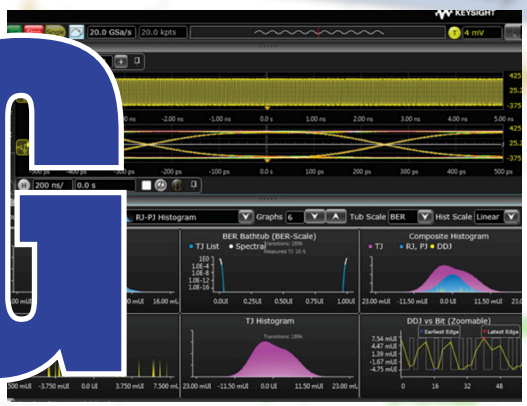
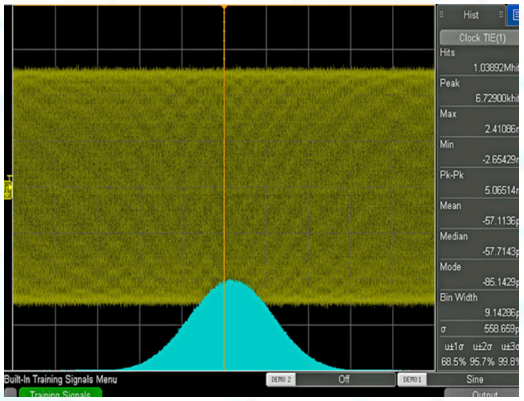
Electronic Design® LIBRARY

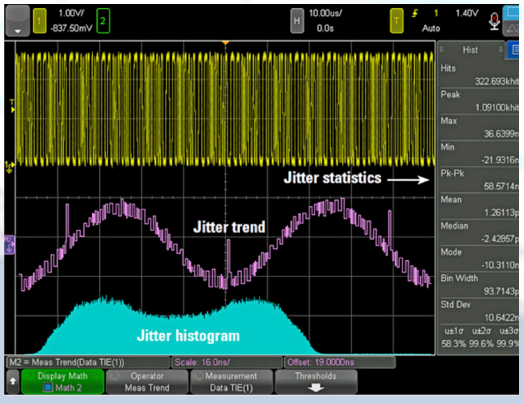
A compendium of articles from *Electronic Design*

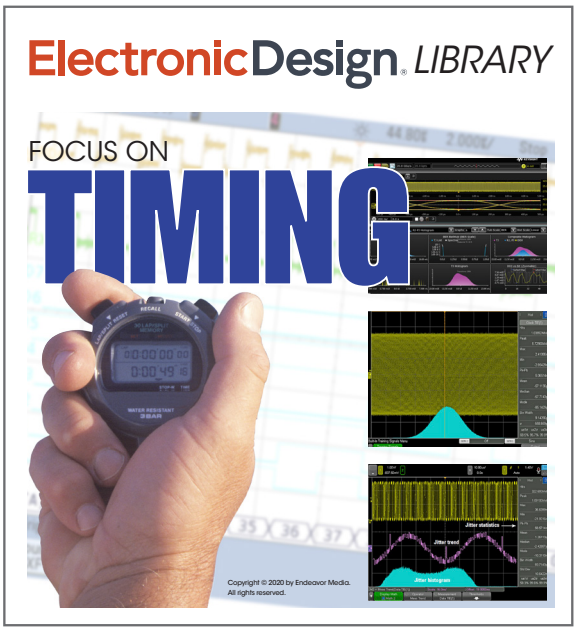
FOCUS ON

TIMING





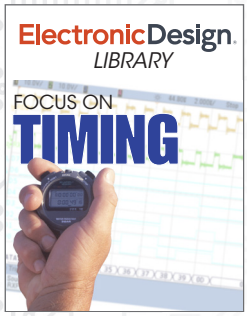


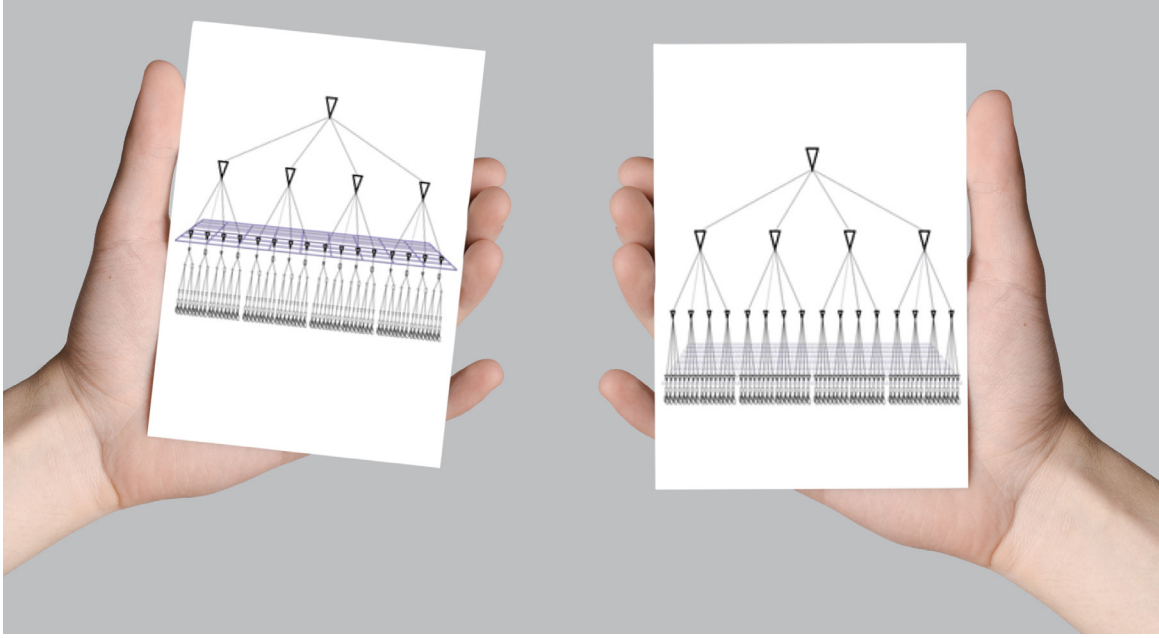


TIMING IS NOT HARD - you just need to stay on the clock. Which clock and how it is synchronized can be a challenge especially when multiple sources are involved and tight timing constraints are imposed due to frequency, jitter and the length of transmission lines. Multisource clock-tree synthesis is an option for clock distribution, joining conventional clock-tree synthesis and clock mesh that is being utilized in more advanced designs. This and other topics like the difference between jitter and noise provide insight into system clock design and distribution.

Bill Wong
Editor,
Senior Content
Director

CONTENTS





CHAPTER 1:

What's The Difference Between CTS, Multisource CTS, And Clock Mesh?

HARVEY TOYAMA, Technical Contributor

Multisource clock-tree synthesis is a relatively new option for clock distribution, joining conventional clock-tree synthesis and clock mesh. This article contrasts and compares these methods, examining the tradeoffs between them.

Up to now, there have been two main methods of clock distribution for large, high-performance designs: conventional clock-tree synthesis (CTS) and clock mesh. Multisource CTS has emerged as a new method that is a hybrid of these two. This article explains the differences between CTS, multisource CTS, and clock-mesh distribution technologies.

