

Modeling Jitter in PLL-based Frequency Synthesizers

Ken Kundert

Designer's Guide Consulting, Inc.

Version 4h, March 2012

A methodology is presented for modeling the jitter in a Phase-Locked Loop (PLL) that is both accurate and efficient. The methodology begins by characterizing the noise behavior of the blocks that make up the PLL using transistor-level simulation. For each block, the jitter is extracted and provided as a parameter to behavioral models for inclusion in a high-level simulation of the entire PLL. This approach is efficient enough to be applied to PLLs acting as frequency synthesizers with large divide ratios.

This paper was written in August, 2002 and was last updated on March 28, 2012. You can find the most recent version at www.designers-guide.org. Contact the author via e-mail at ken@designers-guide.com.

Permission to make copies, either paper or electronic, of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage and that the copies are complete and unmodified. To distribute otherwise, to publish, to post on servers, or to distribute to lists, requires prior written permission.

Designer's Guide is a registered trademark of Kenneth S. Kundert. All rights reserved.

1 Introduction

Phase-locked loops (PLLs) are used in wireless receivers to implement a variety of functions, such as frequency synthesis, clock recovery, and demodulation. One of the major concerns in the design of PLLs is noise or jitter performance. Jitter from the PLL directly acts to degrade the noise floor and selectivity of a transceiver.

Demir proposed an approach for modeling PLLs whereby a PLL is described using high level behavioral models [1,2]. The models are written such that they include jitter in an efficient way. He also devised a powerful new simulation algorithm that is capable of characterizing the circuit-level noise behavior of blocks that make up a PLL that is based on solving a set of nonlinear stochastic differential equations [3,5]. Finally, he gave formulas that can be used to convert the results of the noise simulations on the individual blocks into values for the jitter parameters for the corresponding behavioral models [6]. This approach provides accurate and efficient prediction of PLL jitter behavior once the noise behavior of the blocks has been characterized. However, it requires the use of an experimental simulator that is not readily available.

This paper presents the relevant ideas of Demir, but while he focussed on presenting the conceptual aspects of modeling and simulating jitter in PLLs, this paper concentrates more on the practical aspects. It presents all the information a designer would need to predict the noise and jitter of a PLL synthesizer. This paper is an enhanced version of two previous papers [14,15]. The jitter extraction methodology is based on the commercially available Spectre®RF¹ simulator [24,25] and presents behavioral models for Verilog-A², a standard, non-proprietary analog behavioral modeling language [12,27]. Both SpectreRF and Verilog-A are options to the Spectre circuit simulator [11], available from Cadence Design Systems.³

1.1 Predicting Noise in PLLs

There are two different approaches to modeling noise in PLLs. One approach is to formulate the models in terms of the phase of the signals, producing what are referred to as phase-domain models. In the simplest case, these models are linear and analyzed easily in the frequency domain, making it simple to use the model to predict phase noise, even in the presence of flicker noise or other noise sources that are difficult to model in the time domain. Phase domain models are described more fully in the companion to this manuscript [16].

The other approach formulates the models in terms of voltage, and so are referred to as voltage-domain models. The advantage of voltage-domain models is that they can be refined to implementation. In other words, as the design process transitions to being

-
1. Spectre is a registered trademark of Cadence Design Systems.
 2. Verilog is a registered trademark of Cadence Design Systems licensed to Accellera.
 3. SpectreRF is currently the only commercial simulator that is well suited for characterizing the jitter of the blocks that make up a PLL. SPICE and its descendants are not suitable because they only perform noise analysis about a DC operating point and so do not take into account the time-varying nature of these circuits. Harmonic balance simulators do perform noise analysis about a periodic operating point, which is a critical prerequisite, but they have convergence, accuracy, and performance problems with blocks such as the PFD/CP, FD and VCO that are strongly nonlinear.

more of a verification process, the abstract behavioral models initially used can be replaced with detailed gate- or transistor-level models in order to verify the PLL as implemented.

Voltage-domain models are strongly nonlinear and never have quiescent operating points, making them incompatible with a SPICE-like noise analysis. Often they do have a periodic operating point and so can be analyzed with small-signal RF noise analysis (SpectreRF), but it is also common for that not to be the case. For example, a fractional- N synthesizer does not have a periodic operating point. Occasionally, the circuit is sensitive enough that the noise affects the large-signal behavior of the PLL, such as with bang-bang clock-and-data recovery PLLs, which invalidates any use of small-signal noise analysis.

Modeling large-signal noise in a voltage-domain model as a voltage or a current is problematic. Such signals are very small and continuously, and generally rapidly varying. Extremely tight tolerances and small time steps are required to accurately resolve such signals with simulation. To overcome these problems, the noise is instead represented using the effect it has on the timing of the transitions within the PLL. In other words, the noise is added to the circuit in the form of jitter. In this case there is no need for either small time steps or tight tolerances.

The process of predicting the jitter of a PLL described in this paper involves:

1. Using SpectreRF to predict the noise of the individual blocks that make up the PLL.
2. Converting the noise of the block to jitter.
3. Building high-level behavioral models of each of the blocks that include jitter.
4. Assembling the blocks into a model of the PLL.
5. Simulating the PLL, including the effect of jitter, to find the noise of the overall system.

The simple linear phase-domain model described in the companion paper [16], and the nonlinear voltage-domain model described here represent the two ends of a continuum of models. Generally, the phase-domain models are considerably more efficient, but the voltage-domain models do a much better job of capturing the details of the behavior of the loop, details such as the signal capture and escape processes. The phase-domain models can be made more general by making them nonlinear and by analyzing them in the time domain. It is common to use such models with fractional- N synthesizers. Conversely, simplifications can be made to the voltage-domain models to make them more efficient. It is even possible to use both voltage- and phase-domain models for different parts of the same loop. One might do so to retain as much efficiency as possible while allowing part of the design to be refined to implementation level. In general it is best to understand both approaches well, and use ideas from both to construct the most appropriate approach for your particular situation.

