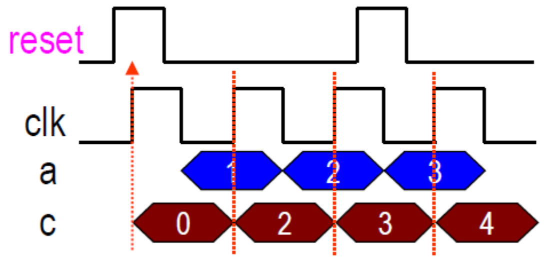
SOC Design
Reset Explained

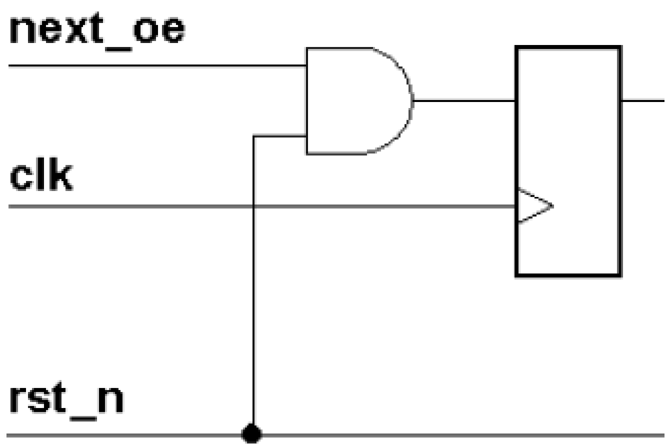
Jiin Lai

Reset: Synchronous/Asynchronous reset
Xilinx prefer synchronous reset

- Register with synchronous reset
  - @(posedge clk): for synchronous reset
    - Ex.

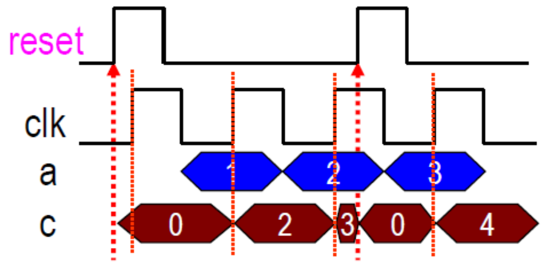
```
always @(posedge clk) begin
  if(reset) c = 0;
  else c = a+1;
end
```

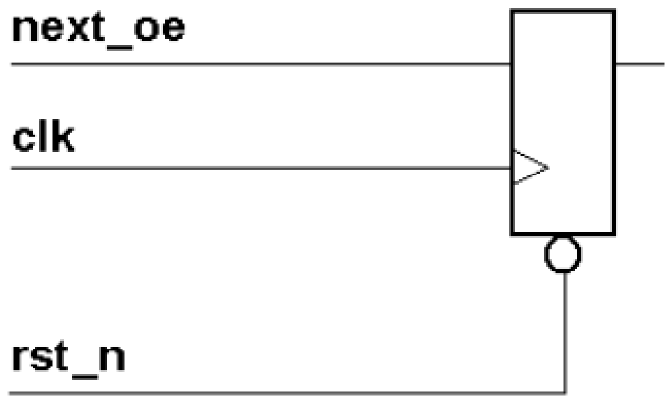




- Register with asynchronous reset
  - @(posedge clk or posedge reset): for asynchronous reset
    - Ex.

```
always @(posedge clk or posedge
reset) begin
  if(reset) c = 0;
  else c = a+1;
end
```





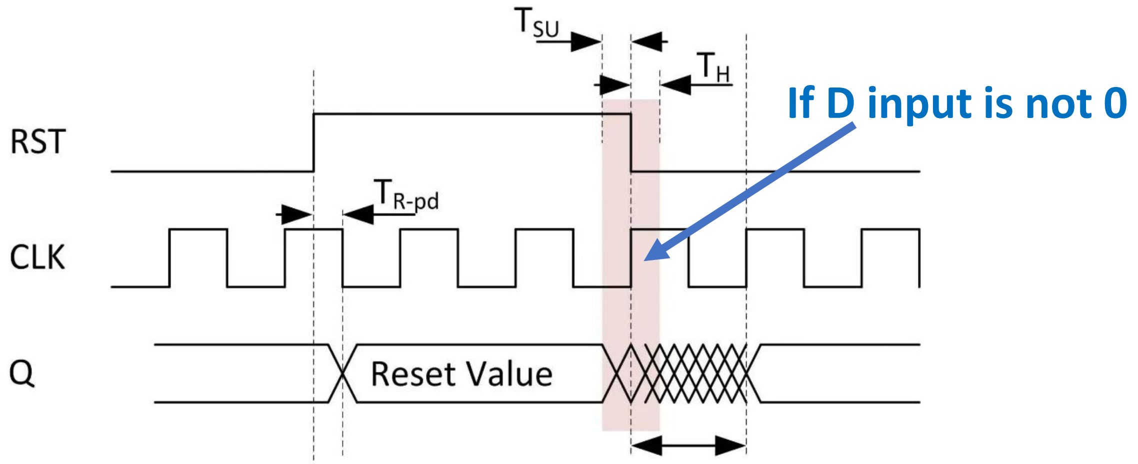
FPGA (Xilinx) Prefers Synchronous Reset

- Synchronous reset
  - Mapped to more resource
  - Affect datapath timing (an extra gate delay)
  - More flexibility for control set remapping for higher density placement
  - DSP48, block RAM have only synchronous reset
  - Clock as a filter for small reset glitch
  - Stretch reset pulse width to ensure sampled by clock
- Asynchronous
  - Impact maximum clock frequency – route this reset to all registers
  - Probability of corrupting memory content during reset assertion
  - Ensure synchronized the de-assertion of the asynchronous reset -> avoid reset release edge synchronization can lead to meta-stability.
  - Reset/hold satisfied for the clock

Xilinx: https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/Synchronous-Reset-vs.-Asynchronous-Reset

