High Speed CMOS Circuit Design

# Lecture 9: Clocking & Latching

**(c) 1997 David Harris**

## 1.0 Introduction to Sequential Logic Design

So far, all logic we have considered computes functions of the current inputs. Real systems must have memory so that they can also compute functions of previous inputs. Such functions are implemented with *sequential* logic.

A major application of sequential logic is for pipelined processor design. Logic moves through stages; for example, the output of the Decode stage becomes the input of the Execute stage. By dedicating one cycle to each stage, multiple instructions can simultaneously occupy the pipeline so that one is being executed while the next is being decoded. On each cycle, the instructions advance to the next stage.

Another application of sequential logic is for finite state machines. On each step, the state machine takes inputs and its current state and produces outputs and advances to the next state.

There are several ways to implement memory elements; they can be categorized as static and dynamic storage. Both rely on the capacitance of the storage node to hold energy which indicates the state. Dynamic storage just places charge on the node. Dynamic storage must be refreshed periodically so that it does not leak away. Static storage uses positive feedback to actively restore the energy; hence static memory retains its contents indefinitely.

It is important to remember that the choice of static or dynamic storage elements is distinct from the choice of static or dynamic logic. For example, static storage elements can be used in a pipeline built out of domino gates. Alternately, dynamic storage elements could be used in a finite state machine constructed with static gates.

In this lecture, we will explore sequential design techniques with application to static logic. The three major options for storage elements are edge-triggered flip-flops, transparent latches, and pulsed latches. We will see how each kind of storage element adds overhead which increases the cycle time. Then we will look at actual circuit designs of various elements. We'll finish with special topics in sequential design, including scan circuits, pulse generators and clock choppers, and wave pipelining.

In the next lecture, we'll move on to sequential design techniques applicable to dynamic logic. We'll see that good design methodologies are essential to minimizing the sequential overhead of dynamic logic. We'll also see that the quality of the clock signal is very important for high-performance design, so in the third lecture we will consider techniques for generating and distributing clocks or for eliminating the clock altogether with asynchronous design.

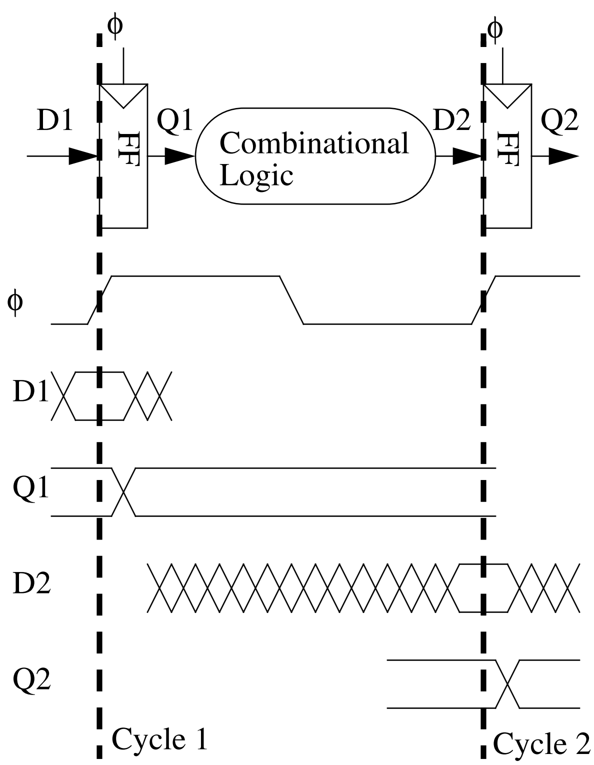
# 2.0 Sequential Elements

Although there are a multitude of possible sequential elements including S-R latches and J-K flip-flops, most CMOS systems are built with just three types of sequential elements: edge-triggered flip-flops, transparent latches, and pulsed latches. There is some confusion about terminology in the industry, so this section will try to be careful with the names of sequential elements. All three sequential elements have clock and data inputs and an output. Depending on the design, the output may use true, complementary, or both polarities.

The three elements are described below. Each has a timing diagram underneath. The logic is drawn along the horizontal axis corresponding to the time at which the data arrives.

