

# Chapter 2

## Clocks and Resets

### 2.1 Introduction

The cost of designing ASICs is increasing every year. In addition to the non-recurring engineering (NRE) and mask costs, development costs are increasing due to ASIC design complexity. To overcome the risk of re-spins, high NRE costs, and to reduce time-to-market delays, it has become very important to design the first time working silicon.

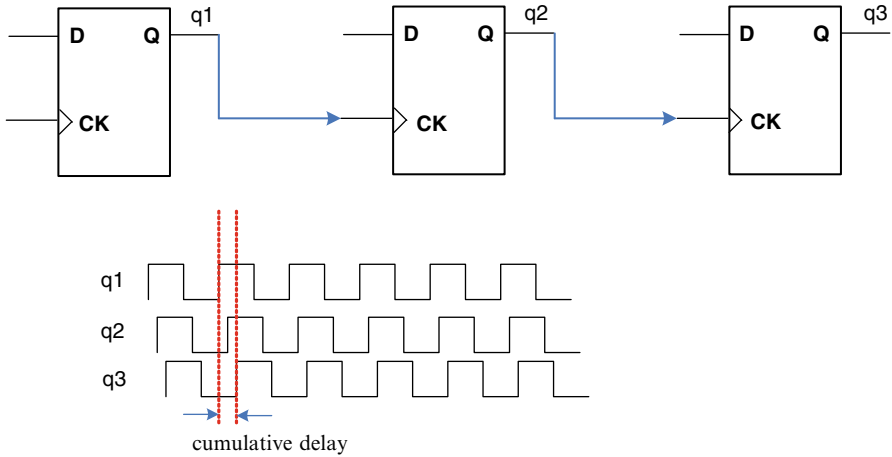
This chapter constitutes a general set of recommendations intended for use by designers while designing a block or an IP (Intellectual Property). The guidelines are independent of any CAD tool or silicon process and are applicable to any ASIC designs and can help designers to plan and to execute a successful System on Chip (SoC) with a well-structured and synthesizable RTL code.

The current paradigm shift towards system level integration (SLI), incorporating multiple complex functional blocks and a variety of memories on a single circuit, gives rise to a new set of design requirements at integration level. The recommendations are principally aimed at the design of the blocks and memory interfaces which are to be integrated into the system-on-chip. However, the guidelines given here are fully consistent with the requirements of system level integration and will significantly ease the integration effort, and ensure that the individual blocks are easily reusable in other systems.

These guidelines can form as a basis of checklist that can be used as a signoff for each design prior to submission for fabrication.

### 2.2 Synchronous Designs

Synchronous designs are characterized by a single master clock and a single master set/reset driving all sequential elements in the design.



**Fig. 2.1** Flip flop driving the clock input of another flip flop (ripple counter)

Experience has shown that the safest methodology for time domain control of an ASIC is synchronous design. Some of the problems with the circuits not being synchronous have been shown in this section.

### 2.2.1 Avoid Using Ripple Counters

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 as shown in Fig. 2.1. Note the cumulative delay gets added on with more number of flip flops and hence the same is not recommended. More details on the ripple counter are given in Sect. 5.6.7.

### 2.2.2 Gated Clocks

Gating in a clock line causes clock skew and can introduce spikes which trigger the flip-flop. This is particularly the case when there is a multiplexer in the clock line as shown in Fig. 2.2.

Simulating a gated clock design might work perfectly fine but the problem arises when such a design is synthesized.

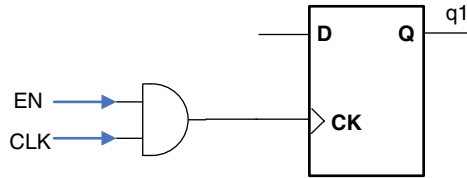


Fig. 2.2 Gated clock line

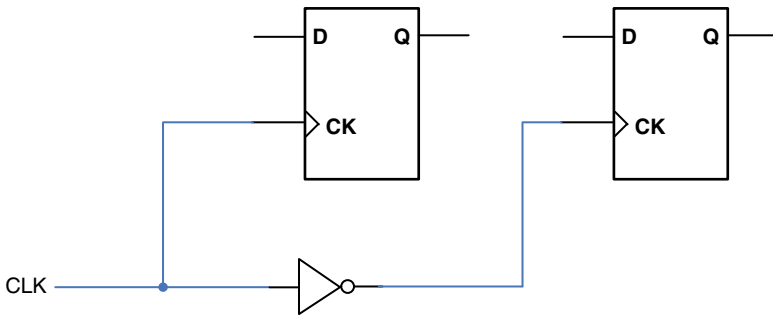


Fig. 2.3 Double-edged clocking

### 2.2.3 Double-Edged or Mixed Edge Clocking

As shown in Fig. 2.3, 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.