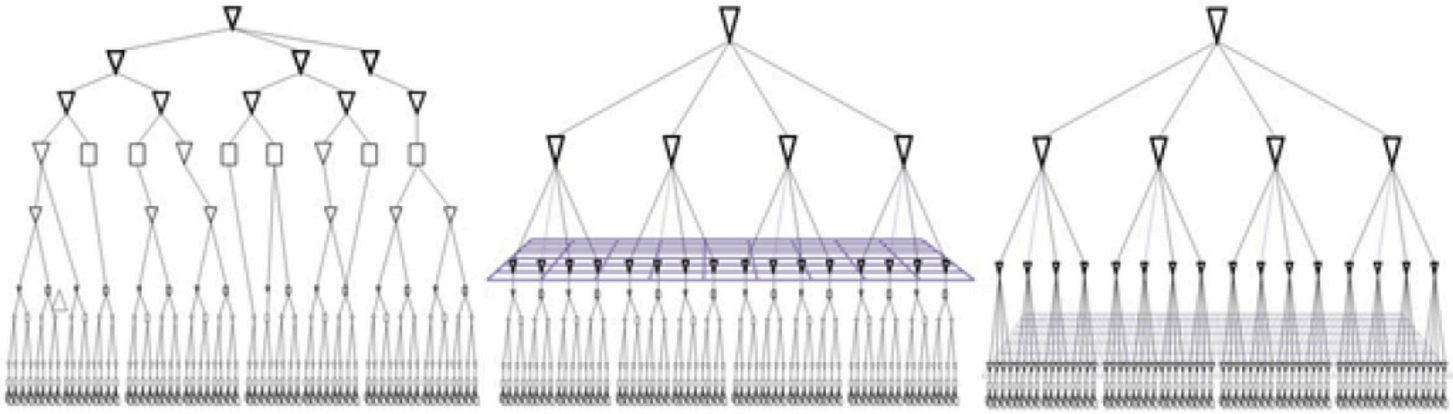
Key Differences Between CTS, Multisource CTS, And Clock Mesh

There are four key differences between conventional CTS, multisource CTS, and clock mesh: shared path, mesh fabric, design complexity, and timing analysis. Each subsequent section discusses each of the three clock distribution methods with respect to these key differences. At the completion of the article, you will know the differences and be better equipped to try a new method that may be better suited to your next design start.

Amount Of Shared Path

The most obvious difference is the structural depth of the shared path between the clock root and the sinks. Consider an example of the same set of sinks addressed by each of the three clock distribution methods **(Fig. 1)**.

A conventional clock tree, shown at left, is characterized by an organic tree structure from the clock root that branches out to each of the sinks in the design. There is unlimited depth for both buffer and clock-gating levels. Most of the sinks in the design share very few paths back to the clock root—so few, in fact, that for any two sinks in the design, the only reliably shared part of the path is the root buffer.



1. The most obvious difference between CTS, multisource CTS, and clock-mesh structures is the depth of the shared path between the clock root and the sinks. Shown here is the same set of sinks addressed by each of the three clock-distribution methods (from left: conventional CTS, multisource CTS, and clock mesh).

Multiple clock trees of small to moderate depth primarily distinguish multisource CTS, shown at center. There are typically between three and nine levels of clock gating and buffers. The multiple clock trees are at the bottom of the structure below the mesh grid and all of the structure above the mesh form a shared path back to the clock root. A substantial portion of the overall insertion delay of the clock is in the form of a shared path.

Clock mesh, shown on the right, is characterized by an extremely shallow logic depth below the mesh, usually just a single buffer or clock gate directly driving the sinks. Most of the insertion delay in a clock mesh design is a large, shared path from the root to the mesh.

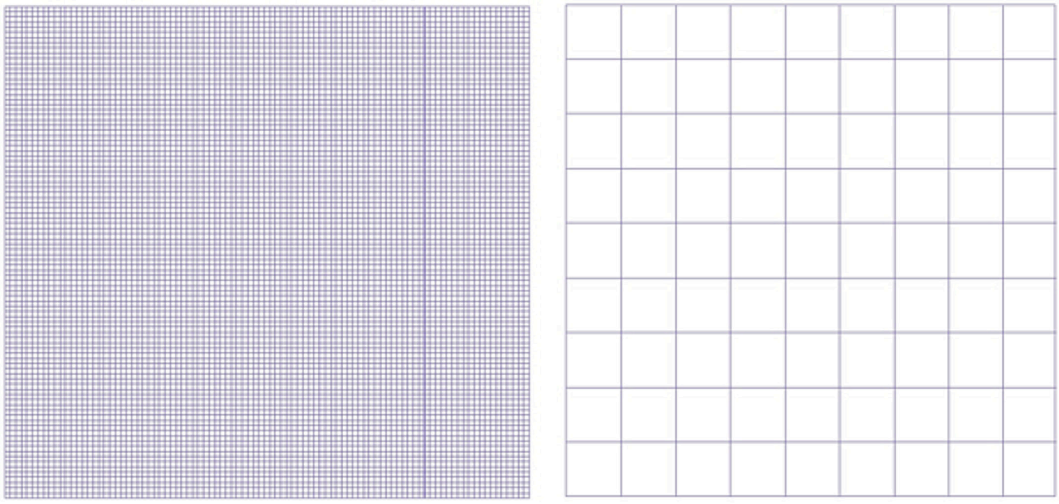
OCV Benefit Of Shared Clock Path

The respective logic depths (unlimited, moderate, and very shallow) are inversely related to the level of shared path between the sinks and the clock root. Path sharing reduces the impact of on-chip variation (OCV) effects on the design because when the sinks share the same clock path to the root, any process-variation occurrence in that path affects both flops equally and all timing assumptions are preserved. In the absence of path sharing, one must increase the clock margin by a derating factor to account for the possibility that either or both the launch and capture flip flops experience a process-variation phenomenon.

We may define the extra margin by multiplying the insertion delay of the non-shared path by a derating scalar, typically between 7% and 10%. Worse yet, it is applied in a range of plus or minus the derating factor. We then add the product to the timed skew of the design and derate the clock-frequency performance of the design.

The current technology nodes encourage large designs with many different functions. As designs grow larger, the impact of OCV derating increases. Of the three clock-distribution methods, conventional CTS is the most adversely affected by OCV derating, and the growing trend is to move away from conventional CTS for high-speed designs.

On the other extreme, the sinks in a clock-mesh design share the overwhelming majority of total clock path. The result is that the measured clock skew increases very little due to OCV derating, preserving the high performance of the design. This is the main reason that clock-mesh design has long been the preferred clock-distribution method deployed



2. The multisource CTS mesh fabric (at right) is one order to two orders of magnitude less dense than the clock-mesh fabric.

by performance-oriented processor designs, whether arithmetic and logical units (ALUs) or graphical processing units (GPUs).

Multisource CTS falls between conventional CTS and clock mesh with regard to the amount of shared path. The high flexibility of multisource CTS enables the designer to trade off the clock level depth of the multiple trees against the OCV immunity of the design. Multisource CTS has other areas of flexibility that we will discuss later.

Mesh Fabric

The mesh fabric is another obvious difference between conventional CTS and the two other methodologies. Clock mesh and multisource CTS both use a mesh fabric, though there is a large difference in the density of the mesh deployed.

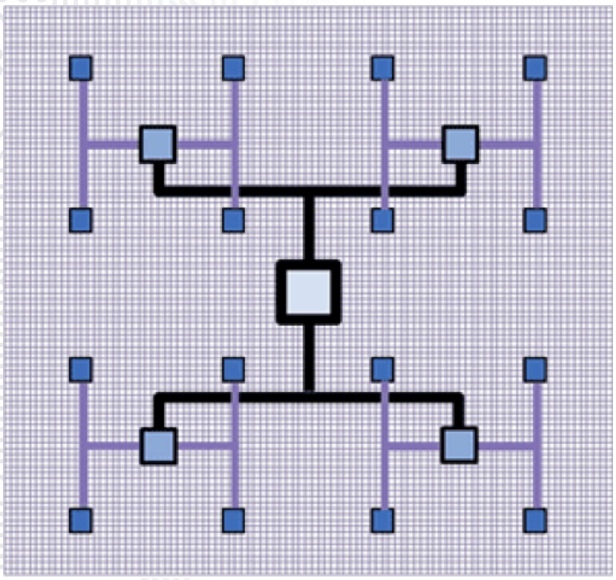
Clock mesh uses a dense mesh fabric as the final component of the shared path of the design. This fabric is typically about as dense as the power/ground meshes and consumes significant routing resources. Most implementations place the fabric at the highest routing layers. This minimizes the impact on data signal routing, which usually takes place at lower levels, while concurrently taking advantage of the low resistance of the wider redistribution layer (RDL) or other higher metal layers for high-speed routes.