2 Frequency Synthesis

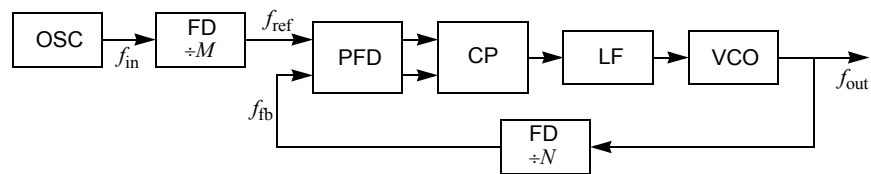
The block diagram of a PLL operating as a frequency synthesizer is shown in Figure 1 [7].⁴ It consists of a reference oscillator (OSC), a phase/frequency detector (PFD), a charge pump (CP), a loop filter (LF), a voltage-controlled oscillator (VCO), and two frequency dividers (FDs). The PLL is a feedback loop that, when in lock, forces

f_{fb} to be equal to f_{ref} . Given a reference frequency f_{in} , the frequency at the output of the PLL is

$$f_{out} = \frac{N}{M} f_{in} \tag{1}$$

By choosing the frequency divide ratios and the input frequency appropriately, the synthesizer generates an output signal at the desired frequency that inherits much of the stability of the reference oscillator. In RF transceivers, this architecture is used to generate the local oscillator (LO) at a programmable frequency, which tunes the transceiver to the desired channel by adjusting the value of N .

FIGURE 1 The block diagram of a frequency synthesizer.



3 Jitter

The signals at the input and output of a PLL are often binary signals, as are many of the signals within the PLL. The noise on binary signals is commonly characterized in terms of jitter.

Jitter is an undesired perturbation or uncertainty in the timing of events. Generally, the events of interest are the transitions in a signal. One models jitter in a signal by starting with a noise-free signal v and displacing time with a stochastic process j . The noisy signal becomes

$$v_n(t) = v(t + j(t)) \tag{2}$$

with j assumed to be a zero-mean process and v assumed to be a T -periodic function. j has units of seconds and can be interpreted as a noise in time. Alternatively, it can be reformulated as a noise in phase, or phase noise, using

$$\phi(t) = 2\pi f_0 j(t), \tag{3}$$

where $f_0 = 1/T$ and

$$v_n(t) = v\left(t + \frac{\phi(t)}{2\pi f_0}\right). \tag{4}$$

4. Frequency synthesis is used as an example, but the concepts presented are easily applied to other applications, such as clock recovery and FM demodulation. In addition, they are also applicable to other types of PLL-based synthesis, such as fractional- N synthesis.

3.1 Jitter Metrics

Define $\{t_i\}$ as the sequence of times for positive-going threshold crossings, henceforth referred to as *transitions*, that occur in v_n . Various jitter metrics characterize the statistics of this sequence.⁵

The simplest metric is the *edge-to-edge jitter*, J_{ee} , which is the variation in the delay between a triggering event and a response event. When measuring edge-to-edge jitter, a clean jitter-free input is assumed, and so the edge-to-edge jitter J_{ee} is

$$J_{ee}(i) = \sqrt{\text{var}(t_i)}. \quad (5)$$

Edge-to-edge jitter assumes an input signal, and so is only defined for driven systems. It is an input-referred jitter metric, meaning that the jitter measurement is referenced to a point on a noise-free input signal, so the reference point is fixed. No such signal exists in autonomous systems. The remaining jitter metrics are suitable for both driven and autonomous systems. They gain this generality by being self-referred, meaning that the reference point is on the noisy signal for which the jitter is being measured. These metrics tend to be a bit more complicated because the reference point is noisy, which acts to increase the measured jitter.

Edge-to-edge jitter is also a scalar jitter metric, and it does not convey any information about the correlation of the jitter between transitions. The next metric characterizes the correlations between transitions as a function of how far the transitions are separated in time.

Define $J_k(i)$ to be the standard deviation of $t_{i+k} - t_i$,

$$J_k(i) = \sqrt{\text{var}(t_{i+k} - t_i)}. \quad (6)$$

$J_k(i)$ is referred to as *k-cycle jitter* or *long-term jitter*⁶. It is a measure of the uncertainty in the length of k cycles and has units of time. J_1 , the standard deviation of the length of a single period, is often referred to as the *period jitter*, and it denoted J , where $J = J_1$.

Another important jitter metric is *cycle-to-cycle jitter*. Define $T_i = t_{i+1} - t_i$ to be the period of cycle i . Then the cycle-to-cycle jitter J_{cc} is

$$J_{cc}(i) = \sqrt{\text{var}(T_{i+1} - T_i)}. \quad (7)$$

Cycle-to-cycle jitter is a metric designed to identify large adjacent cycle displacements. It is like edge-to-edge jitter in that it is a scalar jitter metric that does not contain information about the correlation in the jitter between distant transitions. However, it differs in that it is a measure of short-term jitter that is relatively insensitive to long-term jitter [10]. As such, cycle-to-cycle jitter is the only jitter metric that is suitable for use when flicker noise is present. All other metrics are unbounded in the presence of flicker noise.

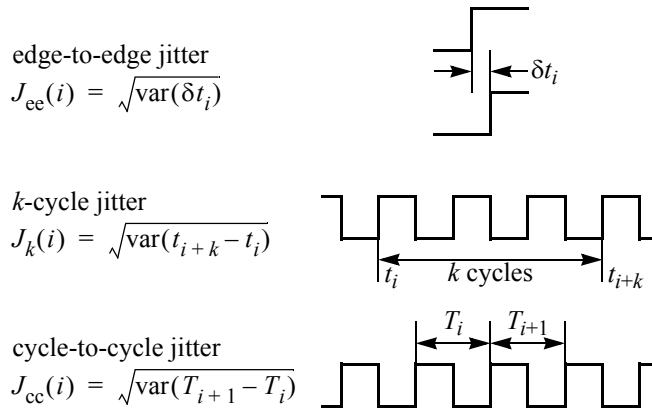
If $j(t)$ is either stationary or T -cyclostationary, then $\{t_i\}$ is stationary, meaning that these metrics do not vary with i , and so $J_{ee}(i)$, $J_k(i)$, and $J_{cc}(i)$ can be shortened to J_{ee} , J_k , and J_{cc} .

