Asynchronous & Synchronous Reset Design Techniques - Part Deux

Clifford E. Cummings
Sunburst Design, Inc.
cliffc@sunburst-design.com
www.sunburst-design.com

Don Mills
LCDM Engineering
mills@lcdm-eng.com
www.lcdm-eng.com

Steve Golson
Trilobyte Systems
sgolson@trilobyte.com
www.trilobyte.com

Agenda

"Resets Update & FAQ"

- Flip-flop coding styles
- Synchronous resets
- Asynchronous resets
- Design For Test (DFT) considerations ← **Reference slides only**
- Reset-buffer tree
  - Distributed synchronous reset flip-flops
  - Distributed asynchronous reset synchronizers
- Synthesis issues with reset nets
- Multi-clock resets

Bad Multi-Flip-Flop Coding Style

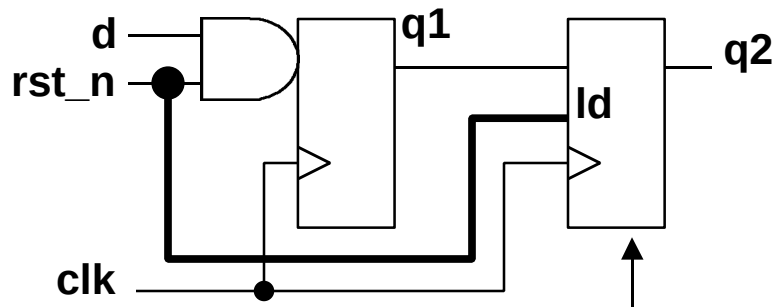
- Problem: dissimilar flip-flops in the same always block

```
module badFFstyle (q2, d, clk, rst_n);
  output q2;
  input  d, clk, rst_n;
  reg    q2, q1;

  always @(posedge clk)
    if (!rst_n) q1 <= 1'b0;
    else begin
      q1 <= d;
      q2 <= q1;
    end
endmodule
```

q2 is only loaded if `rst_n` is high

BAD PARTITIONING Style creates EXTRA LOGIC



`rst_n` becomes a "load-data" signal

VHDL model included in the paper

Good Multi-Flip-Flop Coding Style

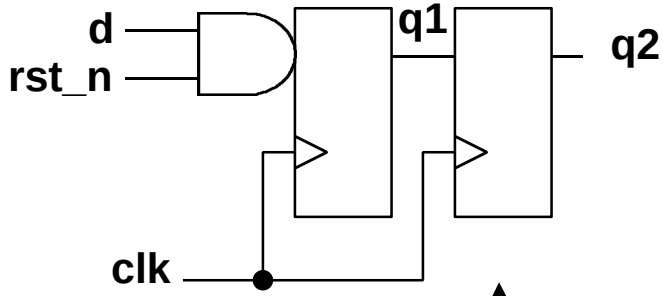
- Solution: put dissimilar flip-flops in separate always blocks

```
module goodFFstyle (q2, d, clk, rst_n);
  output q2;
  input  d, clk, rst_n;
  reg    q2, q1;

  always @(posedge clk)
    if (!rst_n) q1 <= 1'b0;
    else        q1 <= d;

  always @(posedge clk)
    q2 <= q1;
endmodule
```

Good partitioning - no extra logic



No reset on the follower flip-flop

q2 is loaded on every posedge clk

Note: To model sequential logic use nonblocking assignments

VHDL model included in the paper

Modeling Synchronous Resets

- Synchonous reset: **`rst_n`** is not in the sensitivity list

```
module ctr8sr ( q, co, d, ld, rst_n, clk);
  output [7:0] q;
  output       co;
  input  [7:0] d;
  input        ld, rst_n, clk;
  reg    [7:0] q;
  reg          co;

  always @(posedge clk)
    if      (!rst_n) {co,q} <= 9'b0;     // sync reset
    else if (ld)     {co,q} <= d;        // sync load
    else             {co,q} <= q + 1'b1; // sync increment
endmodule
```

rst_n not in the sensitivity list

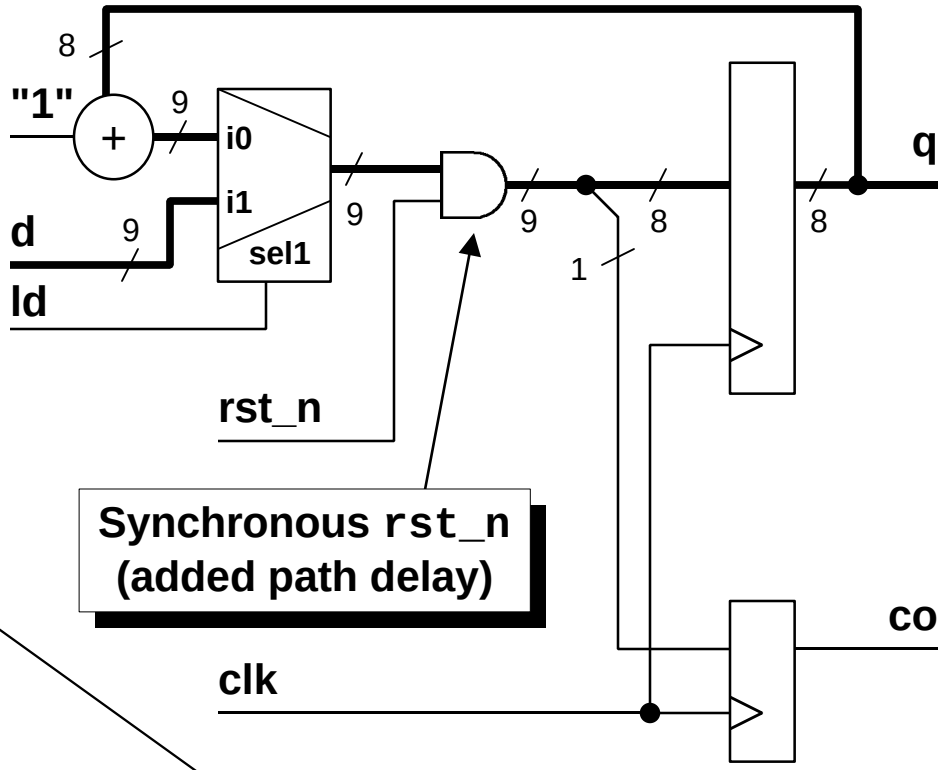
VHDL model included in the paper

VHDL versions would have required too many slides to show the same models ...

... or a microscopic font

Synchronous Resets

Synthesis Results #1



`clk` is in the sensitivity list

Only `clk` can trigger the `always` block

Only `clk` can cause the outputs to change

`rst_n` & `ld` are not in the sensitivity list (synchronous to `clk`)

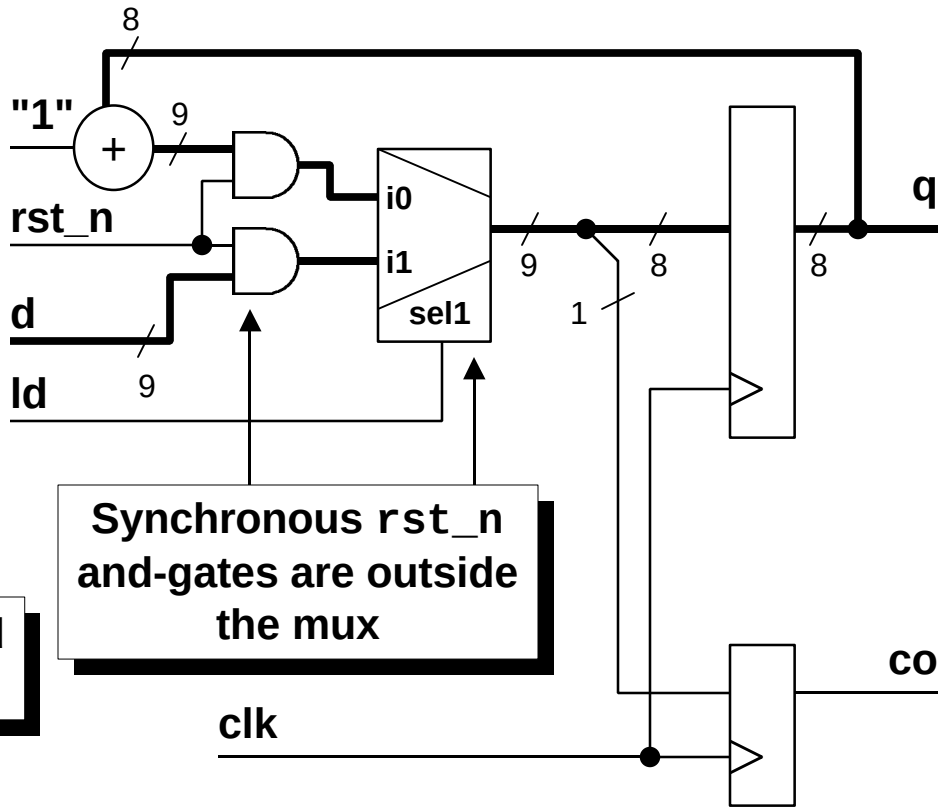
Synchronous `rst_n` (added path delay)

```
always @(posedge clk)
  if      (!rst_n) {co,q} <= 9'b0;
  else if (ld)     {co,q} <= d;
  else             {co,q} <= q + 1'b1;
```

Synchronous Resets

Synthesis Results #2

Synthesis tools could locate synchronous resets signals poorly



Synchronous `rst_n` and-gates are outside the mux

Add this directive to avoid badly placed synchronous reset signals

```
// synopsys sync_set_reset "rst_n"
always @(posedge clk)
  if      (!rst_n) {co,q} <= 9'b0;
  else if (ld)     {co,q} <= d;
  else             {co,q} <= q + 1'b1;
```