The mesh fabric provides uniformity across its entire expanse. The multiply driven fabric smoothes out the arrival time differences of the clock at each driving point. The effective result is that the clock skew at the fabric is zero. The skew component of the design is thus limited to the wire segment attaching the buffer or clock gate to the fabric and the wires and flops thereby driven. This explains the ultra-low skew values achieved with clock mesh.

The multisource CTS mesh fabric is one order to two orders of magnitude less dense than the clock mesh fabric **(Fig. 2)**. One must take greater care with multisource technology to ensure that the skew performance at the fabric meets the design objectives.

Both clock mesh and multisource CTS have more immunity to insertion delay because the prevalence of shared path minimizes the portion of the insertion delay that is exposed to the OCV derating. Both methods also benefit from a highly structured buffer topology that drives the fabric.

We normally configure the pre-mesh drivers as multilevel H-Trees. An H-Tree, as the



3. In this example of H-Tree routing, note the clock-root buffer at the center of the design. For visibility, the routes on the two H layers are in different colors. Here, the top-level "H" lays on its side, while the four lower-level "H" structures (in purple) are in a classic "H" orientation.

name implies, is typically a configuration of five drivers that trace the center and endpoints of an "H" pattern **(Fig. 3)**. Notice the clock root buffer at the center of the design. For visibility, the routes on the two H layers are in different colors. In this example, the top-level "H" lays on its side, while the four lower-level "H" structures, shown in purple, are in a normal "H" orientation.

Power Tradeoff Differences

The coarse pitch of the multisource CTS mesh fabric has the benefit of using considerably less power than the extremely fine pitch of a clock-mesh fabric. Because there is one order to two orders of magnitude less metal in the multisource CTS fabric, the reduction in the power requirement is considerable. Whereas clock mesh consumes between 20% and 40% more power than the same design implemented with conventional CTS, a multisource design will have power numbers much closer to conventional CTS than clock mesh.

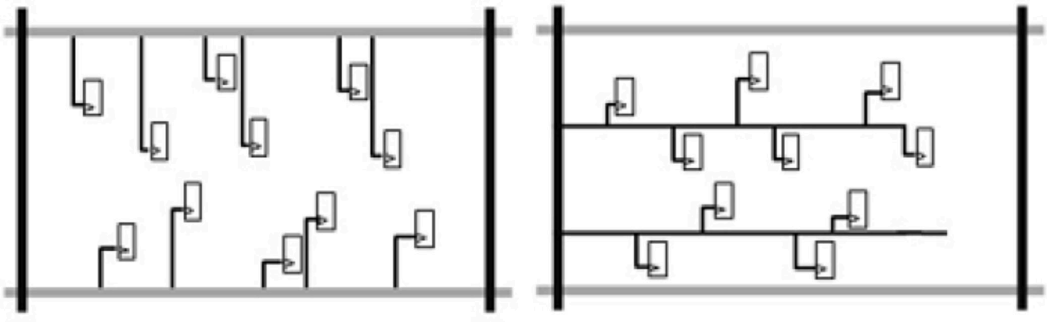
Furthermore, multisource CTS allows greater clock gating depth, enabling more complex clock gating schemes, which contributes to additional power savings. The flexibility of multisource CTS enables clock gating complexity and mesh fabric density tradeoffs versus power. This is not possible in a clock-mesh approach.

Attaching To The Fabric

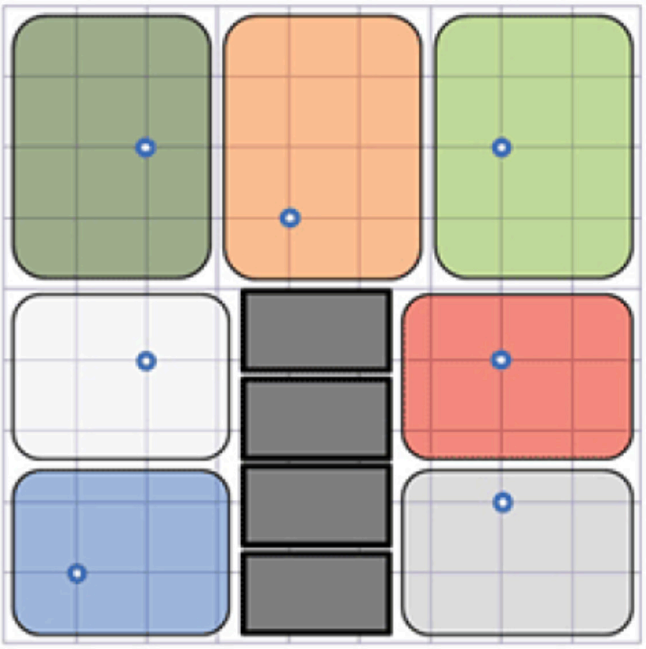
Another difference between multisource CTS and clock mesh is how the design logic attaches to the mesh fabric. In clock mesh, the dense fabric defines relatively small bins that contain cluster or sub-cluster amounts of logic. These structures of buffers or clock gates and the sinks they drive are often called twigs to keep within the arboreal metaphor of trees and branches.

One may attach clock mesh twigs by either comb routes or fishbone routes to the horizontal and vertical spines of the mesh **(Fig. 4)**. Comb routing minimizes the skew but trades this benefit off against routing resources. For most designs, fishbone routing is the better tradeoff.

Whether the routing method is comb or fishbone, clock mesh twigs attach to the nearest



4. One may attach clock mesh twigs by either comb routes or fishbone routes to the horizontal and vertical spines of the mesh. Comb routing minimizes the skew but trades this benefit off against routing resources. For most designs, fishbone routing is the better tradeoff.



5. Multisource CTS clock trees attach to the coarse mesh fabric at locations called tap points, which are defined to provide low skew and insertion delay. We typically create tap points at the intersection of the spines of the coarse mesh fabric. Shown is a design with seven tap points defined.

point along any spine to attach to the mesh fabric.

By contrast, multisource CTS offers much larger logic groupings that are themselves small clock trees. Multisource CTS clock trees attach to the coarse mesh fabric at locations called tap points, which are defined to provide low skew and insertion delay. We typically create tap points at the intersection of the spines of the coarse mesh fabric **(Fig. 5)**.

Each tap point is the local root of one of the multisource CTS clock trees. The root buffer normally attaches to the intersections with stacked vias directly to the input pin of the buffer.

In clock mesh there is no concept of assigning sinks to a clock root because the fabric is so dense and each twig is small. But in multisource CTS, tap-point assignment is an important step. In addition to showing the tap points at grid intersections, Figure 5 shows distinctly colorized areas that have been assigned to the tap point within the colorized boundary. The tap points are the clock tree roots, and the assigned sinks define the boundaries of each clock tree.

Mesh-Less Multisource CTS

There are even differences within the multisource CTS category. One may implement multisource CTS without a mesh fabric at all. In this case, the H-Tree endpoints are the tap points to which the sinks in each region are attached.

Mesh-less multisource CTS offers designers yet another set of tradeoffs. In this case, the tradeoff is between OCV tolerance and the ease of the flow. The flow is easier because there is no need to determine the correct pitch of the mesh fabric. It is also easier because timing the design is simpler.