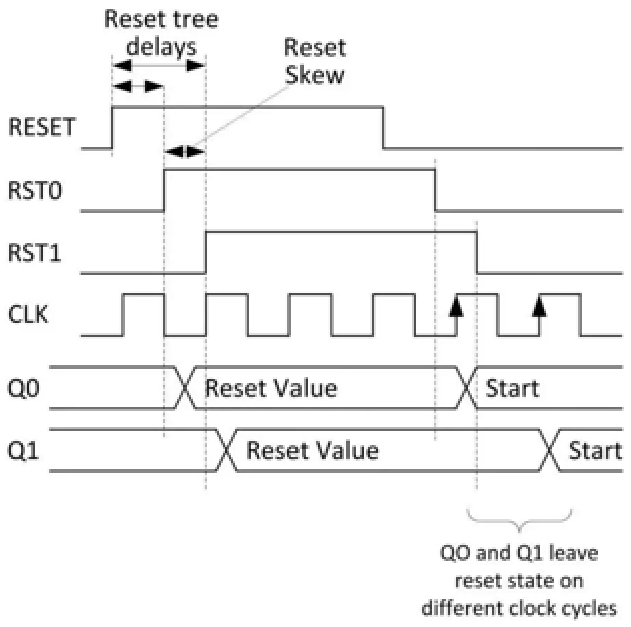
Asynchronous is preferred in Industry

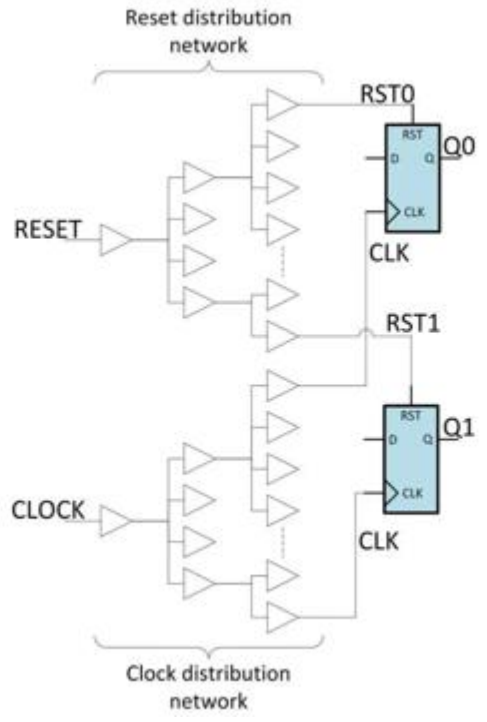
- Save area (not gated logic), and not affect datapath timing
- Case for reset while no clock (low power)
- Reset release must be synchronized to clock
- Synchronization - Glitch filtered and synchronizer
- Distribution – Apply clock-tree style routing



(a) A Deterministic Reset Propagation Delay

(b) A possibly metastable value

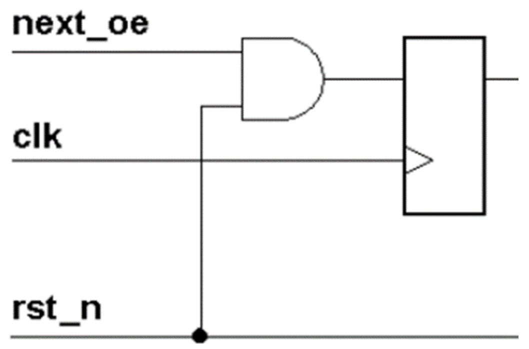




Advantage & Disadvantage of Synchronous Reset

Advantage

- Smaller flip-flops but combinational gate count grows
- Synchronous reset preferred by cycle based simulator
- Filter for small reset glitches
- Use Flip-flops in the reset buffer tree to help timing



Disadvantage

- Need pulse stretcher to guarantee a reset pulse to ensure reset sampled by clock
- Add delay on data path
- Sometime, during reset there is no clock

```
module sync_resetFFstyle (q, d, clk, rst_n);
  output q;
  input  d, clk, rst_n;
  reg    q;

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

Asynchronous reset

```
module async_resetFFstyle (q, d, clk, rst_n);
  output q;
  input  d, clk, rst_n;
  reg    q;

  // Verilog-2001: permits comma-separation
  // @(posedge clk, negedge rst_n)
  always @(posedge clk or negedge rst_n)
    if (!rst_n) q <= 1'b0;
    else        q <= d;
endmodule
```

Set/reset flip-flop modeling

```
// Good DFF with asynchronous set and reset and self-
// correcting set-reset assignment
module dff3_aras (q, d, clk, rst_n, set_n);
  output q;
  input d, clk, rst_n, set_n;
  reg q;

  always @(posedge clk or negedge rst_n or negedge set_n)
    if      (!rst_n) q <= 0; // asynchronous reset
    else if (!set_n) q <= 1; // asynchronous set
    else             q <= d;

  // synopsys translate_off
  always @(rst_n or set_n)
    if (rst_n && !set_n) force   q = 1;
    else                 release q;
  // synopsys translate_on
endmodule
```

Advantage / Disdvantage of Asynchronous reset

Advantage

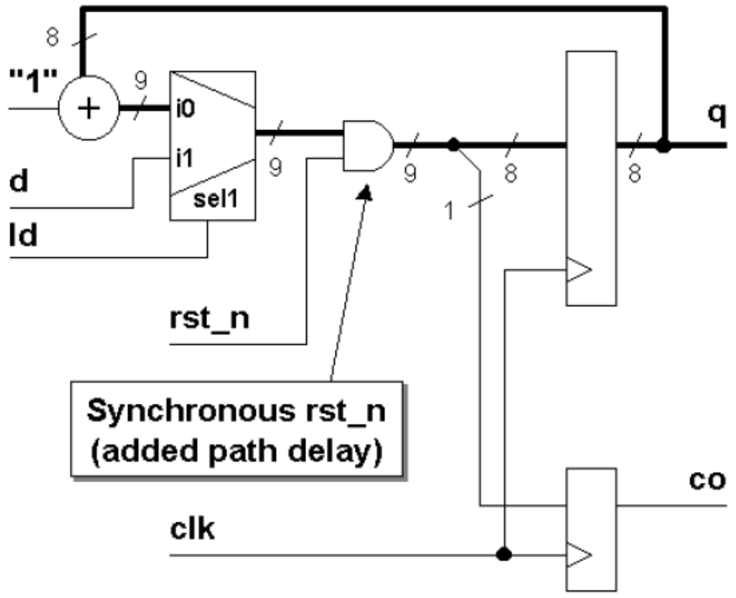
- Datapath is clean, better timing can be reset without a clock

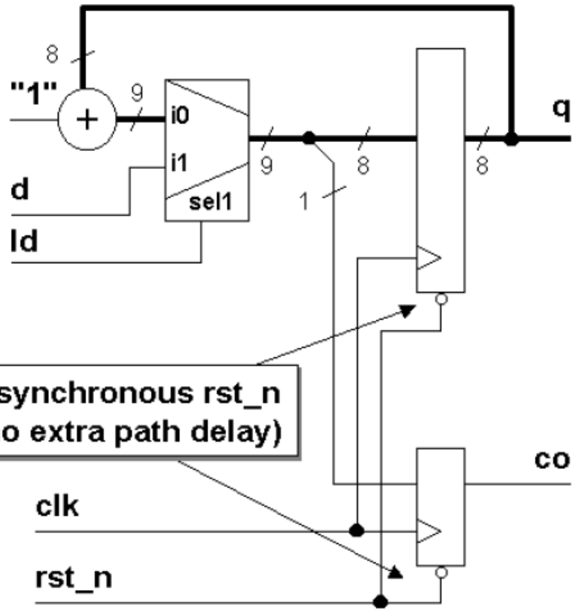
Disadvantage

- DFT – resetability for DFT scanning and testing
- STA is difficult to do Reset deassertion near clock edge , metastable => Removal recovery time problem
- Spurious reset due to noise

Loadable Counter with Synchronous/Asynchronous reset

