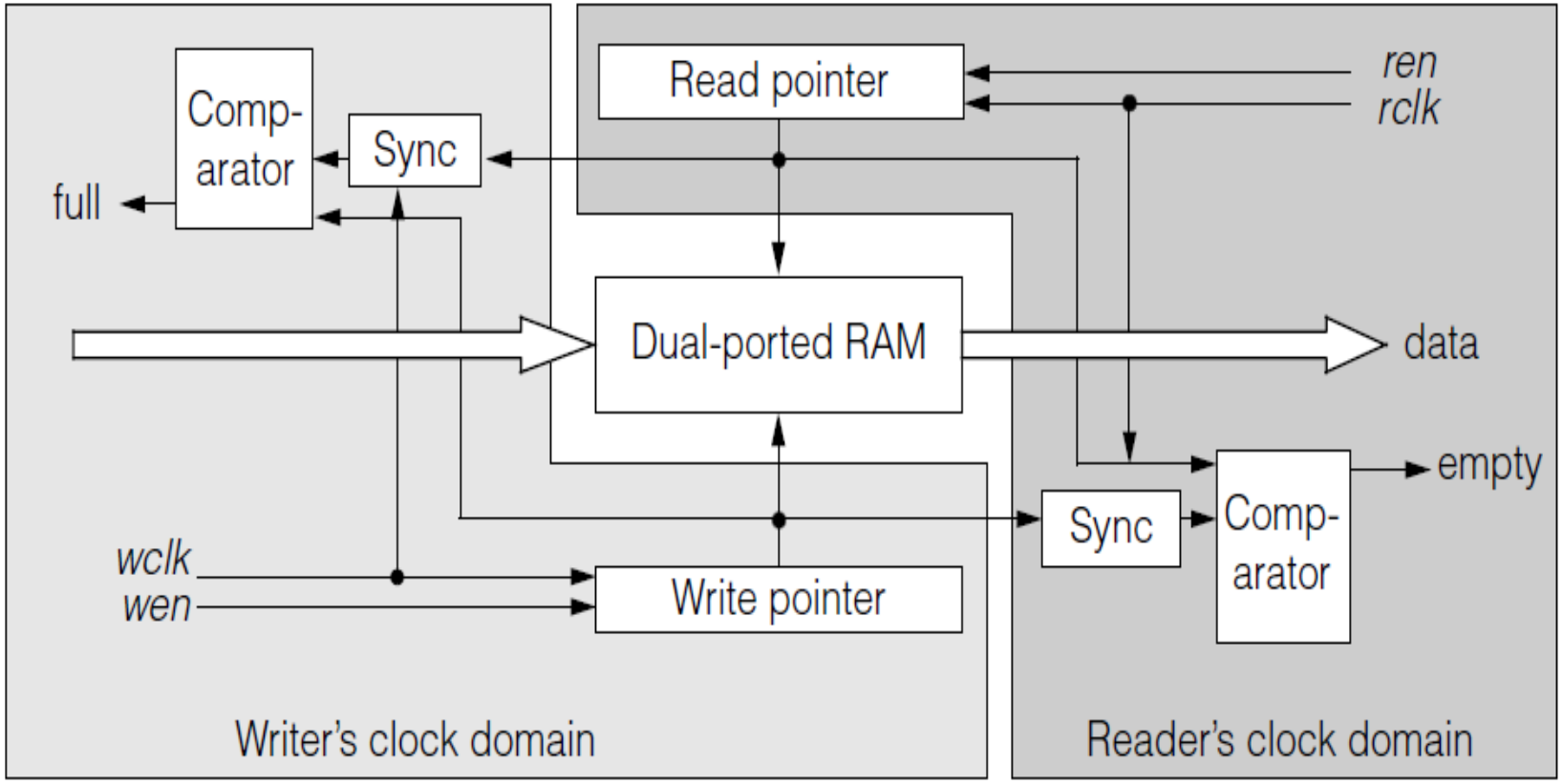
Advanced Digital Design
Asynchronous FIFO Synchronizers

- The asynchronous FIFO synchronizer has two advantages compared with the waiting synchronizer: lower probability of synchronization failure and inherent flow-control.
- The probability of synchronization failure is reduced because synchronizations are performed less frequently. If the read and write pointers are encoded in a one-hot manner, then only a single synchronization is performed per word and only when the FIFO toggles between the empty and nonempty states.
- In contrast the waiting synchronizer performs a synchronization for every bit transmitted.
- More significantly, the FIFO synchronizer provides inherent flow control via the full and empty signals.
- Every symbol that is shifted into the FIFO will be shifted out in the proper order. There is no need to add a protocol to prevent data from being dropped or duplicated because of rate mismatch.

Advanced Digital Design

Two Clock FIFO Synchronizer

- The most common fast synchronizer uses a two clock FIFO buffer as shown in Figure

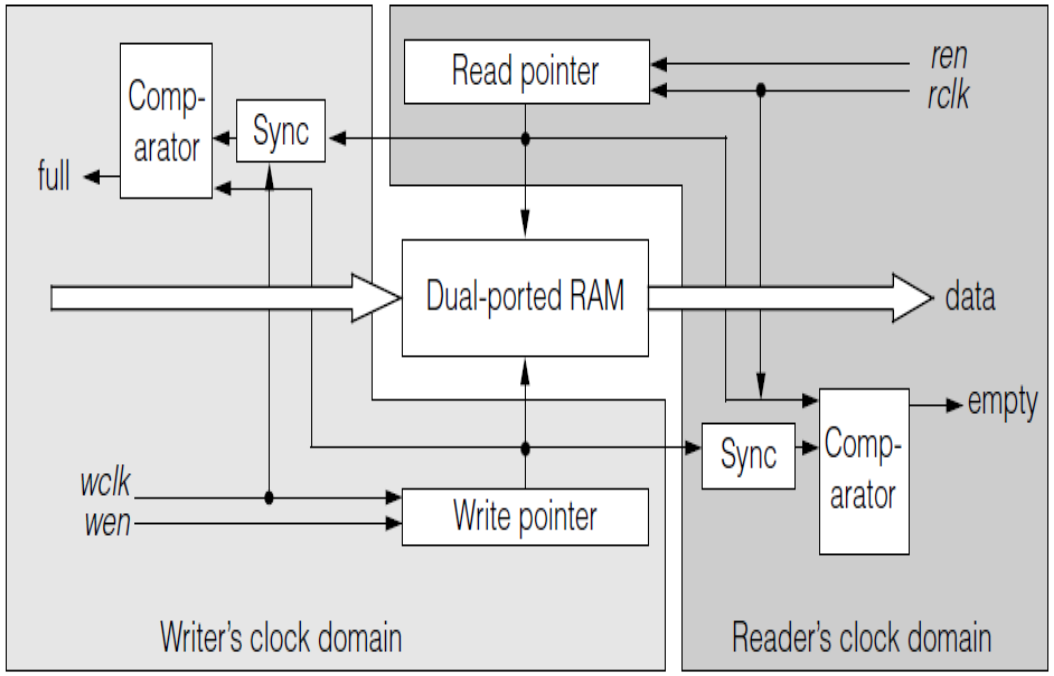


- Two-clock FIFO synchronizer. It contains two separate clock domains and synchronizes pointers rather than data.

Advanced Digital Design

Two –Clock FIFO Synchronizer

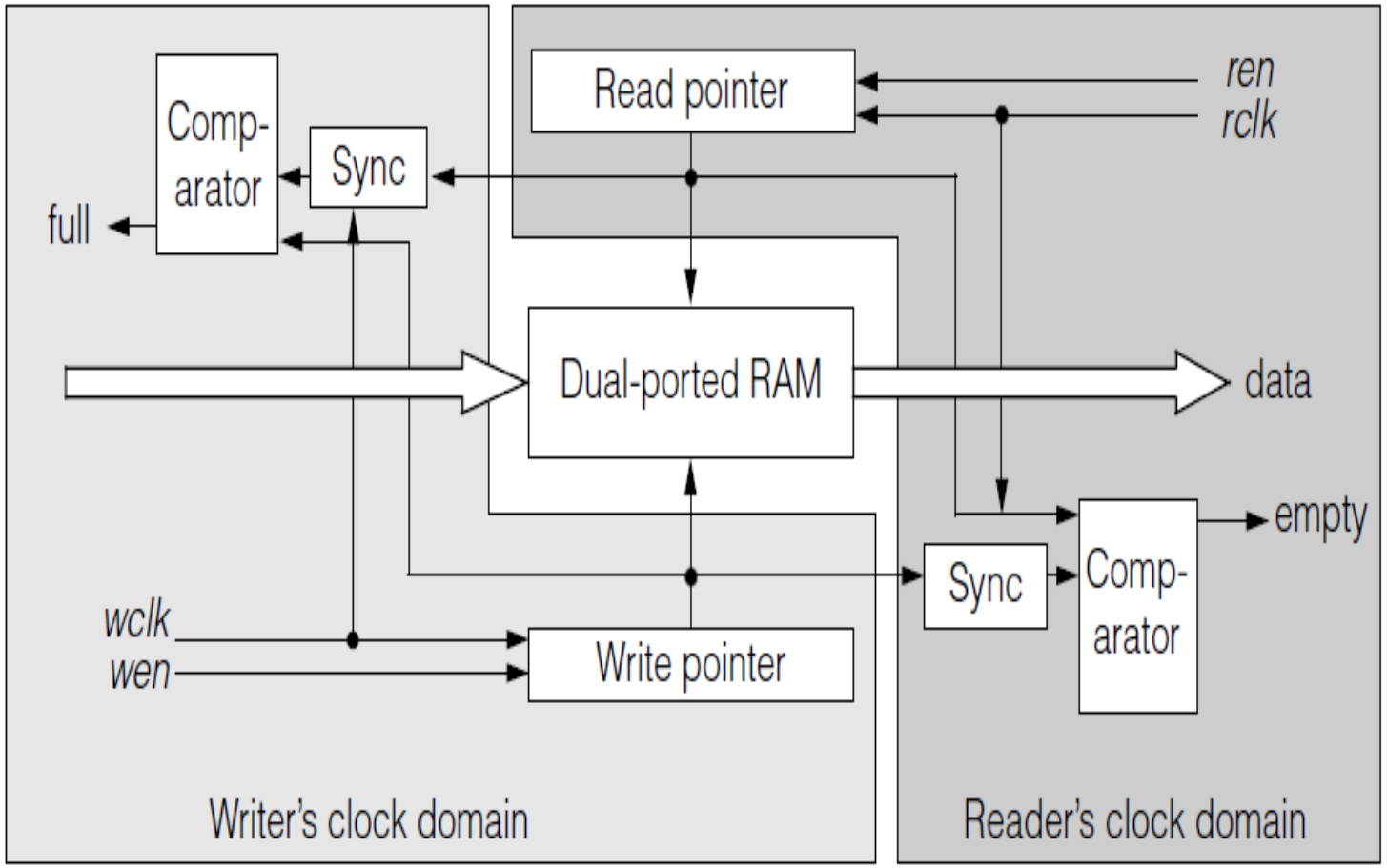
- The writer places data on the input bus and asserts wen (write enable); if full is not asserted, the data was accepted and stored. The reader asserts ren (read enable), and if empty is not asserted then data was produced at the output.
- The RAM is organized as a cyclic buffer. Each data word is written into the cell pointed to by the write pointer and is read out when the read pointer reaches that word.
- On write and on read, the write pointer and the read pointer are respectively incremented. When the read pointer points to the same word as the write pointer, the FIFO buffer is empty.
- To determine that, the two pointers must be compared. However, they belong to two different clock domains thus, the write pointer has to be synchronized with rclk (read clock) when compared.
- That's where the synchronization is; it's applied to the pointers, rather than to the data. That's also where latency is incurred.