Design Complexity

The third area of difference among these methods is how the complexity of the clock-gating plan and the floorplan influences the effectiveness of the clock-distribution approach. Conventional CTS is the most accommodating approach for dealing with design complexity. It is the baseline against which to judge clock mesh and multisource CTS.

Clock mesh is the most rigid of the three approaches. An ideal clock mesh design has no RAMs, ROMs, or other hard blocks. Indeed, it is a flat sea of gates. This is ideal for clock mesh because there are no obstructions that prevent the placement of pre-mesh H-Tree buffers such that each "H" is ideal. The lack of obstructions also enables the H-Tree routes to be perfectly straight, making it easier to ensure an ideal balanced H-Tree. Clock mesh also benefits from a shallow, uniform design topology below the mesh fabric to comply with the limit of two levels of clock buffers or clock gating.

As in other areas of difference, multisource CTS falls between conventional CTS and clock mesh with respect to its handling of design complexity. The depth of the multisource clock trees tolerates most clock-gating plans well, and the smaller pre-mesh H-Tree means fewer drivers to account for amid RAMs and hard blocks in the floorplan.

Timing Analysis

In conventional CTS, we perform timing analysis with standard timing analysis tools, both the accepted signoff static timing engines and the similar timing engines embedded within the place and route tools. This makes conventional CTS the easiest method to time through every stage of the flow.

In the mesh topologies, circuit simulation is required to time the multiply driven mesh fabrics. This adds a level of complexity to the clock mesh and multisource flows that may at first seem prohibitive. However, the standard is for automation within the place and route tool to launch the simulation run and then annotate the timing values onto the design for subsequent static timing reports and analyses. While this mitigates the circuit-simulation learning curve somewhat, it cannot completely obviate some exposure to the underlying simulator technology.

Conclusions

As new technology nodes enable increasingly larger and more feature-rich designs, the choice of clock-distribution methodology becomes ever more important. Conventional CTS, which has traditionally been the default choice for all designs, may no longer be the optimal choice when an extremely high clock frequency is required.

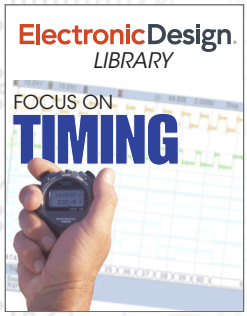
Thus, it is a good idea to broaden the clock-distribution skill set to include clock mesh and multisource CTS technologies. Experience with these methodologies enables designers to make the most optimal design choice given the design goals: clock frequency, OCV tolerance, power consumption, flow ease, and time-to-market pressure.

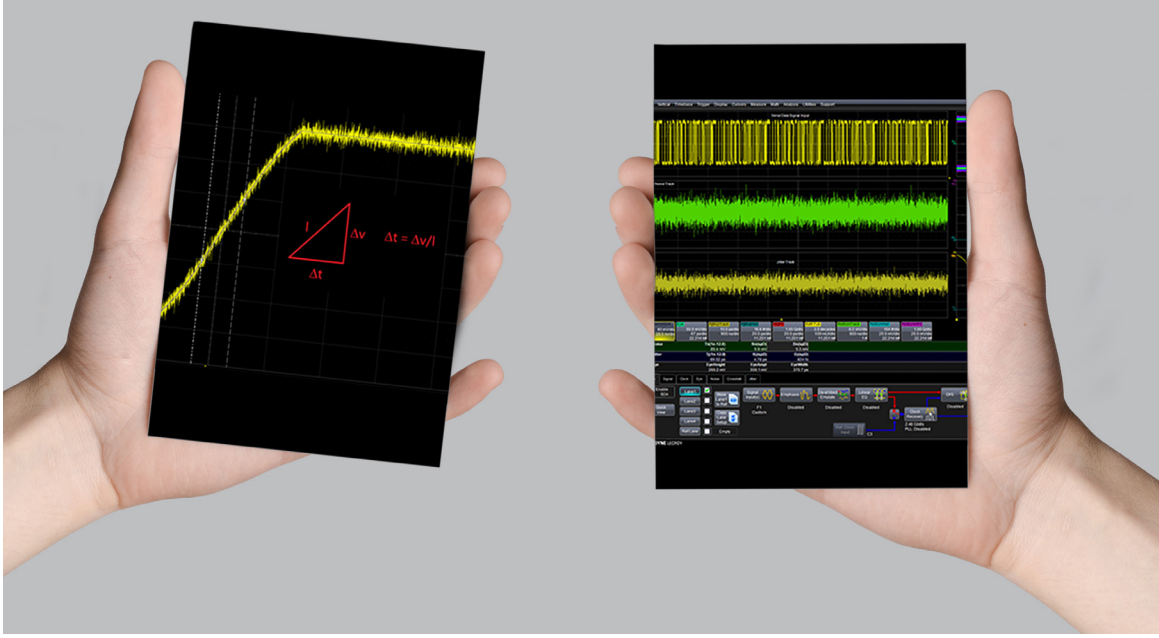
The three technologies explored in this article cover the polar extremes with conventional CTS delivering good to very good clock frequency, moderate OCV performance, best low-power profile, and best ease of use. On the other extreme, clock mesh exhibits the best clock frequency performance, the best OCV tolerance, the worst low-power profile, and the lowest ease of use.

Many designers are finding that multisource CTS offers an attractive, flexible solution between the two extremes of conventional CTS and clock mesh. Knowing the differences between these three clock distribution methods will enable the optimal implementation for your next design.

to view this article online, ☞ *click here*

☞ BACK TO TABLE OF CONTENTS





CHAPTER 2:

What's the Difference Between Jitter and Noise?

PATRICK MANNION, Technical Contributor

Though often comingled, jitter and noise are two completely different effects–gaining a better understanding of them can lead to more effective high-speed signal analysis.

The terms noise and jitter can mistakenly be confused, even more so as data rates and signal amplitudes have changed over time. They are distinct in nature, though, and good analysis tools can identify, isolate, and mitigate them.

This feature will explain the root causes of jitter and noise, and how to leverage that understanding to perform better high-speed signal analysis.

Definitions of Jitter and Noise

Let's start with the basic definitions. Jitter is the variation in a signal's timing from its nominal value. Jitter will manifest itself as variations in phase, period, width, or duty cycle. Noise is the variation of a signal's amplitude from nominal. Both noise and jitter can cause transmission errors and increase the bit error rate of a serial link.

Originally, testing of serial data links focused on characterizing and minimizing timing jitter because serial data amplitudes were relatively high. However, as serial data rates increased, signal amplitudes were being reduced, so noise became more of an issue. Noise and jitter become entangled when noise migrates into timing jitter whenever the signal crosses a logic threshold. Consider the situation shown **(Fig. 1)**.

The signal has additive vertical noise. If applied to a logic gate, the logic threshold is crossed earlier or later depending on the instantaneous amplitude of the noise. The effect on the output time shift (Δt) depends directly on the noise amplitude (Δv) and inversely with the nominal slope (I) of the signal edge, as illustrated.