```
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
```





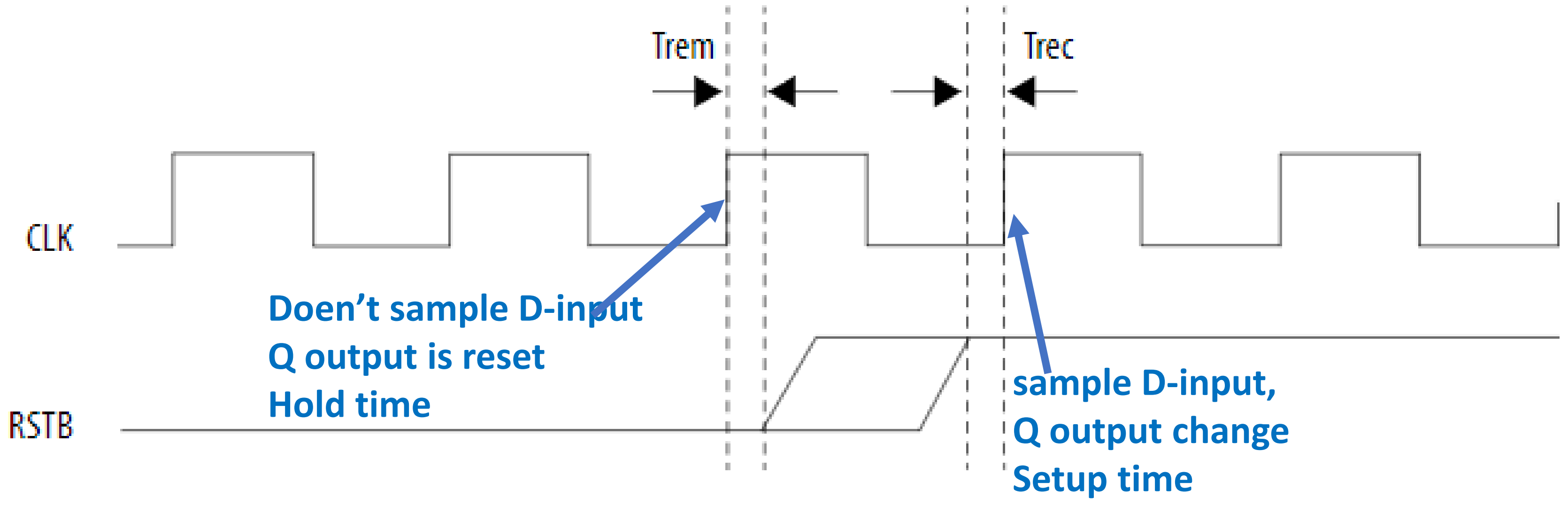
```
always @(posedge clk or negedge rst_n)
  if       (!rst_n) {co,q} <= 9'b0;      // async reset
  else if  (ld)     {co,q} <= d;         // sync load
  else              {co,q} <= q + 1'b1;  // sync increment
```

Asynchronous Reset De-assertion Removal / Recovery Time