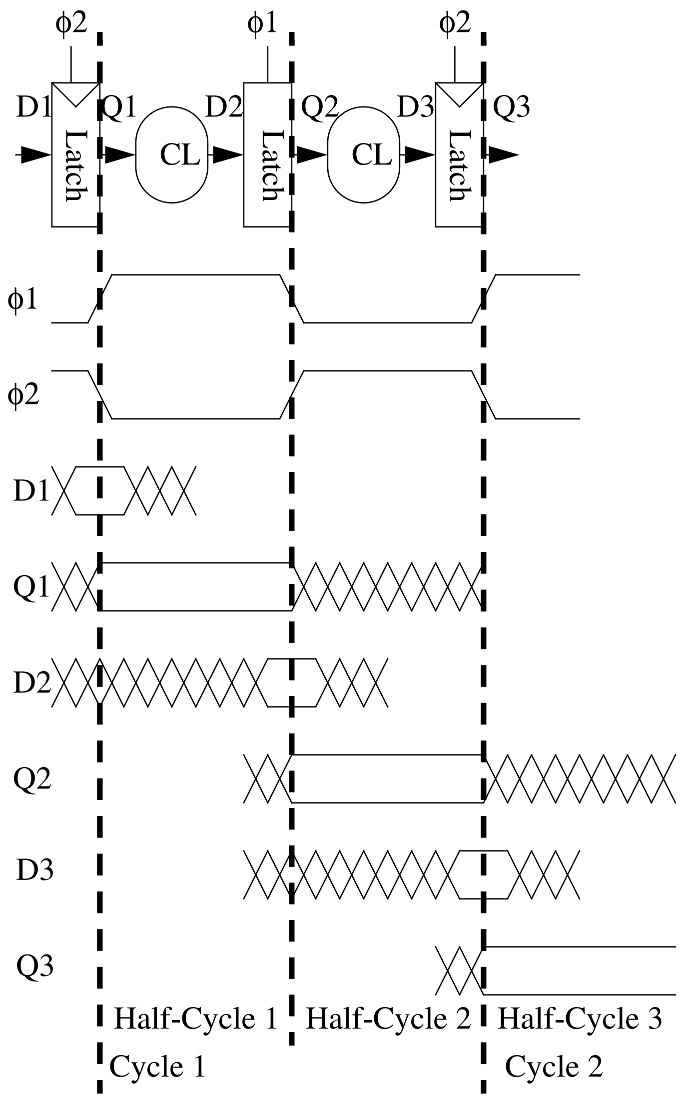
Edge-triggered flip-flops are also known as master-slave or D flip-flops. When the clock rises, the data input is sampled and transferred to the output. At all other times, the data input and output are unrelated.

**FIGURE 1. Flip-flop System**



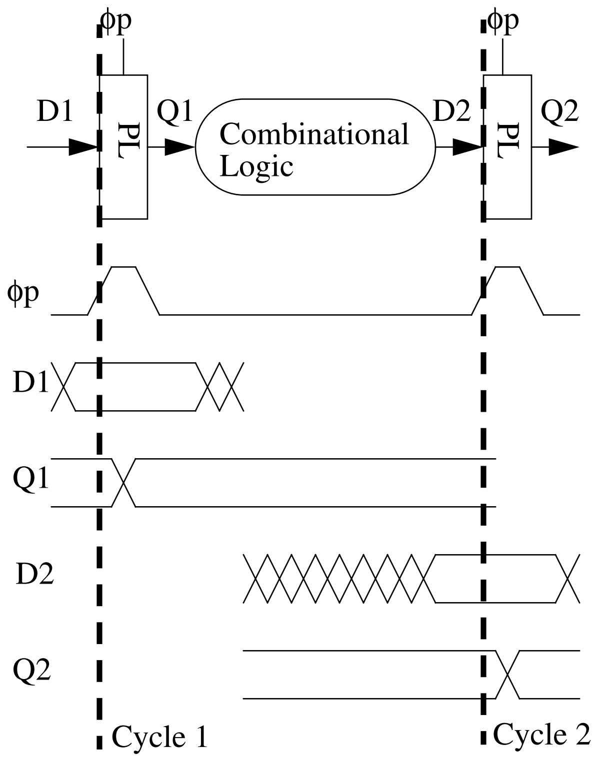

Transparent latches are also known as half-latches. When the clock is high, the output follows the input. When the clock is low, the output holds at its last value and ceases to track the input. It is confusing to call a latch ON or OFF because it is not clear whether ON means that latch is passing data or ON means that latch is holding old data. Instead, the terms transparent and opaque are clearer. To build a sequential system, two half-latches must be used in each cycle. One is transparent for the first part of the cycle, while the other is transparent for the second part of the cycle.

**FIGURE 2. Transparent Latch System**




Pulsed latches behave exactly as transparent latches, but instead of controlling them with a 50% duty cycle clock, they are given a short pulse. As long as the pulse is shorter than the time required for data to propagate through the logic before the next pulsed latch, only one pulsed latch is required in each cycle. Pulsed latches are often misnamed as edge-triggered flip-flops, but they are actually quite distinct.

**FIGURE 3. Pulsed Latch System**




# 3.0 Cycle Time of Sequential Systems

Ideally, the cycle time of a system should equal the propagation delay through the longest logic path. Unfortunately, real sequential systems introduce overhead from many sources that increases the cycle time. This section lists the various sources of overhead and shows how much overhead applies to each of the systems described above. Based on the analysis, it recommends the best latching techniques for various applications.

## 3.1 Sequential Overhead Sources

The most obvious source of overhead from sequential elements is the propagation delay through the element. This depends on the kind of element and the circuit design.

Another source is setup time. This is the time that the data input must have settled before the clock input arrives.

Clock skew is the variation in arrival time of the clock. It may hurt performance because data should be ready by the earliest the clock may arrive, yet the sequential element output may not change until the latest the clock arrives. Sources of clock skew will be discussed later. We will draw clock waveforms at the latest they may arrive; actual clocks may arrive up to $t_{ske}$w earlier.

Time borrowing is the ability for critical logic that nominally operates in one cycle or half-cycle to use time left by non-critical logic in an adjacent half-cycle or cycle. Some sequential elements support time borrowing, while others do not. Latching methods which support time borrowing have numerous advantages. They are easier to design because logic can be balanced between cycles or half-cycles. If there is insufficient time at the end of a cycle for an entire gate, systems using time borrowing can apply the remaining time to logic in another cycle, while systems without time borrowing waste the time. Finally, time borrowing helps compensate for inaccuracies in modeling or variations in processing that make some cycles longer and some shorter than predicted because the long cycles can borrow some of the slack from the short cycles.