### 2.2.4 Flip Flops Driving Asynchronous Reset of Another Flop

In Fig. 2.4, the second flip-flop can change state at a time other than the active clock edge, violating the principle of synchronous design. In addition, this circuit contains a potential race condition between the clock and reset of the second flip-flop.

The subsequent sections show the methods to avoid the above non-recommended circuits.

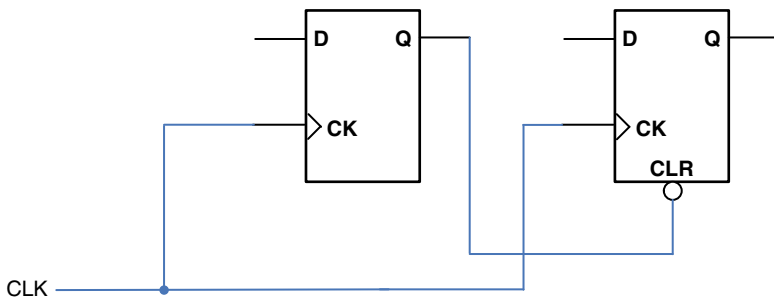


Fig. 2.4 Flip flop driving asynchronous reset of another flop

## 2.3 Recommended Design Techniques

When designing with HDL code, it is important to understand how a synthesis tool interprets different HDL coding styles and the results to expect. It is very important to think in terms of hardware as a particular design style (or rather coding style) can affect gate count and timing performance. This section discusses some of the basic techniques to ensure optimal synthesis results while avoiding several causes of unreliability and instability.

### 2.3.1 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. Combinational loops violate synchronous design principles by establishing a direct feedback with no registers.

In terms of HDL language, combinational loops occur when the generation of a signal depends on itself through several combinational *always*<sup>1</sup> blocks or when the left-hand side of an arithmetic expression also appears on the right-hand side. Combo loops are a hazard to a design and synthesis tools will always give errors when combo loops are encountered, as these are not synthesizable.

The generation of combo loops can be understood from the following bubble diagram in Fig. 2.5. Each bubble represents a combo always block and the arrow going into it represents the signal being used in that always block while an arrow going out from the bubble represents the output signal generated by that output block. It is evident that the generation of signal 'a' depends on itself through signal 'd', thereby generation a combinational loop.

<sup>1</sup>For simplicity, any HDL languages that this book refers to takes Verilog as an example.