During the de-assertion of a reset, to avoid the register entering metastable state, ensure the following timing:

Removal Time, Trem - the minimum time, after the active clock edge, that the reset must be stable before being de-asserted.

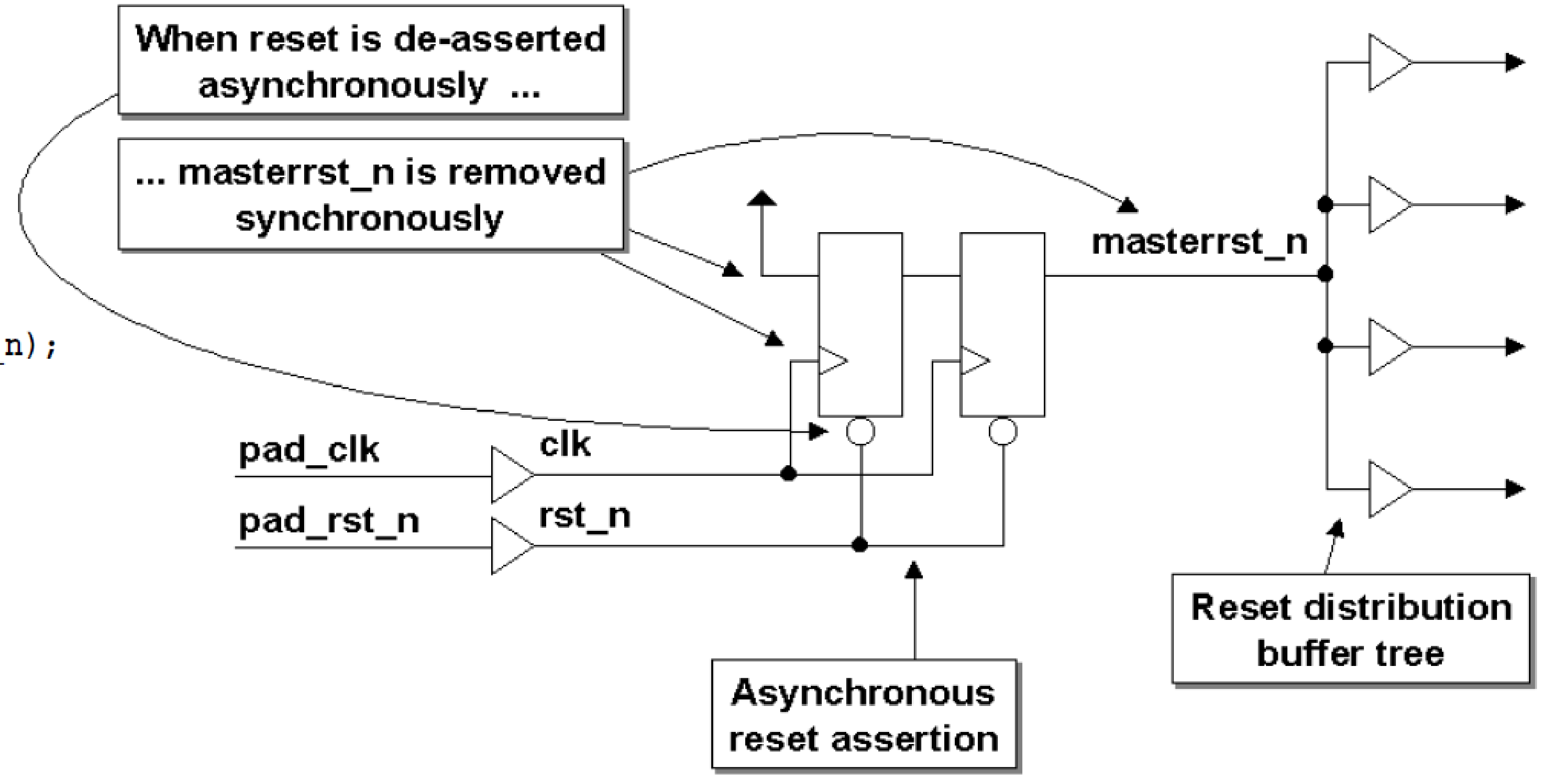
Reset Recovery Time, Trec - the minimum time between the de-assertion of a reset and the clock signal being high again.



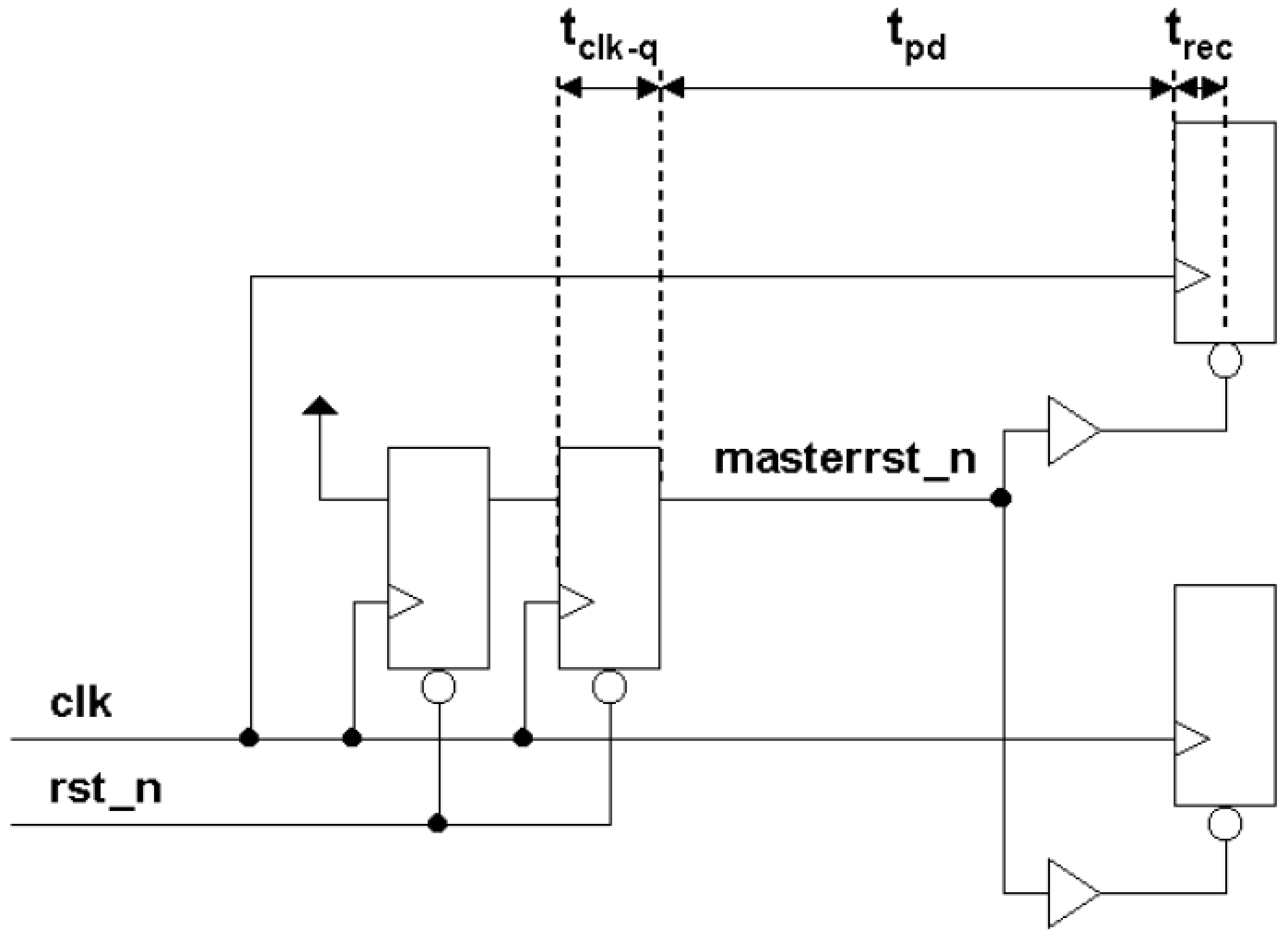
Reset Synchronizer