In addition to these sources of overhead, all synchronous systems operating off a fixed frequency clock pay environmental and manufacturing margins. Manufacturing margins occur because a part is only sold at specific frequencies; for example, if a chip is sold in 800 MHz and 1 GHz speed bins, a part that operates at up to 970 MHz must be sold in the 800 MHz bin. Environmental margins occur because the chip must be able to operate at its rated frequency under worst case voltage and temperature, while in an actual system the true environment is likely to be better. These margins can be reduced if the clock frequency can be arbitrarily adjusted. Manufacturing margin can be minimized by selling the chip at exactly its worst case operating frequency. Environmental margin can also be reduced by measuring on-chip temperature and voltage and adaptively changing the frequency. All of these techniques require accurate knowledge of the worst-case path and good tracking with the worst-case path delay, which can be a difficult problem.

## 3.2 Analysis of Latching Techniques

Manufacturing and environmental margins apply to all latching techniques, so will not be considered further. Let's examine how other sources of overhead impact cycle time.

Flip-flops require the data input to settle a setup time before the earliest a skewed clock might arrive. The output becomes valid a propagation delay after the latest a skewed clock might arrive. Therefore, the cycle time is:

$$t_{cycle} = t_{logic} + t_{d\text{->}q} + t_{setup} + t_{skew} \qquad \text{(EQ 1)}$$

Transparent latches are better because they can use time borrowing to hide overhead. Data must be designed to arrive at the latch $t_{setup} + t_{skew}$ before the falling edge of the latch clock. However, this time is not wasted; instead the next half-cycle can begin using the result as soon as it arrives. Thus, in a system constructed entirely of 2-phase transparent latches:

$$t_{cycle} = t_{logic} + t_{d1\text{->}q1} + t_{d2\text{->}q2} \qquad \text{(EQ 2)}$$

where $t_{d1\text{->}q1}$ and $t_{d2\text{->}q2}$ are the delays of the two static latches. Remember that a flip-flop is just two back-to-back latches, so the propagation delay of a flip-flop is comparable to the propagation delay of two latches.

Also, additional time borrowing is useful in transparent latch systems to balance logic between phases and compensate for uncertainty in delays.

Pulsed latch systems have pros and cons relative to transparent latches. The biggest advantage is that now only a single latch is used, reducing the propagation delay. The data must arrive at least $t_{setup} + t_{skew}$ before the falling edge of the pulse. If the pulse is wide enough, the pulsed latch will be transparent when the data arrives and there is no wasted time. If the pulse is narrower than this window, there is extra overhead. A wide pulse is also beneficial to allow limited amounts of time borrowing. Unfortunately, a wide pulse also increases the number of min-delay problems in which short paths must be padded. The cycle time of a pulsed-latch system is:

$$t_{cycle} = t_{logic} + t_{d\text{->}q} + \max\{0, t_{setup} + t_{skew} - t_{pulsewidth}\} \qquad \text{(EQ 3)}$$

## 3.3 Recommendations

Flip-flops were once popular because they are conceptually simplest. As cycle times are getting shorter, the overhead of flip-flops is too large and they have gone out of favor. Thus, the two main contenders are transparent latches and pulsed latches.

Transparent latches can eliminate all overhead except their propagation delay. They also allow nearly half a cycle of time borrowing. Pulsed latches can be even faster because only a single latch delay is introduced. However, hiding the setup time and clock skew requires a relatively wide pulse. Such a pulse leads to min-delay problems, as will be seen. Pulsed latches also provide very little time borrowing.

Designers willing to do extensive min-delay checking are beginning to favor pulsed latches. Other designers prefer the safety and time borrowing of transparent latches. Fast systems can be built with either technique.

# 4.0 Min-Delay

If there is little logic in a cycle, the clock starting the cycle is skewed early, and the clock ending the cycle is skewed late, data can race through the cycle and be clocked into the next clocked element all in one cycle. This is bad.

The contamination delay through logic in a flop-based system must be at least:

$$t_{cd} = t_{skew} + t_{hold} - t_{cd\text{-}flop} \qquad \text{(EQ 4)}$$

The min-delay constraints on 2-phase logic depend how much the phases may overlap. Academic designs often use non-overlapping clock phases to eliminate min-delay problems. Unfortunately, this requires either distributing two phases around the entire chip or building complex clock generators, so high-performance industrial designs usually clock the logic with true and complementary versions of a global clock with nominal 50% duty cycle. For such systems, the contamination delay constraint is the same as for flop-based systems:

$$t_{cd} = t_{skew} + t_{hold} - t_{cd\text{-}latch1} - t_{cd\text{-}latch2} \quad \text{(EQ 5)}$$

Pulsed latch systems have the most difficult contamination delay restrictions:

$$t_{cd} = t_{skew} + t_{hold} + t_{pulse\text{-}wdith} - t_{cd\text{-}latch} \quad \text{(EQ 6)}$$

In principle, min-delay problems should only occur on paths with a small amount of logic. Min-delay can be avoided by padding the path with extra inverters or with an extra pulsed latch operating in the middle of the cycle.