Modeling Asynchronous Resets

- Asynchonous reset: **`rst_n`** is in the sensitivity list

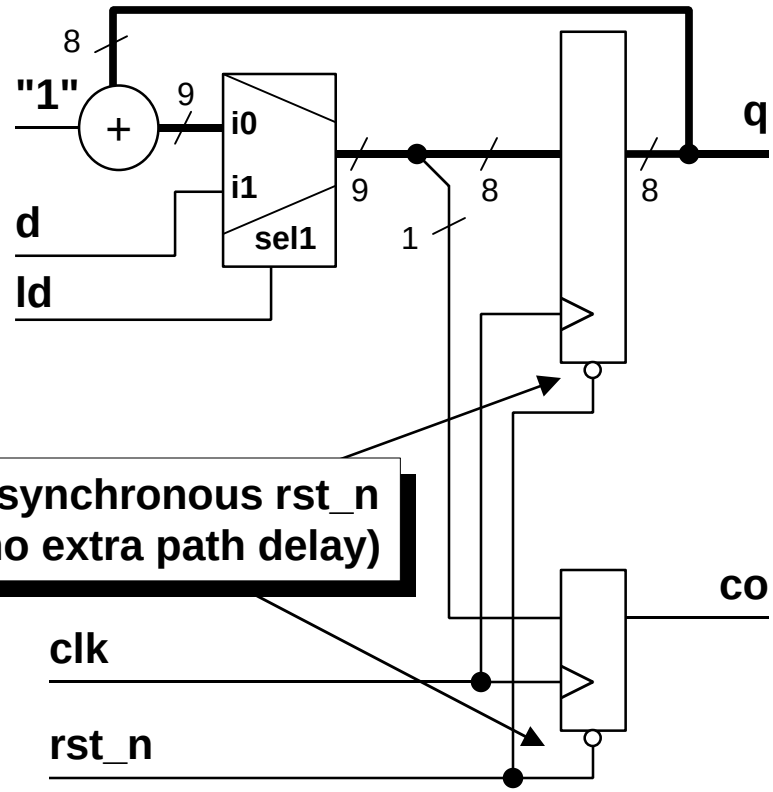
```
module ctr8ar ( q, co, d, ld, rst_n, clk);
  output [7:0] q;
  output       co;
  input  [7:0] d;
  input        ld, rst_n, clk;
  reg    [7:0] q;
  reg          co;

  always @(posedge clk or negedge rst_n)
    if      (!rst_n) {co,q} <= 9'b0;     // async reset
    else if (ld)     {co,q} <= d;        // sync load
    else             {co,q} <= q + 1'b1; // sync increment
endmodule
```

rst_n is in the sensitivity list

VHDL model included in the paper

Synthesizing Asynchronous Resets



Only `clk` *or* `rst_n` can cause the outputs to change

`clk` *or* `rst_n` can trigger the `always` block

Asynchronous `rst_n` (no extra path delay)

asynchronous `rst_n`, in the sensitivity list

synchronous `ld` not in the sensitivity list

```
always @(posedge clk or negedge rst_n)
  if      (!rst_n) {co,q} <= 9'b0;
  else if (ld)     {co,q} <= d;
  else             {co,q} <= q + 1'b1;
```

Synchronous Resets

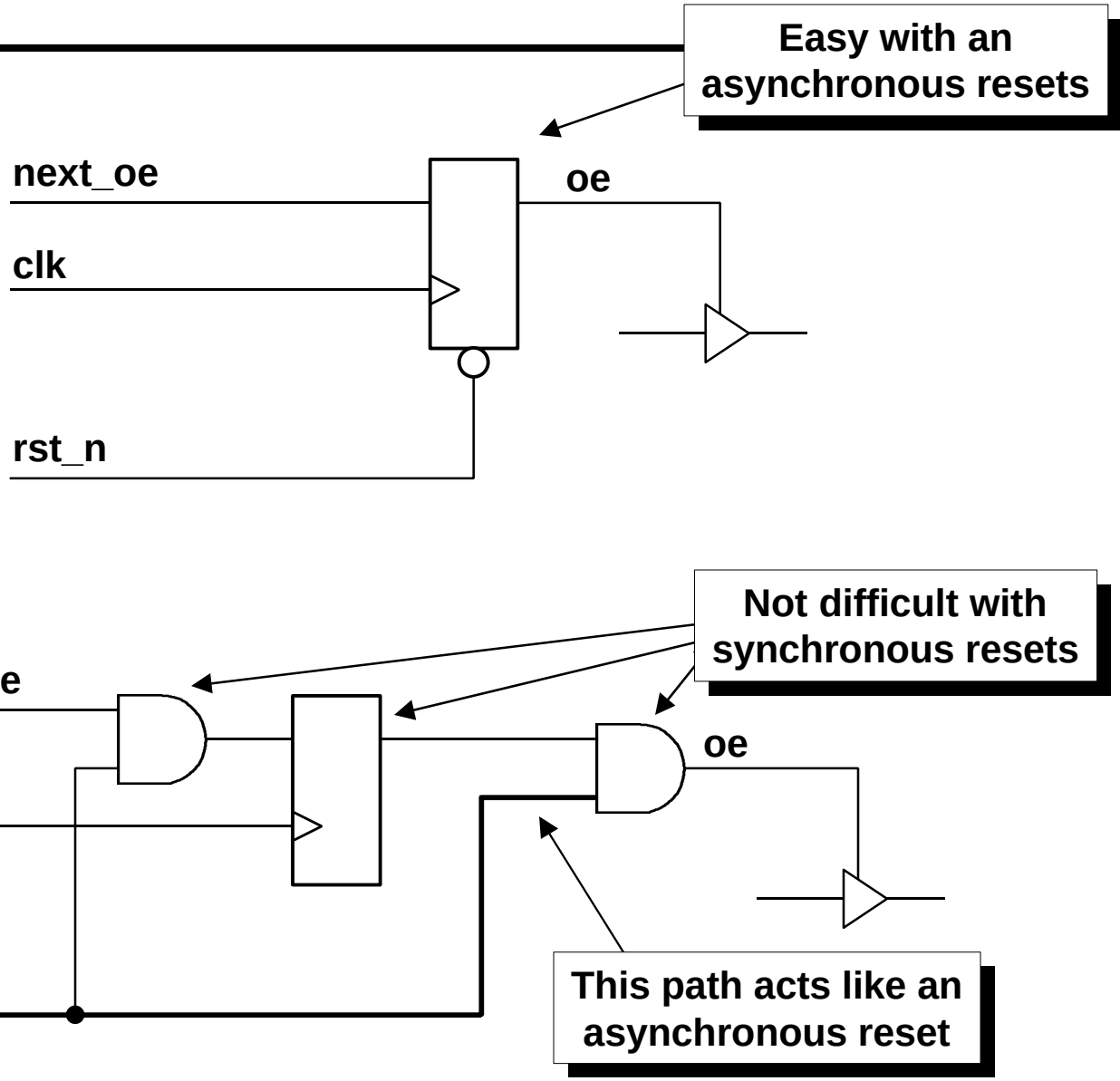
Advantages & Disadvantages

- Advantages
  - Easier to work with cycle based simulators (according to the RMM)
  - Typically recommended for DFT design
  - Glitch filtering from reset combinational logic (to make up for poor design practices)
  - Glitch filtering if reset is in a mission-critical application

- Disadvantages
  - May not be able to come out of Unknown-X during simulation
  - May add delay to data path
  - Power-up reseting of a tri-state bus

... but there is a solution!

Tri-State Enable Drivers



Asynchronous Resets

Advantages & Disadvantages

- Advantages
  - Reset is immediate
  - No problem related to Unknown-X-propagation in design simulation
  - Does not interfere or add extra delay to the data path
  - Very easy to implement on the front end

- Disadvantages
  - Asynchronous signal - can go metastable on release
  - Noisy reset line could cause unwanted resets
  - DFT requires extra steps
  - Requires reset buffer tree manipulation

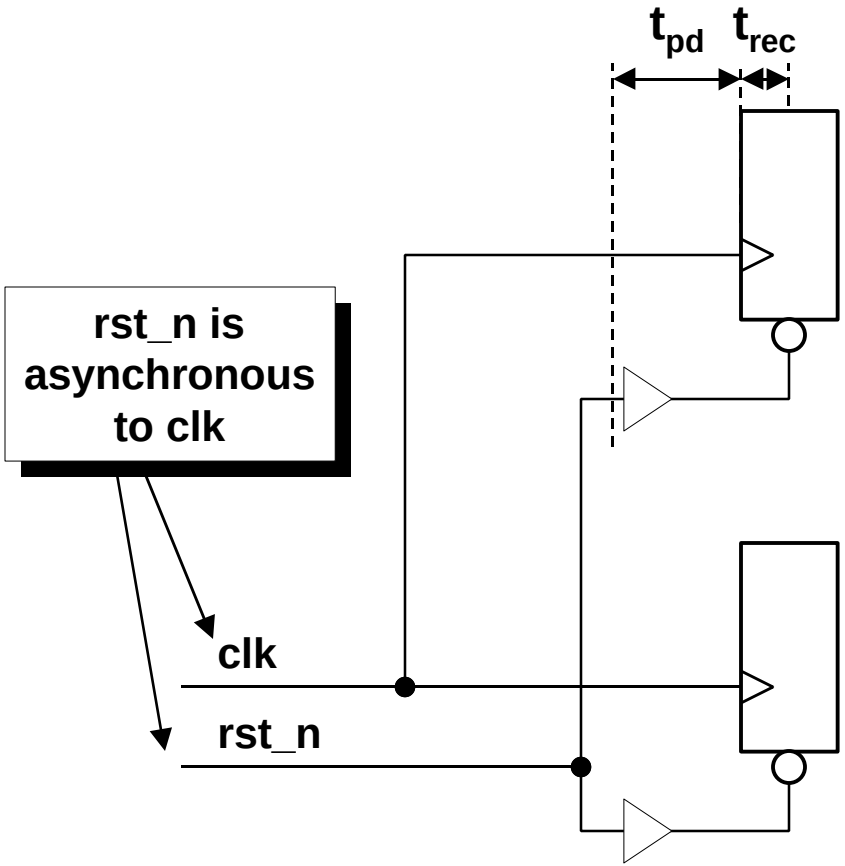
... but there is a solution!