```
module async_resetFFstyle2 (rst_n, clk, asyncrst_n);
  output rst_n;
  input  clk, asyncrst_n;
  reg    rst_n, rff1;

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


When reset is de-asserted asynchronously ...
... masterrst_n is removed synchronously
masterrst_n
pad_clk
clk
pad_rst_n
rst_n
Asynchronous reset assertion
Reset distribution buffer tree

Predictable reset removal to satisfy reset recovery time



Asynchronous Reset Filter & Synchronizer

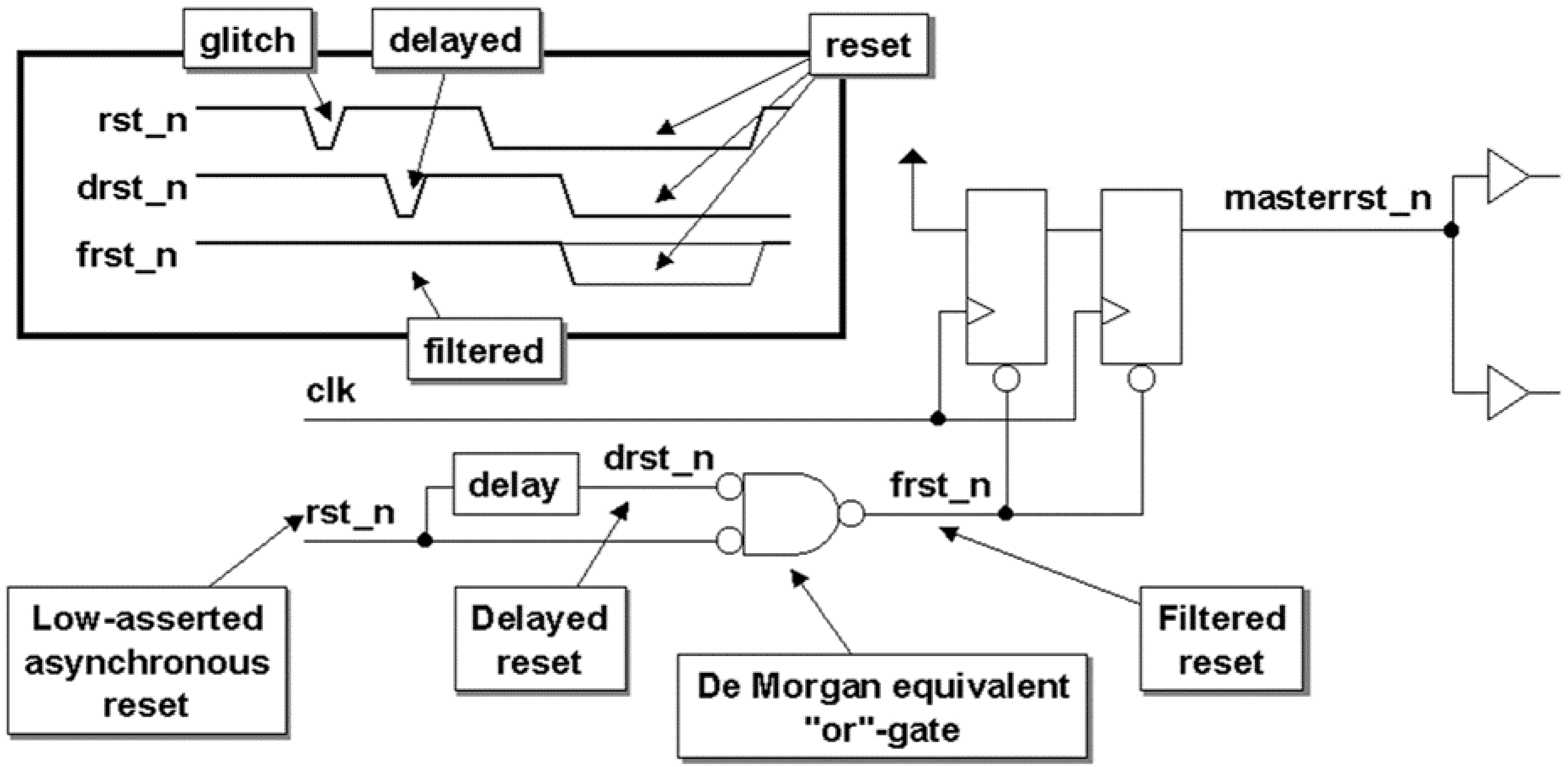
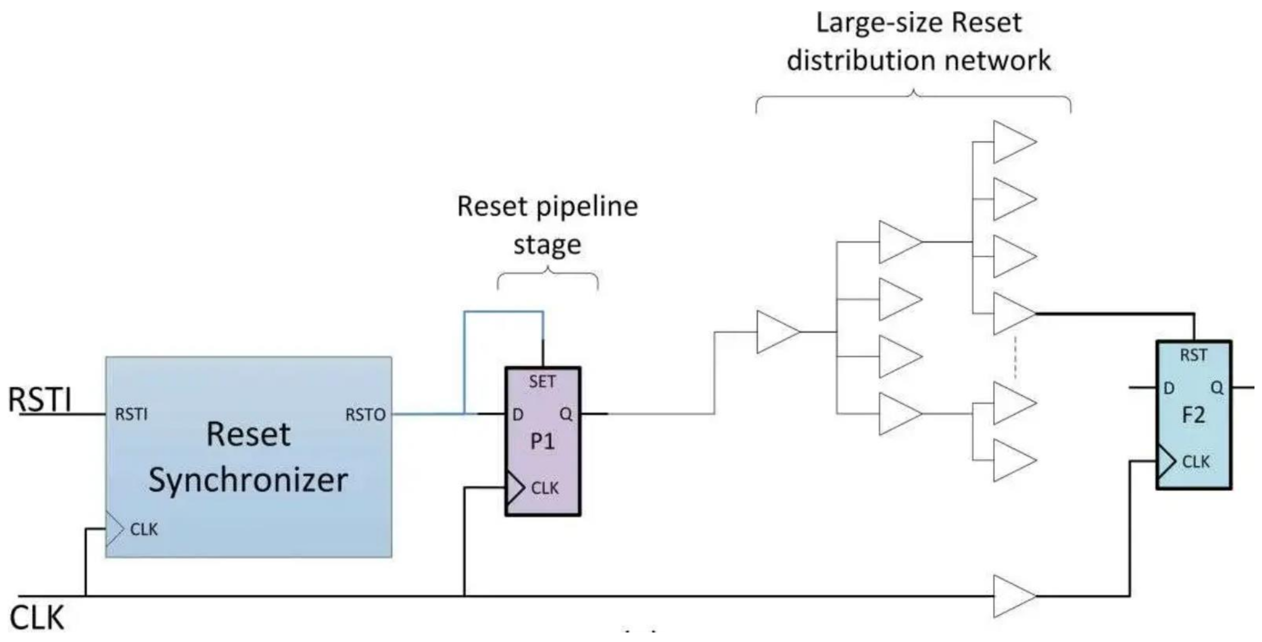
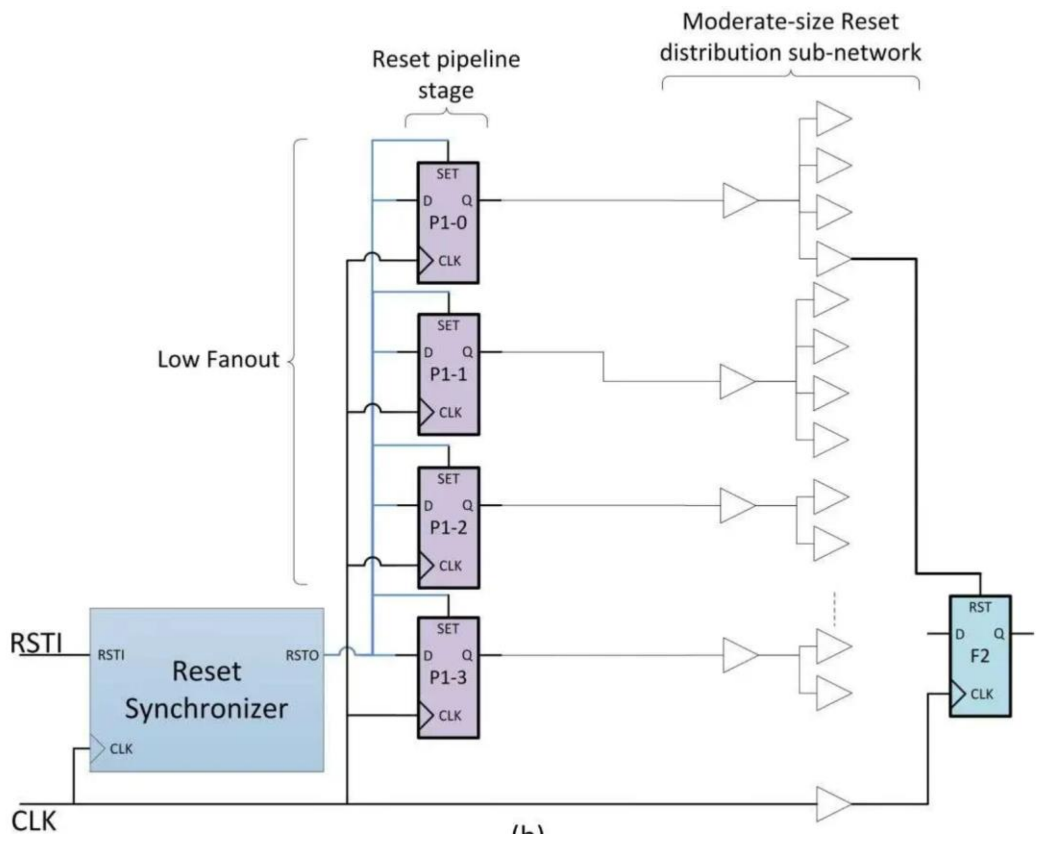