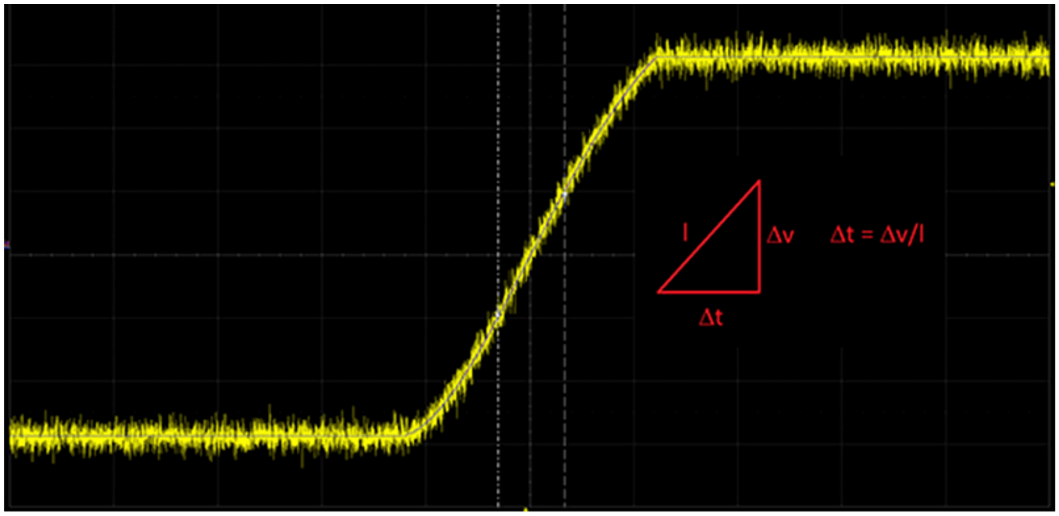
Wide-bandwidth digital oscilloscopes are used to make signal-integrity measurements on high-speed serial data links. Suppliers offer a number of serial-data-analysis options that characterize signal integrity by extracting jitter and noise and extrapolating their effects to predict bit error rates.

Serial Data Analysis

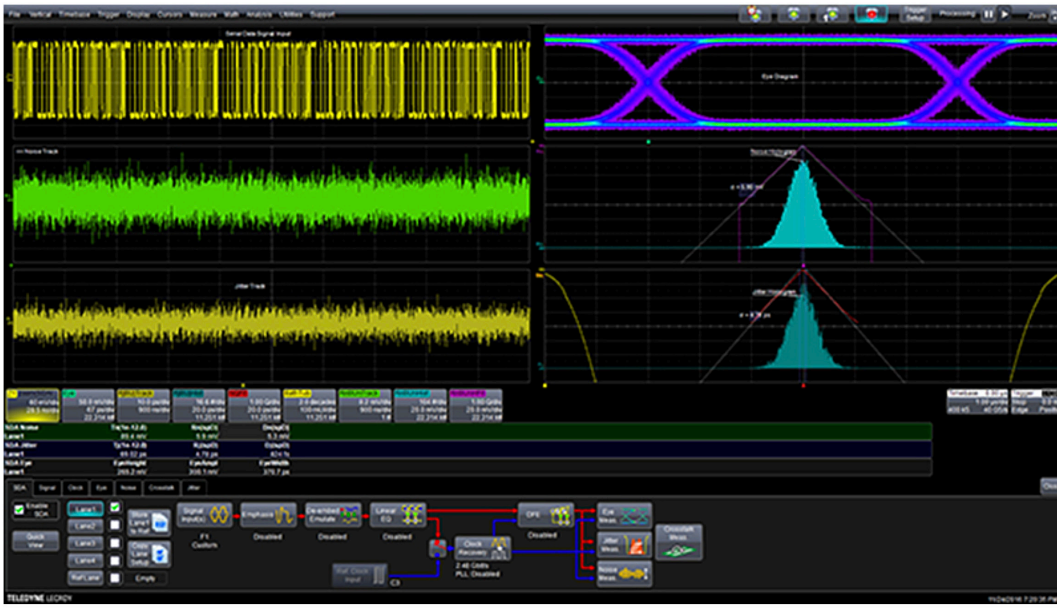
Consider the analysis on an oscilloscope using a serial-data-analysis program **(Fig. 2)**. An expanded view of the 2.48-GHz serial data signal is shown in the upper left display.

Jitter and noise can come from a variety of sources. They may be categorized broadly as having random or deterministic components. Deterministic components can be further defined as periodic, data-dependent, or bounded uncorrelated to the signal.

Knowledge of the type and magnitude of these components helps to identify their source, which is where the software steps in. Using a variety of methods, it separates the



1. When a noisy edge is applied to a logic gate, noise on the input moves the threshold crossing earlier or later in time, depending on the noise amplitude. The resultant output shifts by an amount proportional to the noise amplitude, and inversely proportional to the signal's nominal slope.



2. This typical serial-data-analysis display shows some of the analysis tools used to extract and measure noise and jitter, and predict bit error rate out to any desired record length.

various noise and jitter components, and by extrapolating this data it can predict, based on limited measurement record length, the bit error rates out to 1012 bits.

The two traces below the input signal are the noise-track and jitter-track signals, respectively. The track signals show the noise and jitter separated from the source signal; they are part of the isolation process that takes place in the analysis software.

The upper right grid of Fig. 2 displays an eye diagram. The eye opening represents signal quality in this diagram by the eye opening. Jitter tends to close the eye horizontally and noise closes it vertically.

The deterministic noise and jitter components are bounded; that is, they do not increase with increasing time or number of measurements. Their effect on the eye is predictable. Deterministic components are measured in the frequency domain by taking the fast Fourier transform (FFT) of the noise or jitter.

Random components are unbounded and increase with increasing time. Histograms of both random noise and jitter components, shown in the right-hand grids, second and third from the top, respectively, are used to predict the random components out to the desired number of measurements (generally the aforementioned 1012 bits) using statistical extrapolation techniques.

A bathtub plot is shown on the same grid with the jitter histogram. The bathtub curve shows horizontal eye opening as a function of number of measurements. Random jitter components affect the slope of the curves and deterministic jitter affects the horizontal location (i.e., increasing deterministic jitter moves the curves inward).

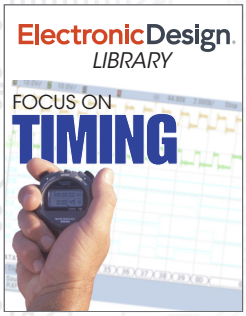
Beneath the graphical display is a table containing noise and jitter measurement parameters. They read the total noise or jitter and the amplitudes of the random and deterministic components. A third row in the table reads the key eye-diagram parameters.

Both noise and jitter affect serial-data bit error rates. As manifested on an eye diagram, noise can affect both vertical and horizontal eye closure. Jitter affects the horizontal closure. Special analysis software is able to isolate, process and measure noise and jitter components, making it possible to predict bit error rates for any number of serial bits.

The major suppliers of wide-bandwidth digital oscilloscopes all offer serial-data-analysis software packages that measure noise and jitter components.

to view this article online, ☞ *click here*

☞ BACK TO TABLE OF CONTENTS



CHAPTER 3:

Jitter Simplified

DANIEL BOGDANOFF, Oscilloscope Product Manager, Keysight Technologies

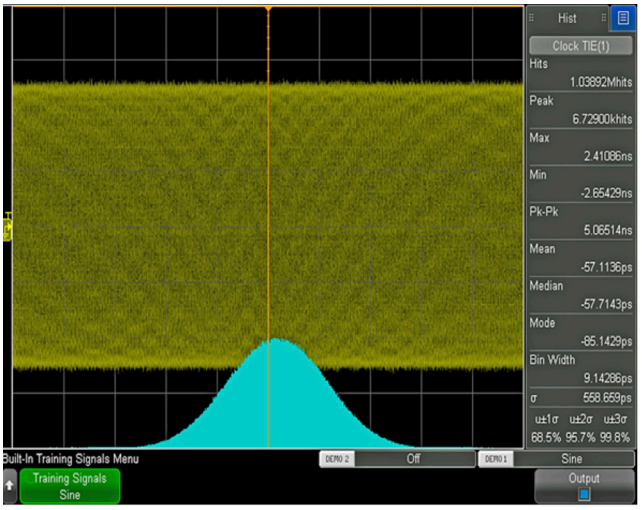
Learning the basics of jitter, specifically what causes it and where it comes from, helps boost confidence in the design and debugging of systems.

Does the word "jitter" strike fear into your heart? It shouldn't. While jitter will always be present in your designs, being able to identify causes and sources of jitter will allow you to confidently design and debug your systems.