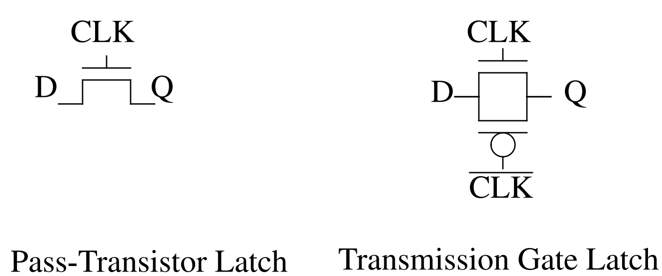
In practice, some paths have widely differing min and max delays. The most difficult ones to manage are those which are critical in the worst case, but so fast in the best case that they could cause min-delay failure.

# 5.0 Latch Circuits

There are many ways to build latches and flip-flops. A flip-flop can just be constructed from two back-to-back transparent latches. A pulsed latch is just a transparent latch controlled by a pulse rather than a 50% duty cycle clock. Thus, the most important building block is the transparent latch.

The simplest dynamic latch is just a pass transistor, shown in Figure 4. To provide rail-to-rail output swings, a full transmission gate is usually a better choice.

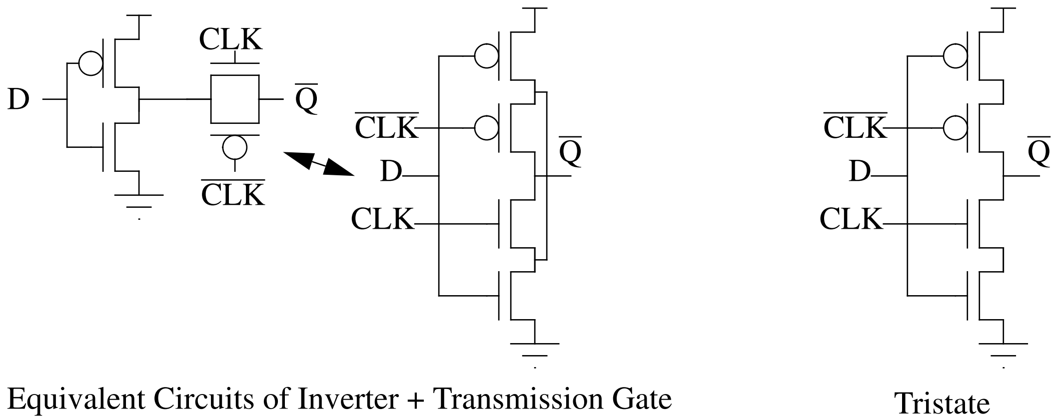
**FIGURE 4. Simple Latches**




Such a latch must drive only a capacitive load; it cannot drive diffusion inputs of other pass transistors because charge sharing could cause the output to glitch when the latch is opaque. Coupling onto the output node is also a problem because it is a dynamic node. Latch setup time depends on both the load and driving gate, just as the delay through an ordinary transmission gate depends on the driver and load.

To solve these problems, inverters may be placed on the input, output, or both. When an inverter is placed on the input, sometimes the inputs to the transmission gate are separated to create a tri-state inverter instead, as shown in Figure 5. The performance of both designs is comparable; removing the shorting connection slightly reduces drive capability, but also reduces internal diffusion parasitics.

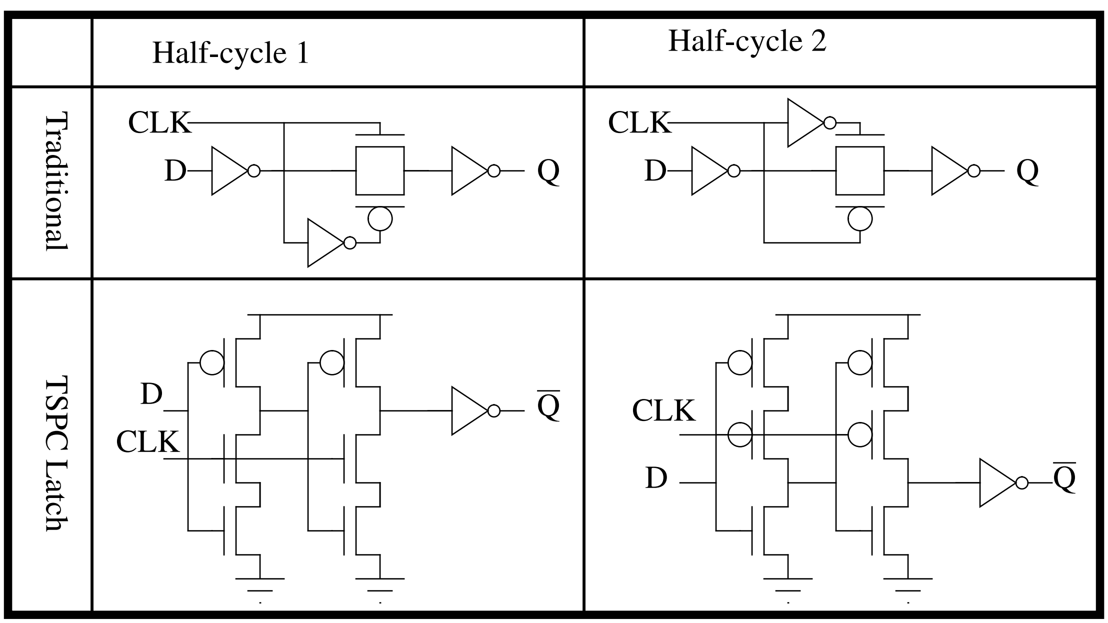
**FIGURE 5. Inverter + Transmission Gate vs. Tri-state**




Note that it is important for the clock input to be on the inside of the stack. If the data input were on the inside, changes in the data value while the latch was off could cause charge sharing and glitches on the output.