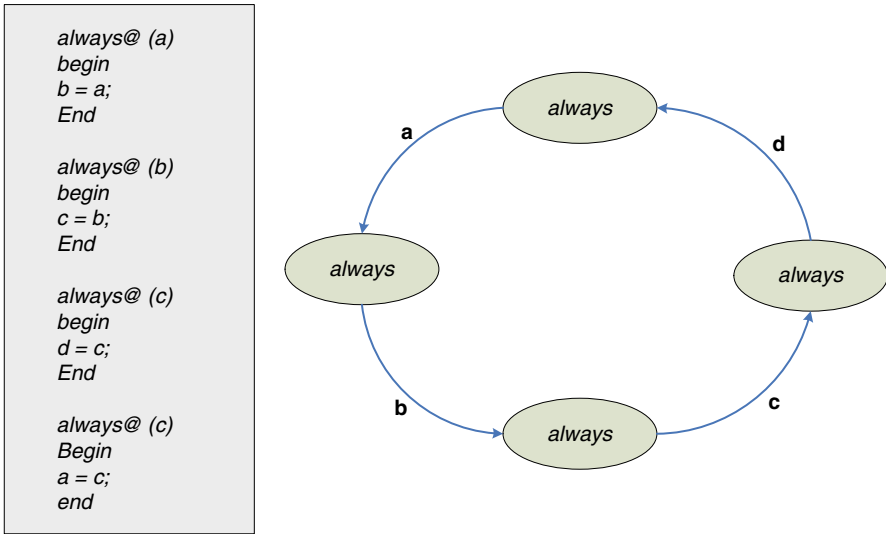


Fig. 2.5 Combinational loop example and bubble diagram

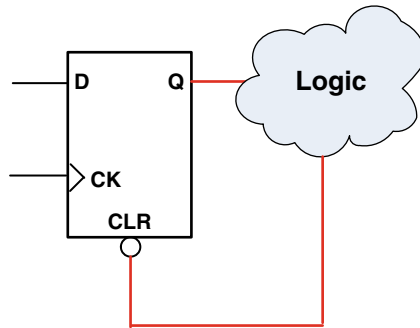


Fig. 2.6 Combinational loop through asynchronous control pins

The code and the bubble diagram are shown below [28]:

In order to remove combo loops, one must change the generation of one of the signals so the dependency of signals on each other is removed. Simple resolution to this problem is to introduce a Flip Flop or register in the combo loop to break this direct path.

Figure 2.6 shows another example where output of a register directly controls the asynchronous pin of the same register through combinational logic.

Combinational loops are inherently high-risk design structures. Combinational loop behavior generally depends on the relative propagation delays through the logic involved in the loop. Propagation delays can change based on various factors and the behavior of the loop may change. Combinational loops can cause endless

computation loops in many design tools. Most synthesis tools break open or disable combinatorial loop paths in order to proceed. The various tools used in the design flow may open a given loop a different manner, processing it in a way that may not be consistent with the original design intent.

### ***2.3.2 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. In both FPGA and ASIC, delays can change with each place-and-route. Delay chains can cause various design problems, including an increase in a design's sensitivity to operating conditions, a decrease in a design's reliability, and difficulties when migrating to different device architecture. Avoid using delay chains in a design, rely on synchronous practices instead.

### ***2.3.3 Avoid Using Asynchronous 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). There are two common methods for pulse generation; these techniques are purely asynchronous and 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. The width of the pulse depends on the relative delays of the path that feeds the gate directly and the one that goes through the delay. This is the same mechanism responsible for the generation of glitches in combinational logic following a change of inputs. This technique artificially increases the width of the spike by using a delay chain.
- A register's output drives the same register's asynchronous reset signal through a delay chain. The register essentially resets itself asynchronously after a certain delay.

Asynchronously generated pulse widths often pose problem to the synthesis and place-and-route software. 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, and the pulse width will change when migrating to a different technology node. In addition, static timing analysis cannot be used to verify the pulse width so verification is very difficult.

Multi-vibrators use the principle of the "glitch generator" to create pulses, in addition to a combinational loop that turns the circuit into an oscillator [25].

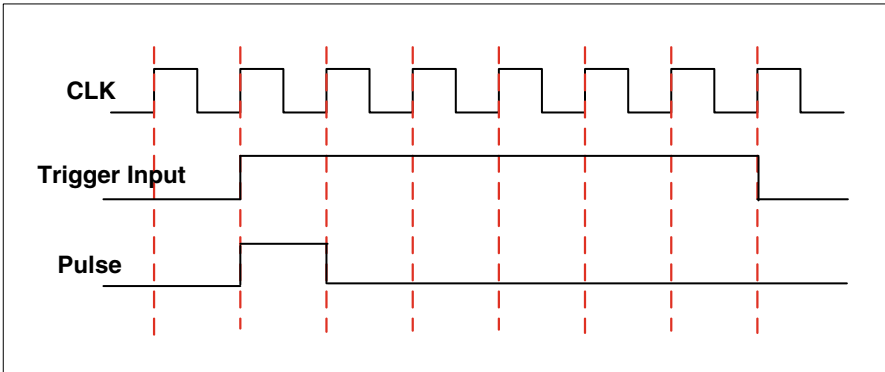
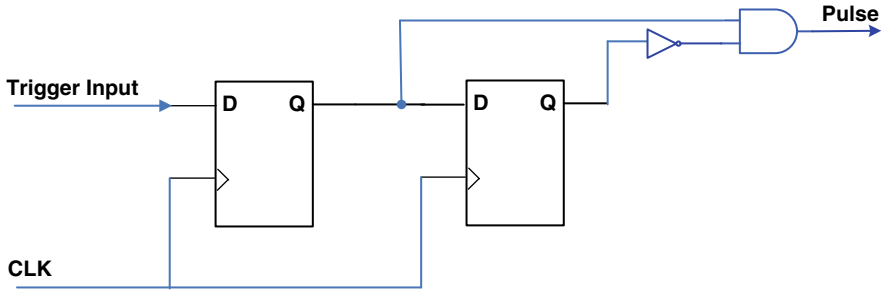


