Advanced Digital Design

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Lecture-I: Clock Generation

Sudeendra kumar K
Department of Electronics and Communication Engineering

Advanced Digital Design

Contents

- Clock generation fundamentals

Clock Generation

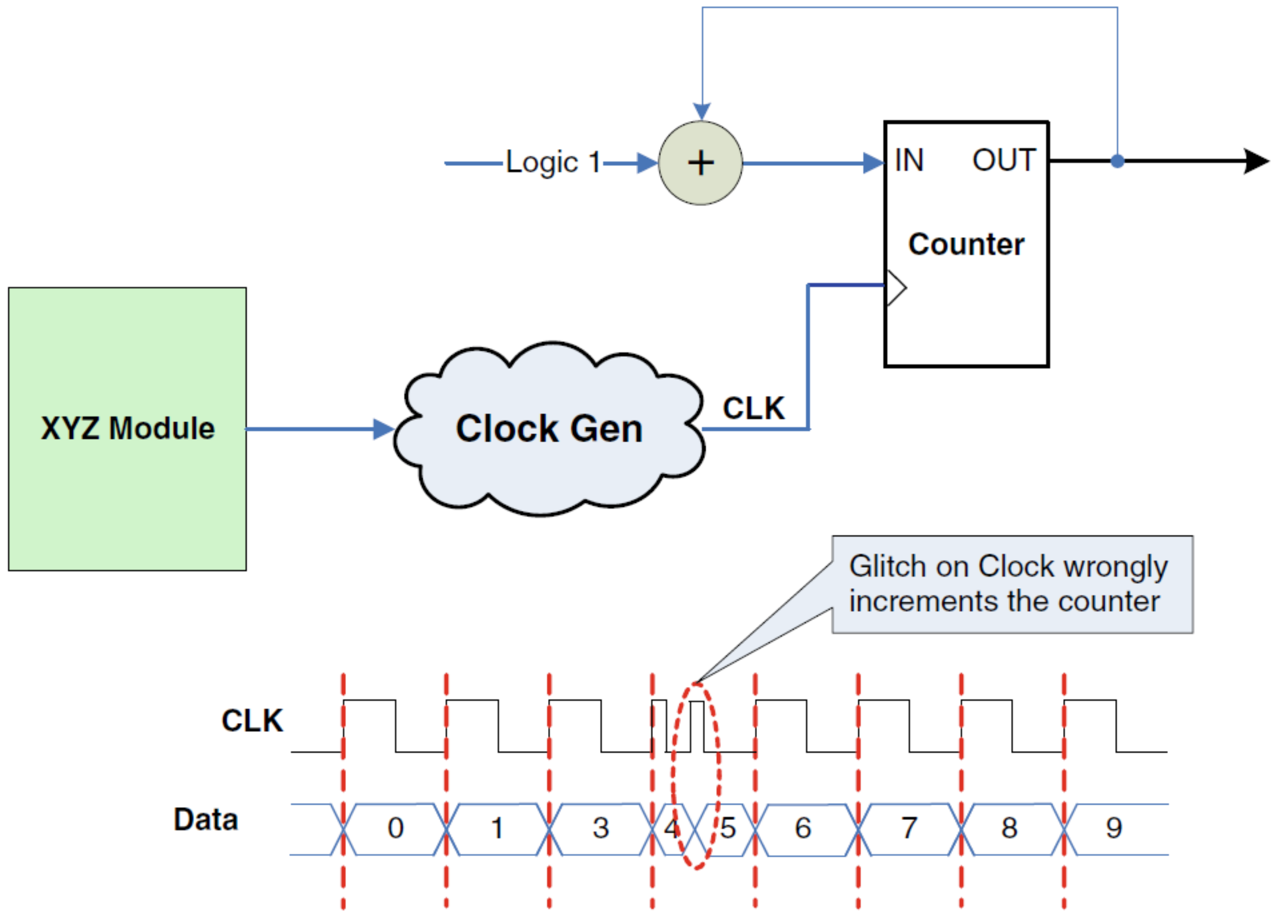
- A designer should avoid internally generated clocks, wherever possible, as they can cause functional and timing problems in the design, if not handled properly.
- Clocks generated with combinational logic can introduce glitches that create functional problems and the delay due to the combinational logic can lead to timing problems.
- In a synchronous design, a glitch on the data inputs does not cause any issues and is automatically avoided as data is always captured on the edge of the clock and thus blocks the glitch.
- A glitch or a spike on the clock input (or an asynchronous input of a register) can have significant consequences.

1. **Piezoelectric Crystal – Stable Clock (external)**
2. **On-chip PLL (free running mode)**
3. **Crystal output as ref to PLL and generate different clocks from PLL. (it takes time to generate stable clock, it is called as PLL locking time)**
4. **Combinational Circuits (Ex: Ring Oscillator)**

Counter as Clock Divider

- For power-of-2 integer division, a simple binary counter can be used, clocked by the input signal.
- The least-significant output bit alternates at 1/2 the rate of the input clock, the next bit at 1/4 the rate, the third bit at 1/8 the rate, etc.
- An arrangement of flipflops is a classic method for integer-n division. Such division is frequency and phase coherent to the source over environmental variations including temperature.

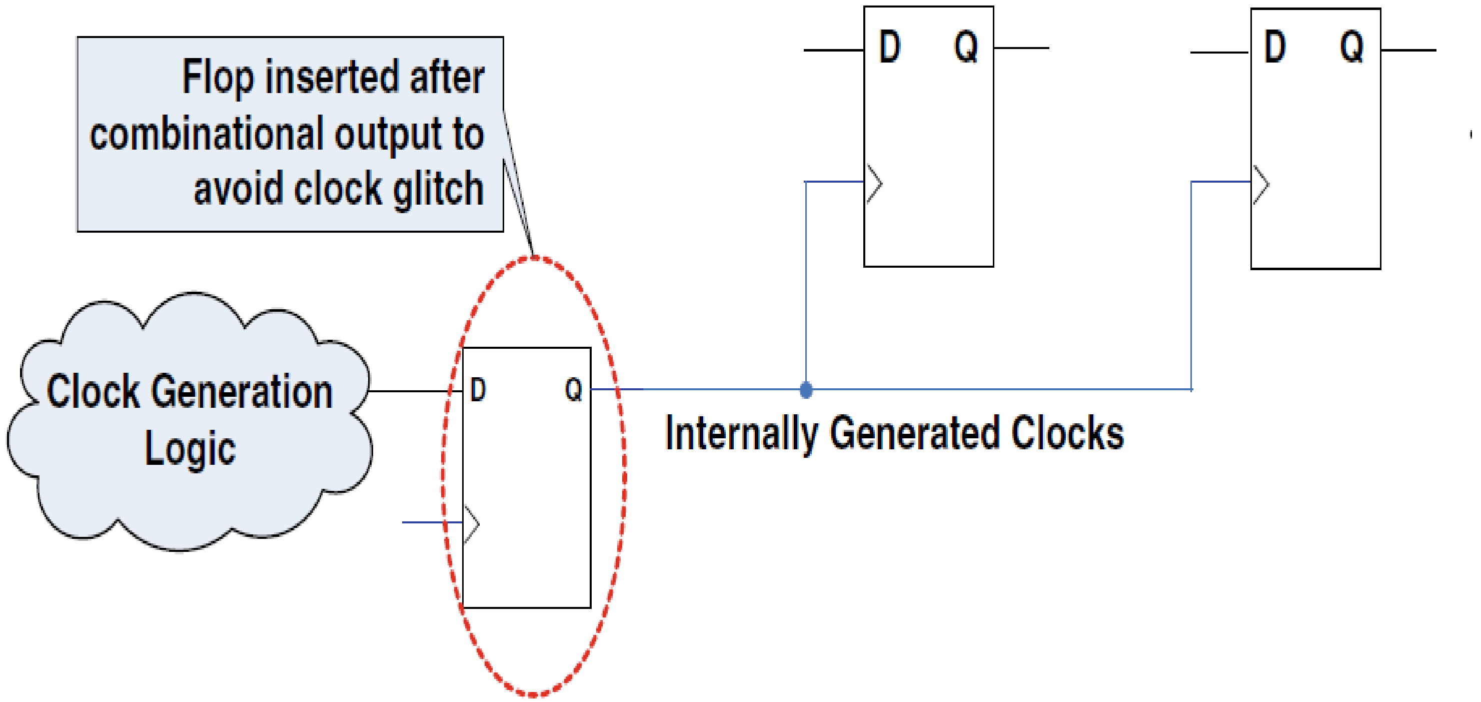
Effects of Glitches in Clock Generation



- Glitch will cause further glitches in the Clock divider circuit

Advanced Digital Design

Recommended Clock generation technique

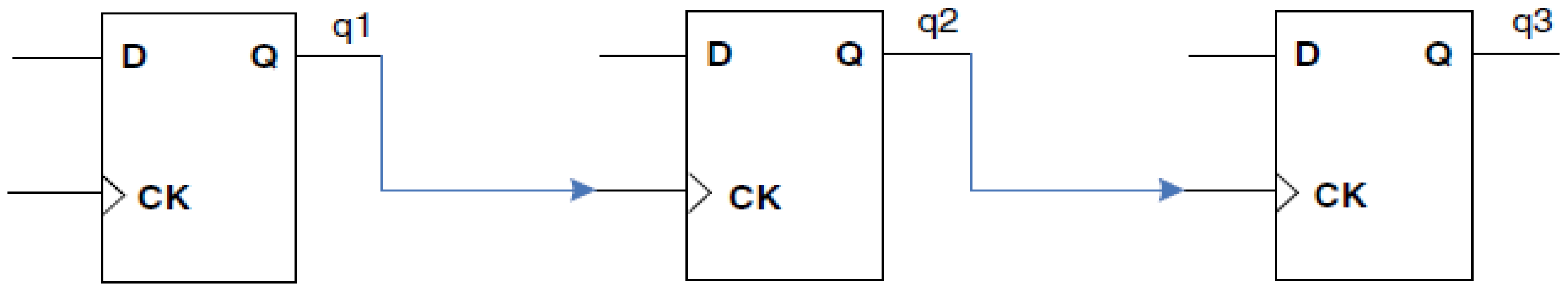

Flop inserted after combinational output to avoid clock glitch
Clock Generation Logic
D
Q
Internally Generated Clocks
D
Q
D
Q

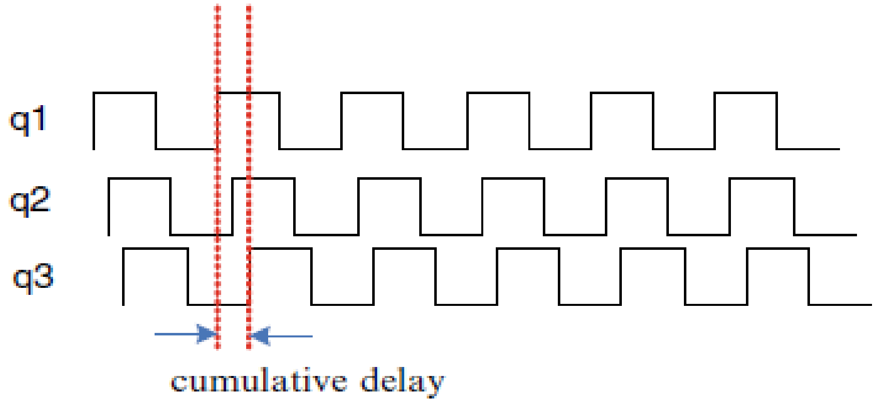
- A simple guideline to the above problem is to always use a registered output of the combinational logic before using it as a clock signal. This registering ensures that the glitches generated by the combinational logic are blocked on the data input of the register.

Advanced Digital Design

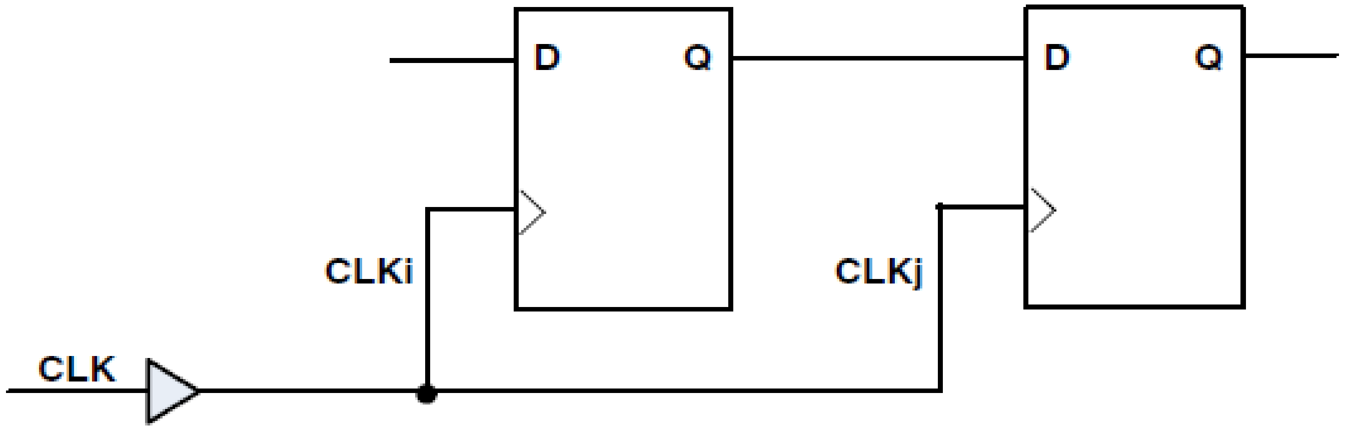
Avoid using Ripple Counters in Clock generation/divider circuit

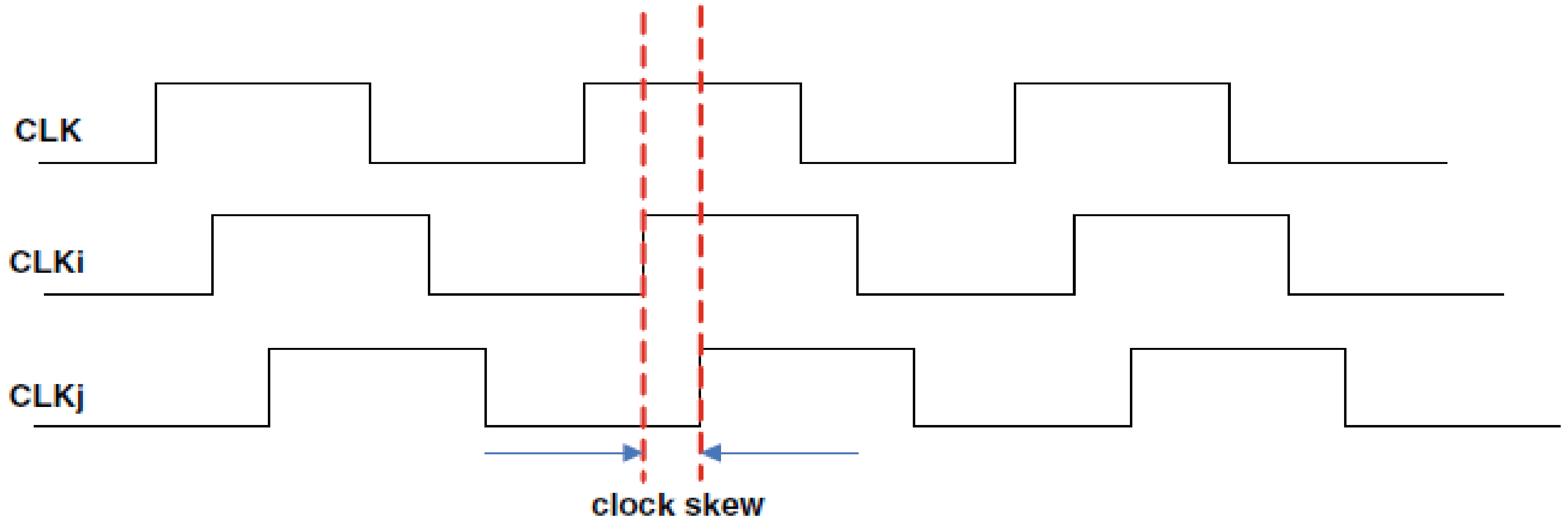
- Flip Flops driving the clock input of other flip flops is somewhat problematic. The clock input of the second flip-flop is skewed by the clock-to-q delay of the first flip-flop, and is not activated on every clock edge. This cumulative effect with more than two Flip Flops connected in a similar manner forms a Ripple counter.