To minimize clock skew, most chips distribute only a single clock signal across the entire chip. Locally, additional phases are derived as necessary. For example, an inverter may be used to create $\overline{\text{CLK}}$ from CLK. Once upon a time, some designer felt that generating local phases was also bad for skew. The True Single-Phase Clocking (TSPC) techniques enjoyed a period of popularity and were actually used on the Alpha 21064. They are shown below:

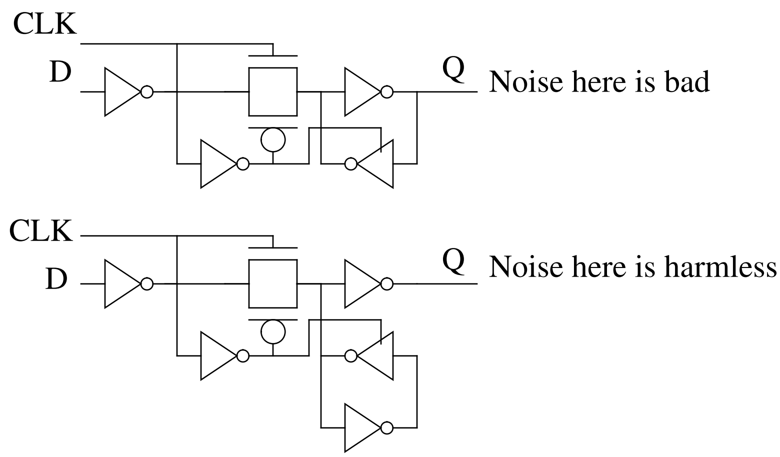
**FIGURE 6. TSPC vs. Traditional Latches**




TSPC latches have a longer propagation delay because the data passes through more gates. Since inverting the clock can be done relatively cheaply and TSPC latches are slower and larger than normal latches, TSPC is seldom used anymore.

All latches presented so far are dynamic; if left indefinitely, they may leak to an incorrect value. To allow a system to run at low frequency or with a stopped clock, the dynamic node must be staticized. This can be done with a weak feedback inverter or with a tri-state feedback inverter. Weak inverters introduce a ratio problem and must be weak enough to be overwritten by the input driver. Tristate inverters are thus safer and avoid contention power. Feeding back from the output inverter is risky because noise on the output can overwrite the contents of the latch. Thus, the output is often buffered with a separate gate, as shown in Figure 7.

**FIGURE 7. Staticizing Latches**



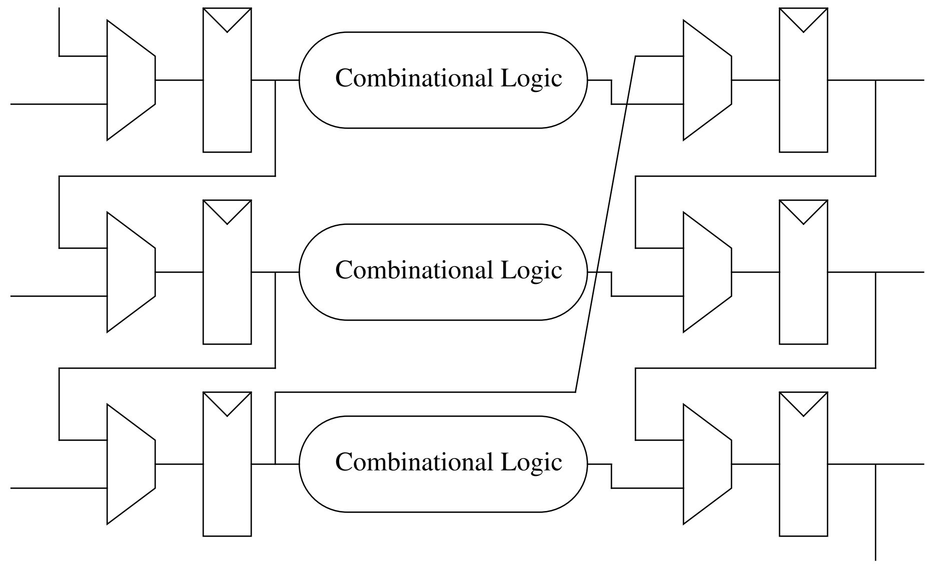

For highest performance custom design, a bare transmission gate is excellent and was used by DEC in the Alpha 21164. Care should be taken with the driver and load that setup time can be met. For synthesized design, an inverter/transmission gate/inverter combination latch is safer to guarantee setup times and safety of the dynamic node. In any case, designs which must support stop clock or low-frequency operation should staticize the dynamic node with cross-coupled weak inverters.