Fig. 2.7 Synchronous pulse generator circuit on start of trigger input

Structures that generate multiple pulses cause even more problems than pulse generators because of the number of pulses involved. In addition, when the structures generate multiples pulses, they also increase the frequency of the design.

A recommended Synchronous Pulse generator is shown in Fig. 2.7.

In the above synchronous pulse generator design, the pulse width is always equal to the clock period. This pulse generator is predictable, can be verified with timing analysis, and is easily migrated to other architectures and is technology independent.

Similar to Fig. 2.7, Fig. 2.8 shows the pulse generator at the end of trigger input.

### 2.3.4 Avoid Using Latches

In digital logic, latches hold the value of a signal until a new value is assigned. Latches should be avoided whereas possible in the design and flip-flops should be used instead.

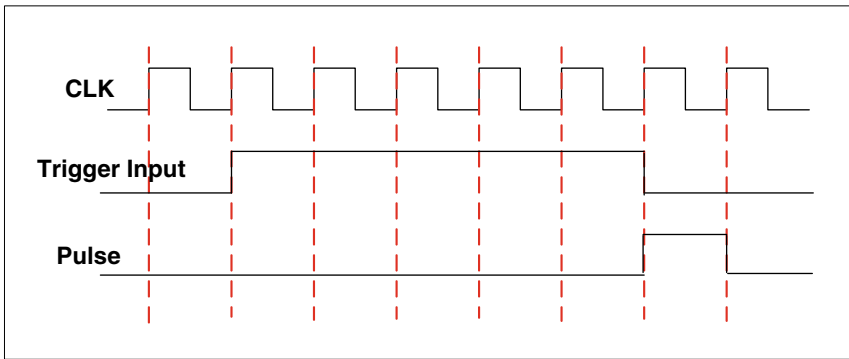
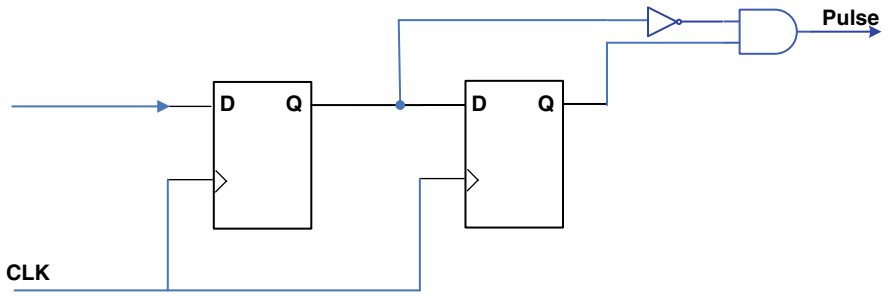