... but this can be filtered

... but this can be fixed

Asynchronous Reset Problem

- Problem: *Asynchronous* reset removal
  - Will reset removal meet recovery time specification?

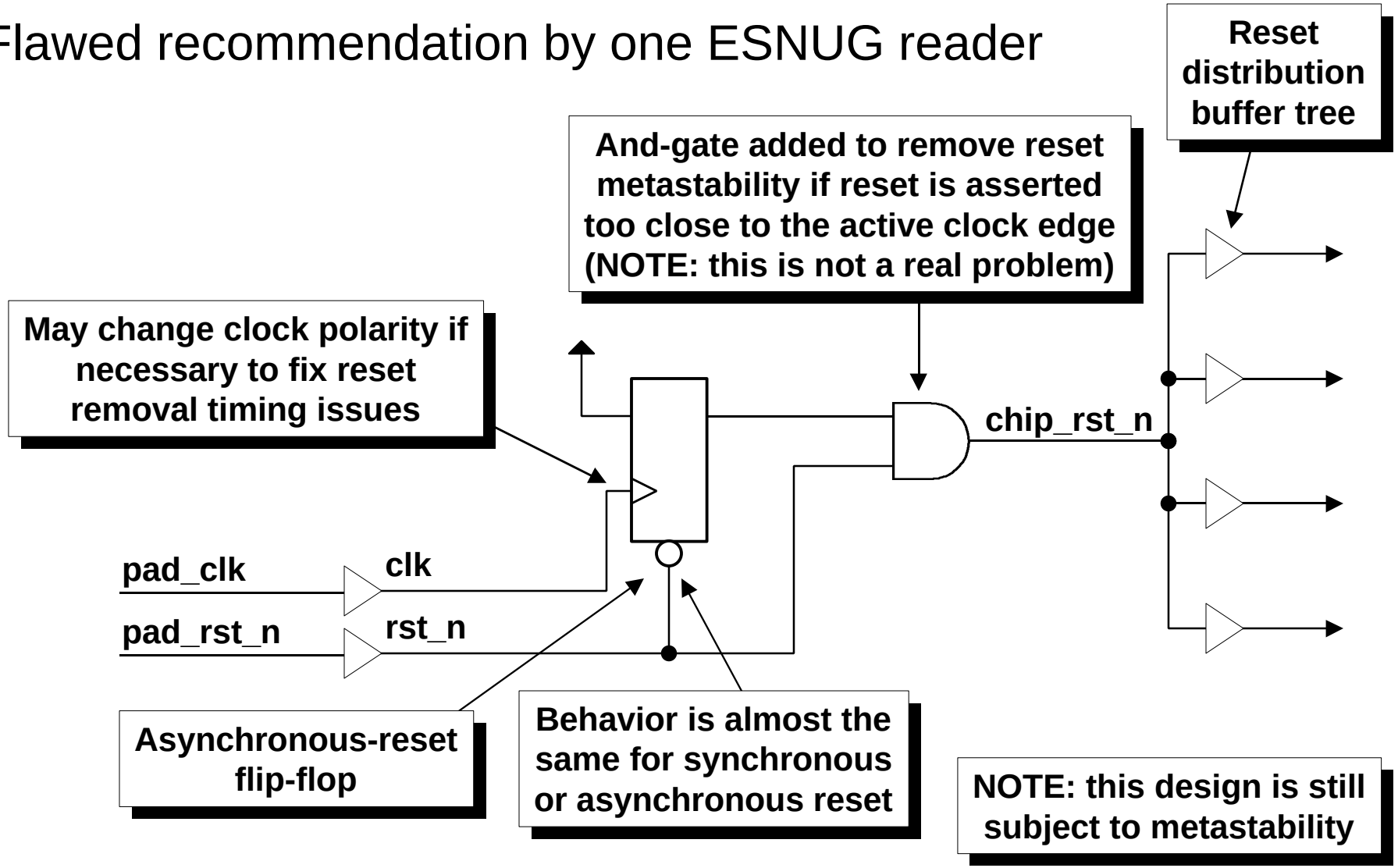


Potential problem: flip-flop could go metastable on reset-release if reset removal violates clock set/hold time

Flawed Reset Synchronizer

(ESNUG 409 Item 11)

- Flawed recommendation by one ESNUG reader



Flawed Reset RTL Thinking

(ESNUG 409 - Also Item 11)

- Flawed RTL coding thoughts by another ESNUG reader

"One reason not to use asynchronous resets is that Verilog cannot model them without a race condition. Typical async reset flop:

```
always @ (posedge ck or negedge rst) begin
  if (!rst) q <= 0;
  else      q <= d;
end
```

A real hardware reset recovery violation!

**"What happens when rst is DEasserted at the same time as ck is asserted?
Either ck goes high first, and rst is still low, so q gets zero.
Or rst goes high first, and when ck fires, q gets d.**

Uh-oh. Do you get your new d or not?"

This is not a Verilog race condition, this is a hardware race condition!

Even with VHDL delta-times this same "race" condition exists

Flawed Reset RTL Thinking

(ESNUG 409 - Also Item 11 - continued)

- Upon further investigation, the real issue was poor testbench practices

```
always @ (posedge ck or negedge rst) begin
  if (!rst) q <= 0;
  else      q <= d;
end
```

Users remove reset on the posedge clk in testbenches (What?? No Way!!)

Non-deterministic behavior between RTL and gates (and between different vendors)

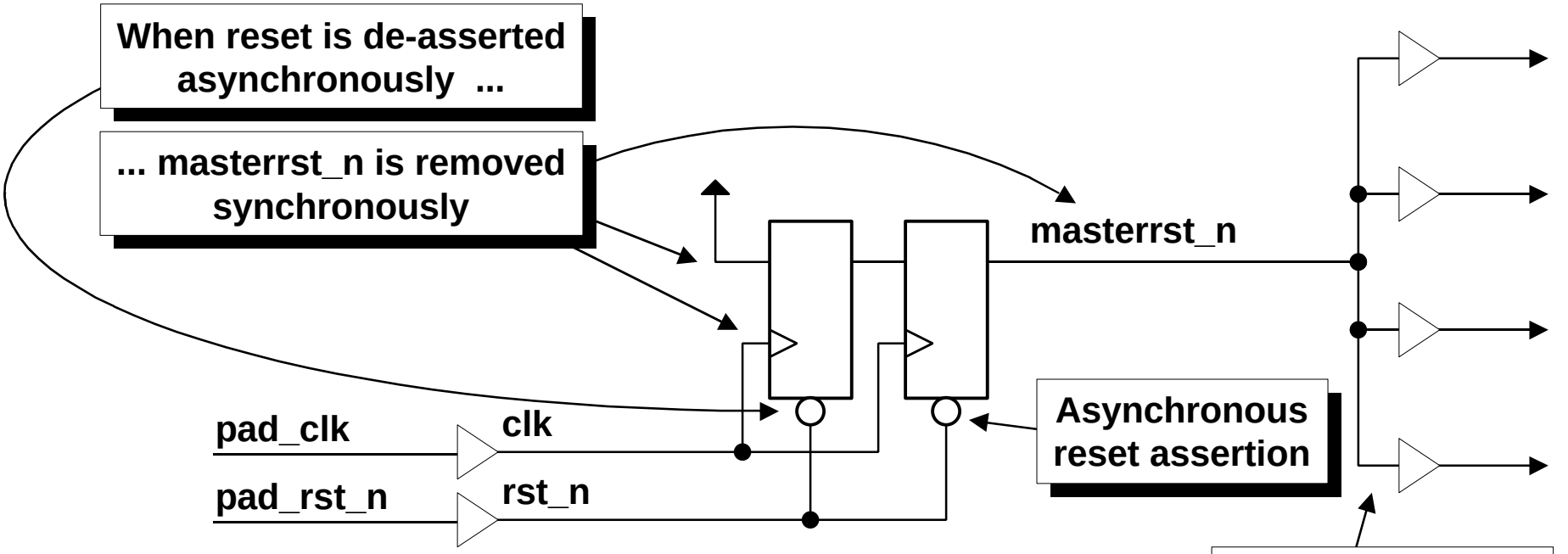
Users blame EDA vendors when RTL differs from gates simulations

EDA vendors cannot fix real hardware race conditions!

Don't be a stupid user! Don't try removing reset on a posedge clk in the testbench! It doesn't work in simulations because it doesn't work in real hardware!!