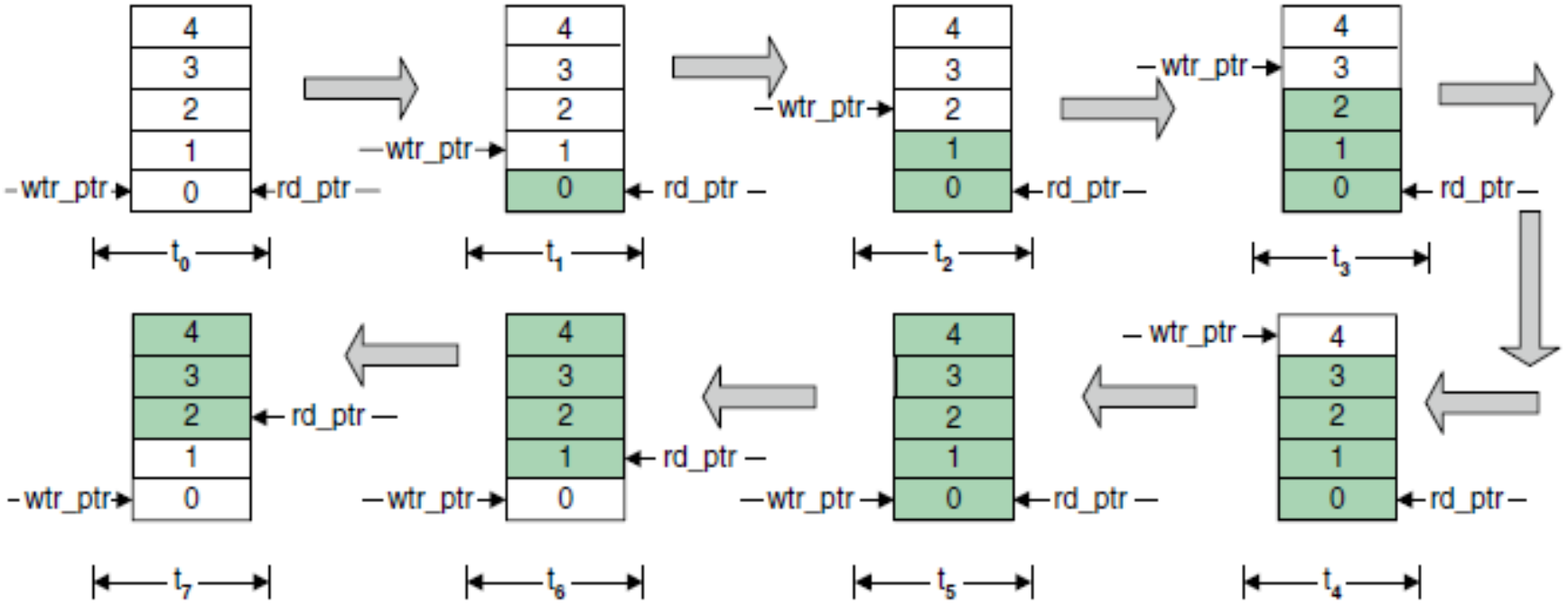
Advanced Digital Design

Two Clock FIFO Synchronizer

- When a new data word is written into an empty FIFO buffer, it might take one or two additional **rclk** cycles before the new write pointer passes through the synchronizer and de-asserts empty.
- But when the two pointers are far from each other, no synchronization latency is incurred; data latency is still there. When the RAM holds k words, a newly inserted word will stay there for at least k **rclk** cycles before it is read out.
- Incidentally, the pointers are usually maintained in Gray code so that only a single bit at a time changes in the synchronizer.



Effect of Synchronization on FIFO full logic

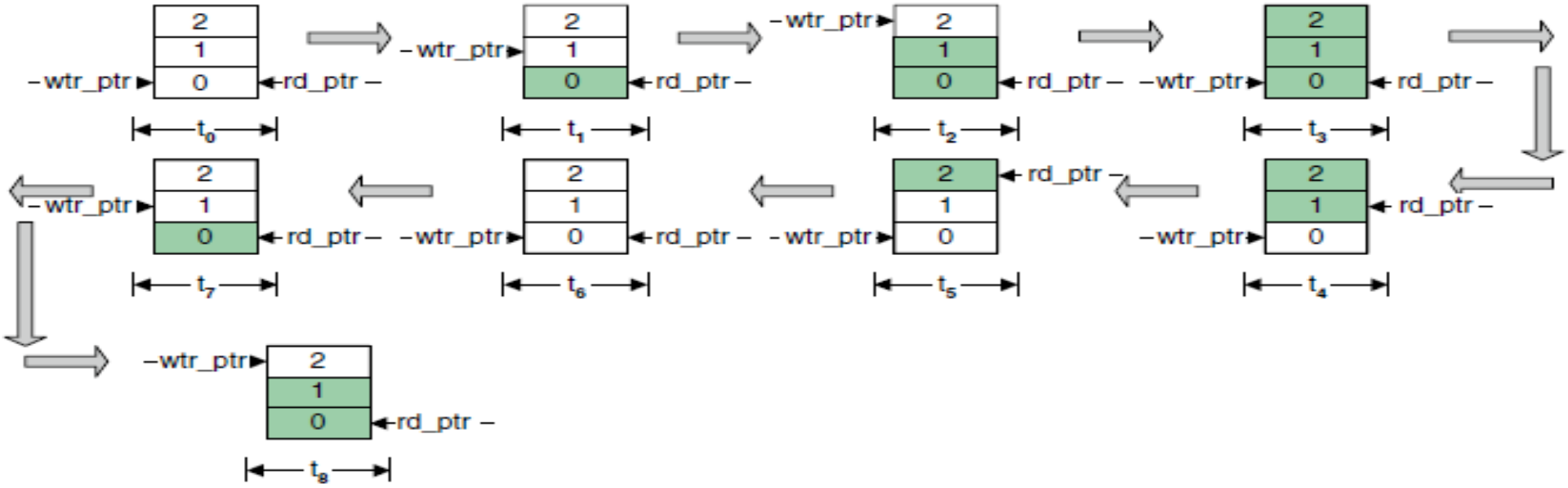


- To calculate the FIFO full condition, the read and write pointer that are incremented on their respective clocks have to be compared. The read pointer (Gray coded) needs to be synchronized to the write clock.
- As shown in Figure, initially read and write pointers are zero at t0 with FIFO empty. As subsequent write takes place on FIFO, write pointer gets incremented.
- A stage is reached when write pointer equals read pointer and FIFO becomes FULL. This happens at t5 as shown in Figure.

Effect of Synchronization on FIFO full logic

- Now incase a read takes place at t6, since a typical synchronizer circuit consists of at least two flip flops, synchronizing read pointer on write clock will result in changed read pointer reflected after two write clocks. This results in blocking additional writes on the FIFO for additional cycles but is harmless. It would have been a problem if writes were not blocked when the FIFO was actually full.
- For the FIFO Empty calculation, write pointer is synchronized to the read clock and compared against the read pointer. Due to this, read side sees delayed writes (two clock delayed signal), and would still indicate FIFO empty even though it actually has some data. This will result in reads getting blocked till the writes becomes visible to the read side.