# 6.0 Scan

Testing chips is becoming very difficult. Traditional test techniques involved probing top-level metal lines with a tiny metal probe or electron beam. As more layers of metal are used, most nodes of interest become blocked by the power, ground, and clock nets which occupy most of the upper levels of metal. Moreover, “flip-chip” packaging is rendering internal nets even more difficult to reach. The technique involves placing solder balls directly on top-level metal pads, then flipping the chip so the pads align with the package or board and heating the solder to make the connection. In such a style, the only way to probe metal lines on the chip is to physically drill or etch a hole through the back of the substrate! This is slow, expensive, and hard to do.

Thus, most designers use scan chains to test their chips. Flip-flops are connected into a chain with a special scan multiplexor so their contents can be marched through the chain instead of captured from a previous stage of logic, as shown in Figure 1. Each flop in the 3-bit datapath can receive data from a normal input or from the previous flop in the chain.

**FIGURE 8. Flops connected in scan chain**




Scan contributes nothing of value except ease in testing, so designers want to minimize its impact on the chip. In particular, a good scan system should feature:

- Zero increase in cycle time
- Zero design time
- Zero timing critical scan signals
- Zero extra area

Adding multiplexors to the critical path is usually unacceptable for speed. With a good scan methodology, the increase in cycle time is just a small amount of extra diffusion loading on latches or domino gates. The design time can be minimized with a clear and simple set of design rules worked out in advance and integrated into the library so designers don't have to concern themselves with scan. A good methodology also does scan with only a small number of scan signals, which may be operated arbitrarily slowly so poor routing and buffering of the scan signals doesn't degrade chip operation. Unfortunately, the area criteria is not met well; most scan schemes require extra "shadow latches."

# 7.0 Clock Generation

Routing a single clock around a chip is a difficult problem. Routing multiple clocks with little skew between the clocks is even harder. Since most circuits two or more phases locally, the usual practice is to distribute a single global clock, then locally derive the multiple phases near where they are necessary. "Near" is usually defined by the maximum wire length a final stage clock buffer can drive before introducing unacceptable RC skews. This is on the order of 2 mm and gets smaller as clock frequencies increase and skews must decrease. Let us first look at the global clock generation, then explore local clock generation.

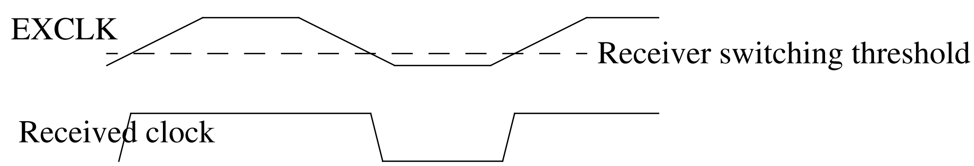
## 7.1 Global

At first glance, it seems a global clock could be produced by simply buffering the input from an external oscillator. There are two difficulties with this technique: phase alignment with the external clock, and duty cycle variation. These difficulties lead most designers to use a phase-locked loop (PLL) or delay-locked loop (DLL) in their global clock generator.

If a chip had no I/O, delays between the external clock and on-chip clock would not matter. However, if the chip communicates synchronously with other chips, the clocks of the various chips must be aligned. A common way to do this is to align all other chip clocks to the external reference with a PLL or DLL.

Even if alignment were not a problem, duty cycle uncertainty would be a challenge. Distributing a high-speed external clock over a PC board and through an IC package is challenging. Since the edge rates are not very sharp, it is difficult to identify exactly when a clock transitions. If the threshold detection is not at exactly 50%, there will be duty cycle variation in the detected signal, as shown in Figure 1.

**FIGURE 9. Duty cycle variation in received clock caused by receiver threshold offset**




To reduce this duty cycle uncertainty and more easily distribute fast clocks, the clock is often sent as a differential small-swing signal. A differential amplifier on the chip must restore the clock to a full-swing single signal. Offset voltages in the amplifier will still cause certain amounts of duty cycle uncertainty.

One way to eliminate duty cycle uncertainty is to provide an external double-frequency clock, then divide by two with a toggle flip-flop to produce a normal frequency clock with exactly 50% duty cycle. However, providing even a 1x clock at high frequency is hard, so providing a 2x clock has gone out of favor. Often a PLL is used to adjust the duty cycle. DEC does not use any loop on the 21164 Alpha, but instead uses a duty-cycle correction circuit which measures the duty cycle of the received clock and adjusts the generator to compensate, reducing duty cycle variation from +/- 5% to +/- 0.5%.

There are many ways to build loops to adjust the clocks. Usually an expert loop designer knows the secret incantations to make robust, reliable loop circuits. We'll just explore the basics to understand some limitations of loops. All loops take a reference clock and use feedback control to create another clock with a particular phase and frequency relationship to the reference.

Since no loop is perfect, the output of the loop may have some phase error relative to the desired output. This error can change over time; the change is called jitter. Important specs on jitter are “cycle-to-cycle” and total. Cycle-to-cycle jitter is the maximum change in phase from one cycle to the next. It has many of the same characteristics as clock skew. Total jitter is the maximum variation of phase over many cycles. Two independent loops may vary in phase by total jitter.

Phase-locked loops use a voltage-controlled oscillator. The control voltage is adjusted until the PLL is oscillating with the same phase and frequency as the reference clock. The PLL can also do frequency multiplication, creating output frequencies of rational multiples of the reference. An error in the control voltage impacts the frequency of the output. Since phase is the integral of frequency, errors can accumulate over multiple cycles. This means that total jitter can be quite large.

