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.

2.4 Clocking Schemes

2.4.1 Internally Generated Clocks

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. However, a glitch or a spike on the clock input (or an asynchronous input of a register) can have significant consequences.

Narrow glitches can violate the register’s minimum pulse width requirements. Setup and hold times may also be violated if the data input of the register is changing when a glitch reaches the clock input. Even if the design does not violate timing requirements, the register output can change value unexpectedly and cause functional hazards elsewhere in the design.

Figure 2.15 shows the effect of using a combinational logic to generate a clock on a synchronous counter. As shown in the timing diagram, due to the glitch on the clock edge, the counters increments twice in the clock cycle shown.
This extra counting may create functional issues in the design where instead of counting the desired count, counter counts an additional count due to the glitch on the clock.

**Note:** That for the sake of simplicity, it is assumed that the Counters Flops did not violate the setup/hold requirements on the data due to the glitch.

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 (Fig. 2.16).
The combinational logic used to generate an internal clock also adds delays on the clock line. In some cases, logic delay on a clock line can result in a clock skew greater than the data path length between two registers. 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 2.17 shows a similar example where setup time on input “IN” is violated due to skew on the clock path.

Note: Data path delay is assumed to be zero for simplicity.

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. Using a low-skew clock tree can help reduce the overall clock skew for the signal.

### 2.4.2 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. Using PLL circuitry will avoid many of the problems that can be introduced by asynchronous clock division logic. 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.
ASIC designers have often implemented ripple counters to divide clocks by a power of 2 because the counters use fewer gates than their synchronous counterparts. Ripple counters use cascaded registers, in which the output pin of each register feeds the clock pin of the register in the next stage (Fig. 2.18).

This cascading can cause problems because the counter creates a ripple clock at each stage. These ripple clocks pose another set of challenges for STA and synthesis tools. One should try to avoid these types of structures to ease verification effort.

Despite all the challenges and problems with respect to using Ripple counters, these are quite handy in systems that eat power and can be good to reduce the peak power consumed by a logic or SoC.

**Note:** Digital designers should consider using this technique in limited cases and under tight control.

Refer Chap. 5 “Low power design” on more details analysis and techniques of using Ripple counters to save power consumption.

### 2.4.4 Multiplexed Clocks

Clock multiplexing can be used to operate the same logic function with different clock sources. Multiplexing logic of some kind selects a clock source as shown in Fig. 2.19.

For example, telecommunications applications that deal with multiple frequency standards often use multiplexed clocks.
Adding multiplexing logic to the clock signal can lead to some of the problems discussed in the previous sections, but requirements for multiplexed clocks vary widely depending on the application.

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.

### 2.4.5 Synchronous Clock Enables and Gated Clocks

Gated clocks turn a clock signal on and off using an enable signal that controls some sort of gating circuitry. As shown in Fig. 2.20, when a clock is turned off, the corresponding clock domain is shut down and becomes functionally inactive.
Gated clocks can be a powerful technique to reduce power consumption. When a clock is gated both the clock network and the registers driven by it stop toggling, thereby eliminating their contributions to power consumption. However, gated clocks are not part of a synchronous scheme and therefore can significantly increase the effort required for design implementation and verification. Gated clocks contribute to clock skew and are also sensitive to glitches, which can cause design failure.

A clock domain can be turned off in a purely synchronous manner using a synchronous clock enable. However, when using a synchronous clock enable scheme, the clock tree keeps toggling and the internal circuitry of each Flip Flop remains active (although outputs do not change values), which does not reduce power consumption. A synchronous clock enable technique is shown in Fig. 2.21.

This Synchronous Clock Enable Clocking scheme does not reduce power consumption as much as gating the clock at the source because the clock network keeps toggling, but it will perform the same function as a gated clock by disabling a set of Flip Flops. As shown in Fig. 2.21, multiplexer in front of the data input of every Flip Flop either load new data or copy the output of the Flip Flop based on the Enable signal.

The next section is dedicated to efficient clock gating methodology that should be used where ever clocking gating is desired due to tight power specifications.
2.5 Clock Gating Methodology

In the traditional synchronous design style, the system clock is connected to the clock pin on every flip-flop in the design. This results in three major components of power consumption:

1. Power consumed by combinatorial logic whose values are changing on each clock edge (due to flops driving those combo cells).
2. Power consumed by flip-flops (this has non-zero value even if the inputs to the flip-flops, and therefore, the internal state of the flip-flops, is not changing).
3. Power consumed by the clock tree buffers in the design.