Effect of Synchronization on FIFO empty logic

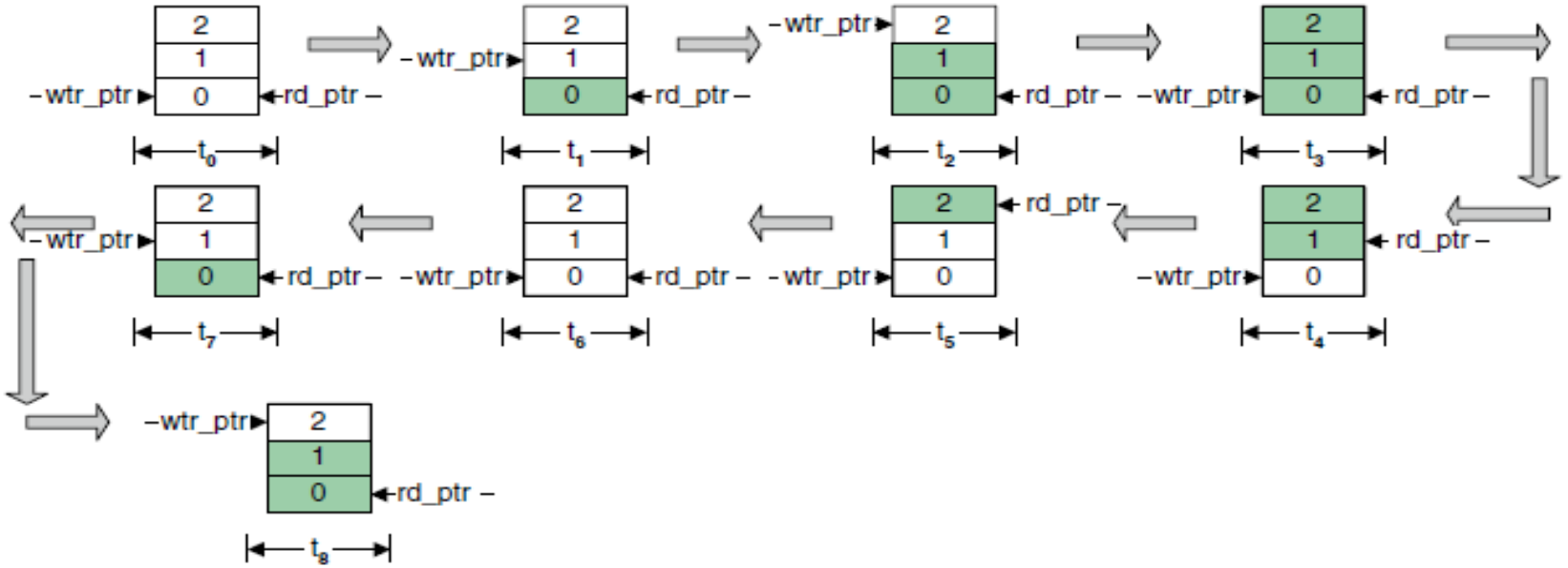


For the FIFO Empty calculation, write pointer is synchronized to the read clock and compared against the read pointer. Due to this, read side sees delayed writes (two clock delayed signal), and would still indicate FIFO empty even though it actually has some data. This will result in reads getting blocked till the writes becomes visible to the read side.

As shown in Figure, initially read and write pointers are zero at t0 with FIFO empty. As subsequent write takes place on FIFO, write pointer is incremented. A stage is reached when write-pointer equals to read-pointer and FIFO becomes FULL. This happens at t3 as shown in Figure.

Effect of Synchronization on FIFO empty logic

- Subsequent reads starts at t4 and again FIFO becomes empty at t6. FIFO is again written back at t7 and t8, now since a typical synchronizer circuit consists of at least two flip flops, synchronizing write pointer on read clock will result in changed write pointer reflected after two read clocks. This results in blocking additional reads on FIFO and is harmless. It would have been a problem if reads were not blocked when the FIFO was actually empty.



Two Clock FIFO Synchronizer

- When a new data word is written into an empty FIFO buffer, it might take one or two additional **rclk** cycles before the new write pointer passes through the synchronizer and de-asserts empty.
- But when the two pointers are far from each other, no synchronization latency is incurred; data latency is still there. When the RAM holds k words, a newly inserted word will stay there for at least k **rclk** cycles before it is read out.
- Incidentally, the pointers are usually maintained in Gray code so that only a single bit at a time changes in the synchronizer.

Two Clock FIFO Synchronizer

- When a new data word is written into an empty FIFO buffer, it might take one or two additional **rclk** cycles before the new write pointer passes through the synchronizer and de-asserts empty.
- But when the two pointers are far from each other, no synchronization latency is incurred; data latency is still there. When the RAM holds k words, a newly inserted word will stay there for at least k **rclk** cycles before it is read out.
- Incidentally, the pointers are usually maintained in Gray code so that only a single bit at a time changes in the synchronizer.

FIFO Depth Calculation:

- **Case – 1 : $f_A > f_B$** with no idle cycles in both write and read.
- **Transmitter frequency** f_A = 80MHz.
- **Receiver Frequency** f_B = 50MHz.
- Burst Length = No. of data items to be transferred = 120.
- There are no idle cycles in both reading and writing which means that, all the items in the burst will be written and read in consecutive clock cycles.

So, the minimum depth of the FIFO should be 45.

- ✓ Time required to write one data item = $\frac{1}{80\ MHz}$ = 12.5 nSec.
- ✓ Time required to write all the data in the burst = 120 * 12.5 nSec. = 1500 nSec.
- ✓ Time required to read one data item = $\frac{1}{50\ MHz}$ = 20 nSec.
- ✓ So, for every 20 nSec, the module B is going to read one data in the burst.
- ✓ So, in a period of 1500 nSec, 120 no. of data items can be written.
- ✓ And the no. of data items can be read in a duration of 1500 nSec = $\left(\frac{1500\ nSec}{20\ nSec}\right) = 75$
- ✓ The remaining no. of bytes to be stored in the FIFO = 120 – 75 = 45.
- ✓ So, the FIFO which has to be in this scenario must be capable of storing 45 data items.

FIFO Depth Calculation

- Case -2 : $f_A > f_B$ with one clk cycle delay between two successive reads and writes.

- This is just, to create some sort of confusion. This scenario is no way different from the previous scenario (case -1), because, always, there will be one clock cycle delay between two successive reads and writes. So, the approach is same as the earlier one

Case – 3 : $f_A > f_B$ with idle cycles in both write and read.

- Writing frequency = f_A = 80MHz.
- Reading Frequency = f_B = 50MHz.
- Burst Length = No. of data items to be transferred = 120.
- No. of idle cycles between two successive writes is = 1.
- No. of idle cycles between two successive reads is = 3.

So, the minimum depth of the FIFO should be 83.

- ✓ Time required to write all the data in the burst = 120 * 25 nSec. = 3000 nSec.
- ✓ Time required to read one data item = $4 * \frac{1}{50\ MHz}$ = 80 nSec.
- ✓ So, for every 80 nSec, the module B is going to read one data in the burst.
- ✓ So, in a period of 3000 nSec, 120 no. of data items can be written.
- ✓ The no. of data items can be read in a period of 3000 nSec = $\left(\frac{3000\ nSec}{80\ nSec}\right) = 37.5 \simeq 37$
- ✓ The remaining no. of bytes to be stored in the FIFO = 120 – 37 = 83.
- ✓ So, the FIFO which has to be in this scenario must be capable of storing 83 data items.

- ✓ The no. of idle cycles between two successive writes is 1 clock cycle. It means that, after writing one data, module A is waiting for one clock cycle, to initiate the next write. So, it can be understood that for every **two** clock cycles, one data is written.
- ✓ The no. of idle cycles between two successive reads is 3 clock cycles. It means that, after reading one data, module B is waiting for 3 clock cycles, to initiate the next read. So, it can be understood that for every **four** clock cycles, one data is read.
- ✓ Time required to write one data item = $2 * \frac{1}{80\ MHz}$ = 25 nSec.

FIFO Depth Calculation

Case – 4 : fA > fB with duty cycles given for wr_enb and rd_enb.

- Writing frequency = **fA = 80MHz.**
- Reading Frequency **= fB = 50MHz.**
- Burst Length = No. of data items to be transferred = 120.
- Duty cycle of **wr_enb** (write enable) = 50 % = ½.
- Duty cycle of **wr_enb (**write enable) = 25 % = ¼.

- **Solution: -This** scenario is no way different from the previous scenario (case - 3), because, in this case also, one data item will be written in 2 clock cycles and one data item will be read in 4 clock cycles.

FIFO Depth Calculation

Case – 5 : fA < fB with no idle cycles in both write and read (i.e., the delay between two consecutive writes and reads is one clock cycle).

- Writing frequency = fA = 30MHz.
- Reading Frequency = fB = 50MHz.
- Burst Length = No. of data items to be transferred = 120.
- There are no idle cycles in both reading and writing which means that, all the items in the burst will be written and read in consecutive clock cycles.

Solution: - In this case, a FIFO of depth ‘1’ will be sufficient because, there will not be any data loss since the reading is faster than writing

Case – 6 : fA < fB with idle cycles in both write and read (duty cycles of wr_enb and rd_enb can also be given in these type of questions).

- Writing frequency = fA = 30MHz.
- Reading Frequency = fB = 50MHz.
- Burst Length = No. of data items to be transferred = 120.
- No. of idle cycles between two successive writes is = 1.
- No. of idle cycles between two successive reads is = 3.

The minimum depth of the FIFO should be 20.