Delay-locked loops use a voltage-controlled delay line. The reference clock is delayed to produce an output with the desired phase offset. For example, a phase offset of 360 degrees gives an output exactly aligned to the reference. Simple DLLs cannot do frequency multiplication. However, errors in the control voltage only impact the delay, not frequency, so jitter does not accumulate over multiple cycles. DLLs therefore usually have better jitter performance.

Since noise on the control voltage causes jitter, it is important to remember that loops are really analog circuits and must be shielded from the noisy power supplies of digital chips.

## 7.2 Local

Once the global clock is generated and shipped to various units around the chip, local clock phases can also be generated. Local clock generators serve to buffer the huge clock load and to reshape the clocks. Examples of shaping include delaying edges, inverting the clock, and producing pulses or longer duty cycle waves.

Local clock generators require careful design to minimize the skew they introduce. Issues include input slope, delay matching, and layout matching.

The final driver should produce sharp edges so that variation of input trip-points on clock receivers doesn’t show up as large clock skew. This usually means the final local buffer should be a fanout-of-3 inverter. The gate should be sized for equal rise and fall time to avoid duty cycle errors.

The other buffers in the local clock generator should be designed to match well across process variation so they don’t introduce skew. For example, if one local clock generator used a 3-input NAND and another used a 3-input NOR, the local clocks could be skewed significantly in the FS and SF corners. Ideally, one could use only inverters in the buffer network, but most real chips need some amount of clock gating with a 2 or 3-input NAND gate. NOR gates are best avoided because they require huge PMOS transistors to drive large loads with equal rise and fall times.

Good layout of local clock buffers is necessary for low skews. Mask misalignment differs in the horizontal and vertical directions, so all transistors should be drawn in the same direction in clock buffers for best matching between buffers. Polysilicon is usually etched with plasma. The etch rate has some dependence on the total amount of material which must be removed. Therefore, when a large amount of poly is etched in a local region of the chip, the rate is slower and channel lengths of the remaining poly are longer than nominal. When a small amount is etched, the rate is faster and the channel lengths are shorter. To get closely matching transistors across the chip, the average polysilicon density in a region should be kept nearly constant. Sometimes, extra polysilicon wires are drawn around buffers to raise the density in sparse regions.

Delay chains in local clock buffers to create overlapping clocks will vary with process and do not track with clock frequency. Thus, a nominal 1/4 cycle delay at full frequency becomes only a small fraction of the cycle delay at low frequency. An alternative to such open-loop delay chains is to use a delay-locked loop in the local clock generator which can produce clock phases of arbitrary timing relationship independent of operating frequency and process parameters. Unfortunately, since it is infeasible to route many clocks all over the chip, a single global clock must be distributed to many local DLLs. Two clocks in different local clock domains therefore see the worst case jitter between the independent DLLs. This jitter is usually far worse than the open-loop skew.

# 8.0 Clock Distribution

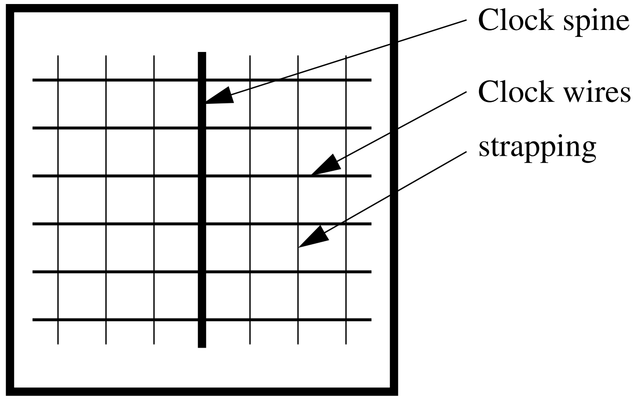
Distributing the clock is another challenge. Again, we can divide the problem into global distribution, from the phase locked loop to the local clock generators, and local distribution, from the local clock generators to clocked elements.

## 8.1 Global

When transistors were much slower than wires, clocks could be routed about the chip in an ad-hoc manner with little penalty. Now that wire delay is important, designers want to minimize the variation in wire RC delay between points on the chip. Popular global distribution networks include grids, H-trees, and hybrids.

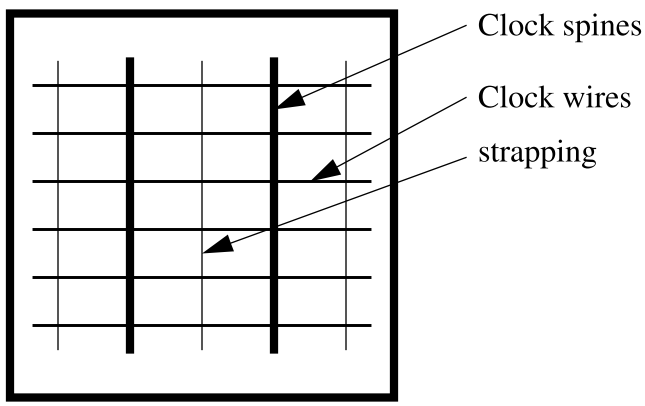
The Alpha 21064 demonstrated a clock grid, as conceptually shown in Figure 2. The final clock driver uses 35 cm of transistor width (!) to drive a 3.25 nF clock load. The clocks are driven horizontally from a clock spine at the center so there is very little skew near the spine and more skew near the edges. The chip is floorplanned so that most skew-sensitive blocks are as close to the spine as possible. The clock lines are strapped vertically. While this does not impact the RC delay from the spine horizontally to a load very much, it does guarantee the local skew between two points on adjacent horizontal lines stays small. The reported RC delay skews are under 300 ps.