Controlling Clock Skew





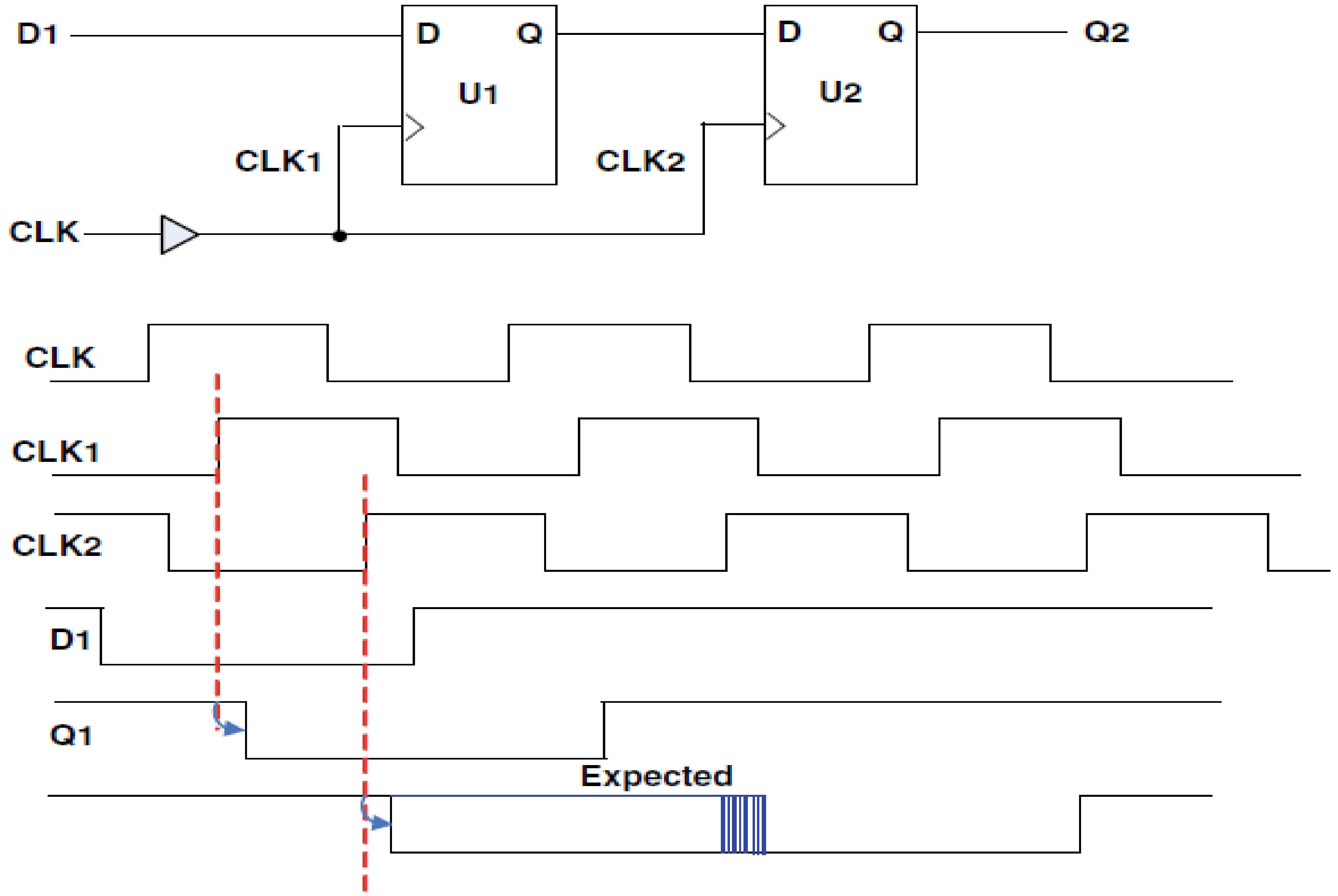
Controlling Clock Skew

- Difference in clock signal arrival times across the chip is called clock skew. It is a fundamental design principle that timing must satisfy register setup and hold time requirements. Both data propagation delay and clock skew are parts of these calculations.
- Clocking sequentially-adjacent registers on the same edge of a high-skew clock can potentially cause timing violations and functional failures. Probably this is one of the largest sources of design failure in an ASIC.

Short Path Problem

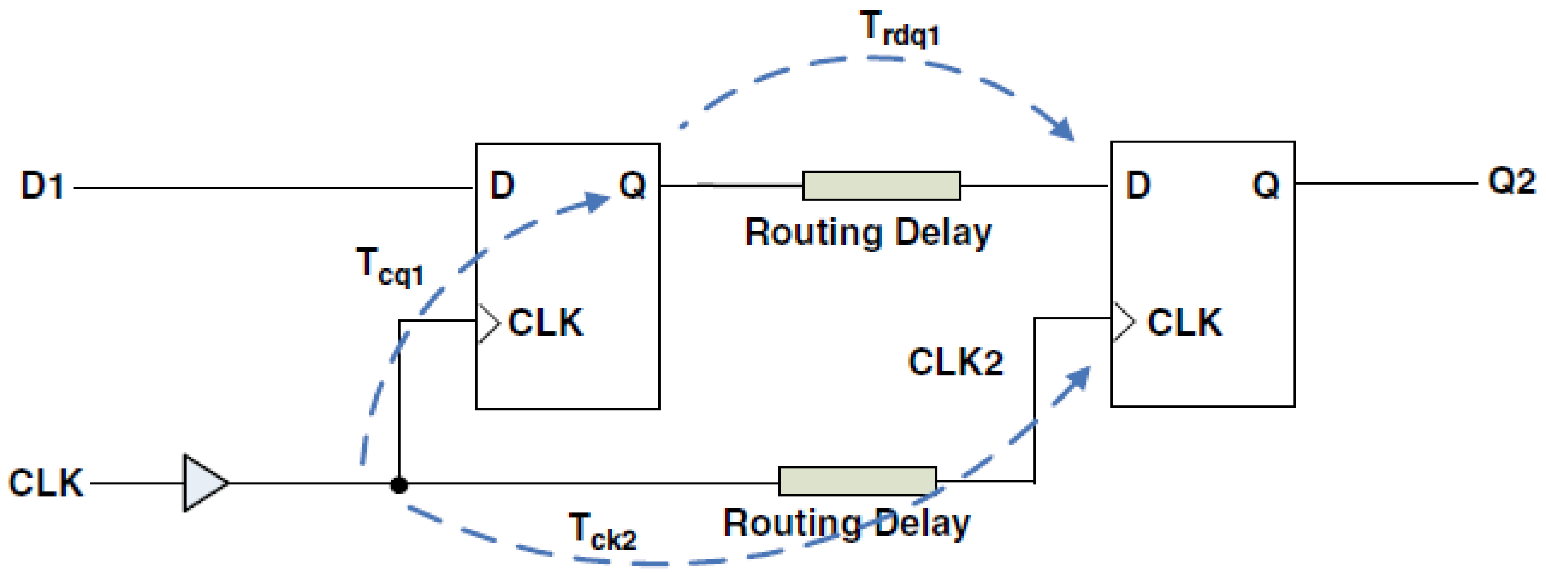
- The problem of short data paths in the presence of clock skew is very similar to hold-time violations in flip-flops.
- The problem arises when the data propagation delay between two adjacent flip-flops is less than the clock skew.
- Since the same clock edge arrives at the second flip-flop later than the new data, the second flip-flop output switches at the same edge as the first flip-flop and with the same data as the first flip-flop. This will cause U2 to shift the same data on the same edge as U1, resulting in a functional error.

Advanced Digital Design
Short Path Problem

- 

Advanced Digital Design

Clock Skew and Short Path Analysis



- Tcq1: The clock to out delay of the first flip-flop.
- Trdq1: The propagation delay from the output of the first flip-flop to the input of the second one.
- Tck2: The clock arrival time at the second flip-flop minus the clock arrival time at the first flip-flop.
- The short-path problem will definitely emerge in this circuit if: -

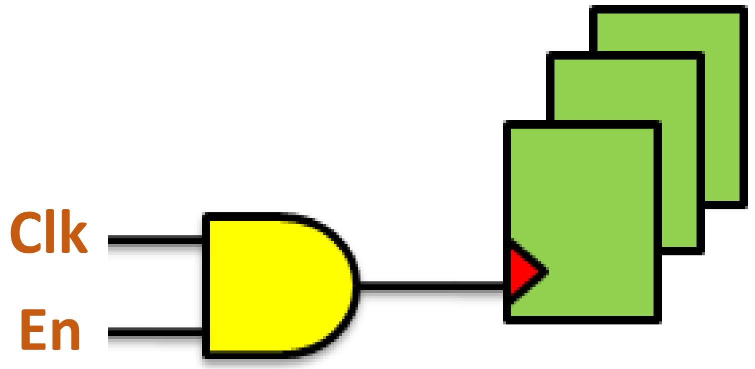
$$T_{ck2} > T_{cq1} + T_{rdq1} - T_{HOLD2}$$

Minimizing Clock Skew

- Adding Delay in Data Path
- Clock Reversing.
- Alternate Phase Clocking.
- Balancing Trace Length

Avoid these during designing Clocks

- **Gated Clocks**: - Gating in a clock line causes clock skew and can introduce spikes which trigger the flip-flop. Simulating a gated clock design might work perfectly fine but the problem arises when such a design is synthesized.
- **Double-Edged or Mixed Edge Clocking: -** The two flip-flops are clocked on opposite edges of the clock signal. This makes synchronous resetting and test methodologies such as scan-path insertion difficult, and causes difficulties in determining critical signal paths.
- **Flip Flops Driving Asynchronous Reset of Another Flop**: - The second flip-flop can change state at a time other than the active clock edge, violating the principle of synchronous design.



Clock Gating

$P = C * V^2 * f$

GALS: Globally Asynchronous and Locally Synchronous

Avoid Combinational Loops in Design

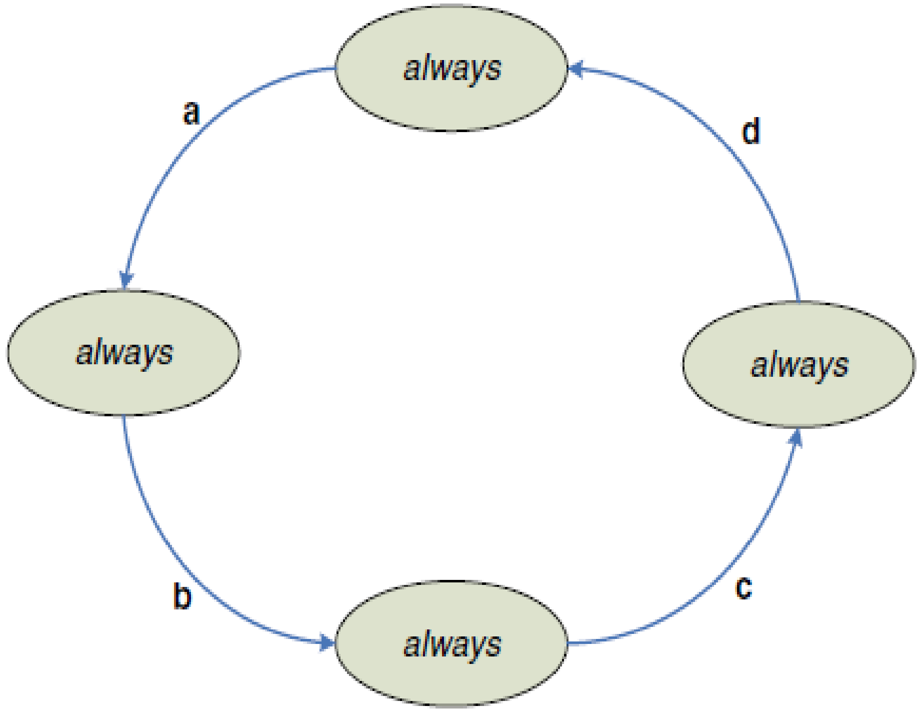
- Combinational loops are among the most common causes of instability and unreliability in digital designs. In a synchronous design, all feedback loops should include registers.
- In terms of HDL language, combinational loops occur when the generation of a signal depends on itself through several combinational always1 blocks or when the left-hand side of an arithmetic expression also appears on the right-hand side.

```
always@ (a)
begin
b = a;
End

always@ (b)
begin
c = b;
End

always@ (c)
begin
d = c;
End

always@ (c)
Begin
a = c;
end
```



Avoid Delay Chains in Digital Logic

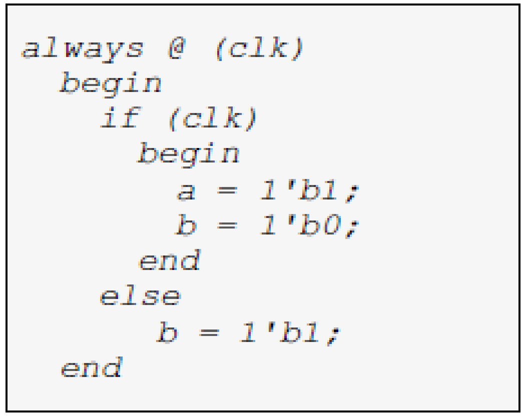
- Delay chains occur when two or more consecutive nodes with a single fan-in and a single fan-out are used to cause delay. Often inverters are chained together to add delay.
- Delay chains generally result from asynchronous design practices, and are sometimes used to resolve race conditions created by other combinational logic.

Avoid Using Synchronous Based Pulse Generator

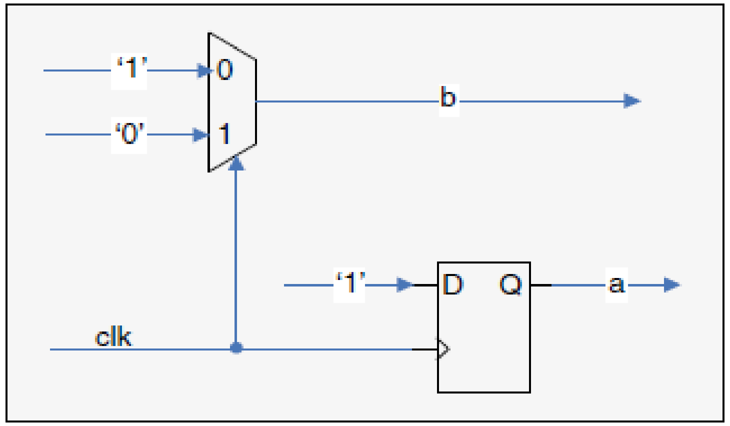
- Often design requires generating a pulse based on some events. Designers sometimes use delay chains to generate either one pulse (pulse generators) or a series of pulses (multi-vibrators).
- The following technique should be avoided: A trigger signal feeds both inputs of a two-input AND or OR gate, but the design inverts or adds a delay chain to one of the inputs. This technique artificially increases the width of the spike by using a delay chain.
- The actual pulse width can only be determined when routing and propagation delays are known, after placement and routing. So it is difficult to reliably determine the width of the pulse when creating HDL.
- The pulse may not be wide enough for the application in all PVT conditions.