- ✓ The no. of idle cycles between two successive writes is 1 clock cycle. It means that, after writing one data, module A is waiting for one clock cycle, to initiate the next write. So, it can be understood that for every **two** clock cycles, one data is written.
- ✓ The no. of idle cycles between two successive reads is 3 clock cycles. It means that, after reading one data, module B is waiting for 3 clock cycles, to initiate the next read. So, it can be understood that for every **four** clock cycles, one data is read.
- ✓ Time required to write one data item = $2 * \frac{1}{30\ MHz}$ = 66.667 nSec.
- ✓ Time required to write all the data in the burst = 120 * 66.667 nSec. = 8000 nSec.
- ✓ Time required to read one data item = $4 * \frac{1}{50\ MHz}$ = 80 nSec.
- ✓ So, for every 80 nSec, the module B is going to read one data item in the burst.
- ✓ So, in a period of 8000 nSec, 120 no. of data items can be written.
- ✓ The no. of data items can be read in a period of 8000 nSec = $\left(\frac{8000\ nSec}{80\ nSec}\right) = 100$
- ✓ The remaining no. of bytes to be stored in the FIFO = 120 – 100 = 20.
- ✓ So, the FIFO which has to be in this scenario must be capable of storing 20 data items.

References

- William J Dally, John W Poulton, “Digital Systems Engineering”, Cambridge University Press.

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu

Advanced Digital Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Reset Domain Crossing

Unit4: Lecture-5

Sudeendra kumar K
Department of Electronics and Communication Engineering

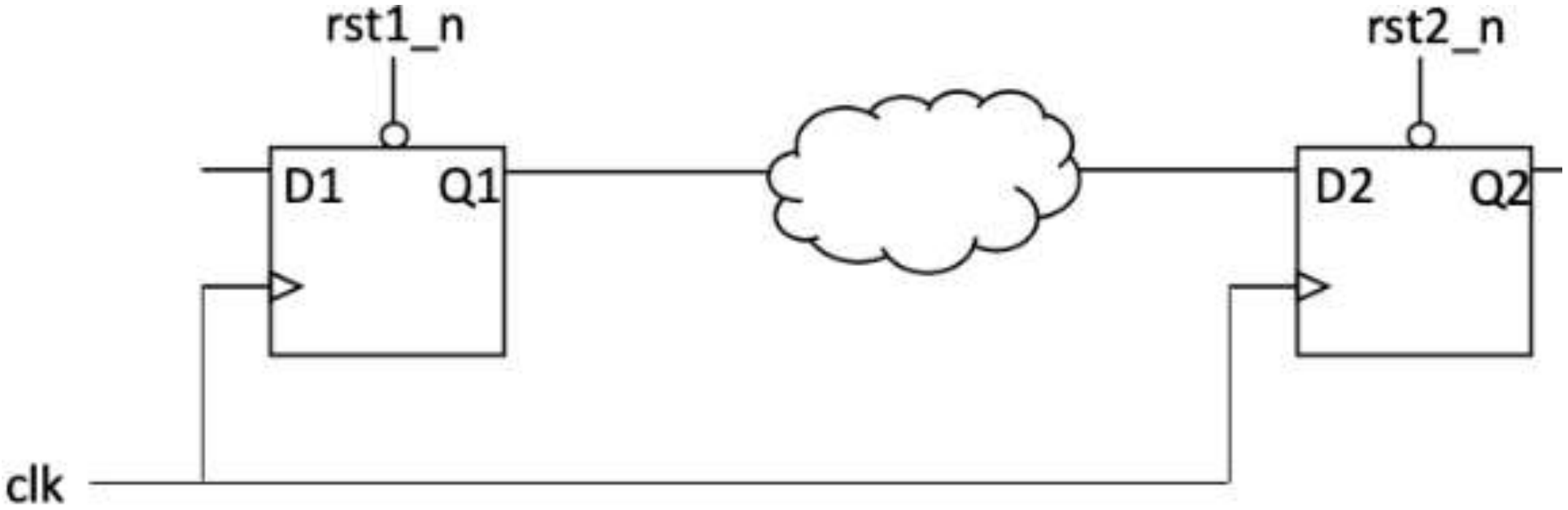
Advanced Digital Design

Contents

- Reset Domain Crossing

Advanced Digital Design

Reset Domain Crossing



Reset Domain Crossing Scenario

Advanced Digital Design

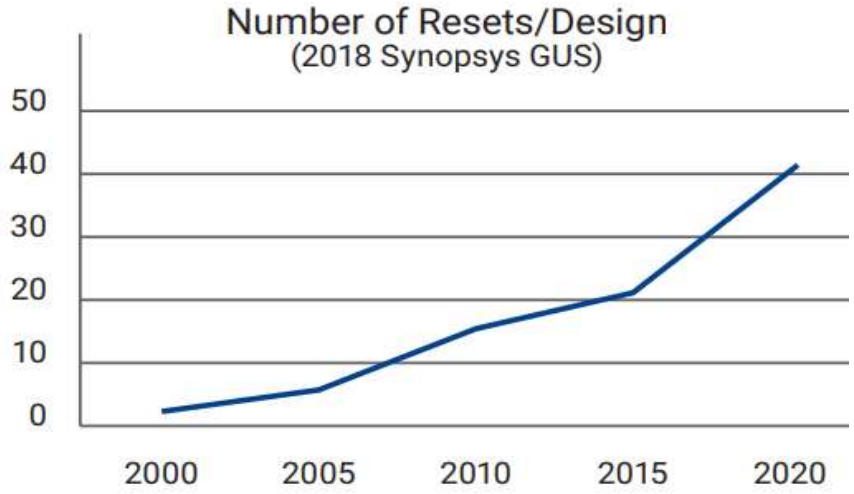
Why Resets are required

- The ability to initialize the entire hardware design and clear all software running through a system-on-chip (SoC) is essential. Stating with a known state avoids propagation of signals with unknown values.
- Despite the best efforts at verification, lingering corner-case bugs may put a system into a state where reset is the only solution to flush unknown values.
- Transient faults such as particle hits, cross talk and environmental effects can also create a situation where reset is required.
- Low-power design techniques that rely on turning parts of the design on and off require the ability to reset. For these reasons and more, the number of reset domains in SoCs has been growing and will continue to increase.

Advanced Digital Design

Why Resets are required

- Studies have shown that the number of resets in a typical chip design has been increasing. Resets provide key functionality, including:
  - Changing from unknown to known state for correct operation when re-booted.
  - State retention and restore after or before their power domain is turned back on from off state.
  - Recovery of SoC or block from critical failures.
  - Isolating independent blocks working on different functionality



Modern chips contain many reset domains

Why Multiple Resets are Required

- Although resets serve many different purposes, they can be grouped into three general categories:
  - An externally applied **power on reset (POR)** is used to get the design into a known initial state when the system starts. –
  - A **cold POR** occurs when the chip has been completely turned off and power is first applied to boot the system. –
  - A **warm POR** occurs when the chip is already operating, and a re-boot is desired. A warm reboot may not perform a power on self-test (POST), the system checks run as part of the full boot process.
  - A warm POR may also be applied to a subsystem or block instead of the entire SoC.

Advanced Digital Design

Why Multiple Resets are Required

- An Internally generated reset occurs when either hardware or software determines that reset is needed. Internal resets may affect only specific portions of the design.
- For example, when a power domain is turned on, the power controller or power control software issues a reset to the domain to ensure that the hardware in the domain is properly initialized.
- Design elements which have been affected due to internal resets may be fewer than for PORs. It is common for a power domain to include retention registers to maintain critical state information while the domain is turned off.
- An internally generated hardware reset on domain power up would not affect these registers, but a POR would initialize them along with all other state. – An internally generated software reset affects even less area of the SoC.

Why Multiple Resets are Required

- This is typically issued by the software/firmware running on the SoC upon a context switch, and it clears design space for the new process.
- Software resets may also be issued when a watchdog timer or automation control internal to the system detects a period of inactivity or critical errors, indicating system malfunction.
- This is common in safety-critical applications such as chips for autonomous vehicles, avionics and medical devices.
- The final category of resets is those issued for test or debug purposes. Depending upon the intent, they may affect only very specific parts of the design.
- For example, the JTAG standard defines an optional pin to reset the test logic, generally with no effect on the core functionality of the chip.

Advanced Digital Design

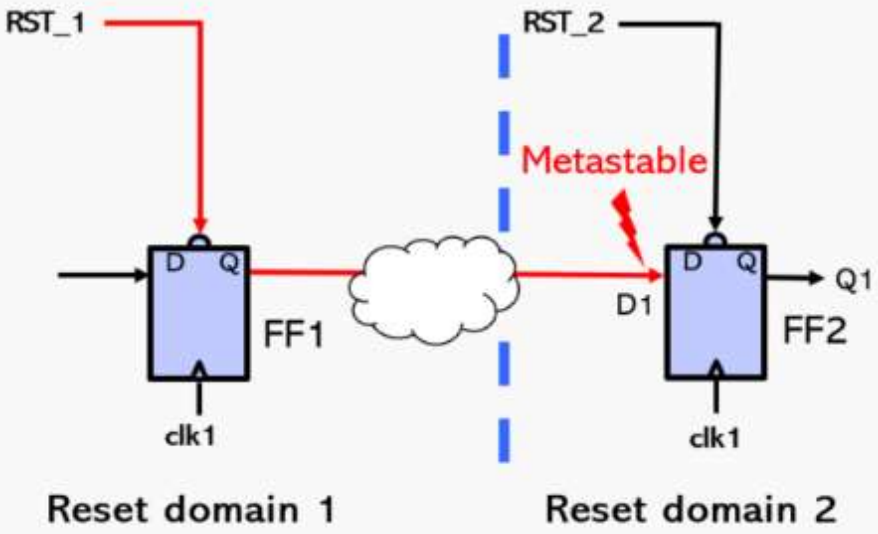
Reset Domain Crossing

- A portion of the chip with a unique reset signal is called a reset domain, and a signal traversing from one reset domain to another creates a reset domain crossing (RDC).
- Although reset domain crossings and clocks domain crossings are completely different in terms of analysis, root cause and debug, the problems are similar.
- These include metastability, in which a signal takes too long to settle to a known value. Note that RDC metastability can occur even within a single clock domain.