Reset Synchronizer

- Advantage: *Asynchronous* reset assertion
- Advantage: *Synchronous* reset removal

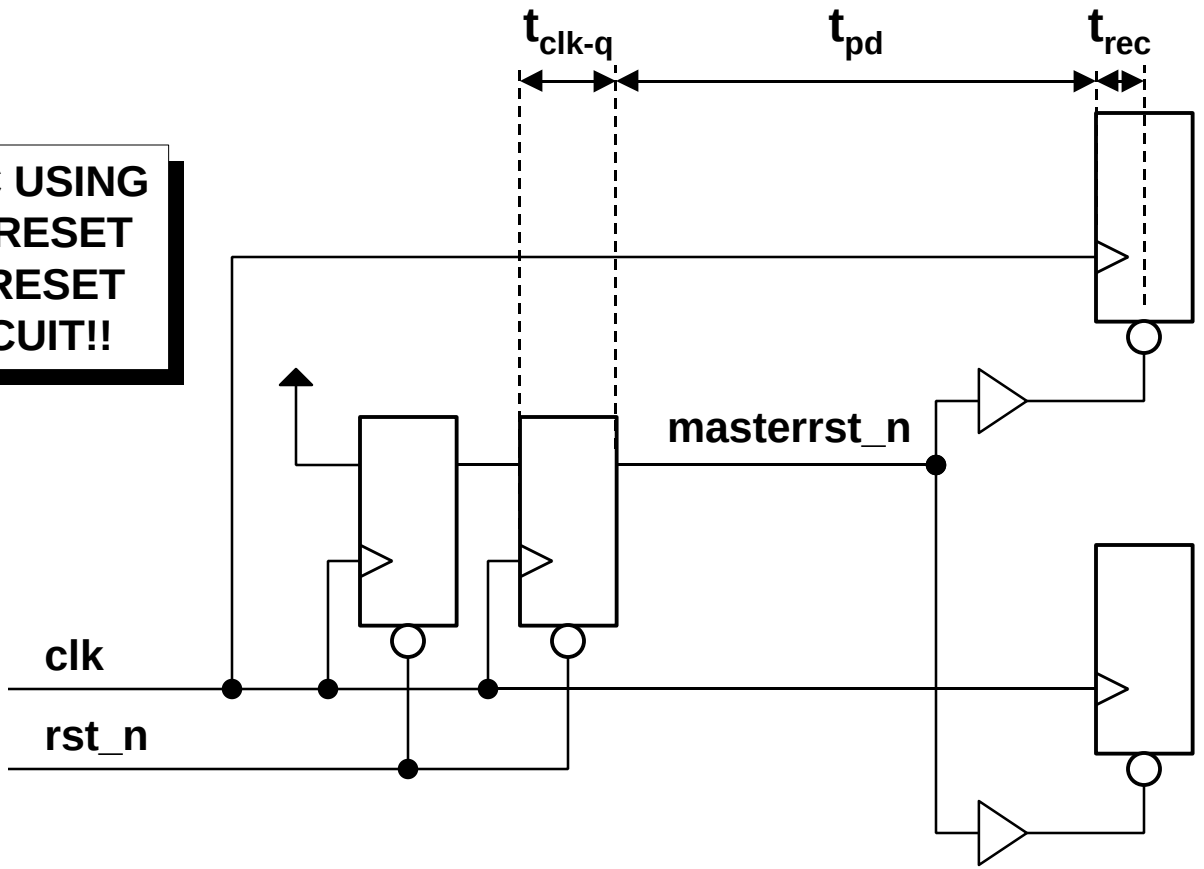


Guideline: EVERY ASIC USING AN ASYNCHRONOUS RESET SHOULD INCLUDE A RESET SYNCHRONIZER CIRCUIT!!

Synchronous Reset Removal Solution

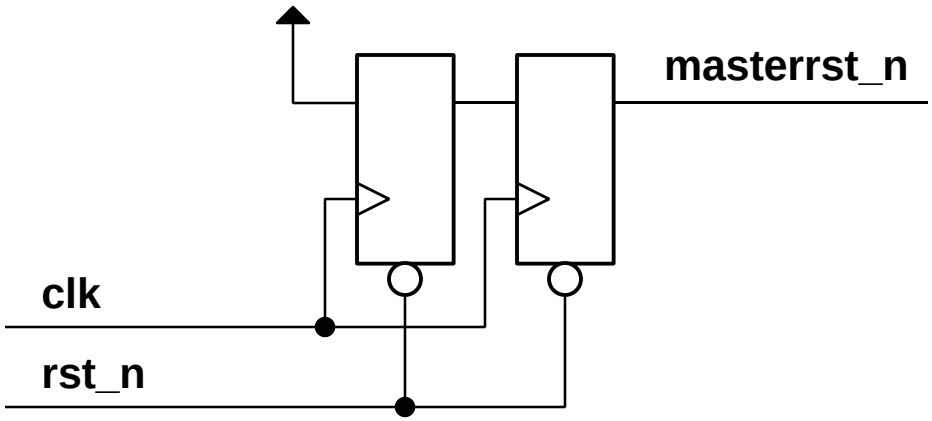
- Advantage: *Synchronous* reset removal
  - Predictable reset removal to meet recovery time specification!

Guideline: EVERY ASIC USING AN ASYNCHRONOUS RESET SHOULD INCLUDE A RESET SYNCHRONIZER CIRCUIT!!



Reset Synchronizer

Verilog RTL Code



Guideline: EVERY ASIC USING AN ASYNCHRONOUS RESET SHOULD INCLUDE A RESET SYNCHRONIZER CIRCUIT!!

```
module reset_synchronizer (masterrst_n, clk, rst_n);
  output masterrst_n;
  input  clk, rst_n;
  reg    rst_n, rff1;

  always @(posedge clk or negedge rst_n)
    if (!rst_n) {masterrst_n,rff1} <= 2'b0;
    else        {masterrst_n,rff1} <= {rff1,1'b1};
endmodule
```

Asynchronous reset assertion (on negedge rst_n)

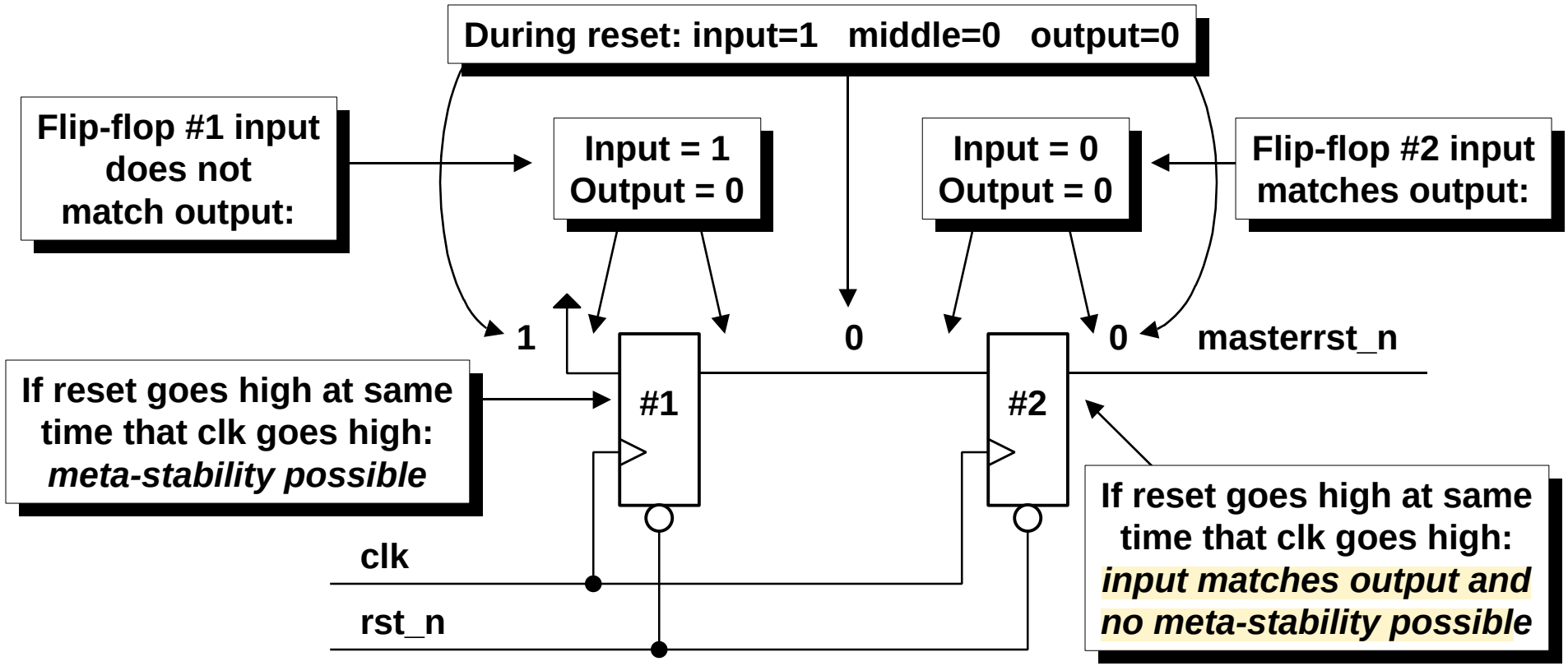
Synchronous reset removal (on posedge clk)

Concatenation makes for efficient coding of the reset synchronizer

Synchronizer Metastability??