5. There is some variability in the jitter metrics that are used and their definitions. The work by Lee documents some other ways of characterizing jitter [17].

6. Some people distinguish between k -cycle jitter and long-term jitter by defining the long-term jitter J_∞ as being the k -cycle jitter J_k as $k \rightarrow \infty$.

These jitter metrics are illustrated in Figure 2.

FIGURE 2 The various jitter metrics.



3.1.1 RMS versus Peak-to-Peak Jitter

All the jitter metrics given so far have been RMS metrics. If you assume that the noise sources have Gaussian distributions, then strictly speaking the metrics do not have peak-to-peak values because the noise is unbounded. However, one can define the peak-to-peak jitter as the magnitude that the jitter exceeds only a for specified fraction of the time, known as the *error rate* [18]. Once the acceptable error rate is specified, then it can converted to α using Table 1, which is the ratio between the peak-to-peak deviation and the standard deviation. Then the RMS jitter can be converted to peak-to-peak jitter using

$$J_{PP} = \alpha J_{RMS} \tag{8}$$

TABLE 1 The ratio of the peak-to-peak deviation of a Gaussian process to its standard deviation where the peak-to-peak deviation is defined as the magnitude that is not exceeded more often than the given error rate.

Error Rate	α
10^{-3}	6.180
10^{-4}	7.438
10^{-5}	8.530
10^{-6}	9.507
10^{-7}	10.399
10^{-8}	11.224
10^{-9}	11.996
10^{-10}	12.723
10^{-11}	13.412
10^{-12}	14.069
10^{-13}	14.698
10^{-14}	15.301

TABLE 1 *The ratio of the peak-to-peak deviation of a Gaussian process to its standard deviation where the peak-to-peak deviation is defined as the magnitude that is not exceeded more often than the given error rate.*

Error Rate	α
10^{-15}	15.883
10^{-16}	16.444

3.2 Types of Jitter

The type of jitter produced in PLLs can be classified as being from one of two canonical forms. Blocks such as the PFD, CP, and FD are driven, meaning that a transition at their output is a direct result of a transition at their input. The jitter exhibited by these blocks is referred to as *synchronous jitter*, it is a variation in the delay between when the input is received and the output is produced. Blocks such as the OSC and VCO are autonomous. They generate output transitions not as a result of transitions at their inputs, but rather as a result of the previous output transition. The jitter produced by these blocks is referred to as *accumulating jitter*, it is a variation in the delay between an output transition and the subsequent output transition. Table 2 previews the basic characteristics of these two types of jitter. The formulas for jitter given in this table are derived in the next two sections.

TABLE 2 *The two canonical forms of jitter.*

Jitter Type	Circuit Type	Jitter
synchronous	driven (PFD/CP, FD)	$J_{ee} = \frac{\sqrt{\text{var}(n_v(t_c))}}{dv(t_c)/dt}$
accumulating	autonomous (OSC, VCO)	$J = \sqrt{cT}$

4 Synchronous Jitter

Synchronous jitter is exhibited by driven systems. In the PLL, the PFD/CP and FDs exhibit synchronous jitter. In these components, an output event occurs as a direct result of, and some time after, an input event. It is an undesired fluctuation in the delay between the input and the output events. If the input is a periodic sequence of transitions, then the frequency of the output signal is exactly that of the input, but the phase of the output signal fluctuates with respect to that of the input. The jitter appears as a modulation of the phase of the output, which is why it is sometimes referred to as phase modulated or PM jitter.

Let η be a stationary or T -cyclostationary process, then

$$j_{\text{sync}}(t) = \eta(t) \quad (9)$$

$$v_n(t) = v(t + j_{\text{sync}}(t)) \quad (10)$$

exhibits synchronous jitter. If η is further restricted to be a white Gaussian stationary or T -cyclostationary process, then $v_n(t)$ exhibits *simple synchronous jitter*. The essential characteristic of simple synchronous jitter is that the jitter in each event is independent or uncorrelated from the others, and (3) shows that it corresponds to white phase noise. Driven circuits exhibit simple synchronous jitter if they are broadband and if the noise sources are white, Gaussian and small. The sources are considered small if the circuit responds linearly to the noise, even though at the same time the circuit may be responding nonlinearly to the periodic drive signal.

For systems that exhibit simple synchronous jitter, from (5),

$$J_{ee}(i) = \sqrt{\text{var}(j_{\text{sync}}(t_i))}. \quad (11)$$

Similarly, from (6), k cycle jitter is

$$J_k(i) = \sqrt{\text{var}(t_{i+k} - t_i)}, \quad (12)$$

$$J_k(i) = \sqrt{\text{var}([(i+k)T + j_{\text{sync}}(t_{i+k})] - [iT + j_{\text{sync}}(t_i)])}, \quad (13)$$

$$J_k(i) = \sqrt{\text{var}(j_{\text{sync}}(t_{i+k})) + \text{var}(j_{\text{sync}}(t_i))}. \quad (14)$$

Since $j_{\text{sync}}(t)$ is T -cyclostationary $j_{\text{sync}}(t_i)$ is independent of i , and so is J_{ee} and J_k .