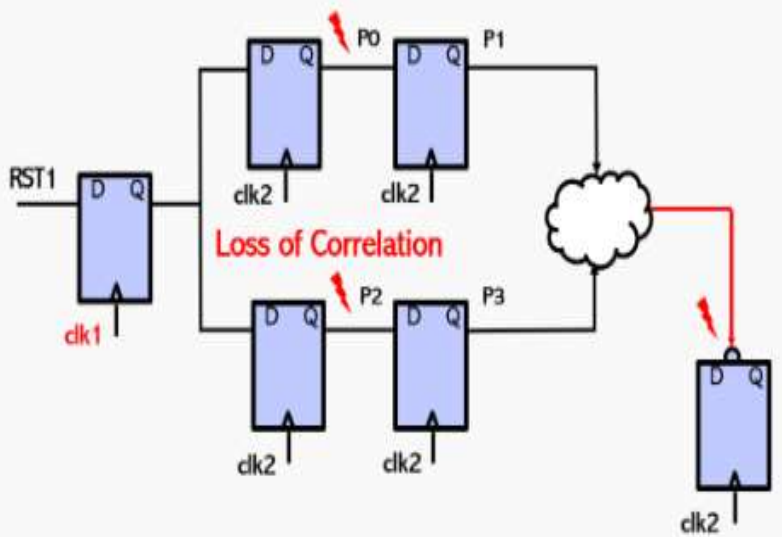
Advanced Digital Design

Reset Domain Crossing: Failures from Asynchronous Resets

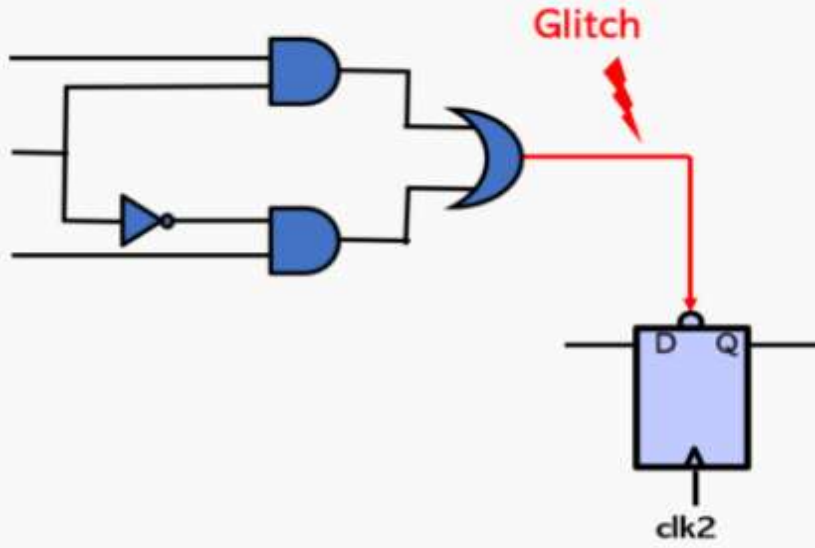
Metastability



Improper Functional Correlation



Glitches



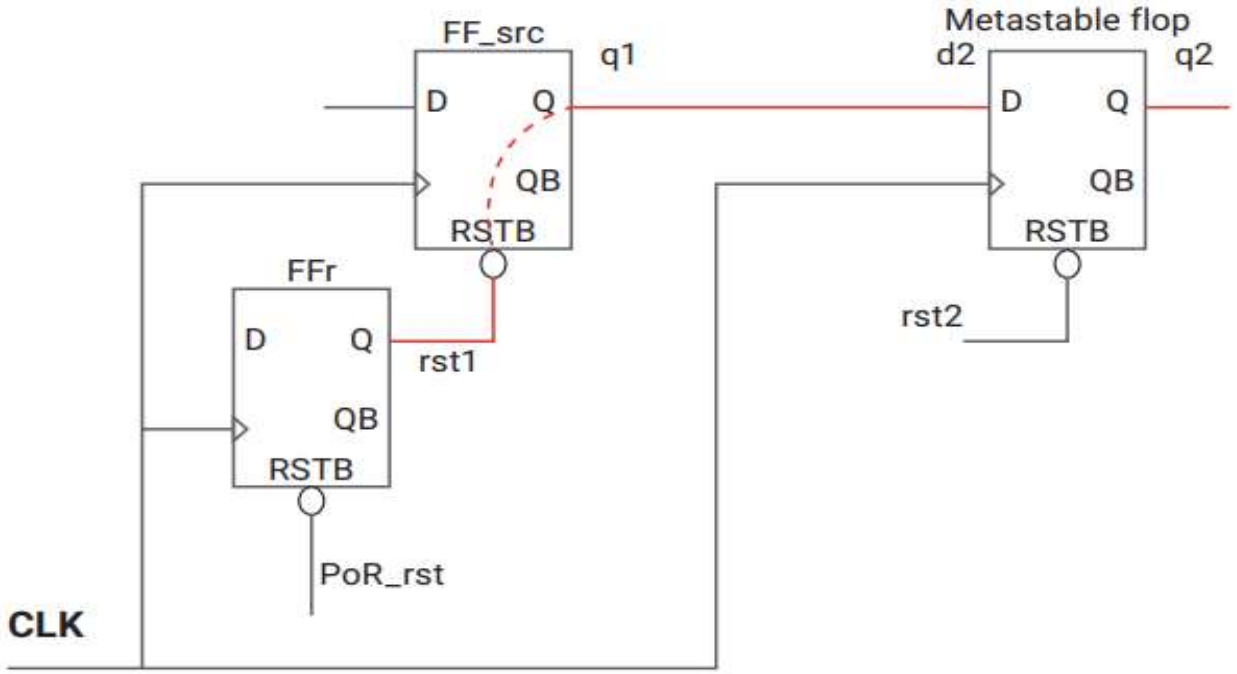
Metastability when asynchronous resets are activated or deactivated. If the asynchronous reset propagates from one reset domain to another, it can **cause metastability that leads to design failure.**

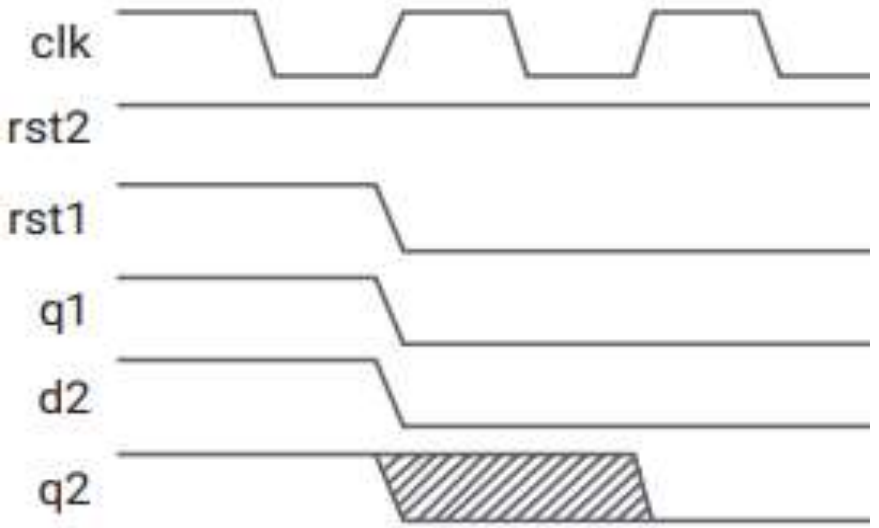
Reconverging synchronized resets can cause uncertainty and functional errors. When two events from one reset propagate through multiple synchronizers, they can create incorrect functional behavior where the flop being driven goes to an

Glitches for **asynchronous resets**. A design can have multiple sources toggling at different times resulting in a glitch at an asynchronous reset. This can cause an intermediate wrong value at the flop it drives and can cause a functional failure.

Advanced Digital Design

RDC Metastability within a clock domain





- Figure shows a case in which a signal from a flip-flop in one reset domain drives a flip-flop in another reset domain whose resets operate independently.
- If the flip-flop which is being reset (FF_src) changes value close to the active clock edge of the capturing flip-flop (Metastable flop), the capturing flip-flop may go metastable and its output will be indeterminate.
- This is because paths going from an asynchronous pin (such as the red path in Figure) typically are not timed. The rst1 signal may be a software reset or may be due to PoR_rst.