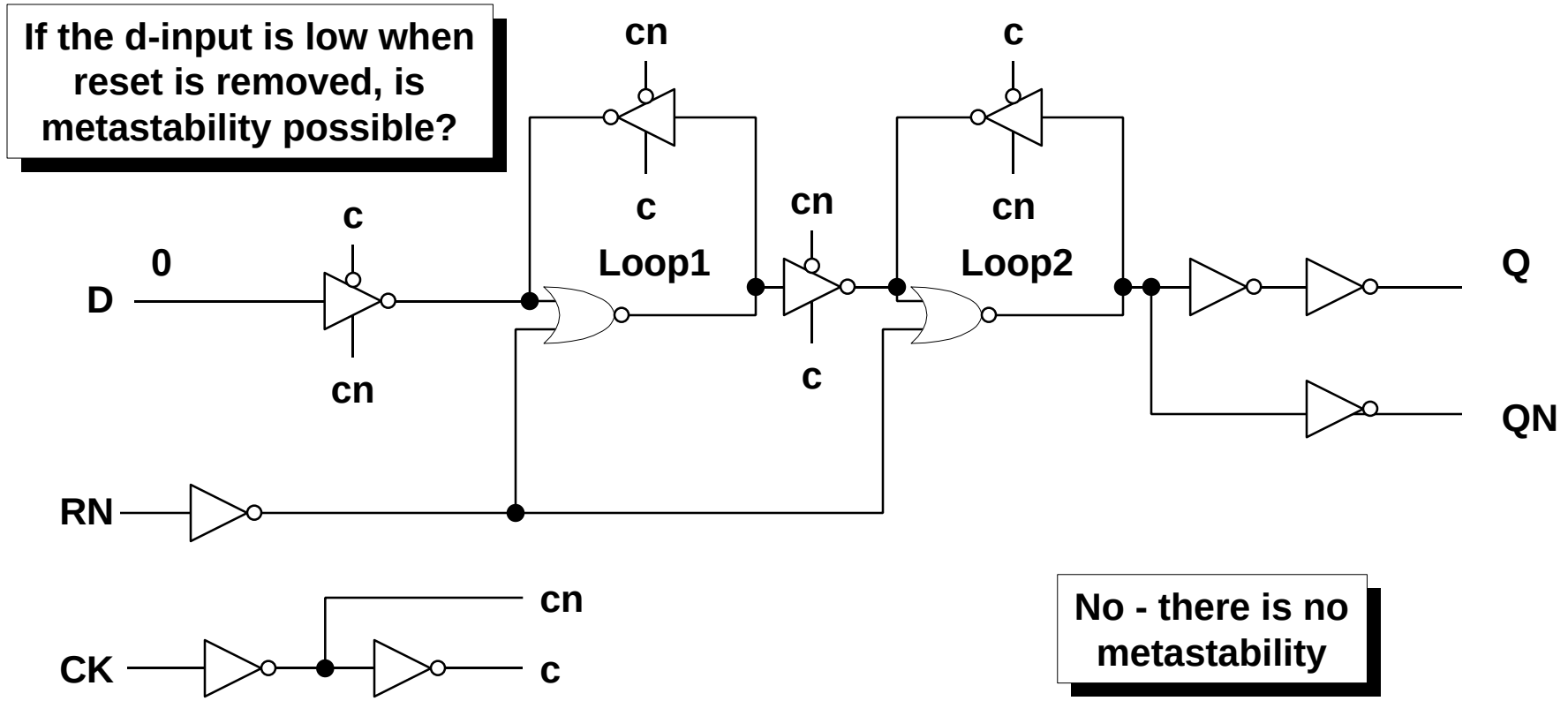
(Very Frequently Asked Question)

- FAQ: Can the 2nd flip-flop of the synchronizer go metastable if rst_n is removed too close to a posedge clk (violating reset recovery time)?
- Answer: **NO !!**



Reset Removal Metastability??

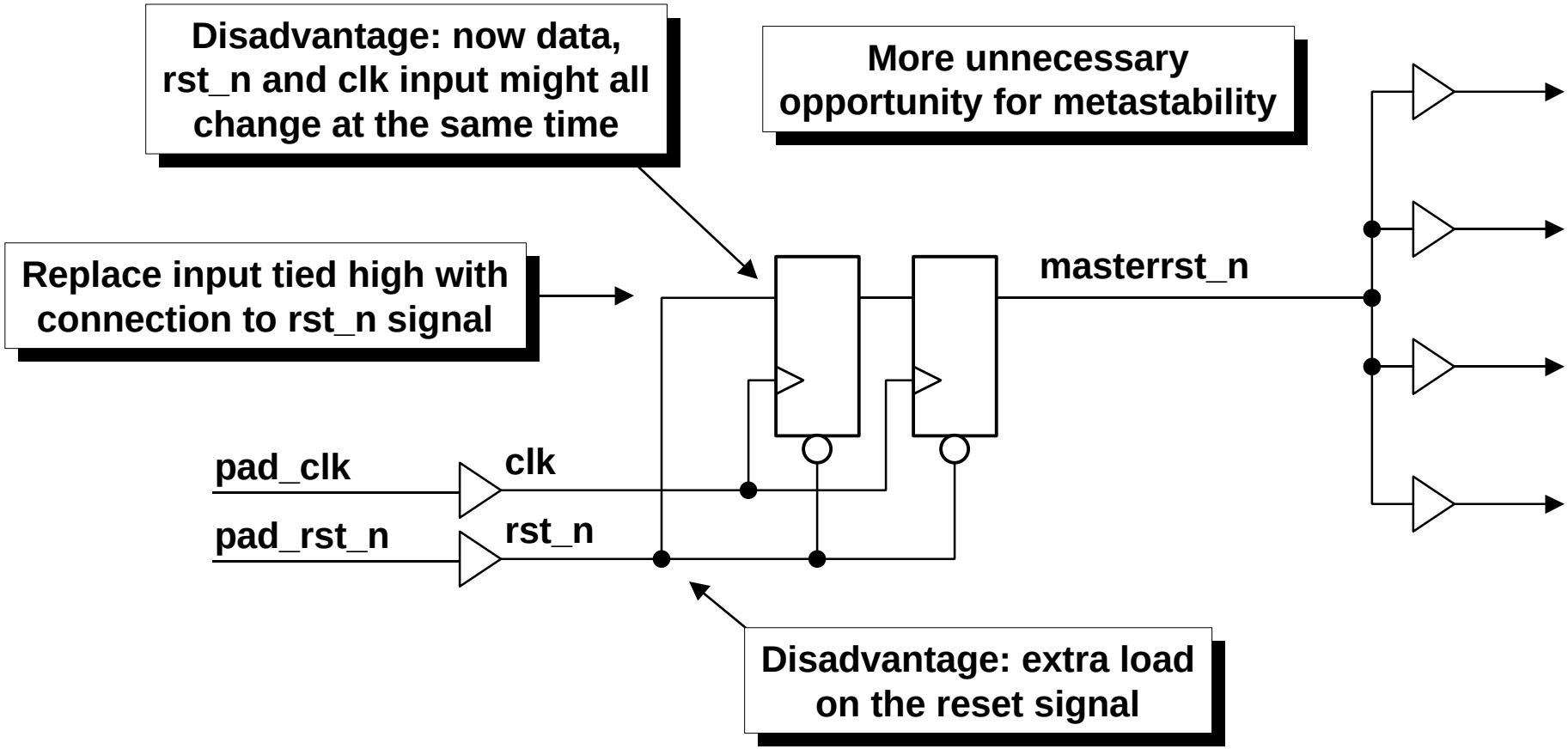
- Example flip-flop implementation



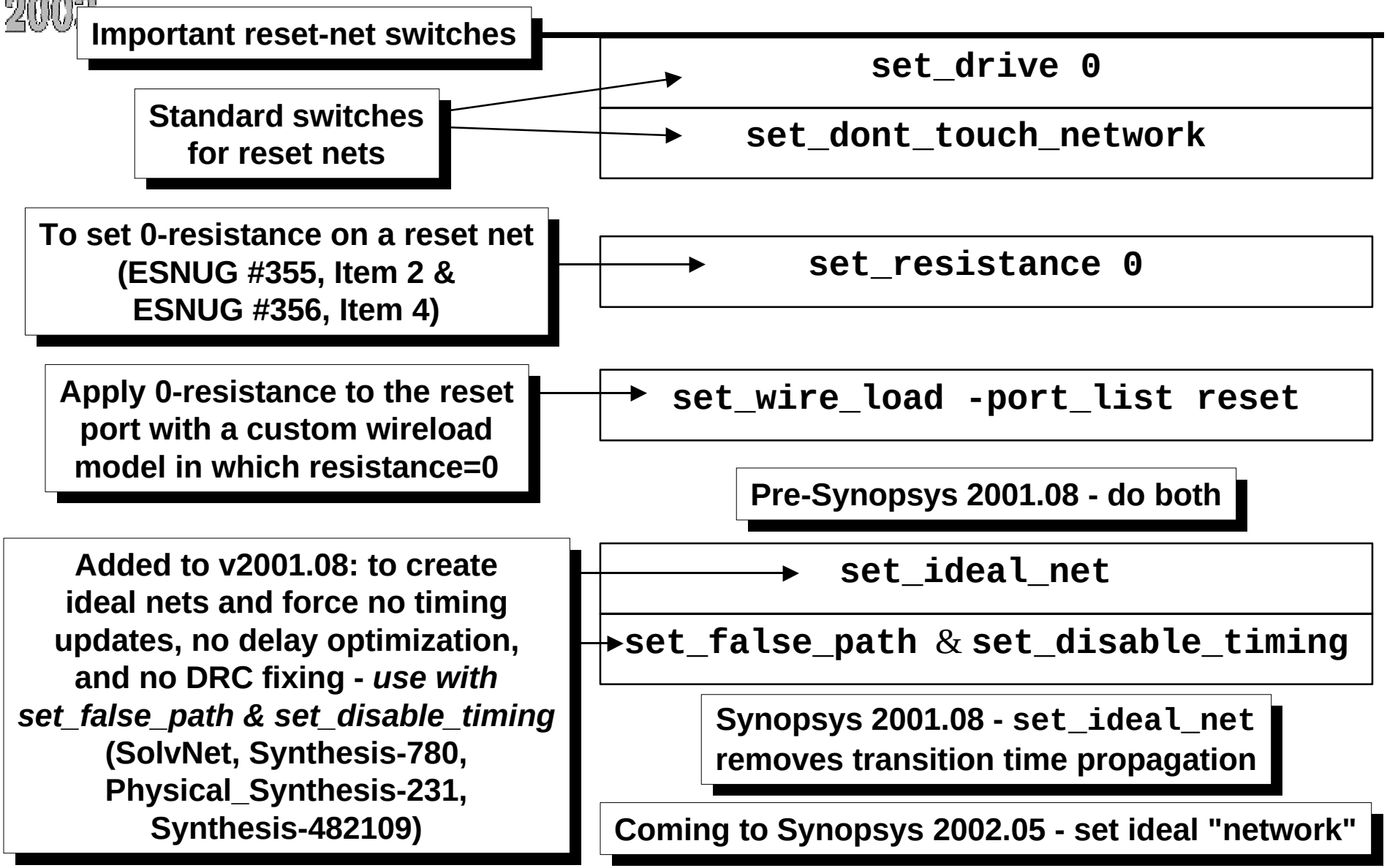
Different Reset Synchronizer

(ESNUG 409 - Also Item 11)