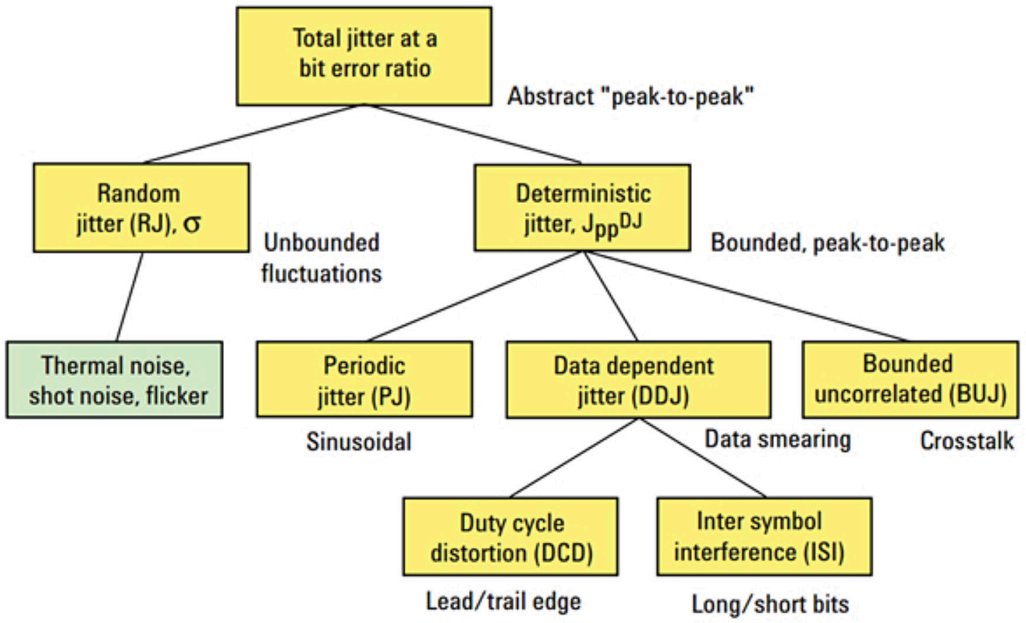
Essentially, jitter is where your signal's edges actually are compared to where you want them to be. If your signal's edges are too far off, it will cause errors in your system. "Total jitter" can be broken down into a number of components that have different root causes with different implications for your designs. Learning the various jitter components and a few key analysis skills (interpreting eye diagrams) is essential when designing and debugging high-speed systems.

Characterizing Jitter

Before discussing how to separate jitter into its components, we'll first look at how jitter is measured plus a few key terms. Jitter is often visually depicted by a probability distribution function (PDF). The best known example of a PDF is a bell curve, which is the PDF of a Gaussian distribution. On your test-and-measurement equipment, PDFs of a system's jitter are often called a histogram. **Figure 1** shows a jittery signal and its histogram/PDF on a Keysight InfiniiVision



1. Because the PDF is Gaussian and only has one peak, we know that RJ is the dominant component of jitter in this system.



2. Jitter consists of multiple components.

6000 X-Series oscilloscope.

Common jitter measurements include cycle-to-cycle jitter, period jitter, and time-interval error (TIE). Cycle-to-cycle jitter is usually measured as a peak-to-peak value, and is the maximum difference of adjacent clock periods. Period jitter, also usually measured as a peak-to-peak value, is the difference between the longest and shortest clock periods over a specified amount of time. TIE is typically measured as an RMS value, and represents the difference between the ideal clock period and the actual clock period.

Jitter may also be described as bounded or unbounded. Bounded jitter has definite maximum and minimum values, while unbounded jitter has no upper or lower limits.

Finally, many people refer to a "jitter budget." A jitter budget is the allowable amount of jitter in a system. This design spec is often defined at the beginning of the project.

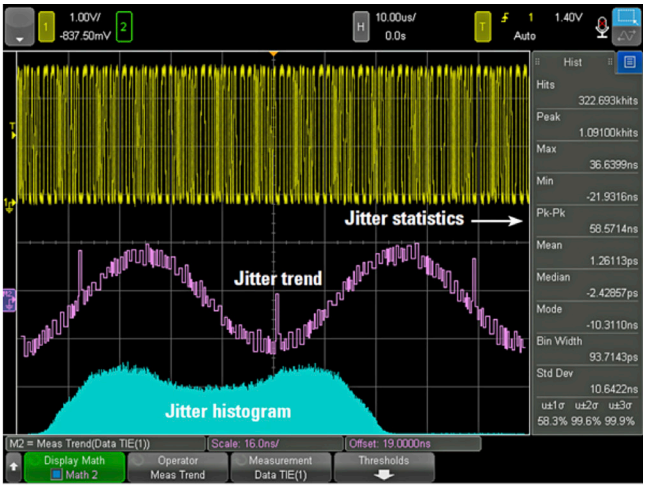
Components of Jitter

When talking about all of the jitter in a system, we use the term "total jitter" (TJ). Total jitter is made up of "random jitter" (RJ) and "deterministic jitter" (DJ). **Figure 2** shows the different components of jitter in the jitter family tree.

Random jitter (RJ)

When talking about random jitter (RJ), we like to use the phrase "jitter happens." Ultimately RJ is unavoidable, but it can be characterized. RJ has a Gaussian distribution (unbounded) and is caused by a combination of three things:

- First, thermal noise causes random jitter, and is described by Noise = kTB, where k is Boltzmann's constant, T is the temperature in Kelvin, and B is the system bandwidth.
- Second, shot noise (or Poisson) noise causes RJ. Shot noise is the inherent noise caused by the quantization of electrons and holes, and is influenced by bias currents.
- Finally, RJ is caused by "pink" noise, which is inversely proportional to the frequency (1/f). All systems will have some level of random jitter.



3. Since this PDF has two peaks (bimodal), we can tell that this system has both DJ and RJ.

Deterministic jitter (DJ)

Deterministic jitter is non-random, bounded, and caused by systematic occurrences in a design. Furthermore, deterministic jitter can be separated into a number of sub-components: periodic jitter (PJ), data-dependent jitter (DDJ), and bounded uncorrelated jitter (BUJ). Figure 3 shows an example of a system with deterministic jitter. Unlike random jitter, the PDF of deterministic jitter will usually have more than one peak.

Bounded uncorrelated jitter (BUJ)

BUJ can be the most frustrating type of jitter to track down because it's the least understood in the jitter family. It's also commonly referred to as crosstalk, but the term "crosstalk" is not exclusive to BUJ. BUJ is considered "uncorrelated" since it's statistically not possible to correlate it with other parts of the system.

BUJ is caused by system phenomena, though. An example of BUJ would be noise inside a chip caused by sources external to the chip, be it power-rail ripple or RF interference. A number of tools are available to help model crosstalk and identify aggressor signals, but causes of BUJ are generally outside of the designer's control.

Periodic jitter (PJ)

Periodic jitter can be either correlated or uncorrelated, but is always periodic. A very common source of periodic jitter is a switch-mode power supply coupling into a data or system clock. This would be classified as uncorrelated, because the power supply is not time-correlated with the victim signal. However, if one data signal was coupling in to a data signal based on the same clock, that would be considered correlated.

A common way to identify the source of PJ is to use a spectrum view. By plotting a trend of the TIE measurement (i.e., "how far off are my edges, and how does that change over time?") and then taking an fast Fourier transform (FFT) or frequency measurement of the trend plot, you can see the frequency of the periodic jitter. Figure 3 shows a jittery signal and a plot of the TIE over time. Knowing the frequency of your PJ will be immensely helpful when troubleshooting the aggressor signal.