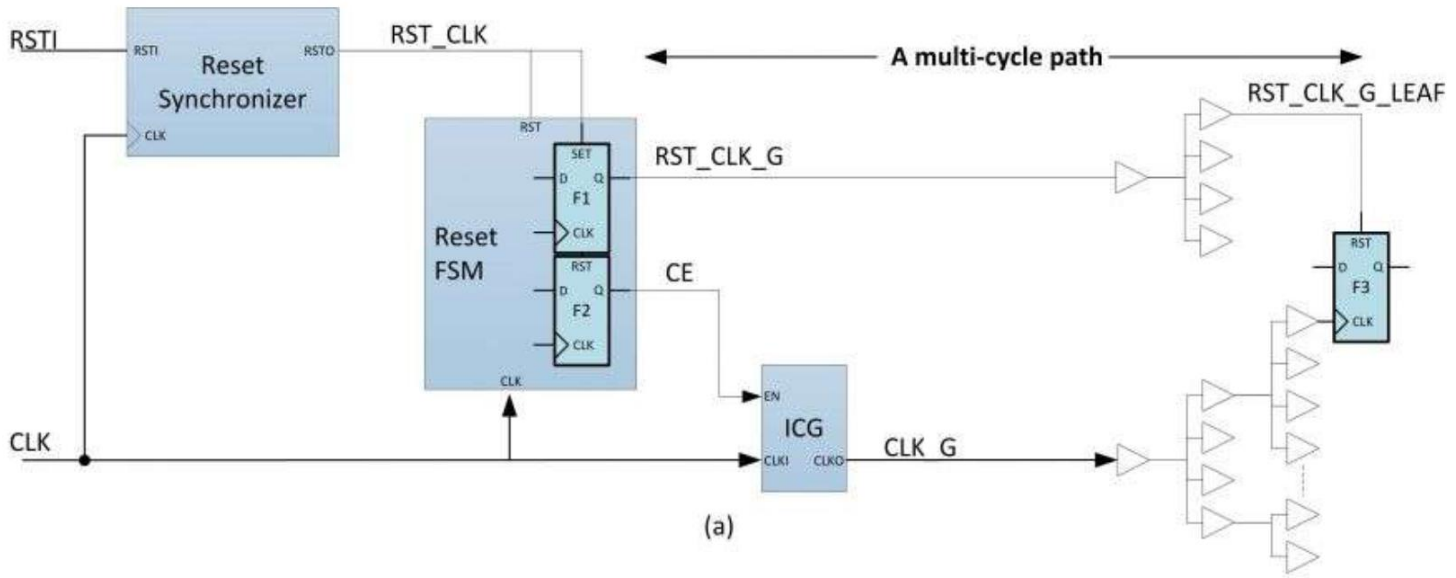
Figure 11 - Reset glitch filtering

Asynchronous Reset Distribution – Distribute Load

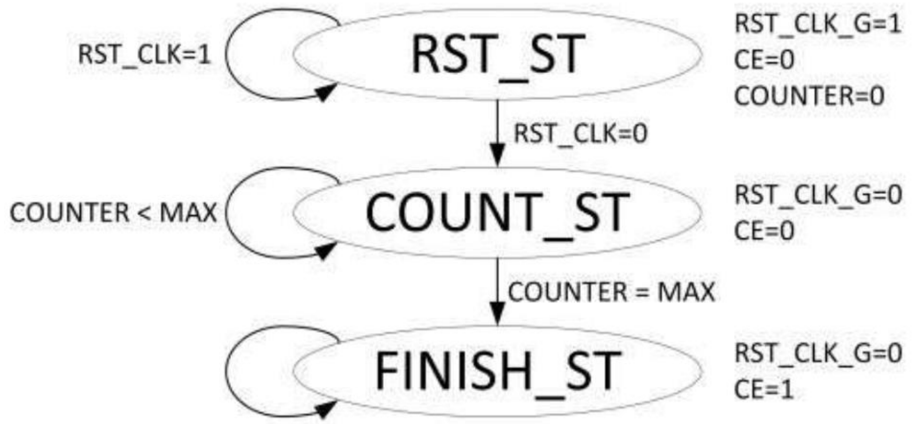