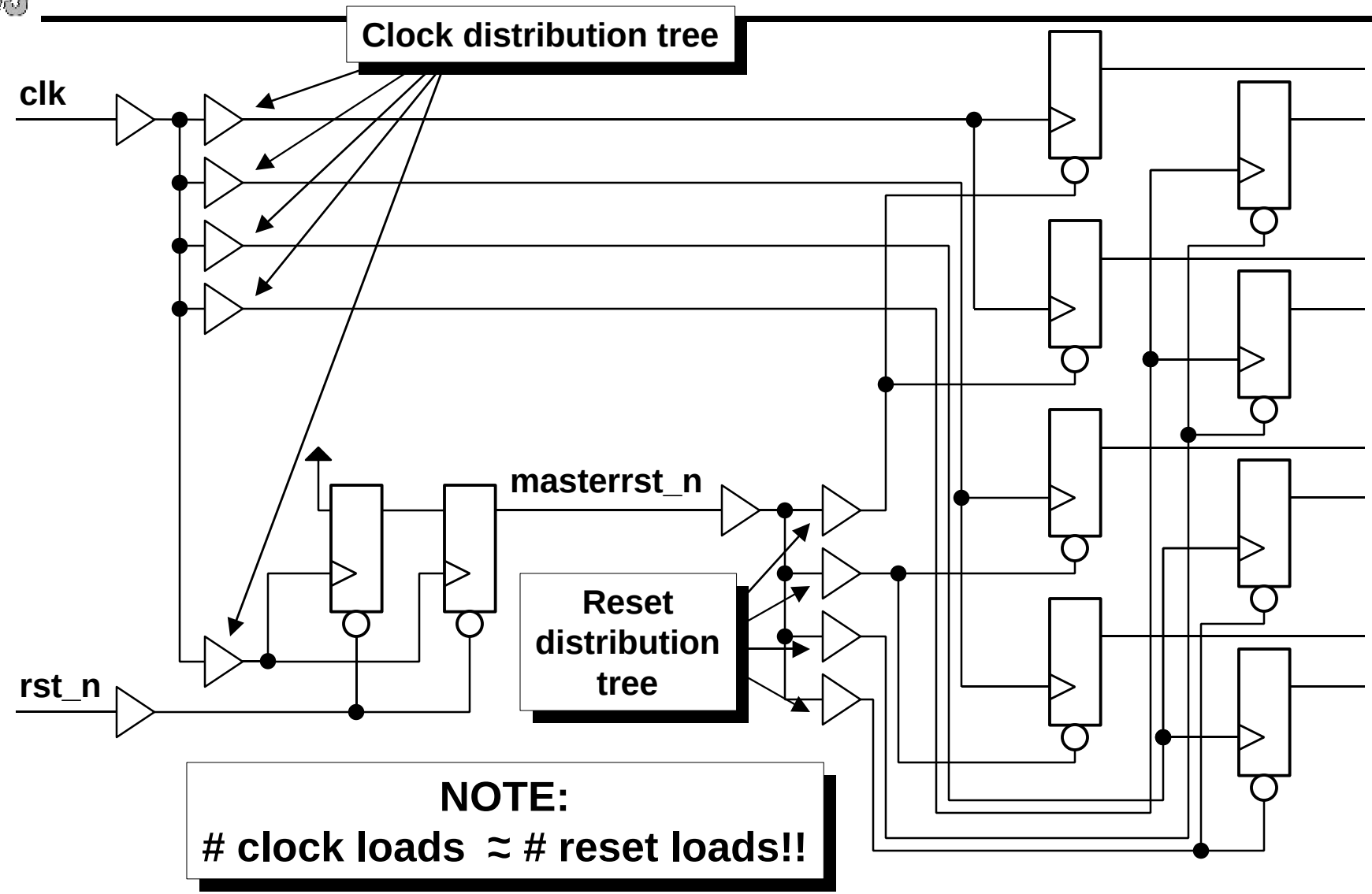
- Tie the synchronizer data input to the reset input signal