$$J_k(i) = \sqrt{2\text{var}(j_{\text{sync}})}, \text{ and} \quad (15)$$

$$J_k(i) = \sqrt{2}J_{ee}. \quad (16)$$

The factor of $\sqrt{2}$ in (16) stems from the length of an interval including the independent variation from two transitions. From (16), J_k is independent of k , and so

$$J_k = J \text{ for } k = 1, 2, \dots. \quad (17)$$

The approach is similar for cycle-to-cycle jitter. From (7),

$$J_{cc}(i) = \sqrt{\text{var}(T_{i+1} - T_i)} \quad (18)$$

$$J_{cc}(i) = \sqrt{\text{var}([t_{i+1} - t_i] - [t_i - t_{i-1}])} \quad (19)$$

$$J_{cc}(i) = \sqrt{\text{var}(t_{i+1} - 2t_i + t_{i-1})} \quad (20)$$

$$J_{cc}(i) = \sqrt{\text{var}([(i+k)T + j_{\text{sync}}(t_{i+k})] - 2[iT + j_{\text{sync}}(t_i)] + [(i-k)T + j_{\text{sync}}(t_{i-k})])} \quad (21)$$

$$J_{cc}(i) = \sqrt{\text{var}(j_{\text{sync}}(t_{i+k})) + \text{var}(2j_{\text{sync}}(t_i)) + \text{var}(j_{\text{sync}}(t_{i-k}))} \quad (22)$$

$$J_{cc}(i) = \sqrt{\text{var}(j_{\text{sync}}(t_{i+k})) + 4\text{var}(j_{\text{sync}}(t_i)) + \text{var}(j_{\text{sync}}(t_{i-k}))} \quad (23)$$

$$J_{cc}(i) = \sqrt{6\text{var}(j_{\text{sync}})} \quad (24)$$

$$J_{cc}(i) = \sqrt{6}J_{ee} \quad (25)$$

Generally, the jitter produced by the PFD/CP and FDs is well approximated by simple synchronous jitter if one can neglect flicker noise.

4.1 Extracting Synchronous Jitter

The jitter in driven blocks, such as the PFD/CP or FDs, occurs because of an interaction between noise present in the blocks and the thresholds that are inherent to logic circuits.

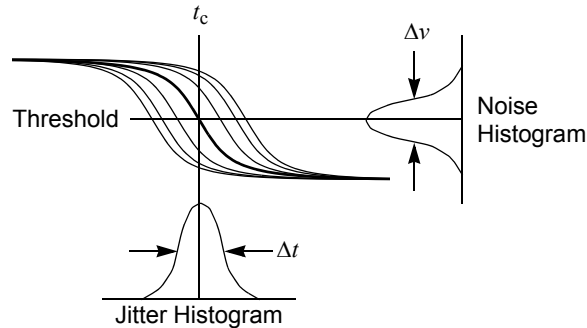
In systems where signals are continuous valued, an event is usually defined as a signal crossing a threshold in a particular direction. The threshold crossings of a noiseless periodic signal, $v(t)$, are precisely evenly spaced. However, when noise is added to the signal, $v_n(t) = v(t) + n_v(t)$, each threshold crossing is displaced slightly. Thus, a threshold converts additive noise to synchronous jitter.

The amount of displacement in time is determined by the amplitude of the noise signal, $n_v(t)$ and the slew rate of the periodic signal, $dv(t_c)/dt$, as the threshold is crossed, as shown in Figure 3 [28]. If the noise n_v is stationary, then

$$\text{var}(j_{\text{sync}}(t_c)) \cong \frac{\text{var}(n_v)}{[dv(t_c)/dt]^2} \quad (26)$$

where t_c is the time of a threshold crossing in v (assuming the noise is small).

FIGURE 3 How a threshold converts noise into jitter.



Generally n_v is not stationary, but cyclostationary. It is only important to know when the noisy periodic signal $v_n(t)$ crosses the threshold, so the statistics of n_v are only significant at the time when $v_n(t)$ crosses the threshold,

$$\text{var}(j_{\text{sync}}(t_c)) = \frac{\text{var}(n_v(t_c))}{[dv(t_c)/dt]^2} \quad (27)$$

The jitter is computed from (11) using (26) or (27),

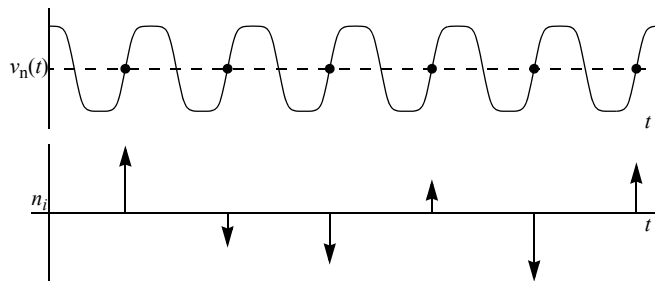
$$J_{\text{ee}} = \frac{\sqrt{\text{var}(n_v(t_c))}}{dv(t_c)/dt} \quad (28)$$

To compute $\text{var}(n_v(t_c))$, one starts by driving the circuit with a representative periodic signal, and then sampling $v(t)$ at intervals of T to form the ergodic sequence $\{v(t_i)\}$

where $t_i = t_c$ for some i . Then the variance is computed by computing the power spectral density for the sequence by integrating from $f = -f_0/2$ to $f_0/2$. Recall that the noise is periodic in f with period $f_0 = 1/T$ because n is a discrete-time sequence with rate T .

In practice, this is done by using the strobed noise capability of SpectreRF⁷ to compute the power spectral density of the sequence. When the strobed noise feature is active, the noise produced by the circuit is periodically sampled to create a discrete-time random sequence, as shown in Figure 4. SpectreRF then computes the power-spectral density of the sequence. The sample time should be adjusted to coincide with the desired threshold crossings. Since the T -periodic cyclostationary noise process is sampled every T seconds, the resulting noise process is stationary. Furthermore, the noise present at times other than at the sample points is completely ignored.

FIGURE 4 Strobed noise. The lower waveform is a highly magnified view of the noise present at the strobe points in v_n , which are chosen to coincide with the threshold crossings in v .



4.1.1 Extracting the Jitter of Dividers

To extract the jitter of a divider, drive the divider with a representative periodic input signal and perform a PSS analysis to determine the threshold crossing times and the slew rate (dv/dt) at these times. Then use SpectreRF’s strobed PNoise analysis to compute $S_n(f)$. The sample point should be set to coincide with the point where the output signal crosses the threshold of the subsequent stage (the phase detector) in the appropriate direction. When running PNoise analysis, assure that the *maxsideband* parameter is set sufficiently large to capture all significant noise folding. A large value will slow the simulation. To reduce the number of sidebands needed, use T as small as possible. SpectreRF computes the power spectral density S_{n_v} , which is integrated to compute the total noise at the sample points,

$$\text{var}(n_v(t_c)) = \int_0^{f_0/2} S_{n_v}(f, t_c) df . \tag{29}$$

Then J_{ee} is computed from (28).