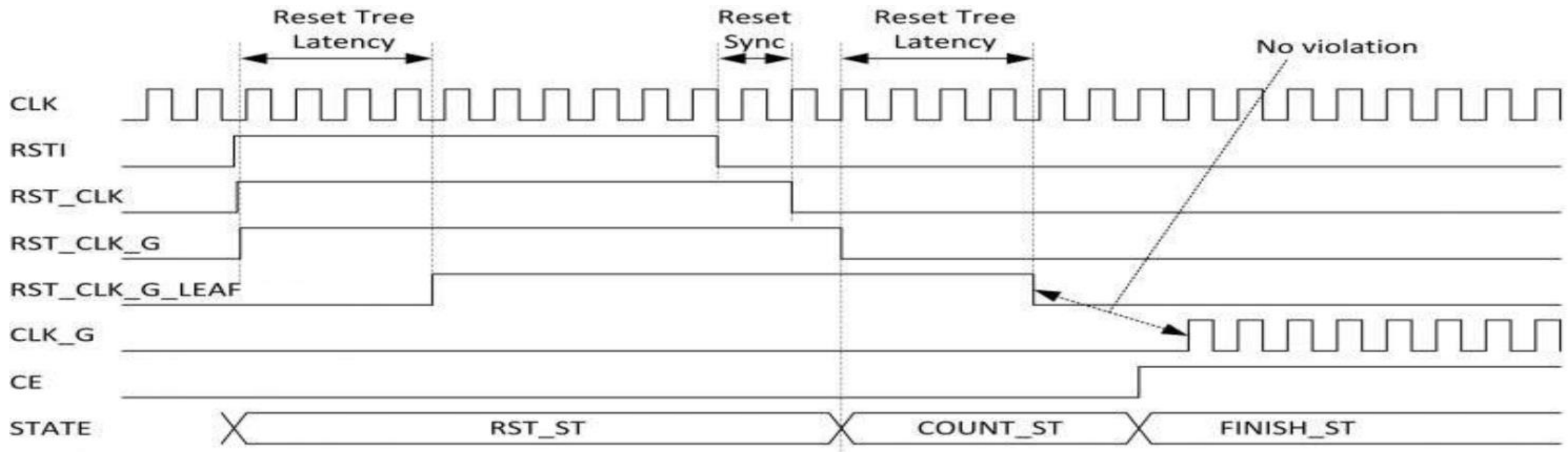
Asynchronous Reset Distribution – Gated Clock



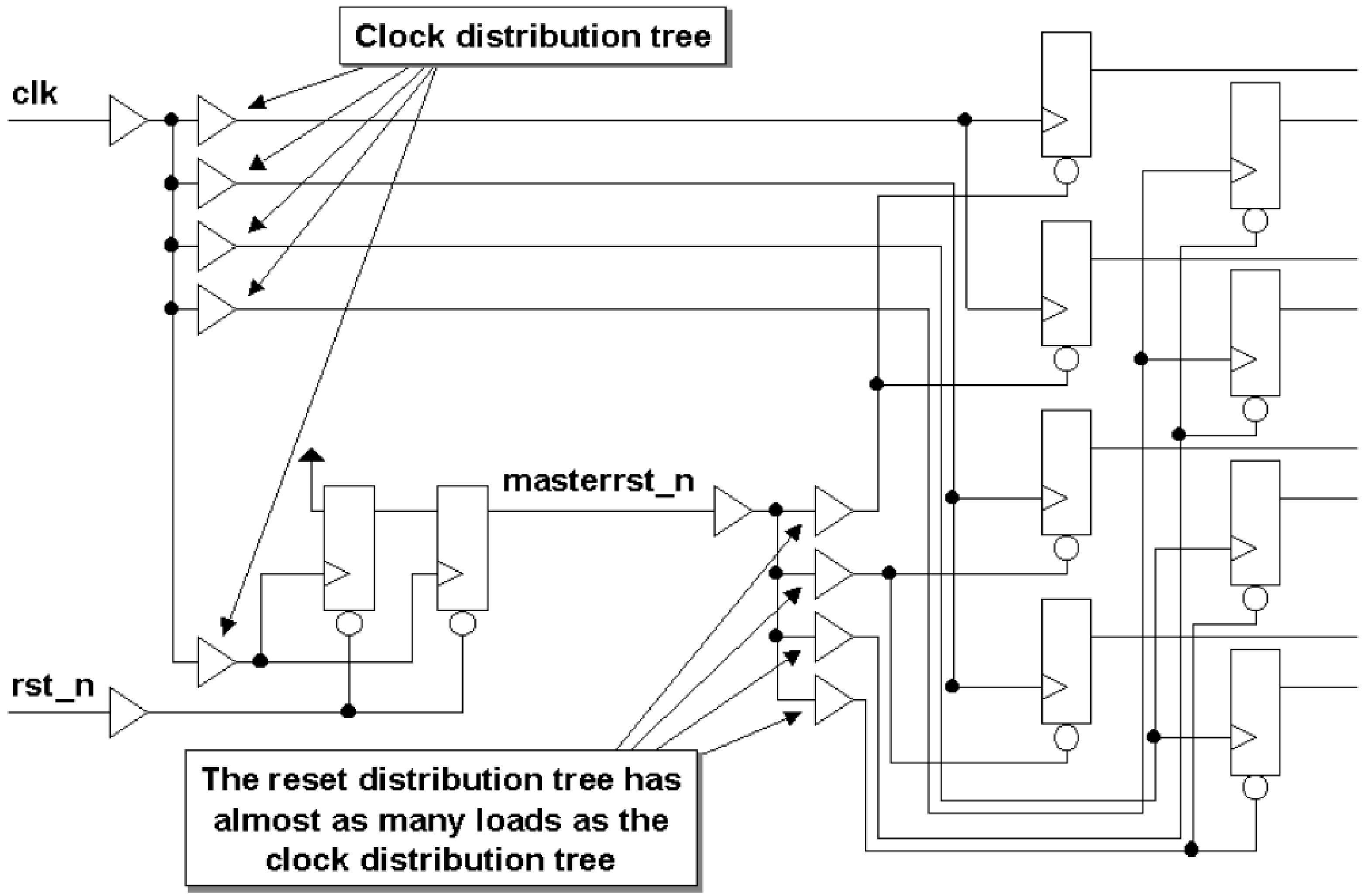
(a)



(b)