Synopsys Reset Switches

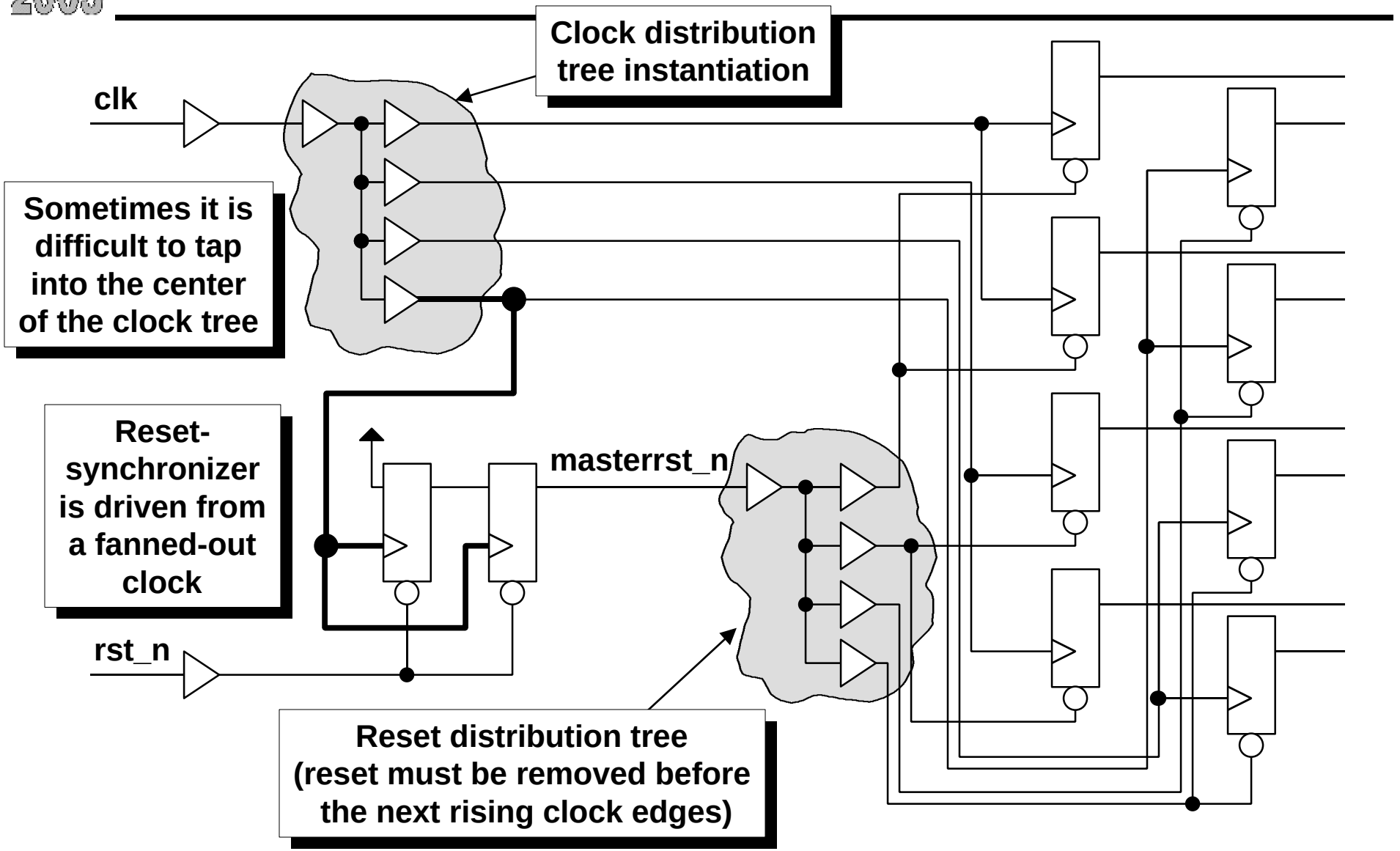


Clock & Reset Loading



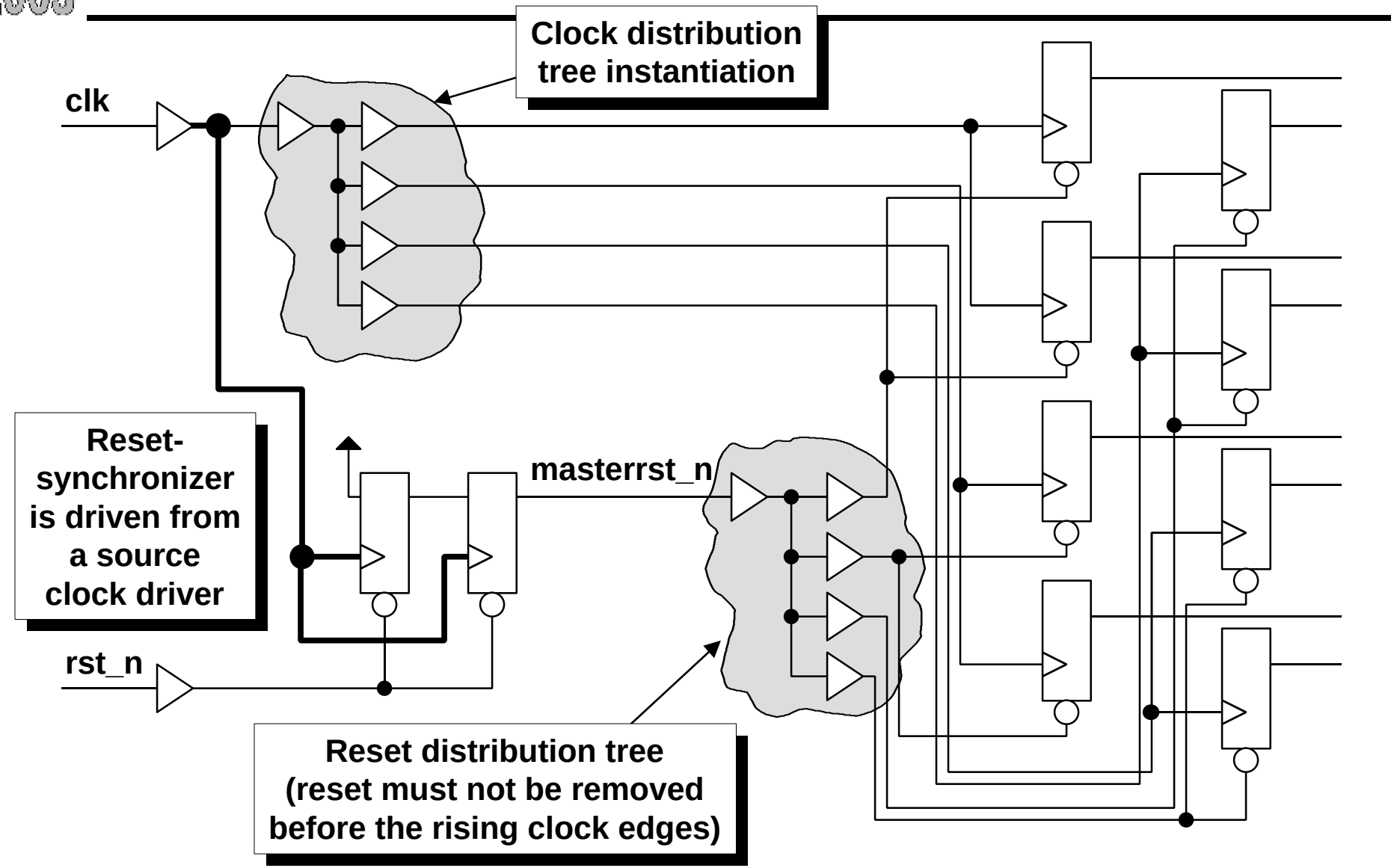
Reset Buffer Tree

Driven from a Leaf-Driver Clock

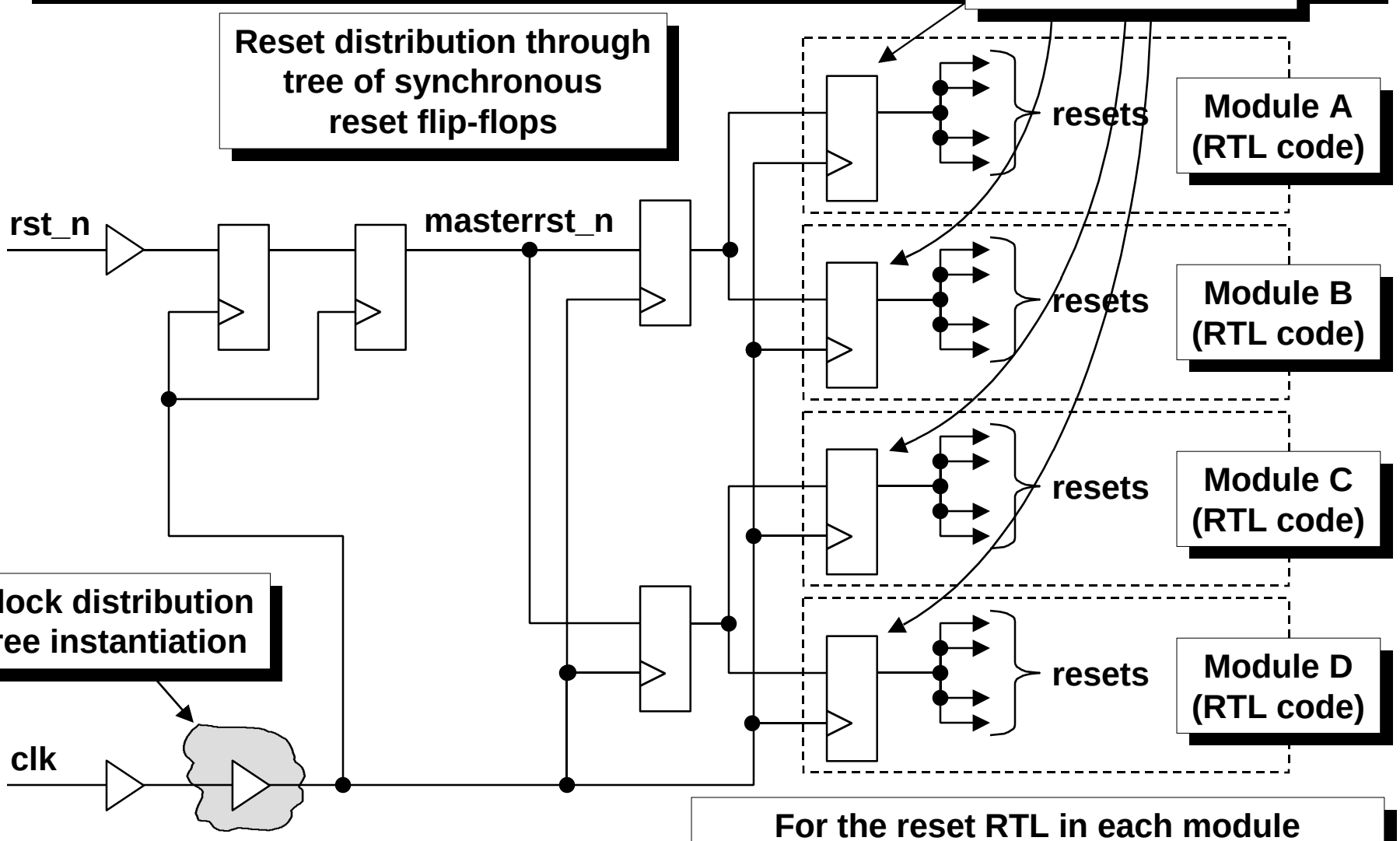


Reset Buffer Tree

Driven from a Source-Driver Clock

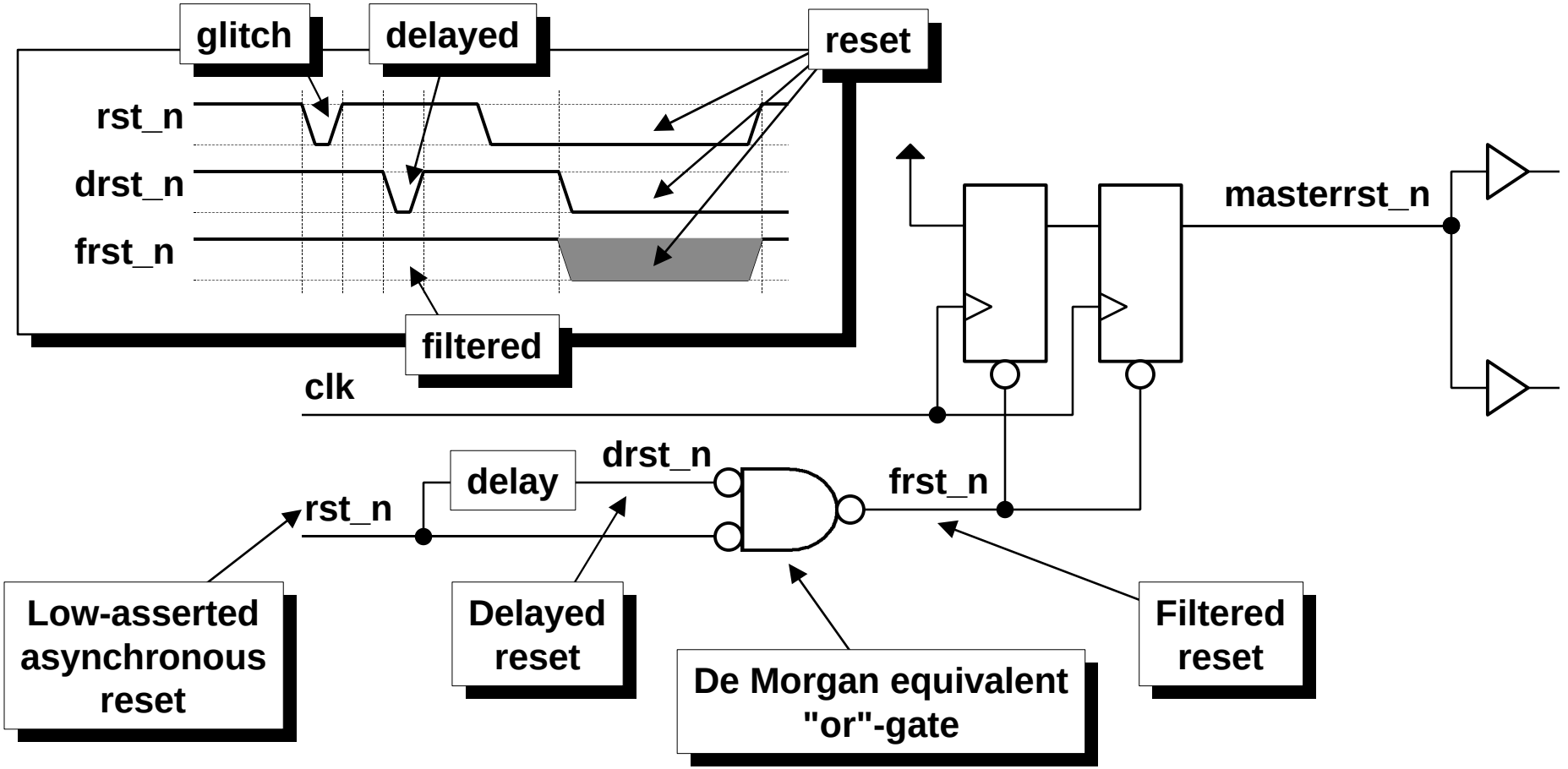


Synchronous Reset Distribution using Flip-Flops



Asynchronous Reset Glitch Filtering

- Glitches on the rst_n input might cause stray resets
- Solution: glitch-filter on the rst_n input



Asynchronous Reset & DFT

- The process of applying the ATPG vectors to create a test is based on:
  - scanning a known state into all the flip-flops in the chip
  - switching the flip-flops from scan shift mode, to functional data input mode
  - applying one functional clock
  - switching the flip-flops back to scan shift mode to scan out the result of the one functional clock while scanning in the next test vector
  - During the above process, the designer must insure that under NO CONDITIONS, an asynchronous set/reset can occur and thus corrupt the input vectors

The asynchronous reset must be held in the inactive state during the entire test

What about coverage to the portion of the chip controlled by the reset?