With ripple counters, one usually only characterizes one stage at a time. The total jitter due to noise in the ripple counter is then computed by assuming that the jitter in each stage is independent (again, this is true for device noise, but not for noise coupling into the divider from external sources) and taking the square-root of the sum of the square of the jitter on each stage.

7. The strobed-noise feature of SpectreRF is also referred to as its time-domain noise feature.

Unlike in ripple counters, jitter does not accumulate with synchronous counters. Jitter in a synchronous counter is independent of the number of stages and consists only of the jitter of its clock along with the jitter of the last stage.

4.1.2 Extracting the Jitter of the Phase Detector

The PFD/CP is not followed by a threshold. Rather, it feeds into the LF, which is sensitive to the noise emitted by the CP at all times, not just during transitions. This argues that the noise of the PFD/CP be modeled as a continuous noise current. However, as mentioned earlier, doing so is problematic for simulators and would require very tight tolerances and small time steps. So instead, the noise of the PFD/CP is referred back to its inputs. The inputs of the PFD/CP are edge triggered, so the noise can be referred back as jitter.

To extract the input-referred jitter of a PFD/CP, drive both inputs with periodic signals with offset phase so that the PFD/CP produces a representative output. Use SpectreRF's PNoise analysis to compute the output noise over the total bandwidth of the PFD/CP (in this case, use the conventional noise analysis rather than the strobed noise analysis). Choose the frequency range of the analysis so that the total noise at frequencies outside the range is negligible. Thus, the noise should be at least 40 dB down and dropping at the highest frequency simulated. Integrate the noise over frequency and apply Wiener-Khinchin Theorem [21] to determine

$$\text{var}(n) = \int_0^{\infty} S_n(f) df, \quad (30)$$

the total output noise current squared [8]. Then either calculate or measure the effective gain of the PFD/CP, K_{det} , in units of amperes per cycle. Scale the gain so that it has the units of amperes per second by dividing K_{det} by the period T seconds per cycle. Then divide the total output noise current by this gain and account for there being two transitions per cycle to distribute the noise over to determine the input-referred jitter for the PFD/CP,

$$J_{\text{eePFD/CP}} = \frac{T}{K_{\text{det}}} \sqrt{\frac{\text{var}(n)}{2}}. \quad (31)$$

As before, when running PNoise analysis, assure that the *maxsideband* parameter is set sufficiently large to capture all significant noise folding. A large value will slow the simulation. To reduce the number of sidebands needed, use T as small as possible.

5 Accumulating Jitter

Accumulating jitter is exhibited by autonomous systems, such as oscillators, that generate a stream of spontaneous output transitions. In the PLL, the OSC and VCO exhibit accumulating jitter. Accumulating jitter is characterized by an undesired variation in the time since the previous output transition, thus the uncertainty of when a transition occurs accumulates with every transition. Compared with a jitter free signal, the frequency of a signal exhibiting accumulating jitter fluctuates randomly, and the phase drifts without bound. Thus, the jitter appears as a modulation of the frequency of the output, which is why it is sometimes referred to as frequency modulated or FM jitter.

Again assume that η be a stationary or T -cyclostationary process, then

$$j_{\text{acc}}(t) = \int_0^t \eta(\tau) d\tau \quad (32)$$

$$v_n(t) = v(t + j_{\text{acc}}(t)) \quad (33)$$

exhibits accumulating jitter. While η is cyclostationary and so has bounded variance, (32) shows that the variance of j_{acc} , and hence the phase difference between $v(t)$ and $v_n(t)$, is unbounded.

If η is further restricted to be a white Gaussian stationary or T -cyclostationary random process, then v_n exhibits *simple accumulating jitter*. In this case, the process $\{j_{\text{acc}}(iT)\}$ that results from sampling j_{acc} every T seconds is a discrete Wiener process and the phase difference between $v(iT)$ and $v_n(iT)$ is a random walk [8]. As shown next, simple accumulating jitter corresponds to oscillator phase noise that results from white noise sources.

The essential characteristic of simple accumulating jitter is that the incremental jitter that accumulates over each cycle is independent or uncorrelated. Autonomous circuits exhibit simple accumulating jitter if they are broadband and if the noise sources are white, Gaussian and small. The sources are considered small if the circuit responds linearly to the noise, though at the same time the circuit may be responding nonlinearly to the oscillation signal. An autonomous circuit is considered broadband if there are no secondary resonant responses close in frequency to the primary resonance.⁸

For systems that exhibit simple accumulating jitter, each transition is relative to the previous transition, and the variation in the length of each period is independent, so the variance in the time of each transition accumulates,

$$J_k = \sqrt{k}J \text{ for } k = 0, 1, 2, \dots, \quad (34)$$

where

$$J = \sqrt{\text{var}(j_{\text{acc}}(t_i + T) - j_{\text{acc}}(t_i))}. \quad (35)$$

Similarly,

$$J_{\text{cc}} = \sqrt{2}J. \quad (36)$$

Generally, the jitter produced by the OSC and VCO are well approximated by simple accumulating jitter if one can neglect flicker noise.

8. Oscillators are strongly nonlinear circuits undergoing large periodic variations, and so signals within the oscillator freely mix up and down in frequency by integer multiples of the oscillation frequency. For this reason, any low frequency time constants or resonances in supply or bias lines would effectively act like close-in secondary resonances. In fact, this is the most likely cause of such phenomenon.

5.1 Extracting Accumulating Jitter

The jitter in autonomous blocks, such as the OSC or VCO, is almost completely due to oscillator phase noise. Oscillator phase noise is a variation in the phase of the oscillator as it proceeds along its limit cycle.