**FIGURE 10. Alpha 21064 Clock Grid**



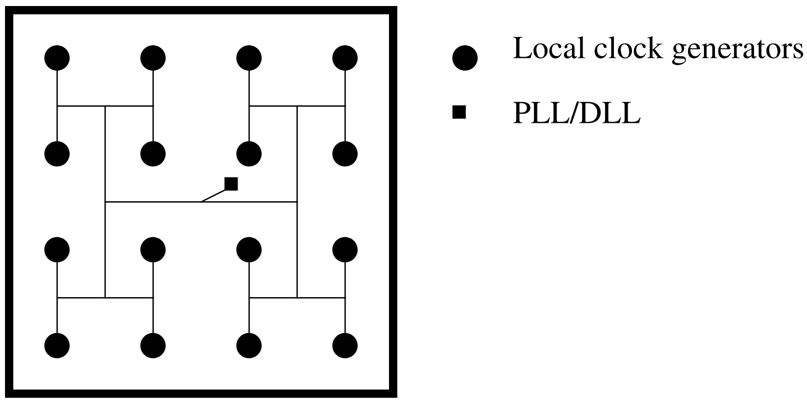

The Alpha 21164 [JSSC Nov. 96] finds that the skew from the spine to the edge of the chip is too large. Thus, it uses two clock spines, so that global clock lines never must travel more than 1/4 the length of the chip. The reported RC delay skews are 90 ps.

**FIGURE 11. Alpha 21164 Clock Grid**




Even with such a grid, a chip will see systematic skews caused by the RC delay driving a heavy load 1/4 of the chip away. An alternative is the H-tree network, shown in Figure 4. An H-tree is a recursive space-filling curve with beautiful mathematical properties that CAD researchers like. A key property is that the distance from the center of the tree to any end is equal. Therefore, the global clock can be distributed to local clock generators at each endpoint with nominally zero RC skew. Unfortunately, difference in loading presented by each generator leads to skew. This skew can be reduced by adding dummy capacitance to the lighter-loaded endpoints, at the expense of power consumption. H-tree layout is also a problem. If the tree were routed in a single layer, it would prevent any long signal routing in that layer. If the tree were routed in two layers, many vias are necessary to change layers each time the wire direction changes.

**FIGURE 12. H-Tree Clock Distribution**




The H-tree works reasonably well for course-grained distribution, but requires many branches for the final distribution. The Alpha 21264 [ISSCC97 slide supplement] uses an H-tree to distribute the clock to 16 regions around the chip, then uses a grid within the regions. They claim an RC skew of under 50 ps with this technique.

## 8.2 Local

Locally, clock distribution depends strongly on the actual logic which must be clocked. A good local distribution scheme will keep local wires short to minimize RC delay. In random logic, it may be best to layout all clocked elements in a small region to avoid a rats nest of clock wiring around the synthesized block. In datapaths, local clock buffers may sit on the edge of the datapath and drive their clock across the bits. If the RC skew across 64 bits of a datapath becomes unacceptably large, it can be cut in 4 by driving the datapath from either the middle or from both edges. Routing signals into the middle of the datapath is often inconvenient, so driving from the edges is usually more popular.

# 9.0 Skew Summary

Measuring total clock skew requires accounting for many components. Most companies presenting chips at conferences neglect to report some components in order to make their specs look better than they actually are.

Any system which uses a loop must deal with cycle to cycle jitter. This jitter can shorten the cycle time, so it has many of the same effects as clock skew. However, almost nobody reports jitter in their skew budget.

Clock skew results from variation in the global distribution, variation in the local buffer, variation in the local distribution, and variation in the switching point of the receiving gates. This variation results from both differences in RC delay and differences in gate delay. DEC makes pretty plots of the maximum difference in RC delay between points on a chip under otherwise nominal conditions and calls it their clock skew. This is very misleading.

The distribution skews are minimized by making the length and load on all global distribution lines as close to equal as possible and by immunizing the length (and hence RC delay) of local distribution lines. It is also important to limit the amount of buffering necessary, since buffers scattered around the chip see different voltage, temperature, and processing and hence have different delays. Finally, matching the local buffers in fanout and structure is vital.

Even doing everything right, reducing skew below 200 ps is very difficult. As processes scale, the raw gate delay improves, which benefits buffer delay variation. However, wires get longer and wire RC per length increases, making wire skew more severe. Also, process tolerances will probably get sloppier, making matching of devices worse across a die. To a certain extent, these problems can be countered by increasing the wiring and power resources dedicated to the clock network.

For current designs, skews are a small enough fraction of the cycle that they can be hidden with transparent latches or skew-tolerant domino. As cycle times shrink more rapidly than clock skew, it may become impossible to tolerate global skew. One solution is to eliminate the clocks, using asynchronous logic. We will look at asynchronous circuits in the future, but such circuits have many severe problems of their own. A more immediate solution may be to operate local regions with better-controlled skew at high frequency, then communicate globally at lower frequency so that the skew is a smaller fraction of the cycle.