- **There are some circuits, which generate clock like signals: Pulse.**
- **Pulse may different duty cycles. Classical pulse generators are Multi-vibrators.**

Advanced Digital Design
Avoid Latches

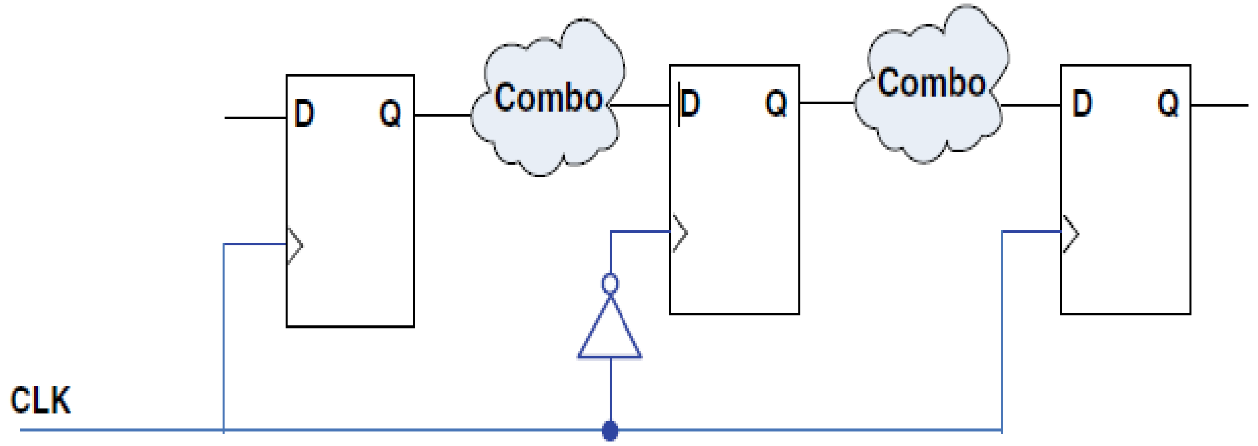


```
always @ (clk)
  begin
    if (clk)
      begin
        a = 1'b1;
        b = 1'b0;
      end
    else
       b = 1'b1;
  end
```



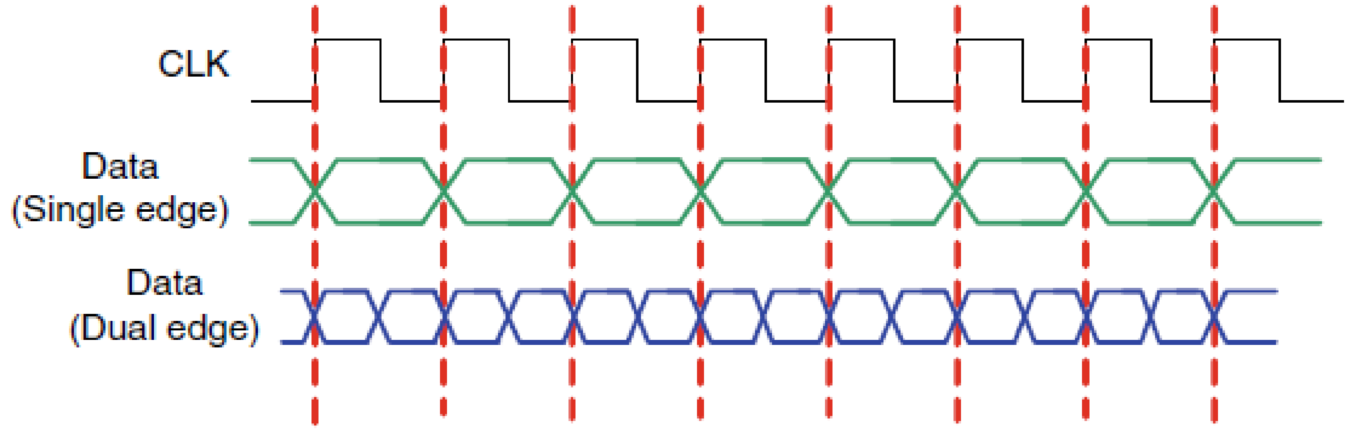
Inferred latch due to incomplete 'if else' statement

Avoid Double Edge Clocking



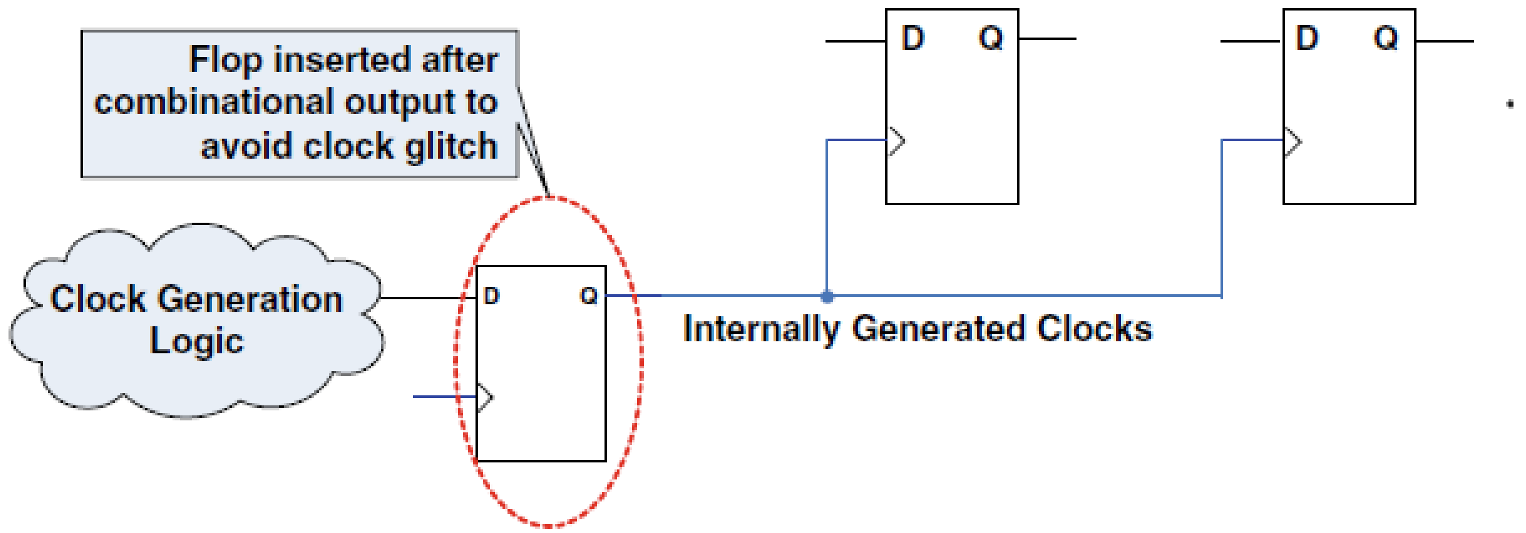
- Double or Dual edged clocking is the method of data transfer on both the rising and falling edges of the clock, instead of just one or the other. The change allows for double the data throughput for a given clock speed.
- Double edge output stage clocking is a useful way of increasing the maximum possible output speed from a design; however this violates the principle of Synchronous circuits and causes a number of problems.
- An asymmetrical clock duty cycle can cause setup and hold violations.
- It is difficult to determine critical signal paths.

Avoid Double Edge Clocking



- Test methodologies such as scan-path insertion are difficult, as they rely on all flip-flops being activated on the same clock edge.
- If scan insertion is required in a circuit with double-edged clocking, multiplexers must be inserted in the clock lines to change to single-edged clocking in test mode.

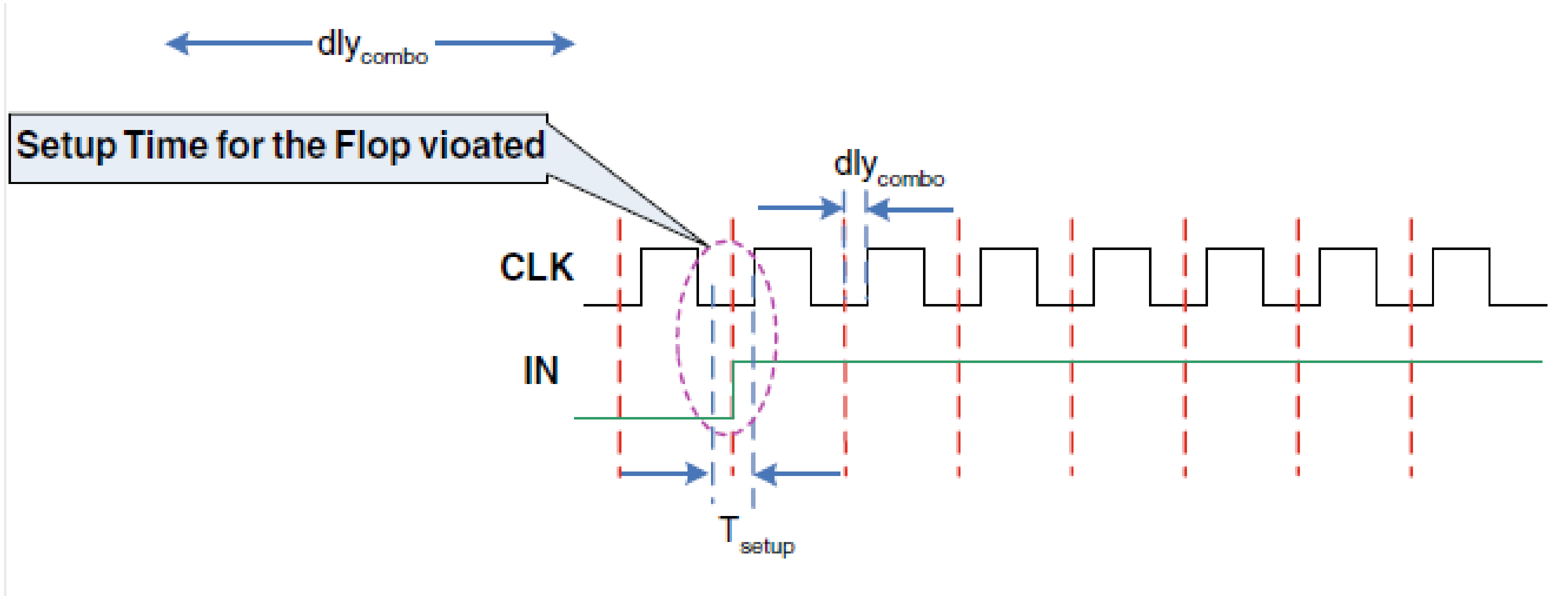
Clock Generation - I


Flop inserted after combinational output to avoid clock glitch
Clock Generation Logic
D
Q
D
Q
D
Q
Internally Generated Clocks

- A simple guideline to the above problem is to always use a registered output of the combinational logic before using it as a clock signal. This registering ensures that the glitches generated by the combinational logic are blocked on the data input of the register

Clock Generation - 2

- The combinational logic used to generate an internal clock also adds delays on the clock line. If the clock skew is greater than the data delay, the timing parameters of the register will be violated and the design will not function correctly.
- Figure shows a similar example where setup time on input "IN" is violated due to skew on the clock path.
- One solution to reduce the clock skew within the clock domain is by assigning the generated clock signal to one of the high-fanout and low-skew clock trees in the SoC.

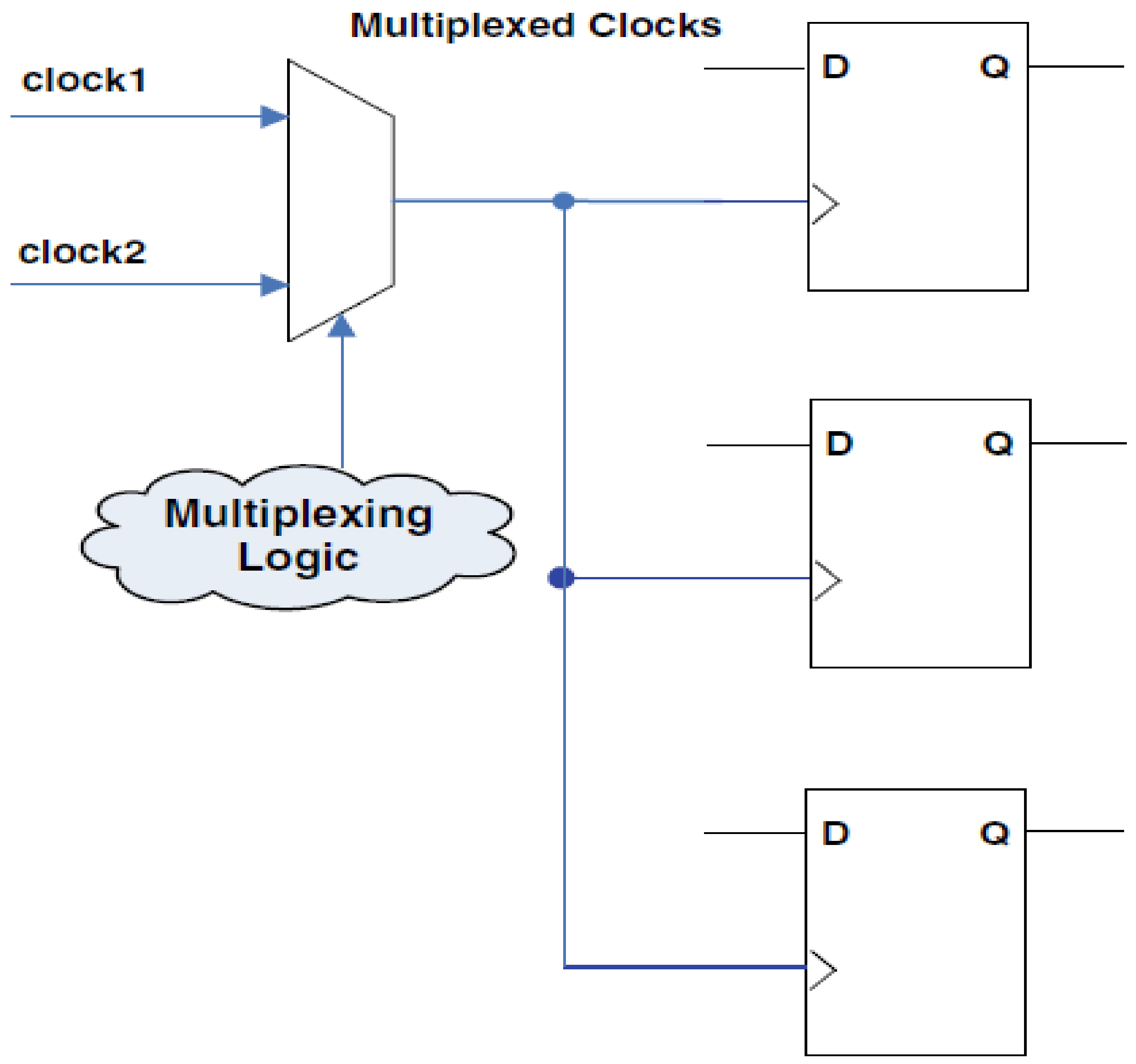


Divided Clocks

- Many designs require clocks created by dividing a master clock.
- Design should ensure that most of the clocks should come from the PLL. When using logic to divide a master clock, always use synchronous counters or state machines.
- In addition, the design should ensure that registers always directly generate divided clock signals.
- Design should never decode the outputs of a counter or a state machine to generate clock signals; this type of implementation often causes glitches and spikes.