Asynchronous Reset & DFT

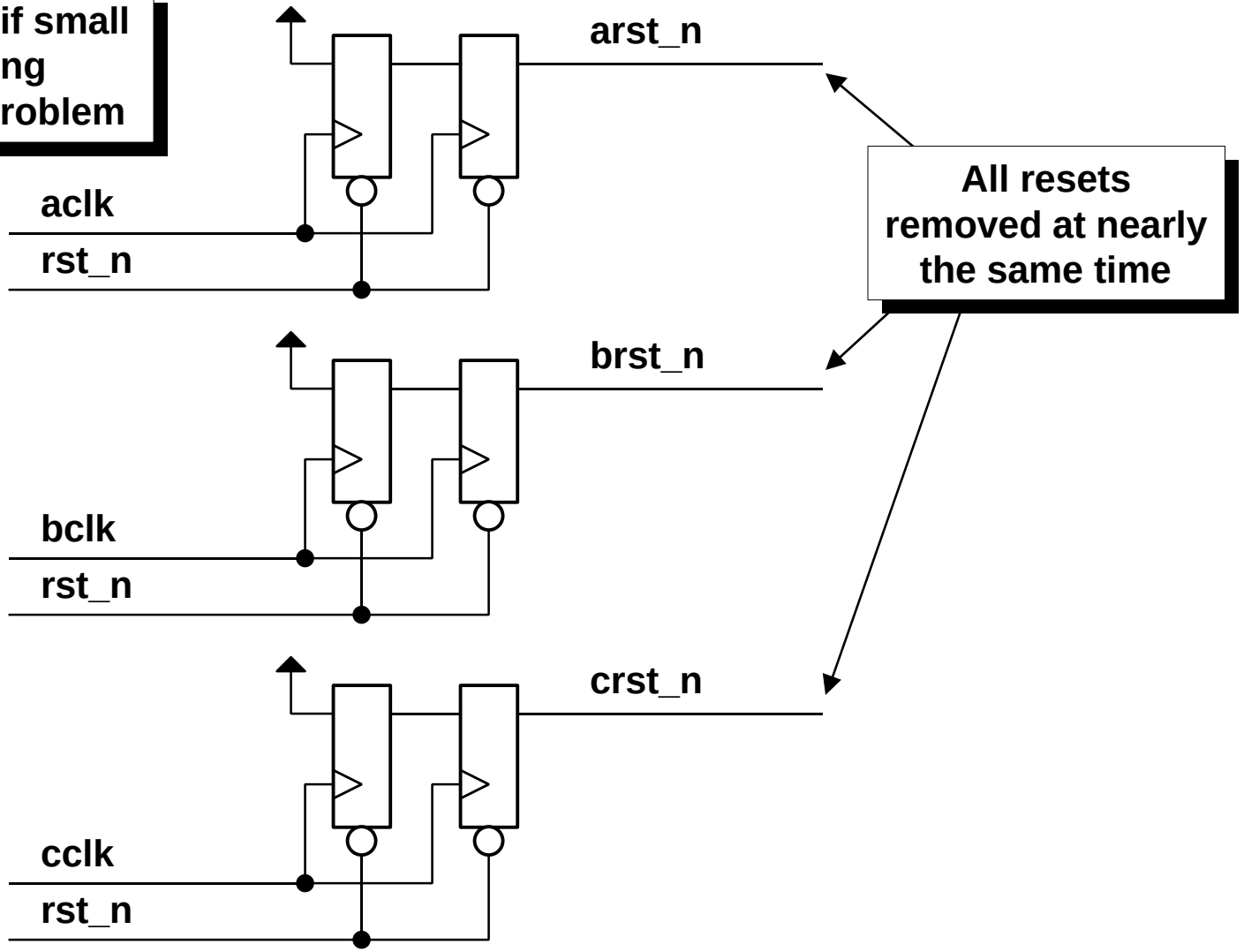
- Can I use an asynchronous reset with Design For Test (DFT) strategies?
  - scan in all ones into the scan chain
  - issue and release the asynchronous reset
  - scan out the result and scan in all zeros
  - issue and release the reset
  - scan out the result
  - set the reset input to the non reset state and then apply the ATPG generated vectors

This will test for the reset line attached to either the asynchronous set or reset of a flip-flop

Multiple Clock Domains

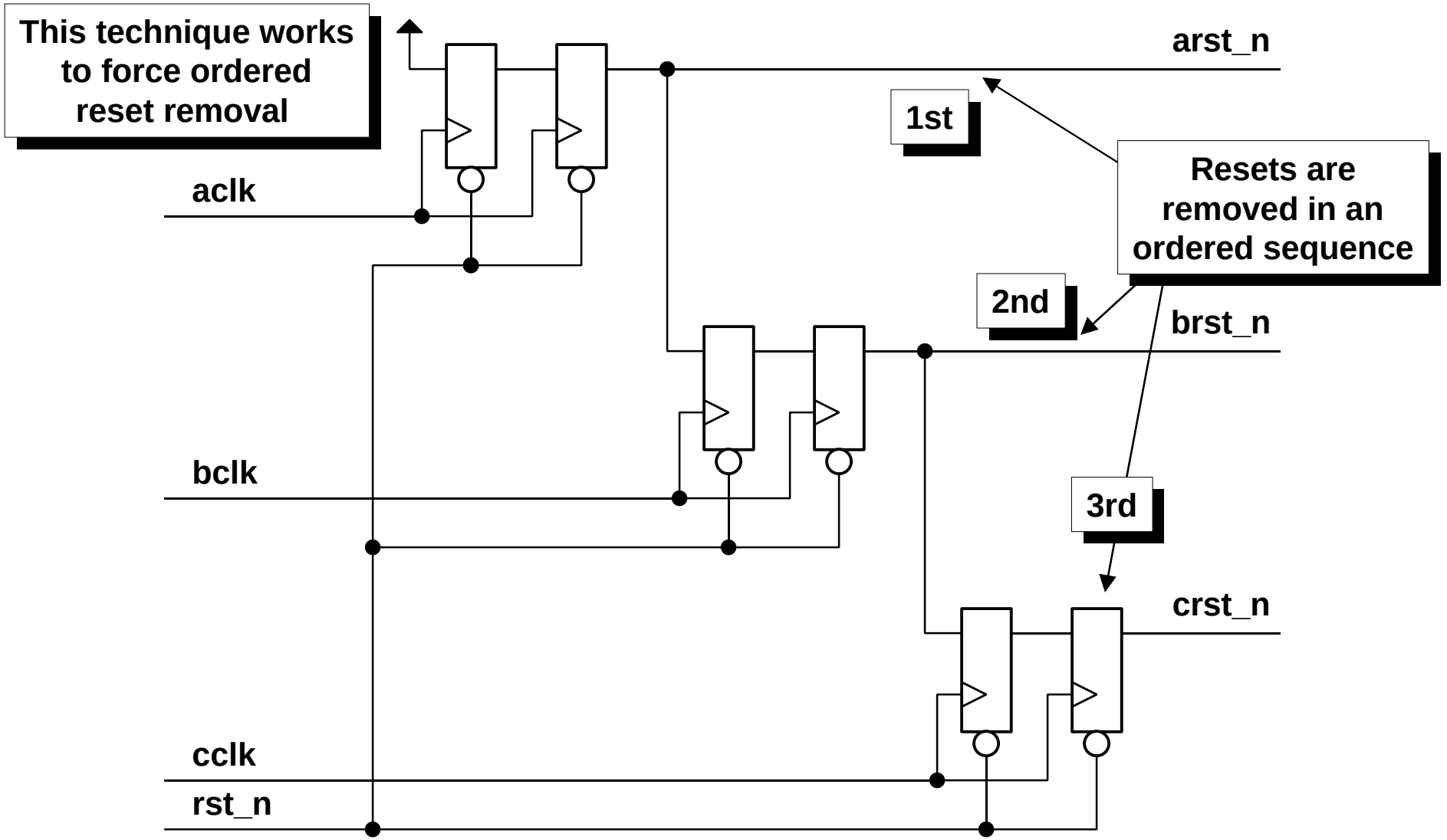
Non-Synchronized Reset Removal

This technique works if small reset removal timing differences are not a problem



Multiple Clock Domains

Synchronized Reset Removal



Conclusions

- Synchronous resets:
  - Add "**`synopsys sync_set_reset`**" directive to optimize synchronous reset layout (and minimize X-state simulation problems)
  - Easiest solution for doing Static Timing Analysis (STA)
  - Easiest solution when working with DFT
- Asynchronous resets and the "Reset Synchronizer":
  - Offers the advantages of asynchronous resets
  - Offers the advantages of synchronous reset removal
  - Still works well with DFT techniques
  - Reset path is not as easily checked using STA
- Know the limitations of each reset strategy
  - If you do it wrong it is gonna hurt!