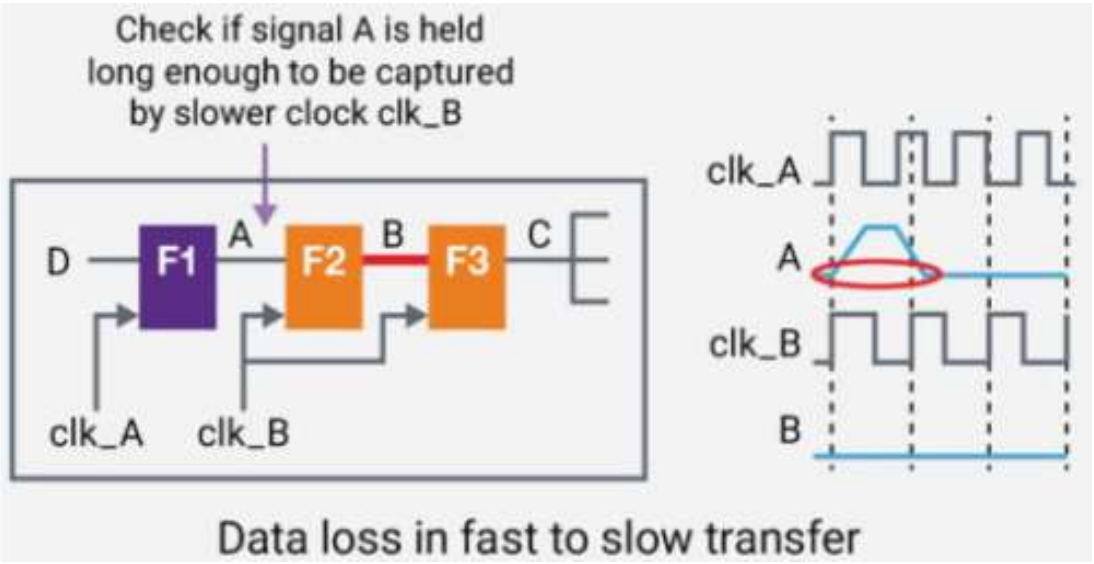
Advanced Digital Design

Reset Domain Crossing

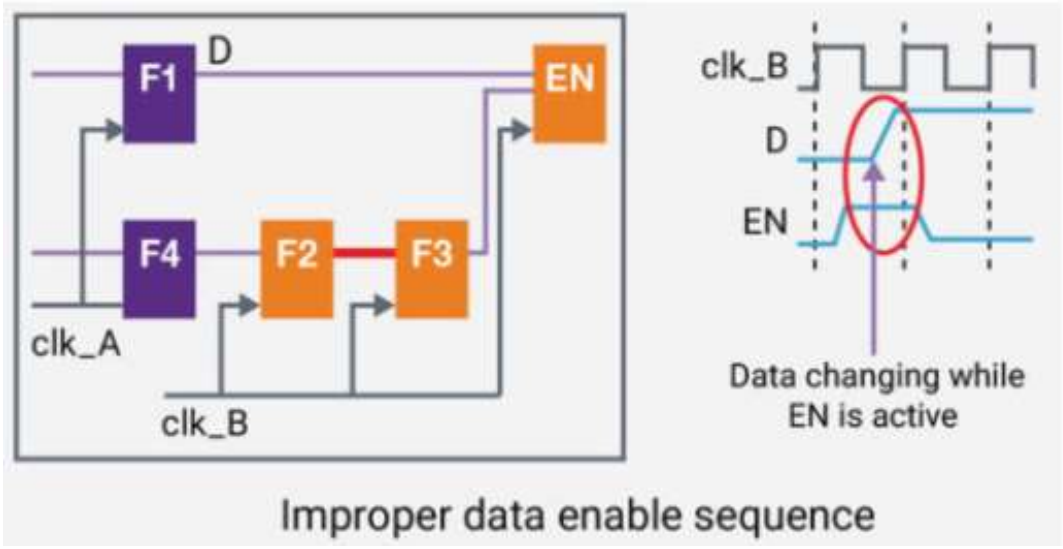
- Any time that data crosses a reset boundary, there is potential for corruption. Scenarios in which this can occur include: -
  - Data transferring between two IP blocks with different resets
  - Data transferring between an IP block and a CPU or core with different resets
  - Data transferring from an interface IP block to memory via a DMA engine
  - A resettable memory controller interfacing to a memory or storage logic that is not receiving any reset
- If these scenarios are not detected and fixed during pre-silicon verification, the resulting fabricated chip may be very difficult to debug and fix. It is rare to be able to control all resets, especially those generated internally, so even testing the chip in the bring-up lab may be very difficult.

Advanced Digital Design

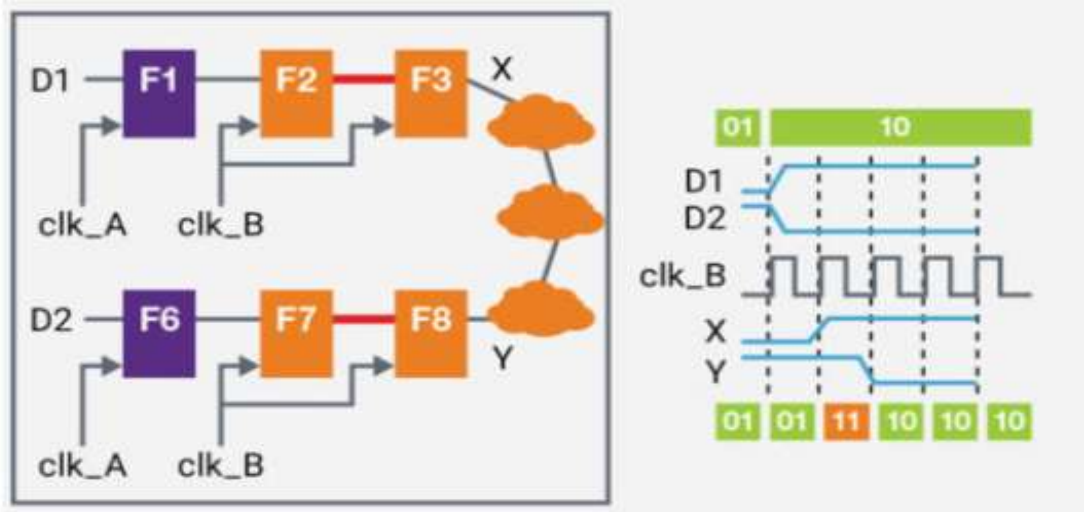
Reset Domain Crossing



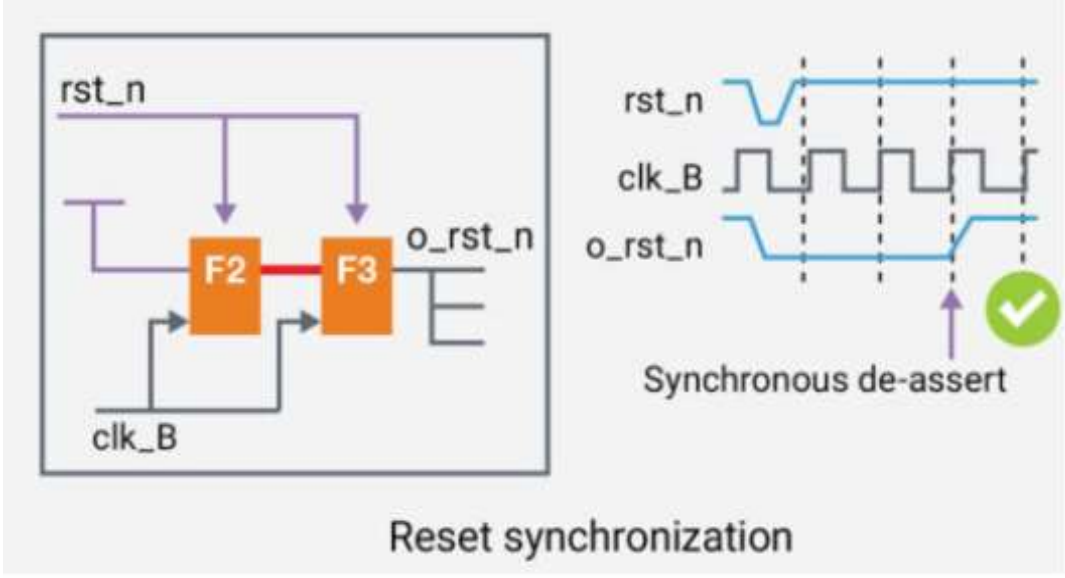
Data loss in fast to slow transfer



Improper data enable sequence



Re-convergence of synced signals



Reset synchronization

Advanced Digital Design

Challenges in Traditional Verification Methods

- Meta-stability arising from jitter between asynchronous clock domains can result in functional failures if the appropriate clock synchronizers are not present.
- What's more, there are more complex paths and scenarios that can be buried deep into the design, such as the re-convergence problem where logic that combines multiple synchronized paths together can result in a timing mismatch due to synchronizer uncertainty.
- It is a class of bug that typically cannot be worked around in the final silicon, so getting it wrong can cost you a chip re-spin. An expensive mistake!