Data-dependent jitter (DDJ)

Data-dependent jitter can be broken down into two parts—duty-cycle distortion (DCD) and inter-symbol interference (ISI). Both of these components of total jitter depend on the data signals.

- *Duty-cycle distortion (DCD)*: Duty-cycle distortion refers to a system's tendency to have one bit (0 or 1) with a characteristically longer cycle than the other. This is usually caused by one of two things. The first trigger often involves differing slew rates for rising and falling edges. A slow rising edge could cause a one-to-zero transition to happen slower. The other common cause of DCD is a non-50% threshold level. If the threshold level is too low or too high, one bit will appear to have a longer period than the other. This creates jitter because 0011 will have different edge timings than 0101. The 00 and 11 will not be altered by the differing slew rates, but a 0101 will be affected. In severe cases, DCD can cause a receiver to read an incorrect bit.
- *Inter-symbol interference (ISI)*: Inter-symbol interference, sometimes called "data-dependent jitter," is caused by long strings of 1s or 0s. By having a long string of 0s or 1s, a settling occurs in the transmitter or physical media. Then, transitioning to the opposite bit can introduce timing inconsistencies. Ultimately, ISI is either caused by bandwidth limitations in the transmitter, receiver, or physical media; or by improper impedance termination. A limited bandwidth will limit the edge speeds; a limited edge speed will cause amplitude variations. Improper termination (or discontinuities in the physical media) will cause signal reflections.

Graphs You Need to Know

Histograms

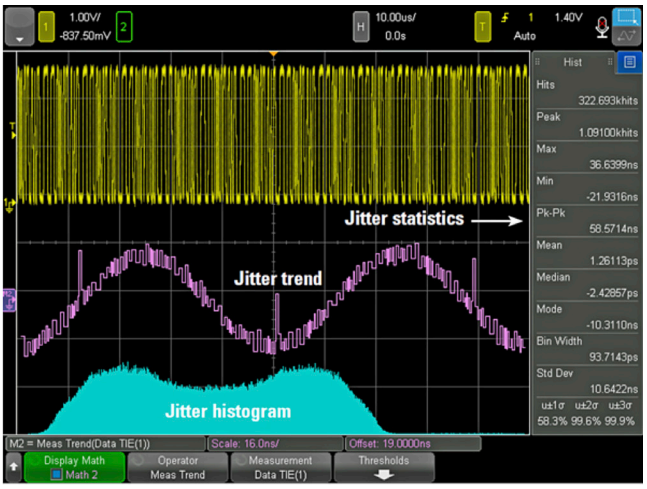
Histograms display the PDF of your time interval error. Knowing the difference between a histogram showing only random jitter (one peak) or deterministic jitter (multiple peaks) will give you a quick start when tracking down jitter in your system.

Trend measurements

Looking at Figure 3, we see a TIE trend measurement. Because it has a frequency of 20 kHz, there's likely something in the system running at 20 kHz that's causing the periodic jitter. In addition, a spike in the trend plot indicates a timing error. This timing error is potentially a result of ISI or DCD.

Eye diagrams

Learning how to read eye diagrams will provide insight into the jitter components at hand. Are there multiple distinct edges, abnormal shapes, varied high/low levels, or non-monotonic edges in your eye diagram? This will help you further analyze what jitter components are the



4. An eye diagram and its corresponding histogram show bimodal DJ and an intermittent, delayed rising edge.

5. Jitter components are automatically separated and compared on a Keysight Infiniium oscilloscope using EZJit Complete.

most egregious in your design without having to resort to complex measurements. **Figure 4** shows an example of an eye diagram on the InfiniiVision 6000 X-Series oscilloscope.

Bathtub plots

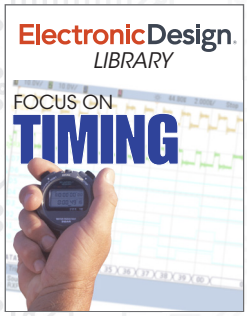
Bathtub plots help you determine your bit error rate (BER) and what portions of the jitter are random or deterministic. Ultimately, when evaluating your design's jitter budget, the bathtub plot will be helpful for pass/fail analysis and applying confidence intervals to your eye diagrams. An example of a bathtub plot is shown in the middle of **Figure 5.**

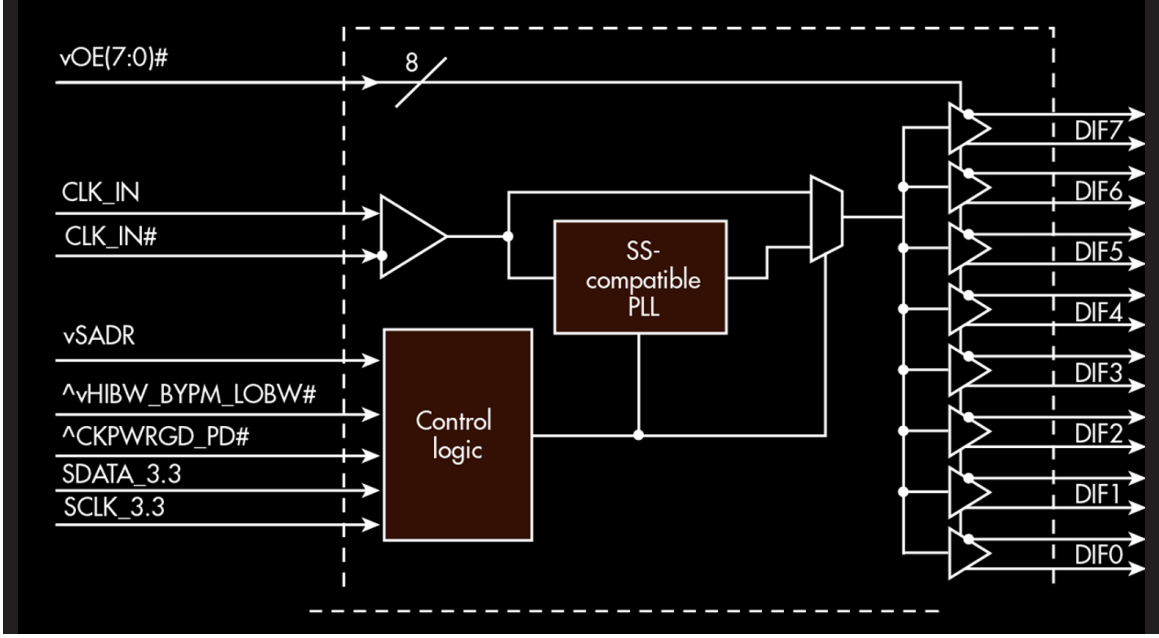
Wrapping Up

Learning the different forms of jitter is just the first step to designing and debugging with a jitter budget in mind. Knowing the different jitter components will become more important as signal speeds get faster and faster. Knowing how to use different graphs and plots to both identify jitter sources and quickly glean necessary information will be a key skill for engineers working with digital systems.

to view this article online, ☞ *click here*

☞ BACK TO TABLE OF CONTENTS





CHAPTER 4:

PCI Express Clock Generators, Buffers Prepare for Next Generation

JOHN BLYLER, Technical Contributor

Sponsored by Renesas: The latest Gen4 clock generator and buffers meet performance and jitter requirements for Gen5 PCI Express systems.

Bracing for the coming 5th generation (Gen5) of PCI Express (PCIe), high-speed serial bus designers are revisiting clocking generators and distribution buffers. This article reviews these fundamental elements in light of the transition from Gen4 to Gen5 PCIe architectures.

PCIe has been around since 2004. It's a high-speed serial computer expansion bus specification that replaces older PCI and PCI-X standards. PCIe currently supports the Generation 4 specification. In June 2017, the PCI-SIG preliminarily announced the PCI Express 5.0 specification, which is expected to be standardized in 2019.