In order to determine the period jitter J of $v_n(t)$ for a noisy oscillator, assume that it exhibits simple accumulating jitter so that η in (32) is a white Gaussian T -cyclostationary noise process (this excludes flicker noise) with a single-sided PSD of

$$S_\eta(f) = 2c, \quad (37)$$

and an autocorrelation function of

$$R_\eta(t_1, t_2) = c\delta(t_1 - t_2), \quad (38)$$

where δ is a Dirac delta function. Then

$$j_{\text{acc}}(t) = \int_0^t \eta_T(\tau) d\tau \quad (39)$$

is a Wiener process [8], which has an autocorrelation function of

$$R_{j_{\text{acc}}}(t_1, t_2) = c \min(t_1, t_2). \quad (40)$$

The period jitter is the standard deviation of the variation in one period, and so

$$J^2 = \text{var}(j_{\text{acc}}(t+T) - j_{\text{acc}}(t)). \quad (41)$$

$$J^2 = E[(j_{\text{acc}}(t+T) - j_{\text{acc}}(t))^2] \quad (42)$$

$$J^2 = E[j_{\text{acc}}(t+T)^2 - 2j_{\text{acc}}(t+T)j_{\text{acc}}(t) + j_{\text{acc}}(t)^2] \quad (43)$$

$$J^2 = E[j_{\text{acc}}(t+T)^2] - 2E[j_{\text{acc}}(t+T)j_{\text{acc}}(t)] + E[j_{\text{acc}}(t)^2] \quad (44)$$

$$J^2 = R_{j_{\text{acc}}}(t+T, t+T) - 2R_{j_{\text{acc}}}(t+T, t) + R_{j_{\text{acc}}}(t, t) \quad (45)$$

$$J^2 = c(t+T) - 2ct + ct \quad (46)$$

$$J = \sqrt{cT} \quad (47)$$

which agrees with Demir [4]. We now have a way of relating the jitter of the oscillator to the PSD of η . However, η is not measurable, so instead the jitter is related to the phase noise S_ϕ . To do so, consider simple accumulating jitter written in terms of phase,

$$\phi_{\text{acc}}(t) = 2\pi f_0 j_{\text{acc}}(t) = 2\pi f_0 \int_0^t \eta(\tau) d\tau, \quad (48)$$

where $f_0 = 1/T$. From (37) and (48) the PSD of ϕ_{acc} is

$$S_{\phi_{\text{acc}}}(\Delta f) = 2c \frac{(2\pi f_0)^2}{(2\pi \Delta f)^2} = \frac{2cf_0^2}{\Delta f^2}. \quad (49)$$

Vendelin [26] showed that

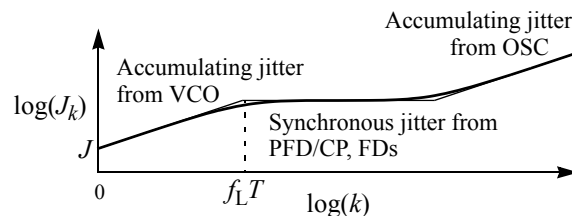
$$J = \sqrt{cT} = \sqrt{\frac{c}{f_0}} = \sqrt{\frac{82.6 \times 10^{-21}}{1.1 \text{ GHz}}} = 8.7 \text{ fs}. \quad (54)$$

In this example, the noise was extracted for the VCO alone. In practice, the LF is generally combined with the VCO before extracting the noise so that the noise of the LF is accounted for.

6 Jitter of a PLL

If a PLL synthesizer is constructed from blocks that exhibit simple synchronous and accumulating jitter, then the jitter behavior of the PLL is relatively easy to estimate [19]. Assume that the PLL has a closed-loop bandwidth of f_L , and that $\tau_L = 1/2\pi f_L$, then for k such that $kT \ll \tau_L$, jitter from the VCO dominates and the PLL exhibits simple accumulating jitter equal to that produced by the VCO. Similarly, at large k (low frequencies), the PLL exhibits simple accumulating jitter equal to that produced by the OSC. Between these two extremes, the PLL exhibits simple synchronous jitter. The amount of which depends on the characteristics of the loop and the level of synchronous jitter exhibited by the FDs and the PFD/CP. The behavior of such a PLL is shown in Figure 6.

FIGURE 6 Long-term jitter (J_k) for an idealized PLL as a function of the number of cycles.



7 Modeling PLLs with Jitter

The basic behavioral models for the blocks that make up a PLL are well known and so are not discussed here in any depth [1,2]. Instead, only the techniques for adding jitter to the models are discussed.

Jitter is modeled in an AHDL by dithering the time at which events occur. This is efficient because it does not create any additional activity, rather it simply changes the time when existing activity occurs. Thus, models with jitter can run as efficiently as those without.

7.1 Modeling Driven Blocks

A feature of Verilog-A allows especially simple modeling of synchronous jitter. The *transition()* function, which is used to model signal transitions between discrete levels, provides a delay argument that can be dithered on every transition. The delay argument must not be negative, so a fixed delay that is greater than the maximum expected deviation of the jitter must be included. This approach is suitable for any model that exhibits synchronous jitter and generates discrete-valued outputs. It is used in the Verilog-A

divider module shown in Listing 1, which models synchronous jitter with (10) where j_{sync} is a stationary white discrete-time Gaussian random process. It is also used in Listing 2, which models a simple PFD/CP.

LISTING 1 *Frequency divider that models synchronous jitter.*

```

`include "disciplines.vams"
module divider (out, in);
input in; output out; electrical in, out;
parameter real Vlo=-1, Vhi=1;
parameter integer ratio=2 from [2:inf);
parameter integer dir=1 from [-1:1] exclude 0; // dir=1 for positive edge trigger
                                                // dir=-1 for negative edge trigger

parameter real tt=1n from (0:inf);
parameter real td=0 from (0:inf);
parameter real jitter=0 from [0:td/5); // edge-to-edge jitter
parameter real ttol=1p from (0:td/5); // recommend ttol << jitter

integer count, n, seed;
real dt;

analog begin
    @(initial_step) seed = -311;
    @(cross(V(in) - (Vhi + Vlo)/2, dir, ttol)) begin
        // count input transitions
        count = count + 1;
        if (count >= ratio)
            count = 0;
        n = (2*count >= ratio);
        // add jitter
        dt = jitter*$rdist_normal(seed,0,1);
    end
    V(out) <+ transition(n ? Vhi : Vlo, td+dt, tt);
end
endmodule