Challenges in Traditional Verification Methods

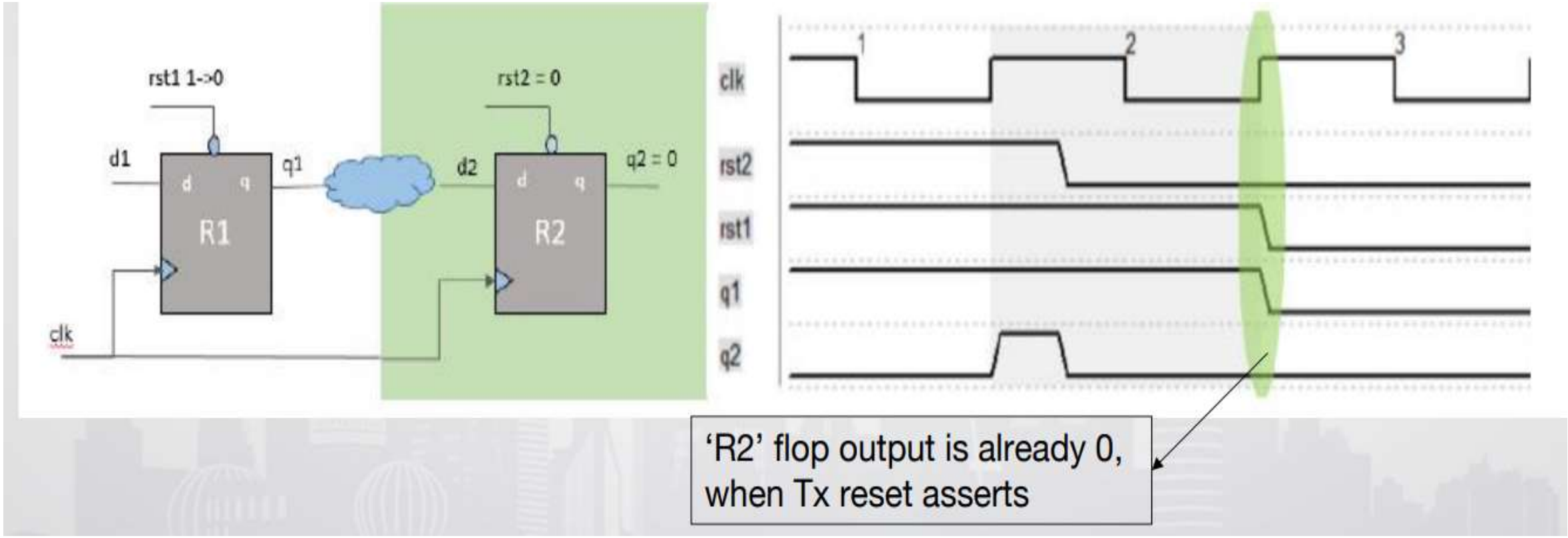
- Traditionally, SoC verification teams have used simulation of register transfer language (RTL) models to verify reset behavior by running bootup sequences or dedicated functional tests.
- This has numerous gaps, starting with the fact that not all resets are easily controllable from the chip inputs, especially software/internally generated reset. RTL tends to model very little sub-cycle timing, so glitches or metastability that could cause problems are unlikely to appear.

Challenges in Traditional Verification Methods

- The bottom line is that simulation is the wrong approach for RDC (or CDC) verification. Asynchronous resets and clocks can vary continually, so any attempt to model that behavior with a series of discrete timing variations is by its nature incomplete.
- Repeating the complete regression suite many times with different reset alignments adds a great deal of overhead to the verification process.
- Finally, no amount of simulation can ever guarantee that all RDC bugs will be found before silicon. Only powerful static analysis can provide exhaustive verification with quick turnaround.

Techniques to Address RDC issues

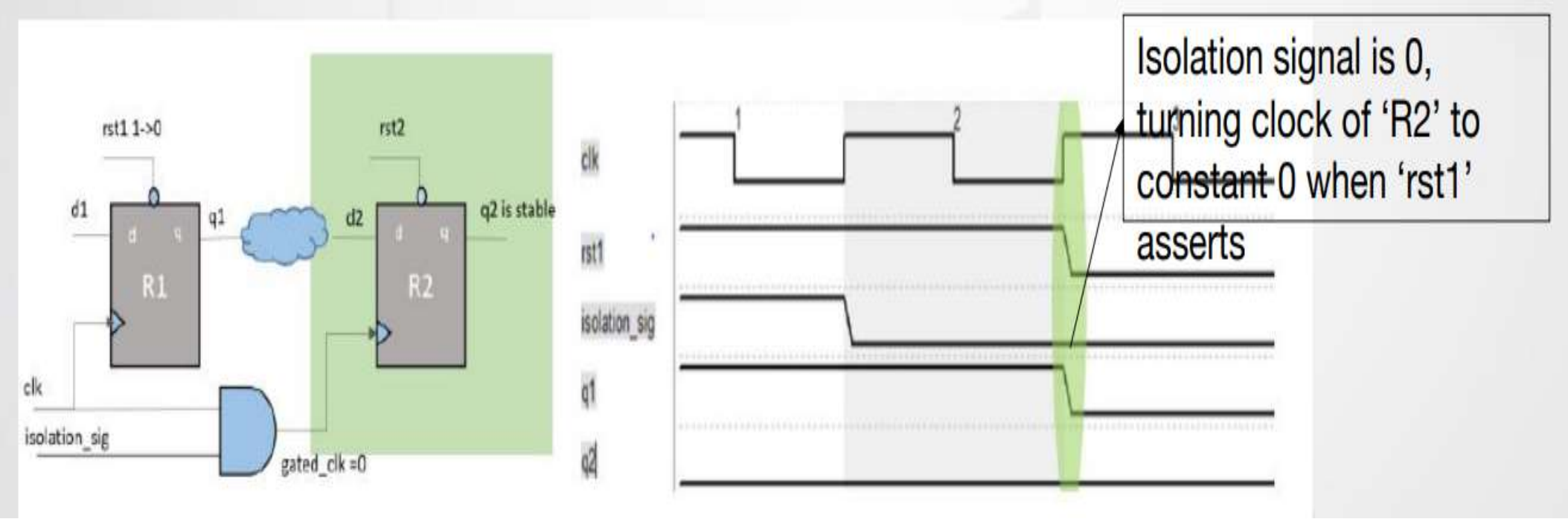
- **Reset Sequencing : -**
  - Async-reset on Rx flop always asserts before async-reset on Tx flop
  - Rx flop already in reset state, so any change on Rx D-pin will not cause metastability.



Advanced Digital Design

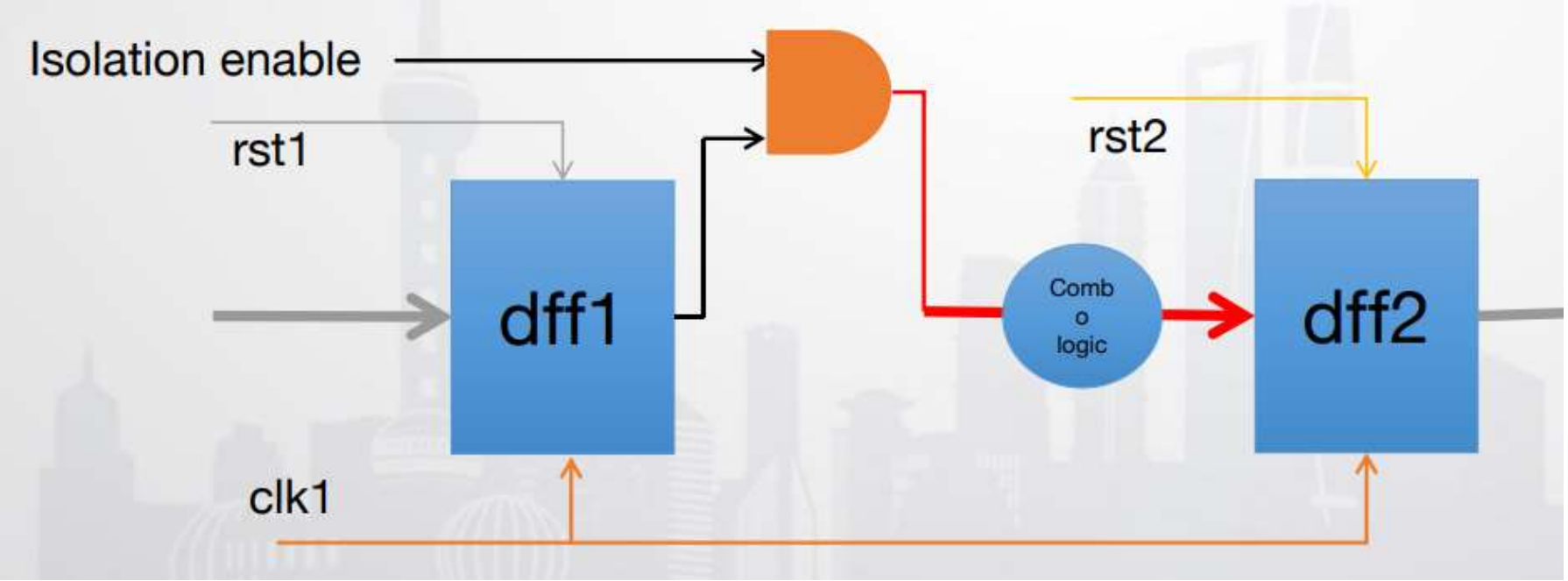
Techniques to Address RDC issues

- **Clock-gate isolation**
  - Turn off clock of Rx flop before Tx reset asserts
  - If clock is off, then any change on Rx D-pin will not cause metastability

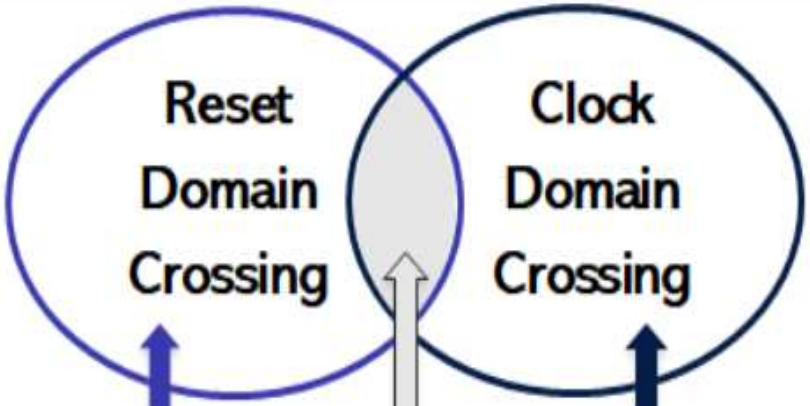


Techniques to Address RDC issues

- **Data Isolation**
  - Isolation signal from a reset controller isolates the output of the first flop when its reset is asserted. There is a handshake protocol between the enables and the corresponding resets.
  - There is a mapping between isolation enables and resets