Advanced Digital Design

Multiplexed Clocks



Multiplexed Clocks

- Clock multiplexing is acceptable if the following criteria are met:
- The clock multiplexing logic does not change after initial configuration
- The design bypasses functional clock multiplexing logic to select a common clock for testing purposes.
- Registers are always in reset when the clock switches

- A temporarily incorrect response following clock switching has no negative consequences.
- If the design switches clocks on the fly with no reset and the design cannot tolerate a temporarily incorrect response of the chip, then one must use a synchronous design so that there are no timing violations on the registers, no glitches on clock signals, and no race conditions or other logical problems.

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-II: Metastability and Resets

Sudeendra kumar K

Department of Electronics and Communication Engineering

Contents

- Reset Handling
- Theory of Metastability

Resets

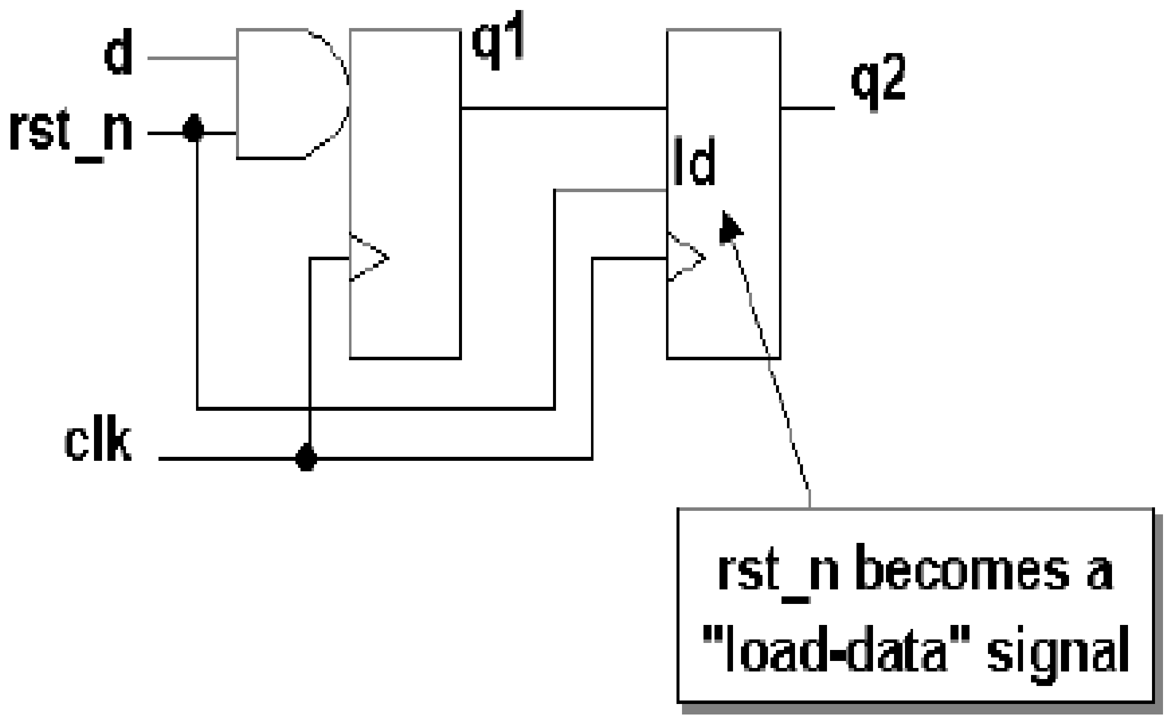
- The primary purpose of a reset is to force the SoC into a known state for stable operations. This would avoid the SoC to power on to a random state and get hanged.
- In some cases, when pipelined flip-flops (shift register flip-flops) are used in high speed applications, reset might be eliminated from some flip-flops to achieve higher performance designs.
- A design may choose to use either an Asynchronous or Synchronous reset or a mix of two.
- **Synchronous Reset: -**
    - Synchronous resets are based on the premise that the reset signal will only affect or reset the state of the flip-flop on the active edge of a clock.

Advanced Digital Design

Synchronous reset flip-flops with non reset follower flip-flops

```
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
```

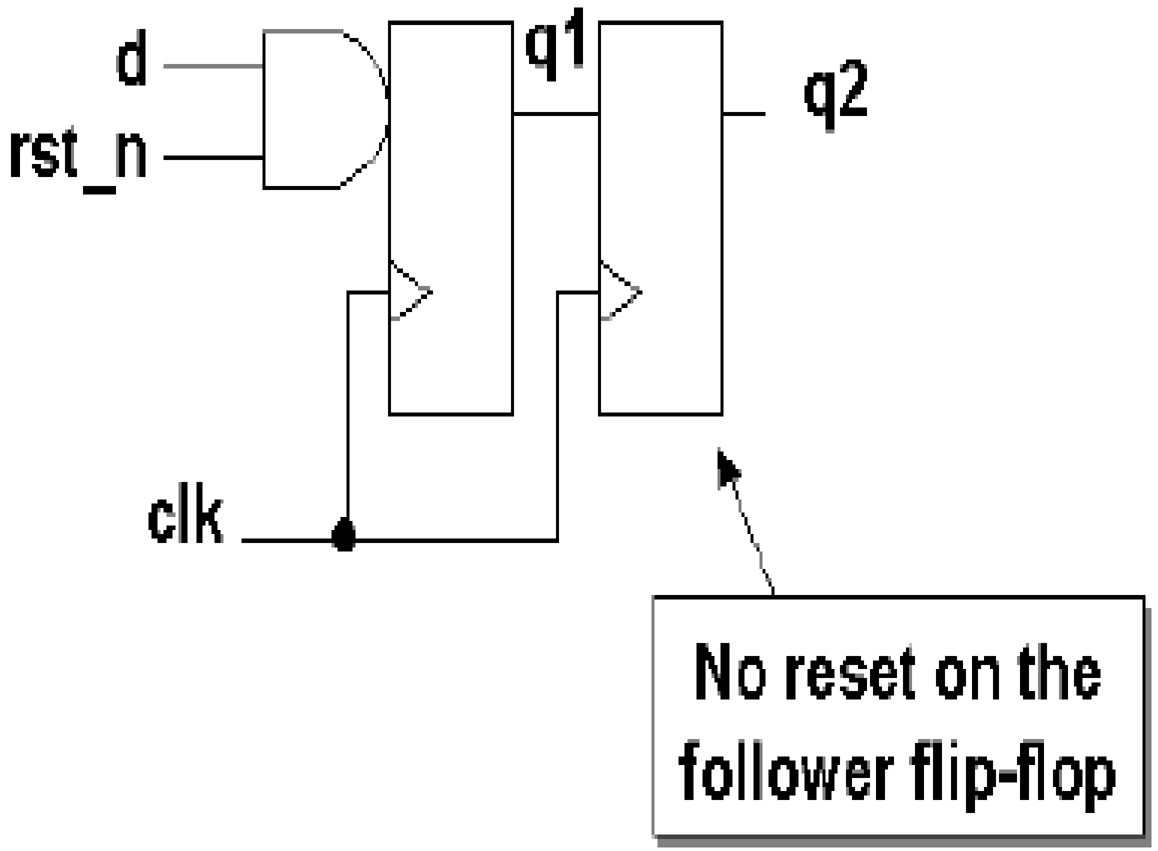


Synchronous reset flip-flops with non reset follower flip-flops

```
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
```



Modeling Verilog flip-flops with asynchronous reset and asynchronous set

```
// 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
```

Advantages of Using Synchronous Resets

- Synchronous resets generally insure that the circuit is 100% synchronous.
- Synchronous resets ensure that reset can only occur at an active clock edge. The clock works as a filter for small reset glitches.
- In some designs, the reset must be generated by a set of internal conditions. Synchronous reset is recommended for these types of designs because it will filter the logic equation glitches between clocks.

Disadvantages of Using Synchronous Resets

- Synchronous resets may need a pulse stretcher to guarantee a reset pulse width wide enough to ensure reset is present during an active edge of the clock. This is an issue that is important to consider when doing multi-clock design. A small counter can be used that will guarantee a reset pulse width of a certain number of cycles.
- A potential problem exists if the reset is generated by combinational logic in the SoC or if the reset must traverse many levels of local combinational logic.
- A synchronous reset will require a clock in order to reset the circuit. This may be a problem in some case where a gated clock is used to save power. Clock will be disabled at the same time during reset is asserted.

Advantages of Using Asynchronous Resets

- The biggest advantage to using asynchronous resets is that, as long as the vendor library has asynchronously reset-able flip-flops, the data path is guaranteed to be clean.
- Designs that are pushing the limit for data path timing, cannot afford to have added gates and additional net delays in the data path due to logic inserted to handle synchronous resets. Using an asynchronous reset, the designer is guaranteed not to have the reset added to the data path.
- The most obvious advantage favoring asynchronous resets is that the circuit can be reset with or without a clock present.

Disadvantages of Using Asynchronous Resets

For DFT, if the asynchronous reset is not directly driven from an I/O pin, then the reset net from the reset driver must be disabled for DFT scanning and testing.

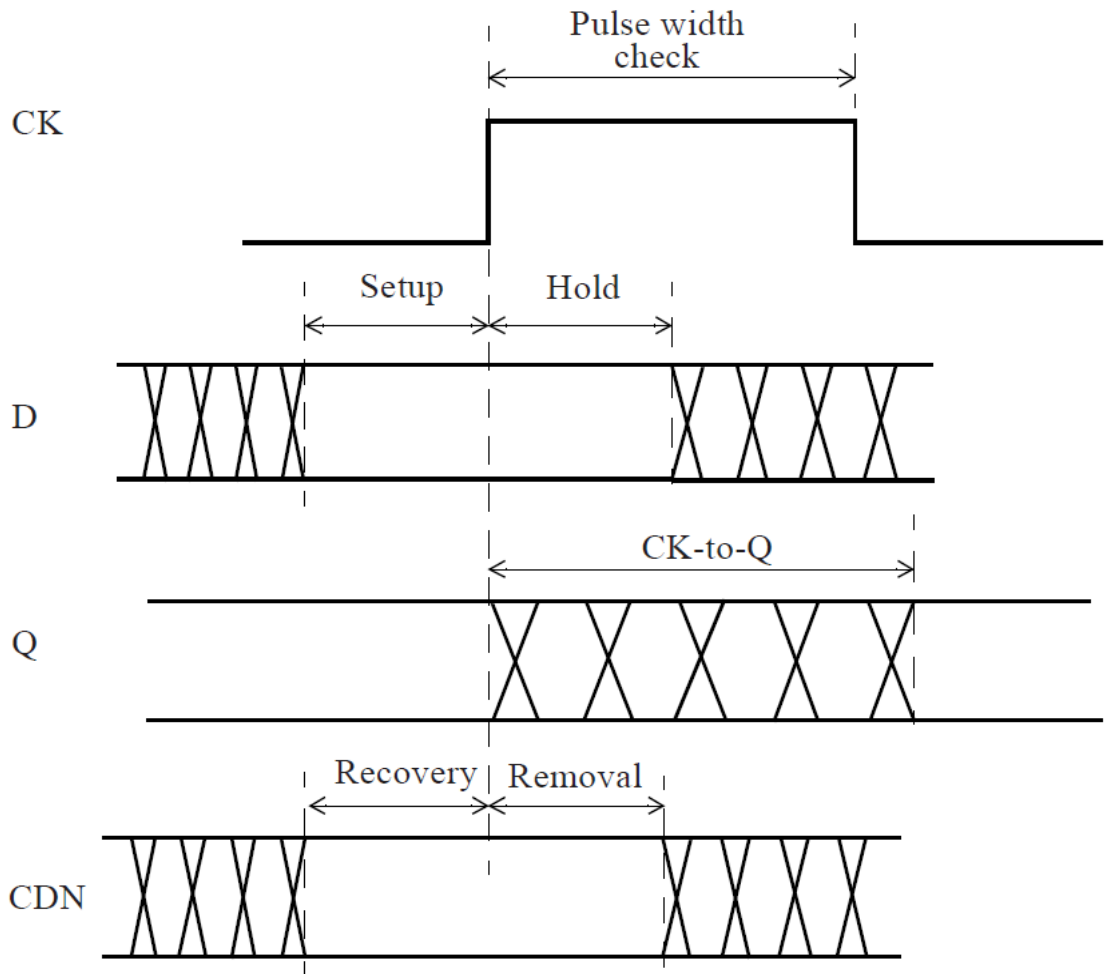
If the asynchronous reset is released at or near the active clock edge of a flip-flop, the output of the flip-flop could go metastable and thus the reset state of the SoC could be lost.

Another problem that an asynchronous reset can have, depending on its source, is spurious resets due to noise or glitches on the board or system reset.

The reset tree must be timed for both synchronous and asynchronous resets to ensure that the release of the reset can occur within one clock period.