Gating the clock path substantially reduces the power consumed by a Flip Flop. Clock Gating can be done at the root of the clock tree, at the leaves, or somewhere in between.

Since the clock tree constitutes almost 50% of the whole chip power, it is always a good idea to generate and gate the clock at the root so that entire clock tree can be shut down instead of implementing the gating along the clock tree at the leaves.

Figure 2.22 shows an example of a clock gating for a three bit Counter.

The circuit is similar to the traditional implementation except that a clock gating element has been inserted into the clock network, which causes the flip-flops to be clocked only when the INC input is high. When the INC input is low, the flip-flops are not clocked and therefore retain the old data. This saves three multiplexers in front of the flip-flops which would had been there in case the gating was implemented by Synchronous Clock Enable as described in Fig. 2.21. This can result in significant area saving when wide banks of registers are being implemented.

2.5.1 Latch Free Clock Gating Circuit

The latch-free clock gating style uses a simple AND or OR gate (depending on the edge on which flip-flops are triggered) as shown in Fig. 2.23.

For the correct operation the circuit imposes a requirement that all enable signals be held constant from the active (rising) edge of the clock until the inactive (falling) edge of the clock to avoid truncating the generated clock pulse prematurely or generating multiple clock pulses (or glitches in clock) where one is required.

Figure 2.24 shows the case where generated clock is truncated prematurely when the above requirement is not satisfied.

This restriction makes the latch-free clock gating style inappropriate for our single-clock flip-flop based design.
Fig. 2.22 Three bit counter with clock gating

Fig. 2.23 Latch free clock gating circuit
2.5.2 Latch Based Clock Gating Circuit

The latch-based clock gating style adds a level-sensitive latch to the design to hold the enable signal from the active edge of the clock until the inactive edge of the clock, making it unnecessary for the circuit to itself enforce that requirement as shown in Fig. 2.25.
Since the latch captures the state of the enable signal and holds it until the complete clock pulse has been generated, the enable signal need only be stable around the rising edge of the clock.

Using this technique, only one input of the gate that turns the clock on and off changes at a time, ensuring that the circuit is free from any glitches or spikes on the output.

Note: Use an AND gate to gate a clock that is active on the rising edge. For a clock that is active on the falling edge, use an OR gate to gate the clock and register the enable with a positive edge-triggered Latch.

When using this technique, special attention should be paid to the duty cycle of the clock and the delay through the logic that generates the enable signal, because the enable signal must be generated in half the clock cycle. This situation might cause problems if the logic that generates the enable command is particularly complex, or if the duty cycle of the clock is severely unbalanced. However, being careful with the duty cycle and logic delay may be acceptable compared with the problems created by other methods of gating clocks.

To ensure high manufacturing fault coverage, it is necessary to make sure the clock gating circuit is full controllable and observable to use within a scan methodology. A controllability signal which causes all flip-flops in the design to be clocked, regardless of the enable term value, can be added to allow the scan chain to shift information normally.

This signal can be ORed in with the enable signal before the latch and can be connected to either a test mode enable signal which is asserted throughout scan testing or to a scan enable signal which is asserted only during scan shifting.

The modified circuit is shown in Fig. 2.26. Most of the ASIC vendors do provide this “Clock Gating Cell” as a part of their standard library cell.
2.5.3 Gating Signals

Effective power implementation can be achieved using gating signals for particular parts of the design. Similar to the concept of gating clock, signal gating reduces the transitions in clock free signals. The most common example is the decoder enable.

As part of an address decoding mechanism, signals used by other parts of the design may toggle as a reflection of activity in these parts. Switching activity on one input of the decoder will induce a large number of toggling gates. Controlling this with an enable or select signal prevents the propagation of their switching activity, even if the logic is slightly more complex (Fig. 2.27).

2.5.4 Data Path Re-ordering to Reduce Switching Propagation

Several data path elements, such as decoders or comparison operators, as well as “glitchy” logic may significantly contribute to power dissipation. The glitches, caused by late arrival signals or skews, propagate through other data path elements and logic until they reach a register. This propagation burns more power as the transitions traverse the logic levels. To reduce this wasted dissipation, designers need to rewrite the HDL code and shorten the propagation paths as much as possible. Figure 2.28 illustrates two implementations of the priority mux where the “glitchy” and “stable” conditions are ordered differently.

2.6 Reset Design Strategy

Many design issues must be considered before choosing a reset strategy for an ASIC design, such as whether to use synchronous or asynchronous resets, will every flip-flop receive a reset etc.