Fig. 2.8 Synchronous pulse generator circuit on end of trigger input

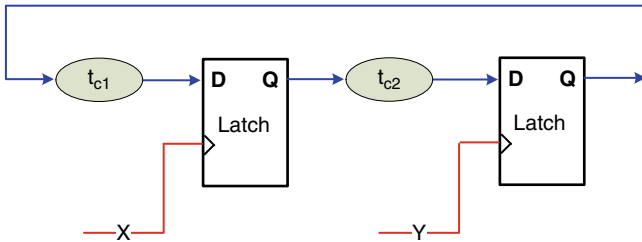
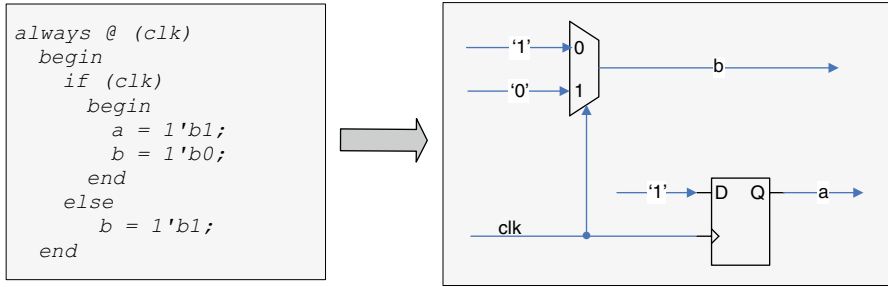


Fig. 2.9 Race conditions in latches

As shown in Fig. 2.9, if both the X and Y were to go high, and since these are level triggered, both the Latches would be enabled resulting in the circuit to oscillate.

Latches can cause various difficulties in the design. Although latches are memory elements like registers, they are fundamentally different. When a latch is in a feed-through mode, there is a direct path between the data input and the output. Glitches on the data input can pass to the output.



**Fig. 2.10** Inferred latch due to incomplete ‘if else’ statement

Static timing analyzers typically make incorrect assumptions about latch transparency, and either find a false timing path through the input data pin or miss a critical path altogether. The timing for latches is also inherently ambiguous. When analyzing a design with a D latch, for example, the tool cannot determine whether you intended to transfer data to the output on the leading edge of the clock or on the trailing edge. In many cases, only the original designer knows the full intent of the design, which implies that another designer cannot easily migrate the same design or reuse the code.

Latches tend to make circuits less testable. Most design for test (DFT) and automatic test program generator (ATPG) tools do not handle latches very well.

Latches pose different challenge in FPGA designs as FPGA’s are register-intensive; therefore, designing with latches uses more logic and leads to lower performance than designing with registers.

Synthesis tools occasionally infer a latch in a design when one is not intended. Inferred latches typically result from incomplete “if” or “case” statements. Omitting the final “else” clause in an “if” or “case” statement can also generate a latch. Figure 2.10 shows a similar example.

As shown in Fig. 2.10, ‘b’ will be synthesized as straight combinational logic while a latch will be inferred on signal ‘a’.

A general rule for latch inferring is that if a variable is not assigned in all possible executions of an always statement (for example, when a variable is not assigned in all branches of an ‘if’ statement), then a latch is inferred.

Some FPGA architectures do not support latches. When such a design is synthesized, the synthesis tool creates a combinational feedback loop instead of a latch (as shown in Fig. 2.11).

Combinational feedback loops as shown above are capable of latching data but pose more problem than latches since they may violate setup, hold requirements which are difficult be determined, whereas latches does not have any setup time, hold time violations since they are level triggered.

**Note:** *The design should not contain any combinational feedback loops. They should be replaced by flip-flops or latches or be eliminated by fully enumerating RTL conditionals.*

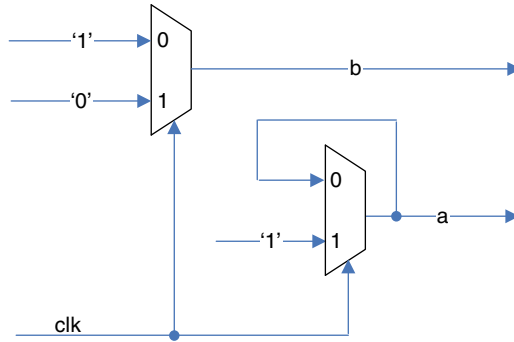


Fig. 2.11 Combinational loop implemented due to incomplete ‘if else’ statement

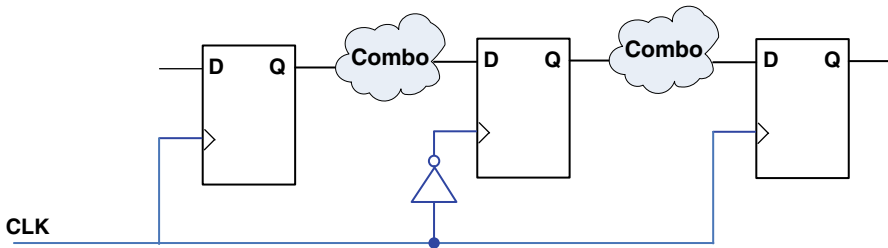


Fig. 2.12 Logic with double edged clocking

To conclude, this does not mean latches should never exist, we will see later how latches could be wonderful when it comes to cycle stealing or time borrowing to meet a critical path in a design.