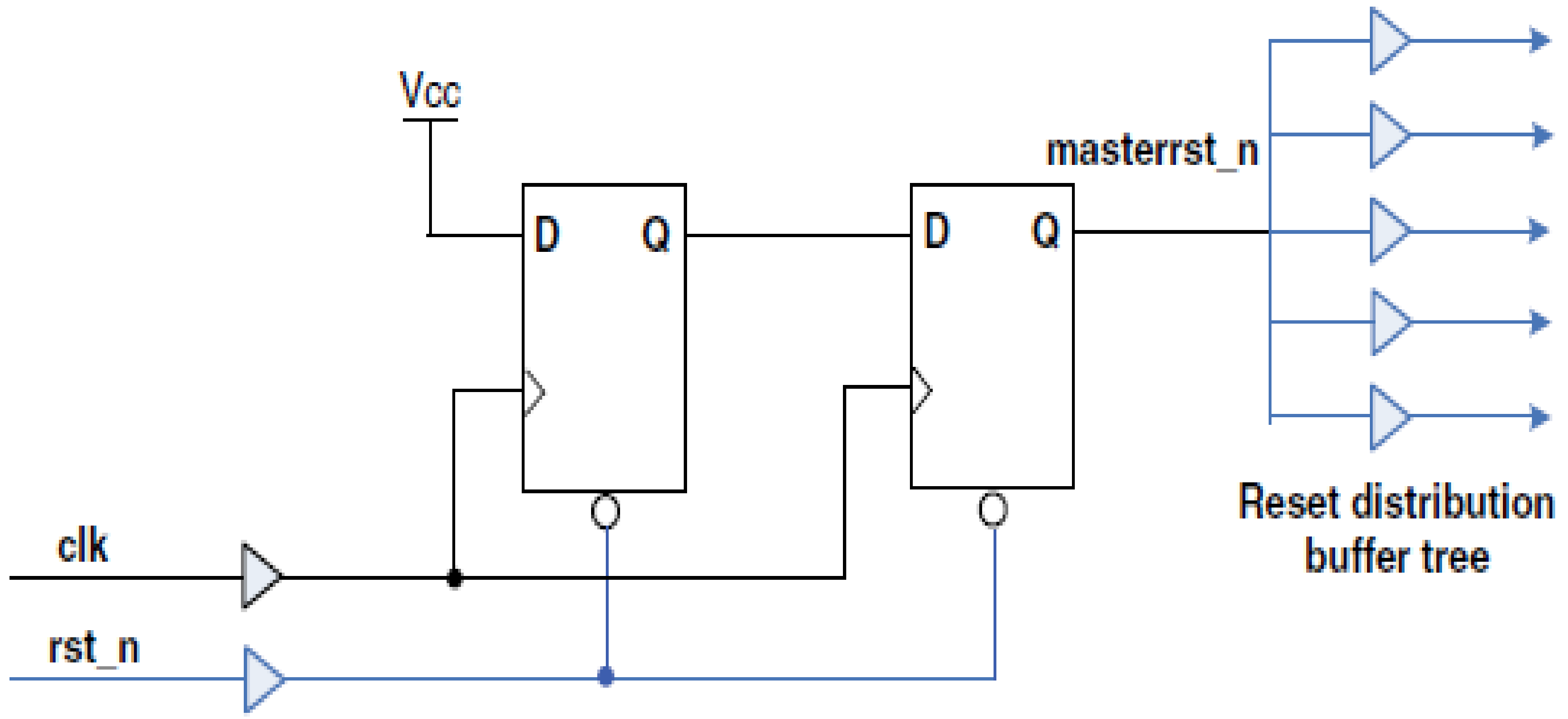
Asynchronous Reset Removal Problem

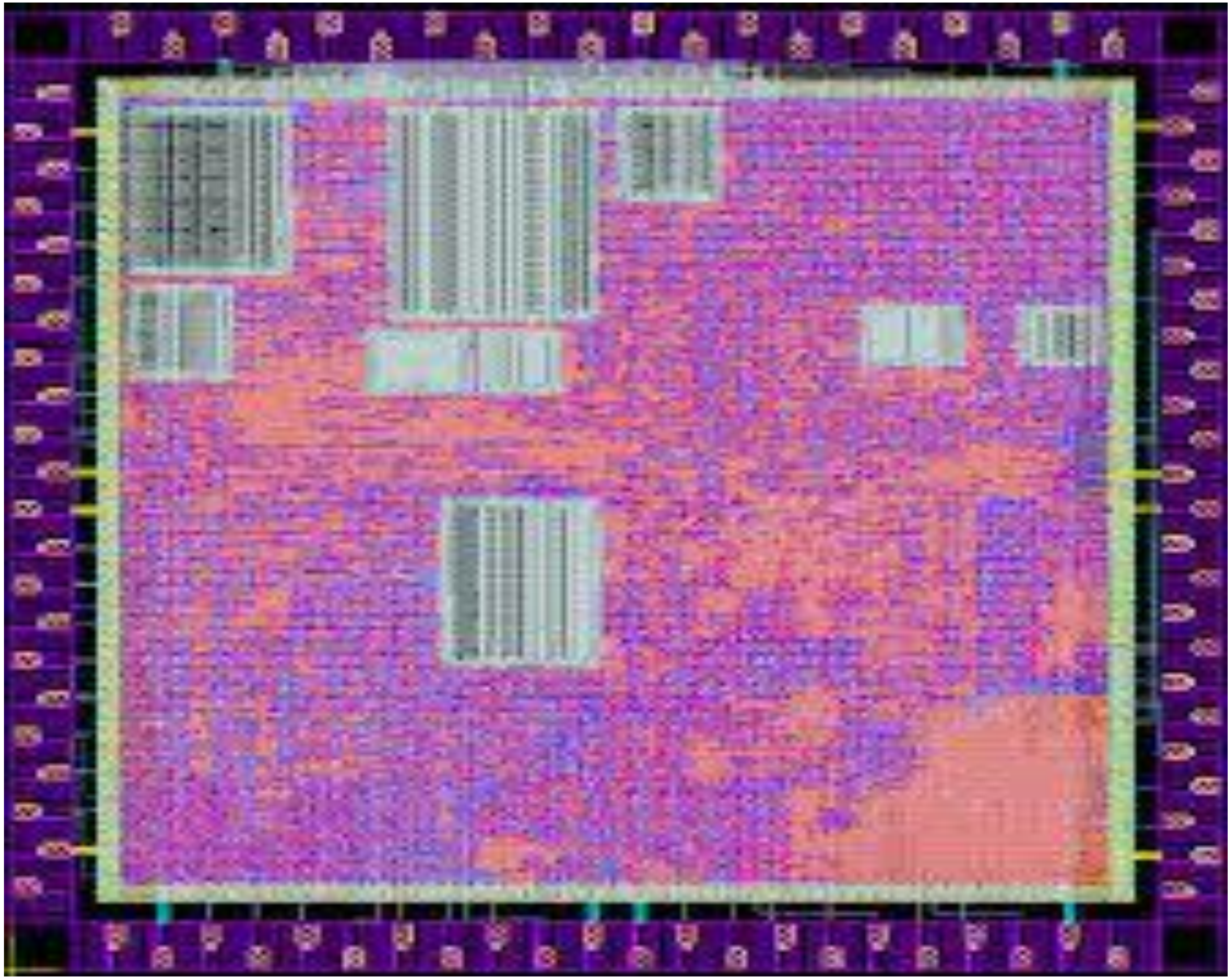
- Releasing the Asynchronous reset in the system could cause the chip to go into a metastable unknown state, thus avoiding the reset all together.
- There are two potential problems when an asynchronous reset signal is de-asserted asynchronous to the clock signal: -
- Violation of reset recovery time: Reset recovery time refers to the time between when reset is de-asserted and the time that the clock signal goes high again. Missing a recovery time can cause signal integrity or Metastability problems with the registered data outputs.
- Reset removal happening in different clock cycles for different sequential elements. When reset removal is asynchronous to the rising clock edge, slight differences in propagation delays in either or both the reset signal and the clock signal can cause some registers or flip-flops to exit the reset state before others.



Reset Synchronizer

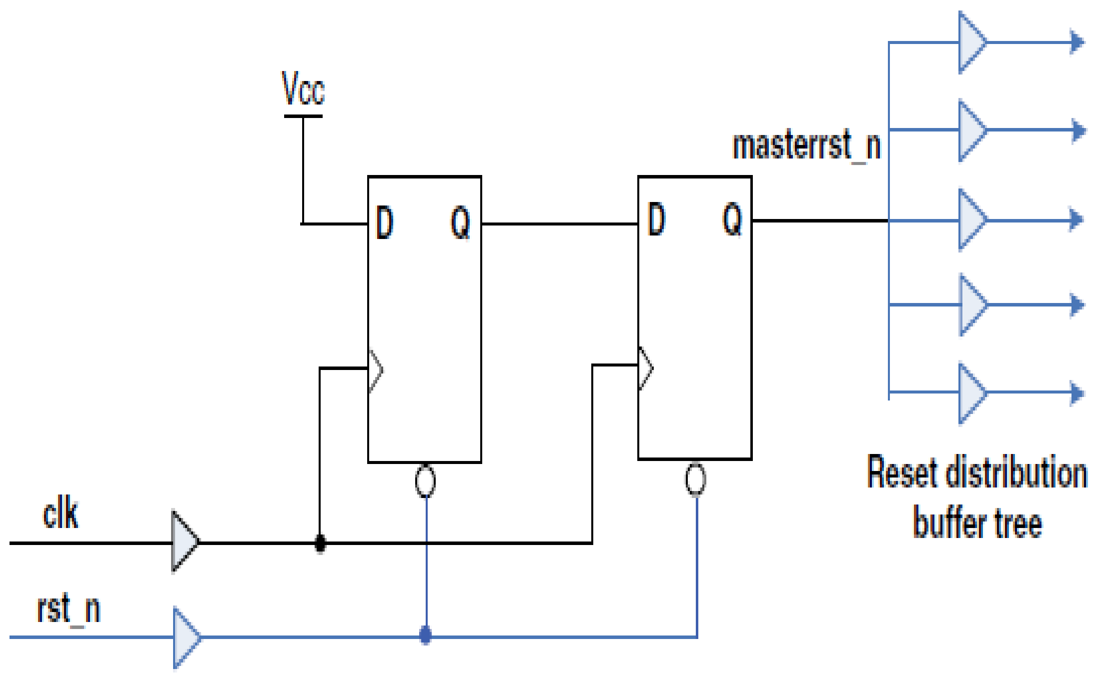
- Solution to asynchronous reset removal problem is to use a Reset Synchronizer.
- Without a reset synchronizer, the usefulness of the asynchronous reset in the final system is void even if the reset works during simulation.





Reset Synchronizer

- An external reset signal asynchronously resets a pair of flip-flops, which in turn drive the master reset signal asynchronously through the reset buffer tree to the rest of the flip-flops in the design.
- The entire design will be asynchronously reset.
- Reset removal is accomplished by de-asserting the reset signal, which then permits the d-input of the first master reset flip-flop (which is tied high) to be clocked through a reset synchronizer. It typically takes two rising clock edges after reset removal to synchronize removal of the master reset.
- Two flip-flops are required to synchronize the reset signal to the clock pulse where the second flip-flop is used to remove any metastability that might be caused by the reset signal being removed asynchronously and too close to the rising clock edge.



Reset Synchronizer

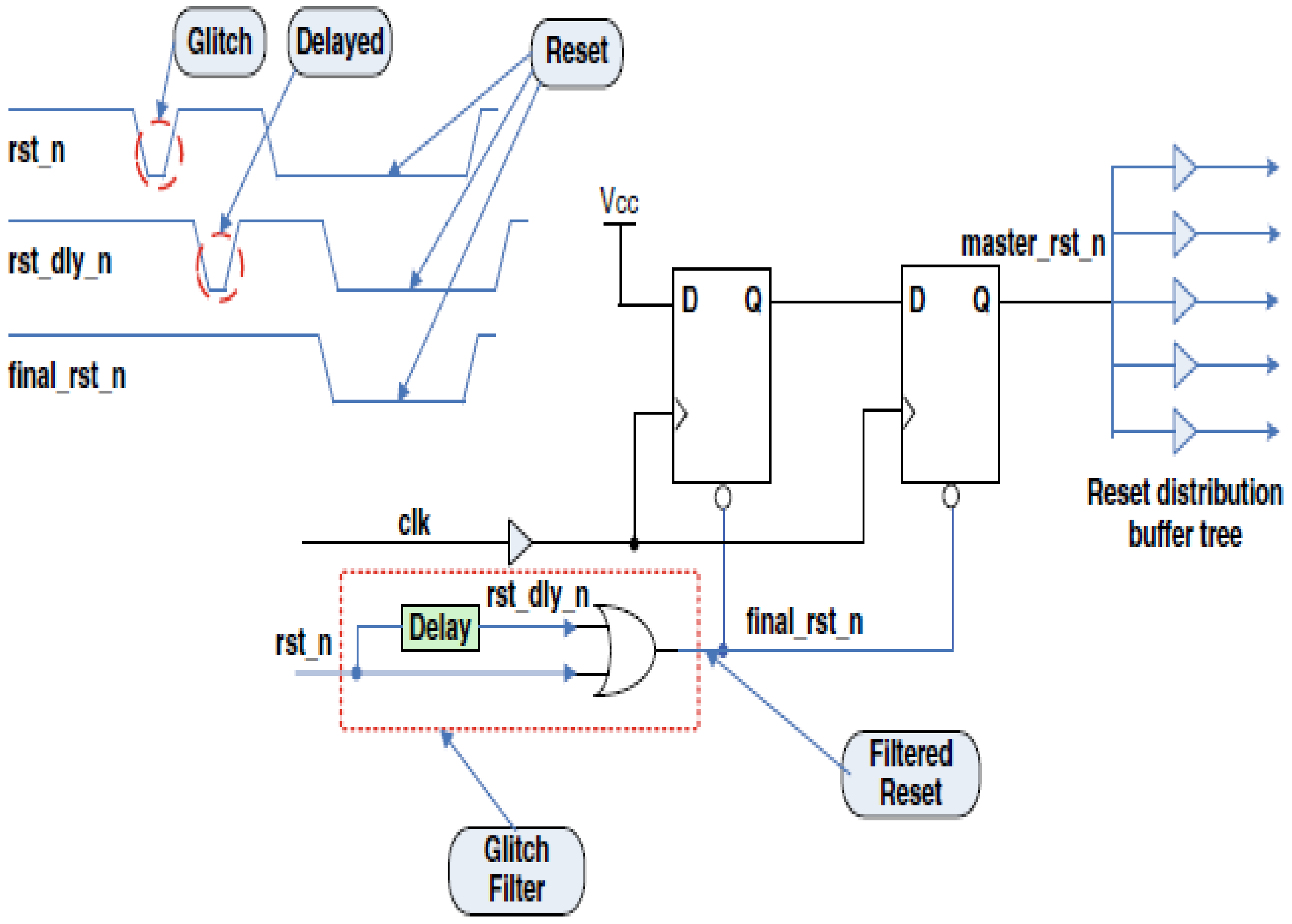
- The first flip-flop of the reset synchronizer does have potential metastability problems because the input is tied high, the output has been asynchronously reset to a 0 and the reset could be removed within the specified reset recovery time of the flip-flop (the reset may go high too close to the rising edge of the clock input to the same flip-flop). This is why the second flip-flop is required.
- The second flip-flop of the reset synchronizer is not subjected to recovery time metastability because the input and output of the flip-flop are both low when reset is removed.
- The following equation calculates the total reset distribution time: -

$$T_{rst\_dis} = t_{clk\text{ - }q} + t_{pd} + t_{rec}$$

  - where
    - tclk-q = Clock to Q propagation delay of the second flip flop in the reset synchronizer
- tpd = Total delay through the reset distribution tree
- trec = Recovery time of the destination flip flop

Reset Glitch Filtering

- Asynchronous Reset are susceptible to glitches, that means any input wide enough to meet the minimum reset pulse width for a flip-flop will cause the flip-flop to reset.
- Since this approach uses delay lines, one of the disadvantages is that this delay would vary with temperature, voltage and process. Care must be taken to make sure that the delay meets the design requirements across all PVT corners.