```

7.1.1 Frequency Divider Model

The model, given in Listing 1, operates by counting input transitions. This is done in the `@cross` block. The cross function triggers the `@` block at the precise moment when its first argument crosses zero in the direction specified by the second argument. Thus, the `@` block is triggered when the input crosses the threshold in the user specified direction. The body of the `@` block increments the count, resets it to zero when it reaches ratio, then determines if count is above or below its midpoint (n is zero if the count is below the midpoint). It also generates a new random dither dT that is used later. Outside the `@` block is code that executes continuously. It processes n to create the output. The value of the `?:` operator is V_{hi} if n is 1 and V_{lo} if n is 0. Finally, the transition function adds a finite transition time of tt and a delay of $td + dt$. The finite transition time removes the discontinuities from the signal that could cause problems for the simulator. The jitter is embodied in dt , which varies randomly from transition to transition. To avoid negative

LISTING 2 PFD/CP model with synchronous jitter:

```

`include "disciplines.vams"
module pfd_cp (out, ref, vco);
input ref, vco; output out; electrical ref, vco, out;

parameter real lout=100u;
parameter integer dir=1 from [-1:1] exclude 0; // dir=1 for positive edge trigger
// dir=-1 for negative edge trigger

parameter real tt=1n from (0:inf);
parameter real td=0 from (0:inf);
parameter real jitter=0 from [0:td/5]; // edge-to-edge jitter
parameter real ttol=1p from (0:td/5); // recommend ttol << jitter

integer state, seed;
real dt;

analog begin
    @(initial_step) seed = 716;
    @(cross(V(ref), dir, ttol)) begin
        if (state > -1) state = state - 1;
        dt = jitter*$rdist_normal(seed,0,1);
    end
    @(cross(V(vco), dir, ttol)) begin
        if (state < 1) state = state + 1;
        dt = jitter*$rdist_normal(seed,0,1);
    end
    I(out) <+ transition(lout*state, td + dt, tt);
end
endmodule

```

delays, td must always be larger than dt . This model expects jitter to be specified as J_{ee} , as computed with (28).

7.1.2 PFD/CP Model

The model for a phase/frequency detector combined with a charge pump is given in Listing 2. It implements a finite-state machine with a three-level output, $-I_{out}$, 0 and $+I_{out}$. On every transition of the VCO input in direction dir , the output is incremented. On every transition of the reference input in the direction dir , the output is decremented. If both the VCO and reference inputs are at the same frequency, then the average value of the output is proportional to the phase difference between the two, with the average being negative if the reference transition leads the VCO transition and positive otherwise [7]. As before, the times of the output transitions are randomly dithered by dt to model jitter. The output is modeled as an ideal current source and a finite transition time provides a simple model of the dead band in the CP.

7.2 Modeling Accumulating Jitter

7.2.1 OSC Model

The delay argument of the *transition()* function cannot be used to model accumulating jitter because of the unbounded nature of this type of jitter. When modeling a fixed frequency oscillator, the *timer()* function is used as shown in Listing 3. At every output transition, the next transition is scheduled using the *timer()* function to be $T/K + J\delta/\sqrt{K}$ in the future, where δ is a unit-variance zero-mean random process and K is the number of output transitions per period. Typically, $K = 2$.

LISTING 3 *Fixed frequency oscillator with accumulating jitter.*

```

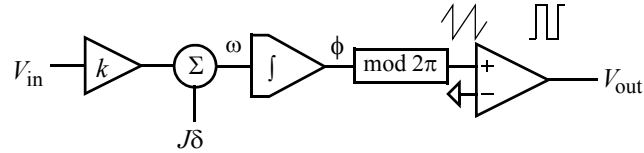
`include "disciplines.vams"
module osc (out);
output out; electrical out;
parameter real freq=1 from (0:inf);
parameter real Vlo=-1, Vhi=1;
parameter real tt=0.01/freq from (0:inf);
parameter real jitter=0 from [0:0.1/freq]; // period jitter
integer n, seed;
real next, dT;
analog begin
    @(initial_step) begin
        seed = 286;
        next = 0.5/freq + $abstime;
    end
    @(timer(next)) begin
        n = !n;
        dT = jitter*$rdist_normal(seed,0,1);
        next = next + 0.5/freq + 0.707*dT;
    end
    V(out) <+ transition(n ? Vhi : Vlo, 0, tt);
end
endmodule

```

7.3 VCO Model

A VCO generates a sine or square wave whose frequency is proportional to the input signal level. VCO models, given in Listings 4 and 5, are constructed using three serial operations, as shown in Figure 7. First, the input signal is scaled to compute the desired output frequency. Then, the frequency is integrated to compute the output phase. Finally, the phase is used to generate the desired output signal. The phase is computed with *idtmod*, a function that provides integration followed by a modulus operation. This serves to keep the phase bounded, which prevents a loss of numerical precision that would otherwise occur when the phase became large after a long period of time. Output transitions are generated when the phase passes $-\pi/2$ and $\pi/2$.

FIGURE 7 Block diagram of VCO behavioral model that includes jitter.



The jitter is modeled as a random variation in the frequency of the VCO. However, the jitter is specified as a variation in the period, thus it is necessary to relate the variation in the period to the variation in the frequency. Assume that without jitter, the period is divided into K equal intervals of duration $\tau = T/K = 1/Kf_0$. The frequency deviation will be updated every interval and held constant during the intervals. With jitter, the duration of an interval is

$$\tau_i = \tau + \Delta\tau_i. \quad (55)$$

$\Delta\tau$ is a random variable with variance

$$\text{var}(\Delta\tau) = \frac{\text{var}(T)}{K} = \frac{J^2}{K}. \quad (56)$$

Therefore,

$$\Delta\tau_i = \frac{J\delta_i}{\sqrt{K}} \quad (57)$$

where δ is a zero-mean unit-variance Gaussian random process. The dithered frequency is

$$f_i = \frac{1}{K(\tau + \Delta\tau_i)} = \frac{\frac{1}{K\tau}}{1 + \frac{\Delta\tau_i}{\tau}} = \frac{f_c}{1 + K\Delta\tau_i f_c} \quad (58)$$

Let $\Delta T_i = K\Delta\tau_i$, then

$$f_i = \frac{f_c}{1 + \Delta T_i f_c}. \quad (59)$$

Finally, $\text{var}(\tau_i) = J^2/K$, and so $\Delta\tau_i = J\delta_i/\sqrt{K}$ and $\Delta T_i = \sqrt{K}J\delta_i$.