Regardless of the generation of the PCIe bus, the clock remains the most basic hardware component. The clock generator produces the timing signal that controls the data rate of the bus. Traditionally, a clock buffer is used to control the rise and fall time of the clock edges. Maintaining the cleanest clock rise and fall time is made difficult by the presence of jitter, which causes the clock edge to degrade from it ideal location and shape. Clock jitter is typically induced by the generator circuitry, thermal noise, power-supply variations, and interference coupled from nearby circuits.

As a high-speed serial interface, the PCIe data channel runs at speeds up to 8 Gb/s, increasing to 16 Gb/s with PCIe Gen4 devices. These devices have been used in data centers and networks including servers, SSDs, NICs, FPGA- and GPU-based accelerator cards, and Ethernet and optical modules.

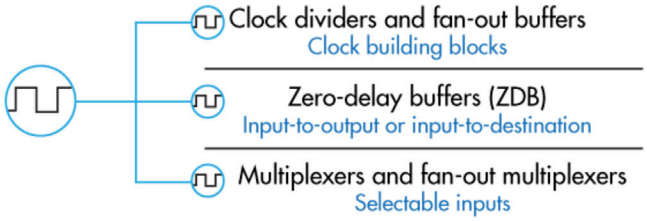
As the data speeds increase with each generation of PCIe bus, so does the potential for jitter. Yet the actual jitter specification for the clock is constantly lowered to meet speed and timing issues. For reference, the PCIe Gen4 jitter limit is 500 femtoseconds (fs) rms,

whereas the Gen3 limit was 1 picosecond (ps) rms. The proposed jitter budget for the reference clock in a PCIe Gen5 system is 250 fs max.

Thus, the clock generator is the most crucial element in maintaining the proper PCIe system performance and reliability.

Clock Buffers

Only a certain number of clocks are allowed in a single device. Such restrictions are the results of a limitation on the number of pin-counts per devices. Also, board wiring routing constraints may limit the number of clocks that are located in a specific area. For these reasons, PCIe clock distribution devices are used in all but the smallest of PCIe bus systems. The distribution devices act as buffers (**Fig. 1**).

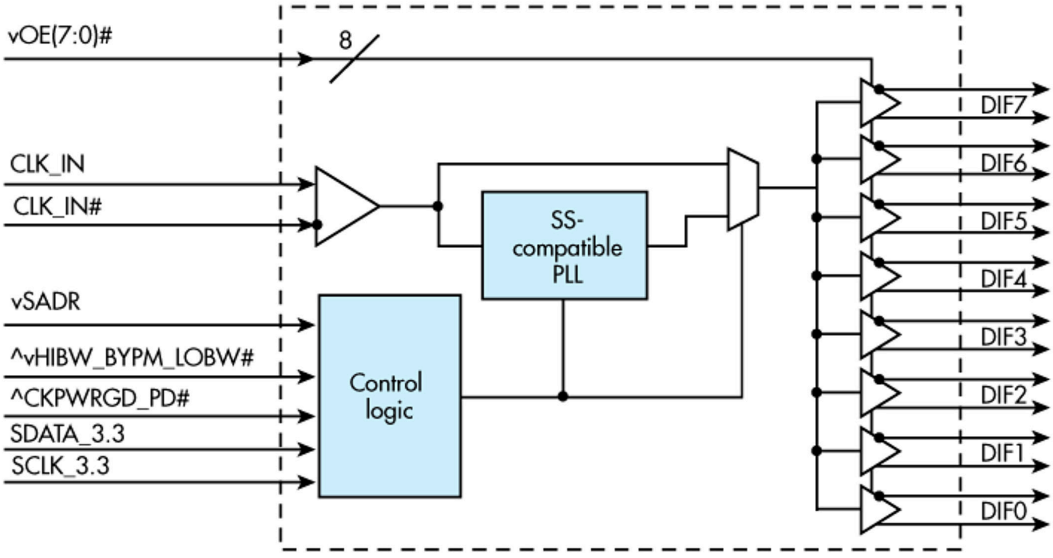


1. Clock-distribution devices are used to condition, manipulate, and distribute clock signals within a system, with or without the use of a phased-locked loop (PLL). (Courtesy of IDT)

Fan-out buffers allow a single clock to be routed to a section needing multiple clocking inputs, where the buffer then fans (or distributes) out the clock locally. Sometimes, the designer only has a single PCIe clock coming from a connector and needs multiple copies. Again, fan-out buffers address this shortcoming.

Clock trees are built using fan-out buffers to provide signal buffering and low-jitter copies of the original clock signal. The clock fan-out from a single input reduces loading on the preceding driver and provides an efficient clock-distribution network (**Fig. 2**).

Finally, zero-delay buffers (ZDBs) provide a synchronous copy (no propagation delay)

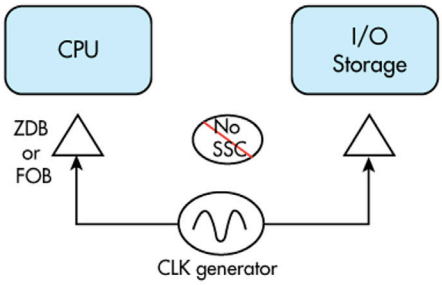


2. This PCIe clock tree with fan-out buffers is from a single clock input source. (Courtesy of IDT)

of the input clock at the outputs. Creating a synchronized copy is accomplished by using phase-locked-loop (PLL)-based devices to regenerate the input clock signal, from which a new fan-out is formed that can drive multiple loads. ZDBs are often used by applications requiring synchronized clocking for FPGAs, CPUs, logic, and synchronous memory.

Clocking Example

Imagine that you have a fairly straightforward design—known as "common clocking"—where all of the clocks come from one source such as a PLL. The clock signal may need to be spread to other parts of the board or not, but it's unlikely that all of the signals will come from one device. This means that fan-out buffers will be needed to distribute extra copies of the PCI Express clock.



3. This PCIe common clock architecture uses no spread spectrum copies (SSC), but rather zero-delay buffers (ZDBs) or fan-out buffers (FOBs). (Courtesy of IDT)

If the design isn't using spread-spectrum clocking (SSC), then the fan-out to the buffers could be a ZDB with a PLL inside or simply a fan-out buffer (**Fig. 3**). Either implementation will satisfy the common clock requirement of the PCIe standard. SSC is a technique where the clock frequency is modulated slightly to lower the peak energy generated by a clock to help reduce total system electromagnetic interference (EMI).

Renesas family of PhiClock programmable clock generators represents a good example of timing devices. These generators can be configured via select hardware pins or software. Part of the family provides several non-spread-spectrum copies of a single output frequency, copies of the crystal reference input, and fractional frequency copies of the original clock signal.

In addition, Renesas PCIe clock generators support 1.5-, 1.8-, or 3.3-V versions, allowing the designer to power the clocks from the same power supply as an FPGA or system on chip (SoC).

On the buffer side, Renesas 9Z enhanced PCIe buffer family offers buffers with and without a PLL inside. The PLL creates a ZDB that eliminates propagation delay through the device, which reduces transport delay. If the PLL is bypassed, the unit converts into a pure fan-out buffer (no PLL). If the design uses spread spectrum, then a pure fan-out buffer is needed most of the time.

Taken together, both the PhiClock clock generator and 9Z buffers offer performance upgrades by providing a 50% margin to the PCIe Gen4 clock jitter limit of 0.5 ps rms. Initial Renesas testing shows both families satisfy upcoming Gen5 PCIe requirements. For more information, please visit www.Renesas.com/pcietiming.

Summary

Clock generators and buffers will continue to play a critical role in PCIe specifications and performance. Renesas has Gen4 PCIe clock-tree devices that meet the soon-to-be-released Gen5 standard.

to view this article online, ☞ *click here*

☞ BACK TO TABLE OF CONTENTS