Theory of Metastability

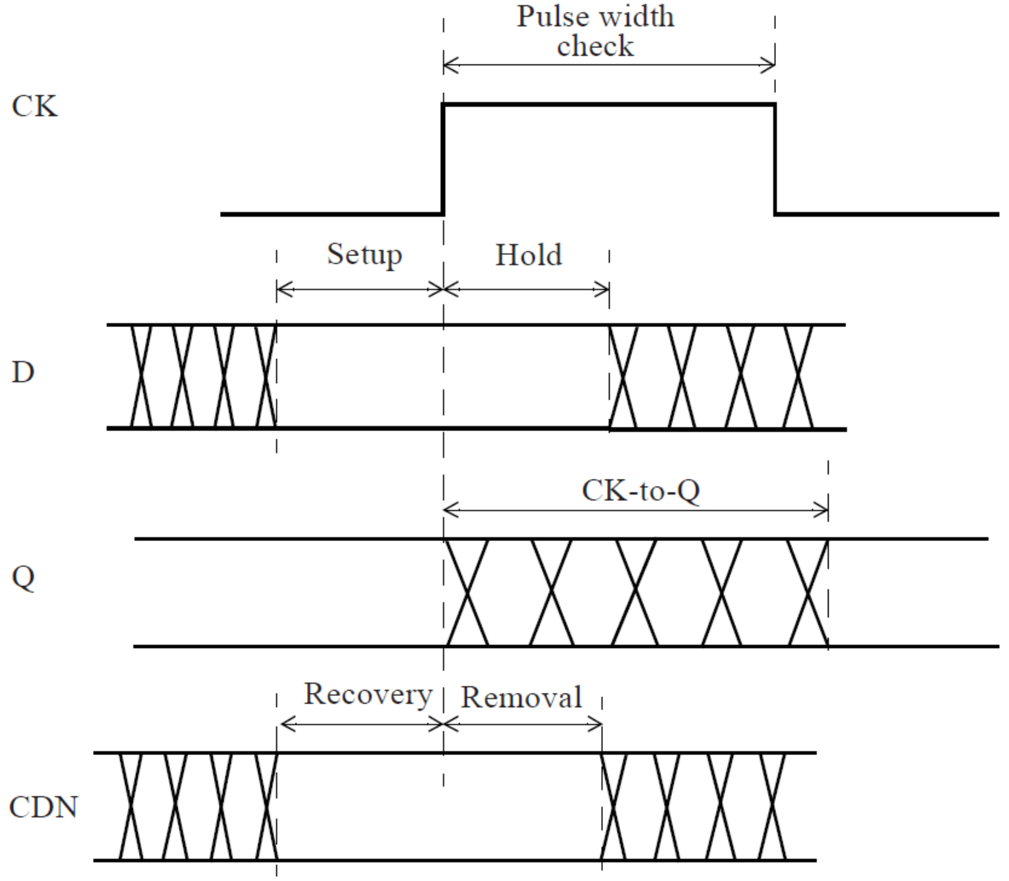
- Metastability arises as a result of violation of setup and hold times of a flip flop.
- This propagation of unwanted state is called Metastability. As a result the output of a flip-flop can produce a glitch or remain temporarily in metastable state, thus taking longer to return to stable state.
- When a flip-flop is in a metastable state, the output hovers at a voltage level between high and low, causing the output transition to be delayed beyond the specified clock-to-output delay (tco).
- The additional time beyond tco that a metastable output takes to resolve to a stable state is called the settling time (tMET).

Theory of Metastability

- Not every transition that violates the setup or hold times results in a metastable output.
- When a signal is changing in one clock domain (src_data_out) and is sampled in another clock domain (dest_data_in), then this causes the output to become metastable. This is known as Synchronization Failure.

Advanced Digital Design
Metastability Window

- Metastability Window is defined as the specific length of time, during which both the data and clock should not change. If both signals do occur, the output may go metastable.
- The larger the window, the greater the chance the device will go Metastable.
- In most cases, newer logic families have smaller Metastability windows which reduce the chance of the device going Metastable.



Calculating MTBF

- Mean (Average) Time Between Failures or MTBF of a system, is the reciprocal of the failure rate in the special case when the failure rate is constant.

$$\frac{1}{FailureRate} = \mathrm{MTBF}_1 = \frac{e(t_r/\tau)}{W \times f_c \times f_d}$$

where

t_r = resolve time allowed in excess of the normal propagation delay time of the device

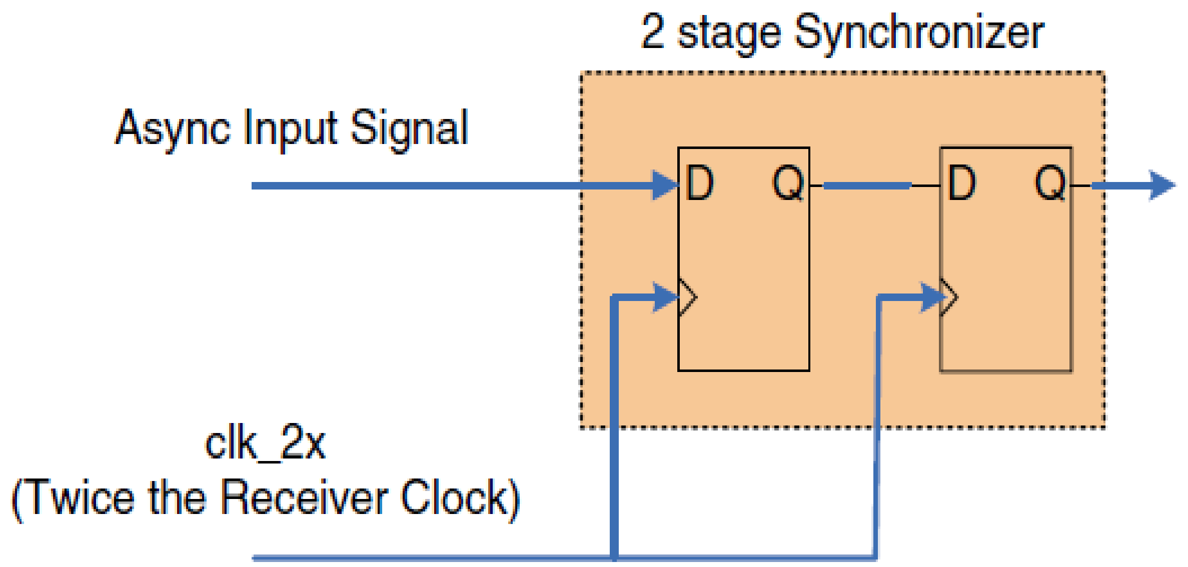
τ = metastability (resolving) time constant for a flip-flop

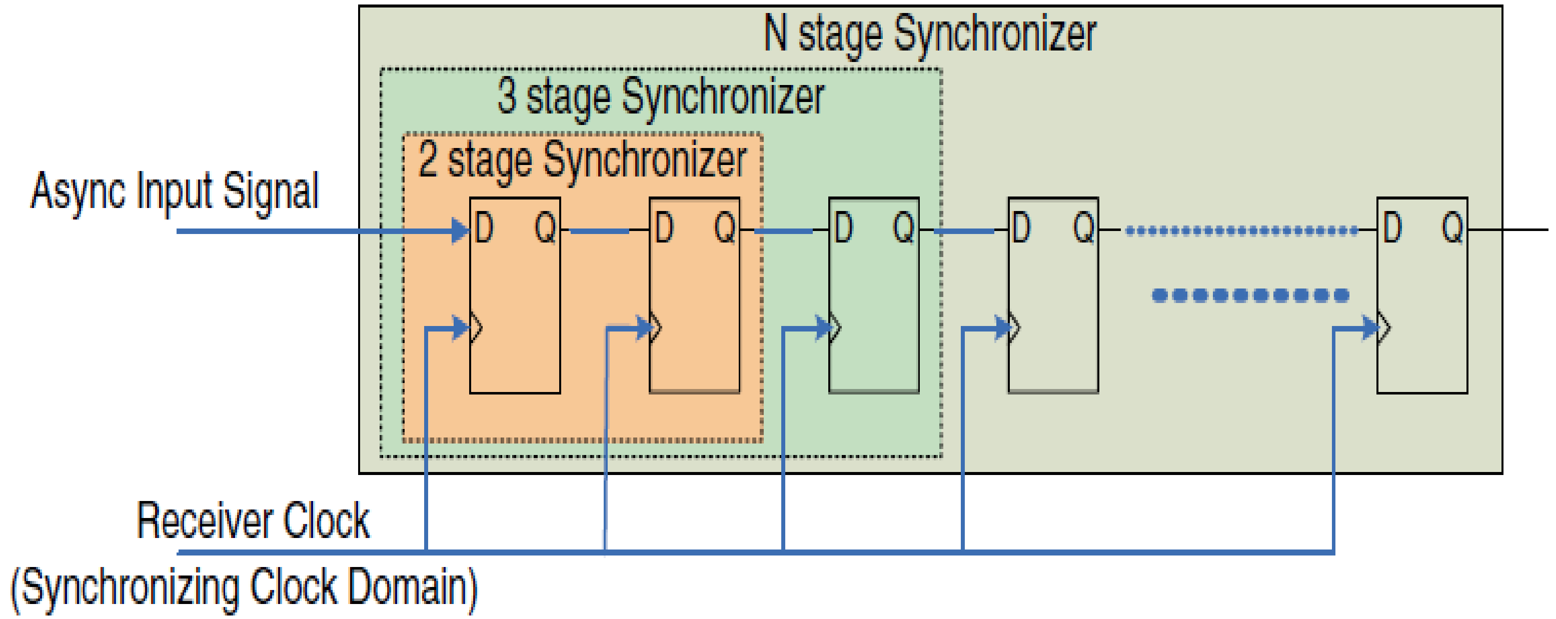
W = Metastability Window

f_c = Clock frequency

f_d = Asynchronous data edge frequency

Metastability Synchronizers





Metastability Synchronizers

- The most common way to avoid metastability is to add one or more synchronizing flip-flops at the signals that move from one clock domain to the other.
- These approaches can guarantee that metastability cannot pass through the synchronizer; they simply reduce the probability of occurrence of metastability.

Calculating MTBF

- The constants W and τ are related to electrical characteristics of the device and may vary according to the process technology node. Therefore, different devices manufactured with the same process have similar values for W and τ.

$$\tau = \frac{t_{r2} - t_{r1}}{\ln(N1 / N2)}$$

where

t_{r1} = resolve time 1
t_{r2} = resolve time 2
$N1$ = numbers of failures at tr1
$N2$ = numbers of failures at tr2

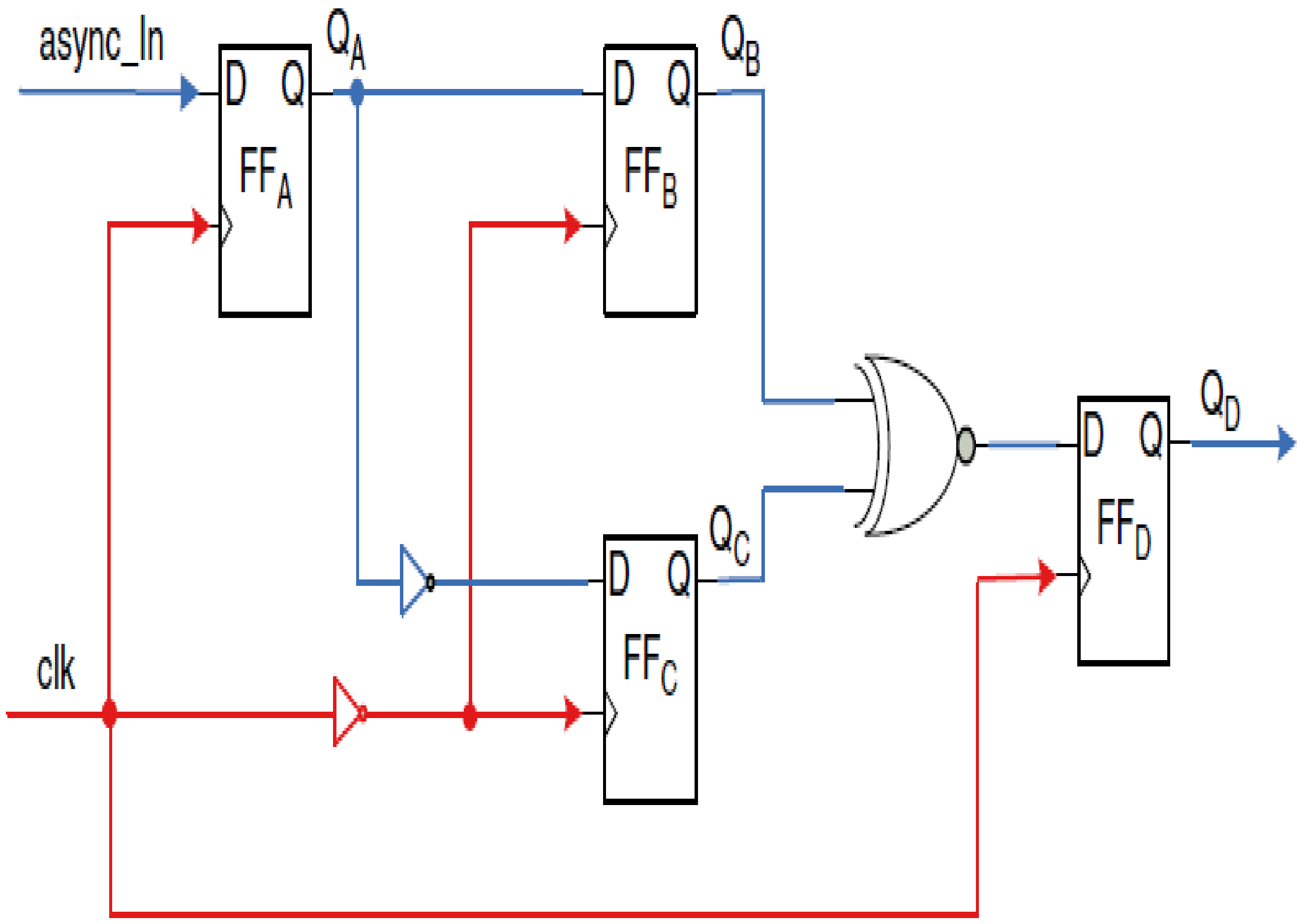
$$\mathrm{MTBF}_2 = \frac{e(t_{r1} / \tau)}{W \times f_c \times f_d} \times e(t_{r2} / \tau)$$

where

t_{r1} = resolve time allowed for the first stage of synchronizer
t_{r2} = resolve time in access of normal propagation delay

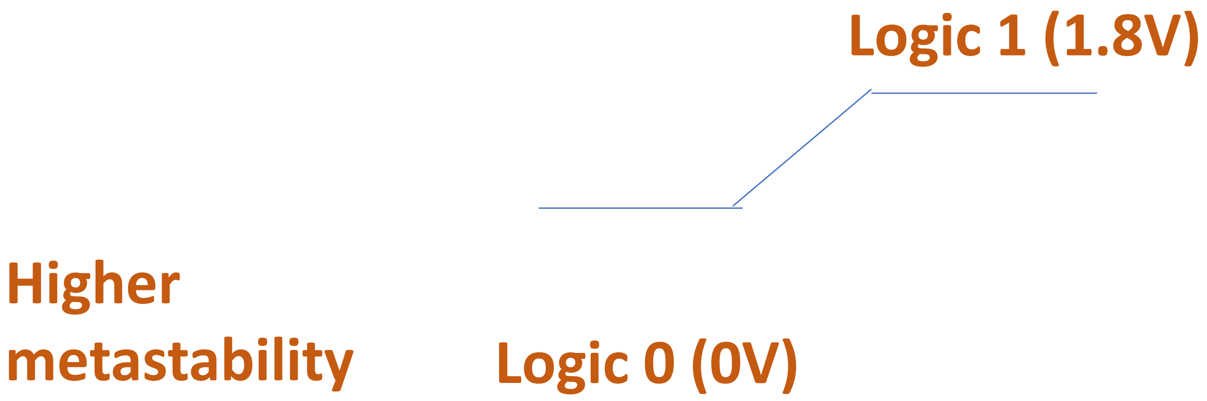
Metastability Test Circuitry

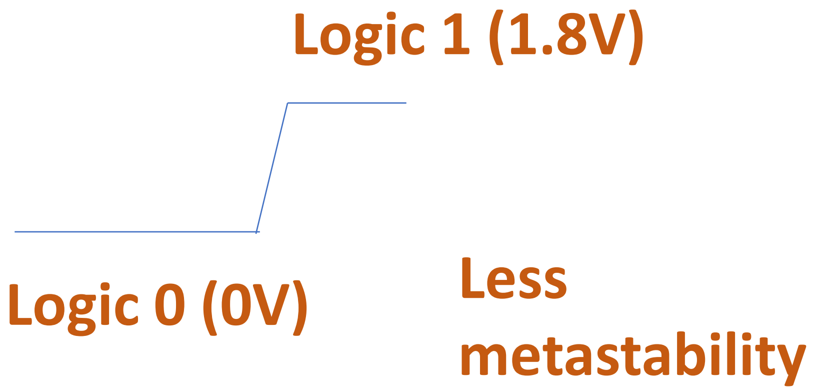
- Whenever a flip-flop samples an asynchronous input, a small probability exists that the flip-flop output will exhibit an unpredictable delay.
- This happens not only when the input transition violates setup and hold time specifications, but also when the transition actually occurs within a small timing window during which the flip-flop accepts the new input. Under these circumstances, the flip-flop can enter a metastable state.