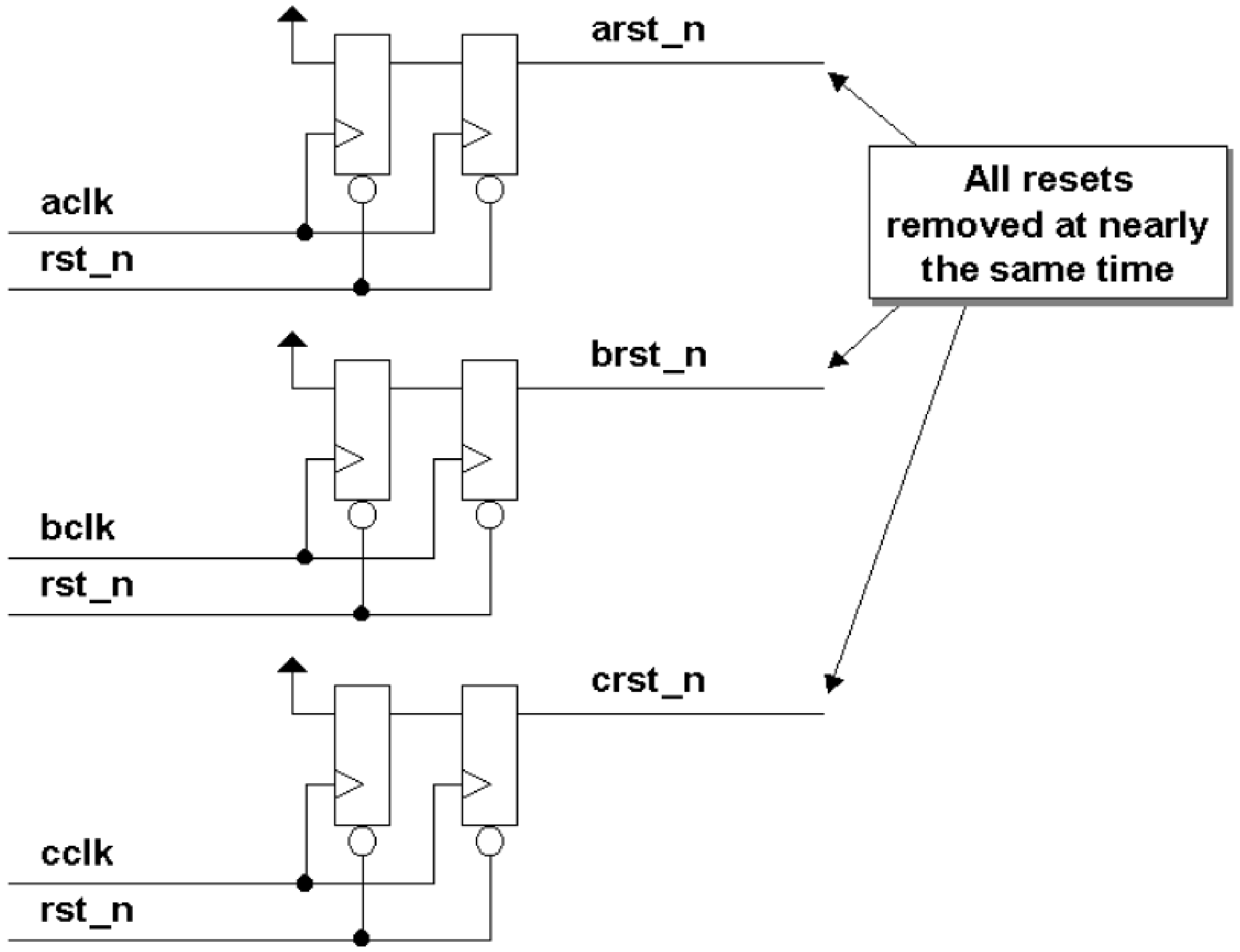
Reset Distribution Tree – **Reset Synchronizer uses an early clock**



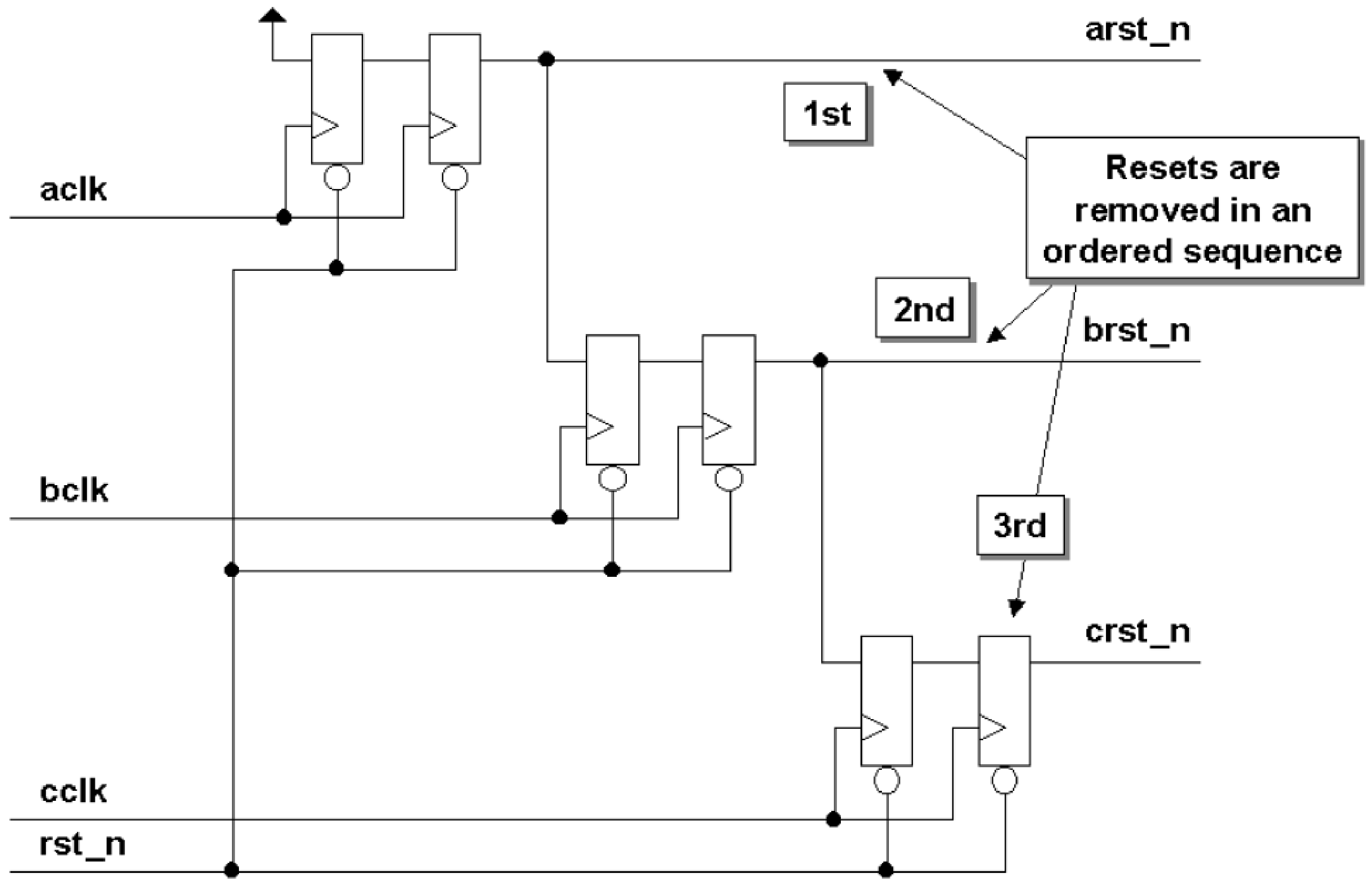
Clock distribution tree
clk
masterrst_n
rst_n
The reset distribution tree has almost as many loads as the clock distribution tree

Multi-clock Domain Reset

Multi-clock reset – non-coordinated reset removal



Multi-clock reset – Sequenced coordination of reset removal


arst_n
1st
Resets are removed in an ordered sequence
aclk
2nd
brst_n
bclk
3rd
crst_n
cclk
rst_n