RDC Sign-Off has Critical Differences from CDC Sign-Off



a. RDC errors - can occur in same clock domain

b. Analysis Scope - Global

c. RDC-specific analysis required to identify all RDC issues with low noise

d. Mean Time Between Failures - High

Sign-off uses static analysis

Can cause metastability, glitches & functional correlation errors

a. CDC errors - occur across clock domains

b. Analysis Scope - Local at CDC interfaces

c. CDC-specific analysis required to identify all CDC issues with low noise

d. Mean Time Between Failures - Low

Reset domain crossing errors can occur within the same clock domain

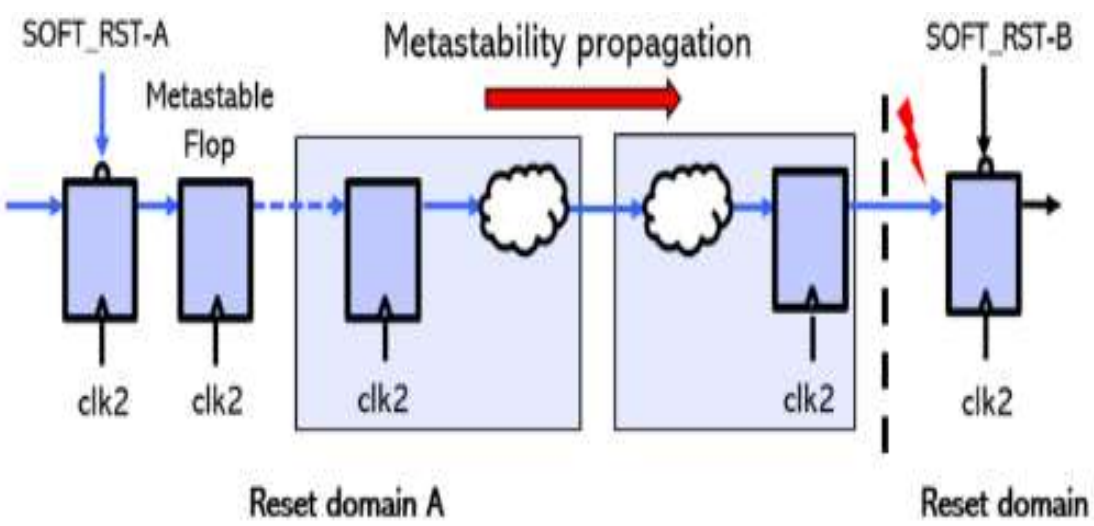
CDC	RDC
Design problems only occur between **asynchronous** clock domains.	Design problems can occur between **synchronous & asynchronous** clock domains.

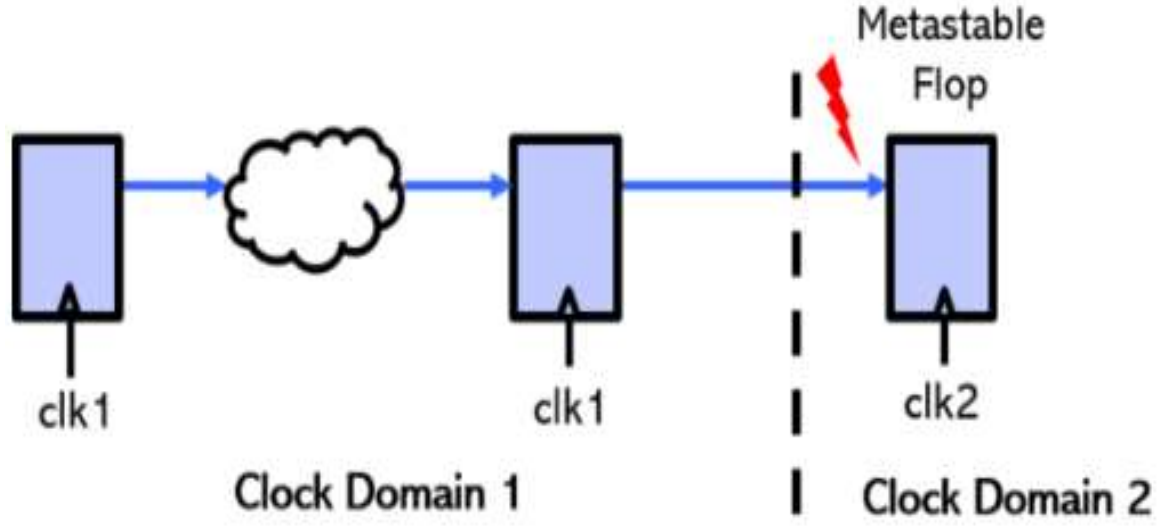
In a sequential design, if the source flop makes an asynchronous transition to a reset state, it can cause a transition inside the setup and hold window of the destination flop and cause a metastability issue.

Advanced Digital Design

Reset Domain Crossing needs global analysis. CDC has local analysis

- The scope of analysis required to identify design issues differs for RDC and CDC.
- RDC metastability analysis & debug is GLOBAL. The RDC analysis path can go from source in one reset domain through an intermediate set of flops and modules to destination flop in a different reset domain.
- CDC metastability analysis & debug is LOCAL. The CDC analysis path goes from source flop in one clock domain to destination flop in a different clock domain.





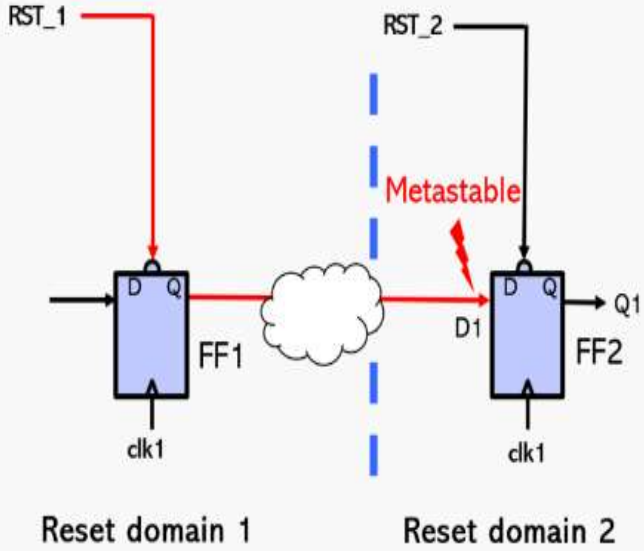
Advanced Digital Design

Reset domain crossing requires RDC-specific design principles & analysis

- RDC domain-specific analyses — both structural and functional — are needed to identify all RDC design issues with low noise.
- The example below shows how structural analysis alone can show a metastability error, while advanced RDC functional analysis of the design can recognize design protection such as a blocking signal.
- Without this advanced RDC functional analysis, the number of false errors reported goes up tremendously. Engineers must review the false errors in the reported violations, so a high-noise violation report requires a lot more engineering time and effort to reach sign-off.
- Additionally, if there are too many false positives reported, engineers can end up classifying actual errors as waivers – resulting in missed design bugs.

Structural analysis only

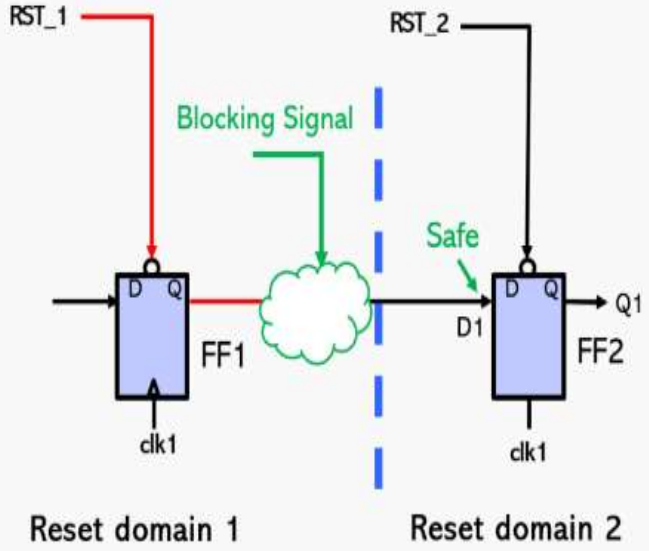
False positive error



Analysis shows the assertion of RST_1 creates an untimed path from FF1 to FF2, which can cause metastability when RST_1 asserts and RST_2 to FF2 is de-asserted.

Advanced RDC functional analysis

Identifies protection



Running RDC-specific functional analysis of the combinational logic shows there is a blocking signal. The reset domain crossing path is proven safe, and no false error is reported.

RDC mean time between failures (MTBF) is higher than CDC

- One practical challenge faced by design teams is that chip failures due to RDC can be missed because the MTBF is much higher for RDC than for CDC.
- For both RDC and CDC, the MTBF is correlated with the source flop change rate, the destination flop capture rate, and the setup-hold time window.
- The source flop change rate. Clocks toggle continuously; thus, if a CDC flop changes value once every 10 clock cycles, a clock domain crossing for a 1 GHz frequency clock can occur 100 million times per second. In contrast, resets only assert based on specific events.
- The destination flop capture rate, which is defined as the number of possible destination flop clock edges that can capture a transition from the source flop. The probability that a signal transition falls within the setup and hold time window of the capture flop is much higher for CDC as compared to RDC.

References

- This Lecture Slides and Lecture notes

THANK YOU

Sudeendra kumar K

Department of Electronics and Communication Engineering

sudeendrakumark@pes.edu