Recommendations for Avoiding Metastability

- Use Synchronizers.
- Use Faster Flip Flops (narrower metastable window TW).
- Use metastable hardened Flip Flops (designed for very high bandwidth and reduced sampling times that are optimized for clock domain input circuitry).
- Cascade flip-flops as Synchronizers (two or more)
- Reduce Sampling rate.
- Avoid input signals with low dV/dt.





Advanced Digital Design

References

- Mohit Arora, “The Art of Hardware Design”, Springer

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-III: The Art of Pipelining

Sudeendra kumar K

Department of Electronics and Communication Engineering

Contents

Pipelining

- Pipelining splits the critical path (path with maximum combinational delay) with memory elements between the clock cycles.
- This reduces the delay of each stage in the critical path and thus a circuit can operate at higher clock frequency.
- Pipelining a circuit increases the calculations per second since the clock period per stage is reduced but increases the overhead by adding memory elements.

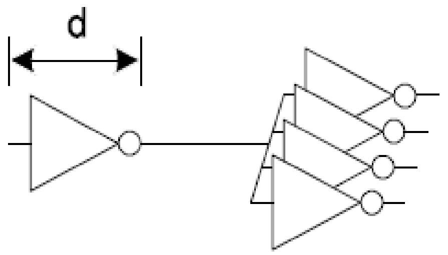
Advanced Digital Design

FO4 (Fan –out of 4)

- Digital circuit delays vary with feature size, process corner, operating voltage, and junction temperature. Delays are steadily decreasing with advances in process technology, so comparing results reported in nanoseconds between process generations is difficult. The delay of a fanout-of-4 inverter (FO4) to normalize process and operating condition variations and quantifies how well this normalization works.

- **UMC :180nm, 90nm, 40nm….**
- **MUX (180nm) = 12ps**
- **MUX (90nm) = 8ps**
- **Delay:**
  - **Input signal transition**
  - **Output Capacitance**

Estimate the delay of a fanout-of-4 (FO4) inverter



The FO4 delay is about

200 ps in 0.6 μm process

60 ps in a 180 nm process

$f/3$ ns in an f μm process

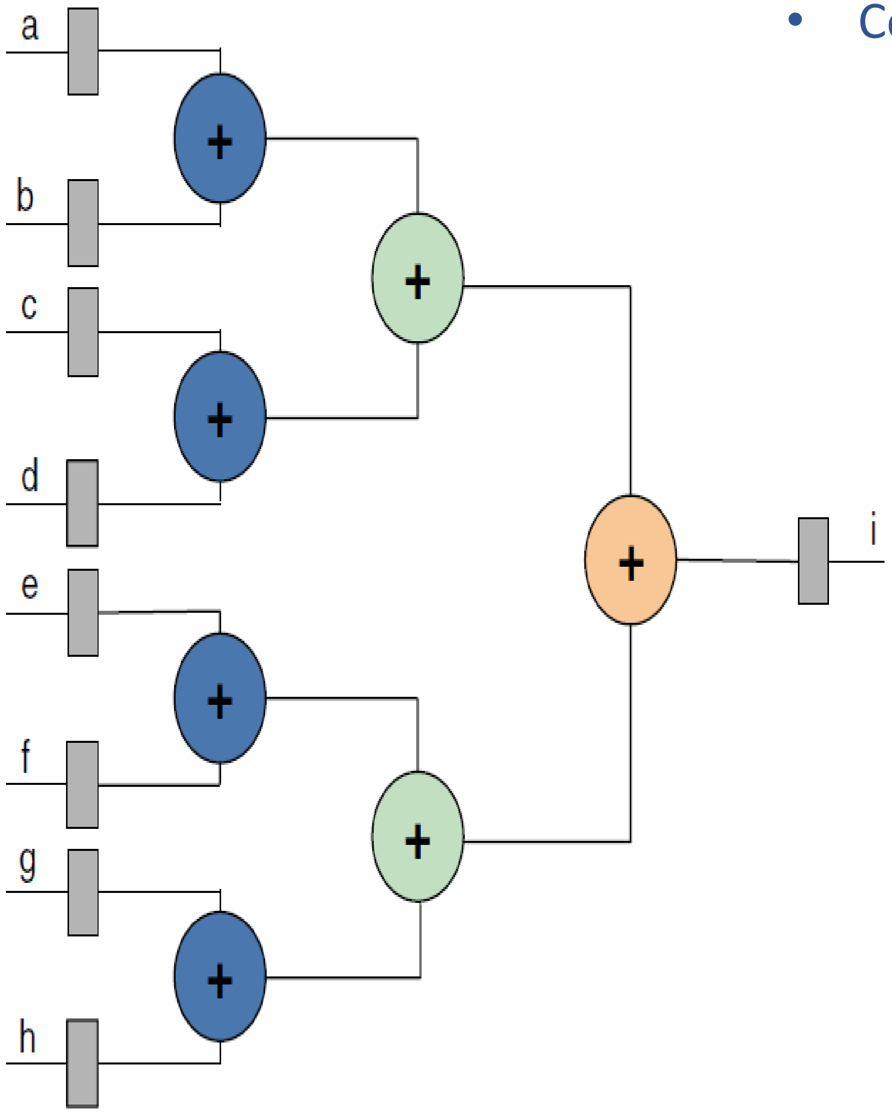
Logical Effort:	$g = 1$
Electrical Effort:	$h = 4$
Parasitic Delay:	$p = 1$
Stage Delay:	$d = gh + p = 5$

- Reporting the FO4 delay of a process along with any circuit performance results (both taken at the same operating conditions) will facilitate comparing the circuit to alternative implementations in other processes.

Reference

"The Fanout-of-4 Inverter Delay Metric", by David Harris, Ron Ho, Gu-Yeon Wei, and Mark Horowitz, Stanford University

Pipelining



Eight Input Adder Before Pipelining

- Consider the circuit shown in Figure performing the operation

$$i=(a+b+c+d)+(e+f+g+h)$$

$$\{T_{FF}\}_{max}=T_{CQ}+T_{stp}+T_{SKW}+T_{JIT}+\{T_{combo}\}_{max}$$

Here $\{T_{combo}\}_{max}=3*T_{adder}$
So the final value of clock period is

$$\{T_{FF}\}_{max}=T_{CQ}+T_{stp}+T_{SKW}+T_{JIT}+\{3*T_{adder}\}$$

Assuming the following values of the constraints

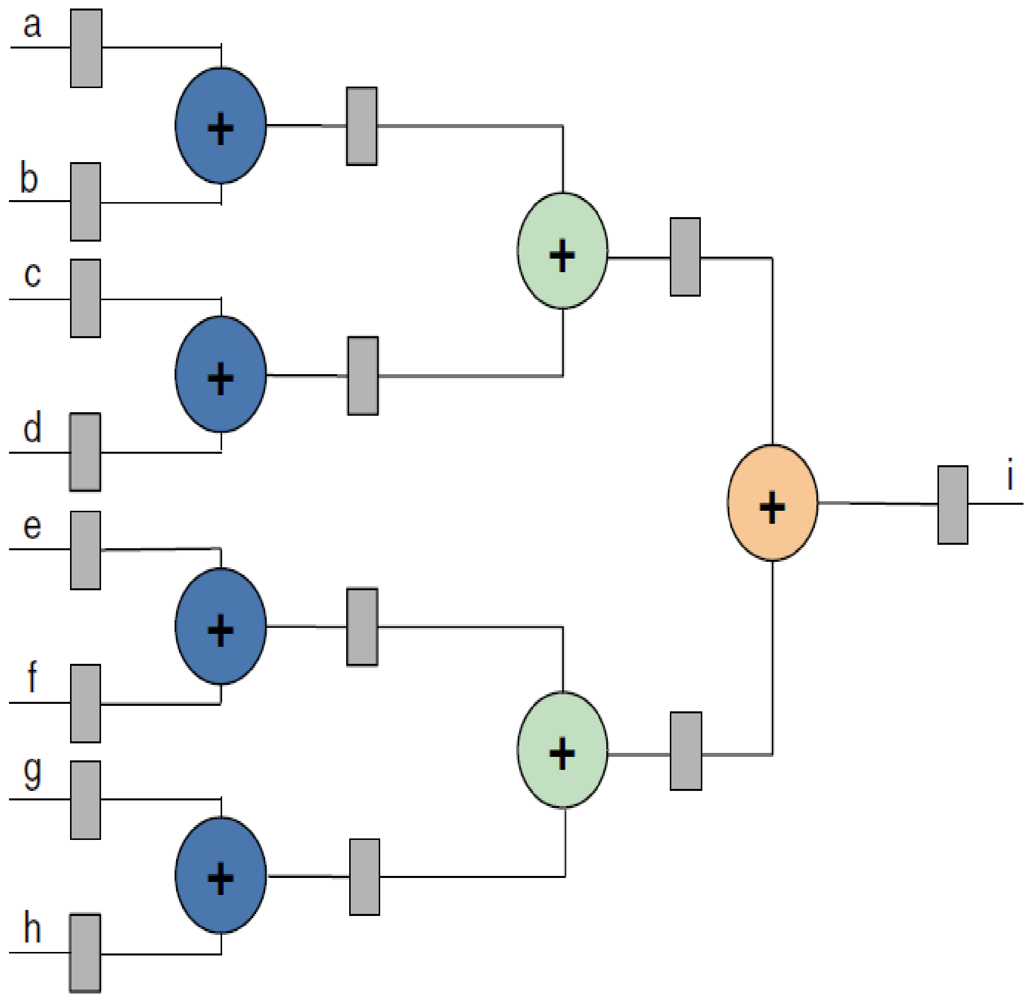
$$T_{CQ}=4\text{ FO4}$$
$$T_{stp}=2\text{ FO4}$$
$$T_{SKW}+T_{JIT}=4\text{ FO4}$$
$$T_{adder}=10\text{ FO4}$$

where FO4 is Fan-out of 4 inverter delay.

$$\{T_{FF}\}_{max}=4+2+4+3*10$$
$$=40\text{ FO4}$$

Advanced Digital Design

Pipelining



The delay for each pipeline state is as below

$$\{T_{FF}\}_{max} = T_{CQ} + T_{stp} + T_{SKW} + T_{JIT} + \{T_{combo}\}_{max}$$

here $\{T_{combo}\}_{max} = 1 * T_{adder}$

Note: *Instead of three adders we just have a single adder between any two flip-flops.*

$$\{T_{FF}\}_{max} = T_{CQ} + T_{stp} + T_{SKW} + T_{JIT} + \{1 * T_{adder}\}$$

$$\{T_{FF}\}_{max} = 4 + 2 + 4 + 1 * 10$$

$$= 20 \text{ FO4}$$

Eight Input Adder after Pipelining

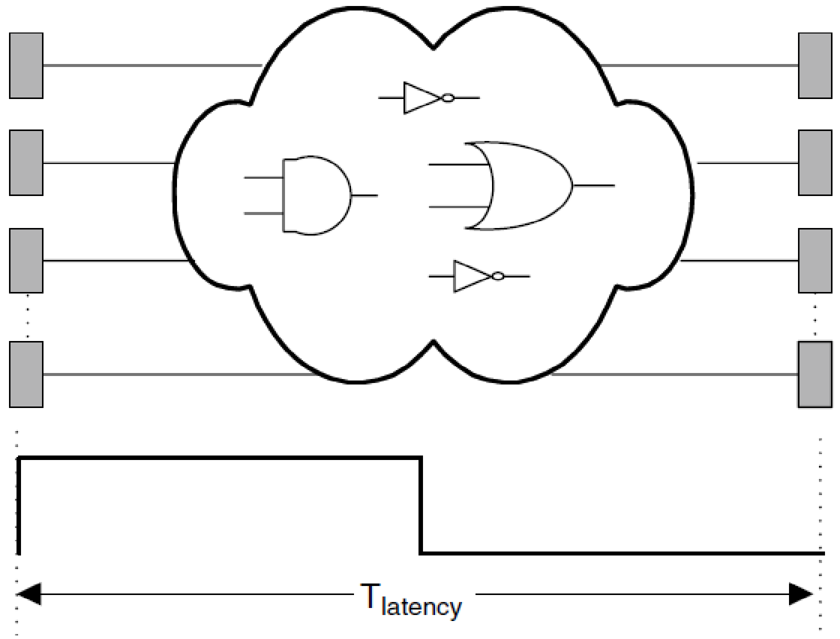
Pipelining

- By implementing the sum of eight inputs directly with seven pipelined adders (as shown in Fig), the throughput (number of calculations per clock cycle) is increased to calculate one eight input sum per clock cycle. The latency for the summation is three clock cycles.
- Compared to using a single adder to perform the above calculation, seven adders would mean at least seven times the area and power consumption.
- The area and the power cost of parallelization the circuit is substantial. Generally computing the same operation k times in parallel increases the power and the area by the replicated logic by more than a factor of k, as there is more wiring due to greater number of flip-flops and extra logic.

Advanced Digital Design

Performance Increase from Pipelining

- Consider Figure. below, as a big array of combinational logic between flip-flops registers. The latency of the pipeline is the time from the arrival of the pipeline inputs to the pipeline, to the exit of the pipeline outputs corresponding to a given set of inputs. The logic in Figure below just includes a single pipeline stage (also called an unpipelined stage). The latency of the above circuit is also the clock period



Logic before Pipelining (Non-pipelined Circuit)

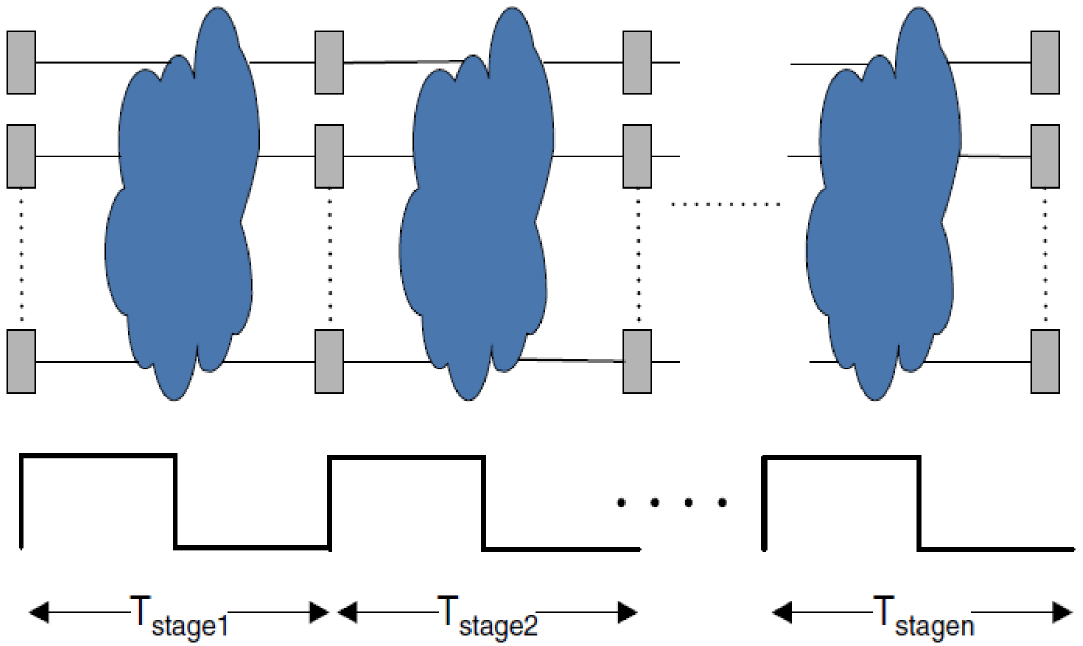
$$T_{latency} = T_{comb} + T_{register} + T_{clocking}$$

Where $T_{register}$ is the register overhead $= T_{CQ} + T_{stp}$

$T_{clocking}$ is the clocking overhead $= T_{SKW} + T_{JIT}$

Performance Increase from Pipelining

- Consider the same circuit to be pipelined into n stages of combinational logic between registers as shown in Figure below.