### 2.3.5 Avoid Using Double-Edged 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.

Figure 2.12 shows a circuit triggered by both edges of clock.

Some of the problems encountered with Double Edged clocking are mentioned below:

- An asymmetrical clock duty cycle can cause setup and hold violations.
- It is difficult to determine critical signal paths.

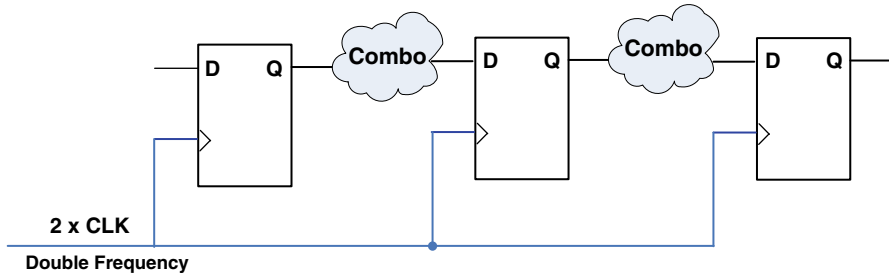


Fig. 2.13 Logic with single edged clocking

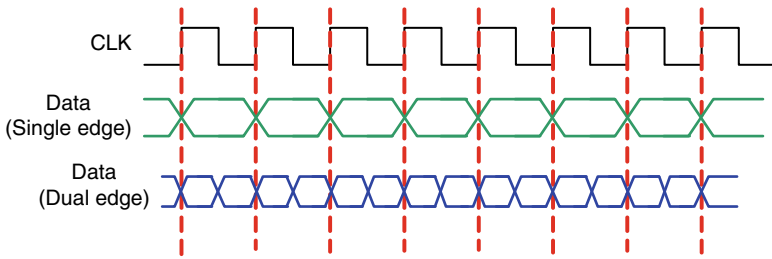


Fig. 2.14 Single/double edged data transfer

- 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.

Figure 2.13 shows the normal equivalent pipelined logic with single edge clocking. Note that this synchronous circuit requires a clock frequency that is double the one shown in Fig. 2.12.

Figure 2.14 shows the single transition and double transition clocked data transfer.

The green and blue signals represent data; the “hexagon” shapes are the traditional way of representing a signal that at any given time can be either a one or a zero.

In the circuit shown in Fig. 2.12, an asymmetrical clock duty cycle could cause setup and hold time violations, and a scan-path cannot easily be threaded through the flip-flops.

The above does not mean that circuits with dual edge clocking should never be used unless there is an intense desire for higher performance/speed that cannot be met with the equivalent synchronous circuits as the latter comes with an additional overhead of complexity in DFT and verification.

### 2.3.5.1 Advantages of Dual Edge Clocking

The one constant in the PC world is the desire for increased performance. This in turn means that most interfaces are, over time, modified to allow for faster clocking, which leads to improved throughput. Many newer technologies in the PC world have gone a step beyond just running the clock faster. They have also changed the overall signaling method of the interface or bus, so that data transfer occurs not once per clock cycle, but twice or more.

There are other advantages of circuit operating on dual edge rather than the same synchronous circuit being fed with double the clock frequency. Whatever extent possible, interface designers do regularly increase the speed of the system clock. However, as clock speeds get very high, problems are introduced on many interfaces. Most of these issues are related to the electrical characteristics of the signals themselves. Interference between signals increases with frequency and timing becomes more “tight”, increasing cost as the interface circuits must be made more precise to deal with the higher speeds.

The other advantage using double edged clocking is lower power consumptions as clock speeds are decreased by half and hence the system consumes less power than the equivalent synchronous circuits.

So to conclude system integrator should only use dual or double edged clocking unless the same desired performance cannot be met with the equivalent synchronous circuits.