Logic after Pipelining into 'n' stages

Period for any stage

$$T_{stage} = (T_{comb})_{stage} + T_{register} + T_{clocking}$$

Pipeline stage with worst delay limits the clock period so that the clock period for any state is

$$T_{pipeline} = \max\{T_{comb}\} + T_{register} + T_{clocking}$$

The latency is n times the clock period, as the delay through each state is the clock period

$$T_{latency} = n \times T_{pipeline}$$

Performance Increase from Pipelining

- Ideally, each pipeline stage should have equal delay so that the max combinational logic delay in any pipeline is the same

$$T_{comb_i} = \frac{T_{comb}}{n}$$

So the minimum possible clock period for any pipeline stage

$$T_{pipeline} = \frac{T_{comb}}{n} + T_{register} + T_{clocking}$$

Thus the final latency with this ideal clock period is

$$T_{pipeline\_ideal} = nT_{pipeline} = T_{comb} + n\,(T_{register} + T_{clocking})$$

We can now calculate the speed increase of a circuit after pipelining

$$Speed\ Increase = \frac{F_{after}}{F_{before}}$$

Where F_{after} is the clock frequency of the circuit after pipelining
F_{before} is the clock frequency of the unpipelined circuit

$$Speed\ Increase = \frac{T_{before}}{T_{after}}$$

Performance Increase from Pipelining

$$Speed\ Increase = \frac{T_{before}}{T_{after}}$$

$$= \frac{T_{comb} + T_{register} + T_{clocking}}{\left(\frac{T_{comb}}{n} + T_{register} + T_{clocking}\right)}$$

If we specify the register and clock overhead as a fraction **k** of the total clock period of an unpipelined circuit, then we have:

$$k = \frac{T_{register} + T_{clocking}}{T_{comb} + T_{register} + T_{clocking}}$$

$$Speed\ Increase = \frac{1}{\left(\frac{1-k}{n}\right) + k}$$

Throughput of a system can be defined as calculations completed per clock cycle.

Performance Increase from Pipelining

Hence performance of a pipelined system can be defined as

$$= \frac{Average\,calculation\,time\,per\,instruction\,before\,pipelining}{Average\,calculation\,time\,per\,instruction\,after\,pipelining}$$

Suppose the number of instructions per clock cycle T = IPC
Then average calculation time per instruction = T/IPC
Substituting the above value in above, we have
Performance increase

$$PerformaceIncrease = \frac{IPC_{after} \times T_{before}}{IPC_{before} \times T_{after}}$$

$$= \frac{IPC_{after}}{IPC_{before}} \times \frac{1}{\left(\frac{1-k}{n}\right) + k}$$

Here, the above equation assumes that $\text{IPC}_{\text{after}} <= \text{IPC}_{\text{before}}$ assuming no additional micro-architectural feature to improve the IPC after pipelining the circuit.

Performance Increase from Pipelining: Example

Let's take a practical example of performance increase due to the increase in pipeline stages.

Numbers of pipeline stages were increased from 10 in Pentium III to 20 in Pentium IV resulting in a 20% reduction in instructions per cycle (IPC). Assuming a constant 2% timing overhead as a fraction of the total non-pipelined delay, performance increase is calculated as

$$= \frac{IPC_{after}}{IPC_{before}} \times \frac{\left(\frac{1-k}{n_{before}}\right)+k}{\left(\frac{1-k}{n_{after}}\right)+k}$$

$$= 0.8 \times \frac{\left(\frac{1-0.02}{10}\right)+0.02}{\left(\frac{1-0.02}{20}\right)+0.02}$$

$$= 1.37$$

Thus from above, Pentium IV has only about 37% better performance than the Pentium III in the same technology, despite having twice the number of pipeline stages.

Advanced Digital Design

References

- Mohit Arora, “The Art of Hardware Design”, Springer

THANK YOU

Sudeendra kumar K
Department of Electronics and Communication Engineering
sudeendrakumark@pes.edu

Architectures for Hardware Acceleration

Dr. Sudeendra kumar K

Department of Electronics and Communication Engineering

ADVANCED DIGITAL DESIGN

Multipliers

Unit-5: Lecture 5

Sudeendra kumar K

Department of Electronics and Communication Engineering

Advanced Digital Design

Array Multiplier

- An array multiplier is a parallel multiplier that generates the partial products in a parallel fashion. The various partial products are added as soon as they are available.
- Two 4-bit unsigned numbers, X3,X2,X1,X0 and Y3,Y2,Y1,Y0, are multiplied to generate a product that is possibly 8 bits. Each of the $X_{i,}$ Y_j , product bits can be generated by an AND gate.
- Each partial product can be added to the previous sum of partial products using a row of adders. The sum output of the first row of adders, which adds the first two partial products, is S13 S12 S11 S10, and the carry output is C13, C12 C11 C10.
- Similar results occur for the other two rows of adders. (We have used the notation S_{ij} and C_{ij} to represent the sums and carries from the ith row of adders.)

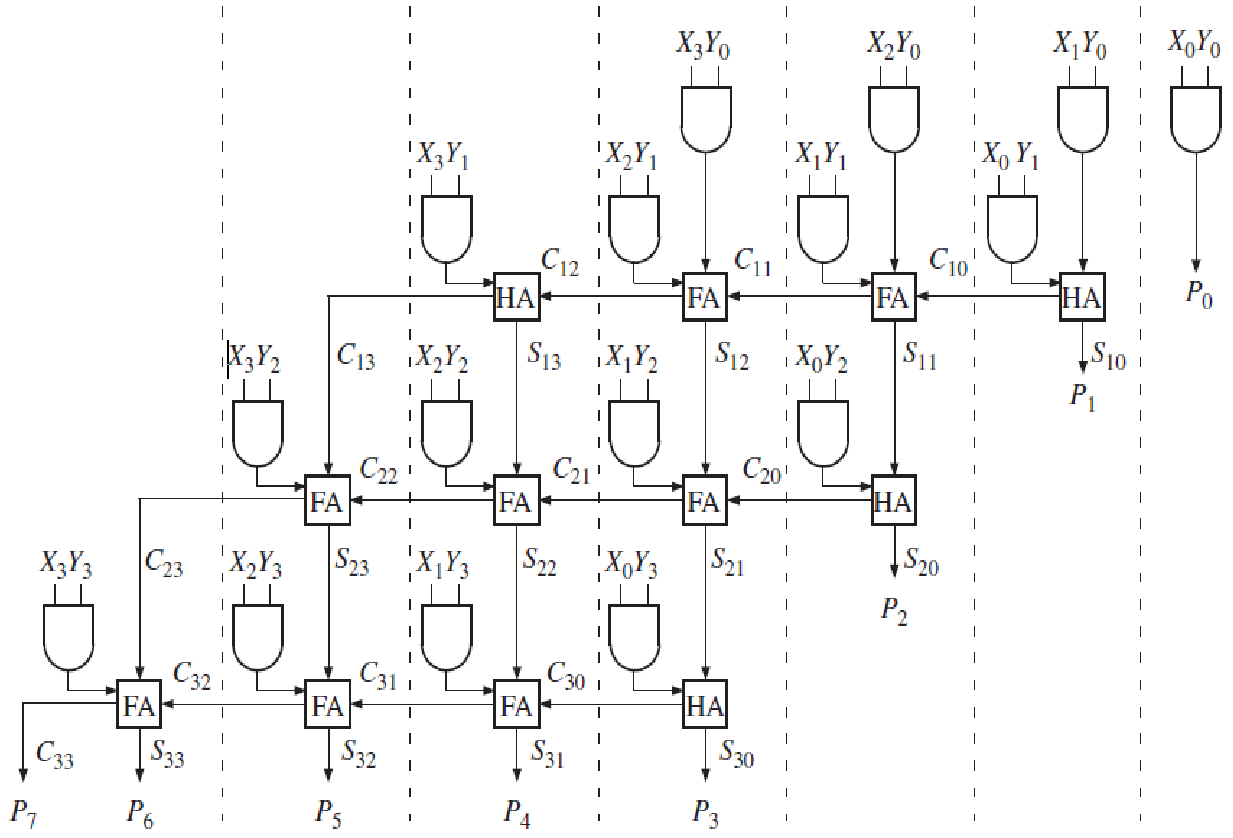
			X_3	X_2	X_1	X_0	Multiplicand
			Y_3	Y_2	Y_1	Y_0	Multiplier
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0	partial product 0
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1		partial product 1
		C_{12}	C_{11}	C_{10}			1st row carries
	C_{13}	S_{13}	S_{12}	S_{11}	S_{10}		1st row sums
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2			partial product 2
	C_{22}	C_{21}	C_{20}				2nd row carries
C_{23}	S_{23}	S_{22}	S_{21}	S_{20}			2nd row sums
X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3				partial product 3
C_{32}	C_{31}	C_{30}					3rd row carries
S_{33}	S_{32}	S_{31}	S_{30}				3rd row sums
P_6	P_5	P_4	P_3	P_2	P_1	P_0	final product

Advanced Digital Design

Array Multiplier

			X_3	X_2	X_1	X_0	Multiplicand
			Y_3	Y_2	Y_1	Y_0	Multiplier
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0	partial product 0
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1		partial product 1
		C_{12}	C_{11}	C_{10}			1st row carries
	C_{13}	S_{13}	S_{12}	S_{11}	S_{10}		1st row sums
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2			partial product 2
	C_{22}	C_{21}	C_{20}				2nd row carries
C_{23}	S_{23}	S_{22}	S_{21}	S_{20}			2nd row sums
X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3				partial product 3
C_{32}	C_{31}	C_{30}					3rd row carries
S_{33}	S_{32}	S_{31}	S_{30}				3rd row sums
P_6	P_5	P_4	P_3	P_2	P_1	P_0	final product

4-bit Multiplier



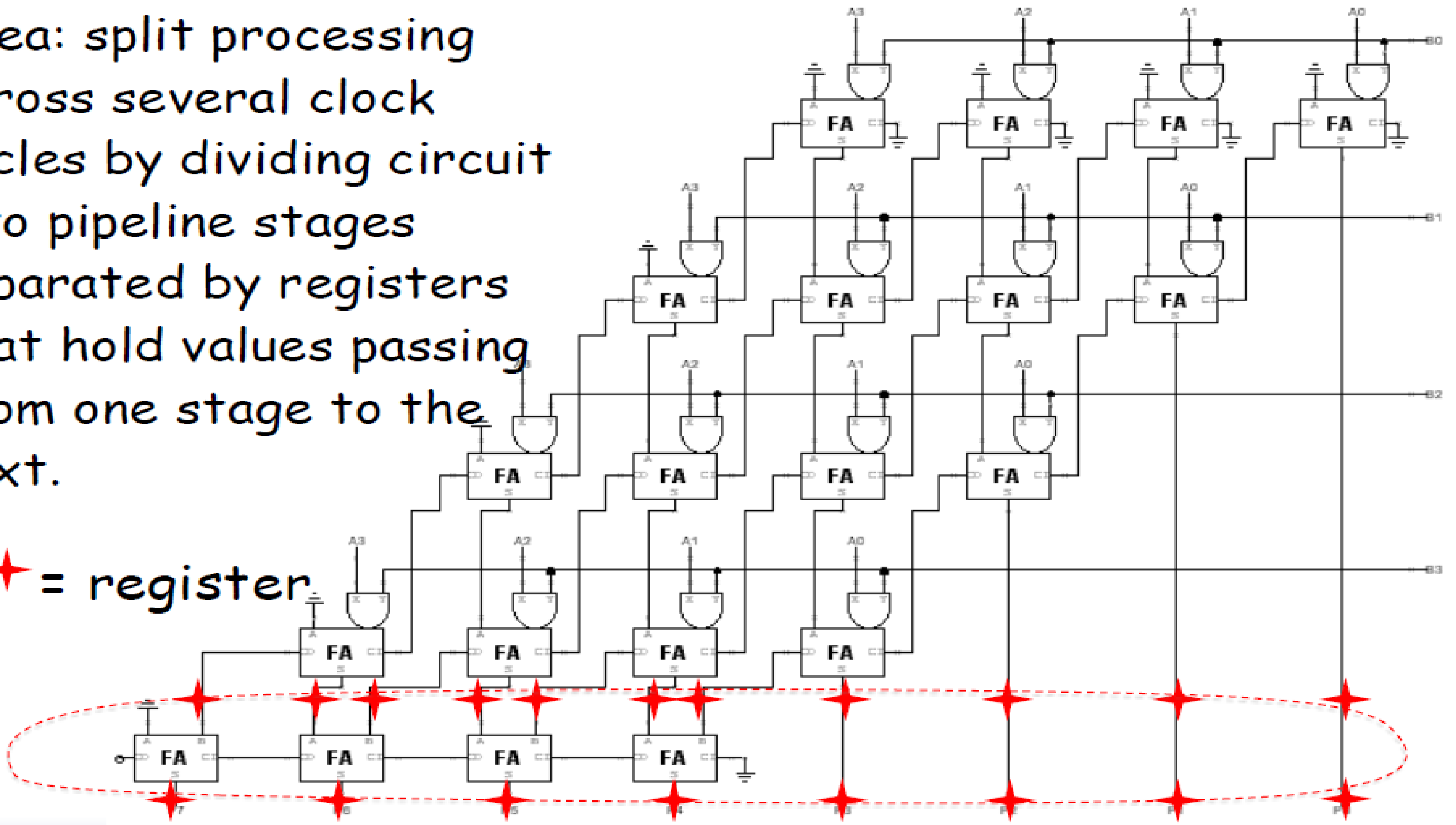
Block Diagram of 4 X 4 Array Multiplier

Advanced Digital Design

Pipelined Array Multiplier

Idea: split processing across several clock cycles by dividing circuit into pipeline stages separated by registers that hold values passing from one stage to the next.

✦ = register



PES UNIVERSITY ONLINE

THANK YOU

Sudeendra kumar K
Department of Electronics and Communication Engineering
sudeendrakumark@pes.edu