The @cross statement is used to determine the exact time when the phase crosses the thresholds, indicating the beginning of a new interval. At this point, a new random trial δ_i is generated.

The final model given in Listing 4. This model can be easily modified to fit other needs. Converting it to a model that generates sine waves rather than square waves simply requires replacing the last two lines with one that computes and outputs the sine of the phase. When doing so, consider reducing the number of jitter updates to one per period, in which case the factor of 1.414 should be changed to 1.

LISTING 4 *VCO model that includes accumulating jitter.*

```

`include "disciplines.vams"
`include "constants.vams"

module vco (out, in);
input in; output out; electrical out, in;

parameter real Vmin=0;
parameter real Vmax=Vmin+1 from (Vmin:inf);
parameter real Fmin=1 from (0:inf);
parameter real Fmax=2*Fmin from (Fmin:inf);
parameter real Vlo=-1, Vhi=1;
parameter real tt=0.01/Fmax from (0:inf);
parameter real jitter=0 from [0:0.25/Fmax]; // period jitter
parameter real ttol=1u/Fmax from (0:1/Fmax);

real freq, phase, dT;
integer n, seed;

analog begin
    @(initial_step) seed = -561;

    // compute the freq from the input voltage
    freq = (V(in) - Vmin)*(Fmax - Fmin) / (Vmax - Vmin) + Fmin;

    // bound the frequency (this is optional)
    if (freq > Fmax) freq = Fmax;
    if (freq < Fmin) freq = Fmin;

    // add the phase noise
    freq = freq/(1 + dT*freq);

    // phase is the integral of the freq modulo 2π
    phase = 2*M_PI*idtmod(freq, 0.0, 1.0, -0.5);

    // update jitter twice per period
    // 1.414=sqrt(K), K=2 jitter updates/period
    @(cross(phase + `M_PI/2, +1, ttol) or cross(phase - `M_PI/2, +1, ttol)) begin
        dT = 1.414*jitter*$rdist_normal(seed,0, 1);
        n = (phase >= -`M_PI/2) && (phase < `M_PI/2);
    end

    // generate the output
    V(out) <+ transition(n ? Vhi : Vlo, 0, tt);
end
endmodule

```

Listing 5 is a Verilog-A model for a quadrature VCO that exhibits accumulating jitter. It is an example of how to model an oscillator with multiple outputs so that the jitter on the outputs is properly correlated.

7.4 Efficiency of the Models

Conceptually, a model that includes jitter should be just as efficient as one that does not because jitter does not increase the activity of the models, it only affects the timing of particular events. However, if jitter causes two events that would normally occur at the same time to be displaced so that they are no longer coincident, then a circuit simulator

LISTING 5 *Quadrature Differential VCO model that includes accumulating jitter.*

```

`include "disciplines.vams"
`include "constants.vams"

module quadVco (Plout,Nlout, PQout,NQout, Pin,Nin);
electrical Plout, Nlout, PQout, NQout, Pin, Nin;
output Plout, Nlout, PQout, NQout;
input Pin, Nin;

parameter real Vmin=0;
parameter real Vmax=Vmin+1 from (Vmin:inf);
parameter real Fmin=1 from (0:inf);
parameter real Fmax=2*Fmin from (Fmin:inf);
parameter real Vlo=-1, Vhi=1;
parameter real jitter=0 from [0:0.25/Fmax]; // period jitter
parameter real ttol=1u/Fmax from (0:1/Fmax);
parameter real tt=0.01/Fmax;

real freq, phase, dT;
integer i, q, seed;

analog begin
    @(initial_step) seed = 133;

    // compute the freq from the input voltage
    freq = (V(Pin,Nin) - Vmin) * (Fmax - Fmin) / (Vmax - Vmin) + Fmin;

    // bound the frequency (this is optional)
    if (freq > Fmax) freq = Fmax;
    if (freq < Fmin) freq = Fmin;

    // add the phase noise
    freq = freq/(1 + dT*freq);

    // phase is the integral of the freq modulo 2π
    phase = 2*M_PI*idtmod(freq, 0.0, 1.0, -0.5);

    // update jitter where phase crosses π/2
    // 2=sqrt(K), K=4 jitter updates per period
    @(cross(phase - 3*M_PI/4, +1, ttol) or cross(phase - M_PI/4, +1, ttol) or
    cross(phase + M_PI/4, +1, ttol) or cross(phase + 3*M_PI/4, +1, ttol)) begin
        dT = 2*jitter*$rdist_normal(seed,0,1);
        i = (phase >= -3*M_PI/4) && (phase < M_PI/4);
        q = (phase >= -M_PI/4) && (phase < 3*M_PI/4);
    end

    // generate the I and Q outputs
    V(Plout) <+ transition(i ? Vhi : Vlo, 0, tt);
    V(Nlout) <+ transition(i ? Vlo : Vhi, 0, tt);
    V(PQout) <+ transition(q ? Vhi : Vlo, 0, tt);
    V(NQout) <+ transition(q ? Vlo : Vhi, 0, tt);

end
endmodule

```

will have to use more time points to resolve the distinct events and so will run more slowly. For this reason, it is desirable to combine jitter sources to the degree possible.

To make the HDL models even faster, rewrite them in either Verilog-HDL or Verilog-AMS. Be sure to set the time resolution to be sufficiently small to prevent the discrete nature of time in these simulators from adding an appreciable amount of jitter.

7.4.1 Including Synchronous Jitter into OSC

One can combine the output-referred noise of FD_M and FD_N and the input-referred noise of the PFD/CP with the output noise of OSC. A modified fixed-frequency oscillator model that supports two jitter parameters and the divide ratio M is given in Listing 6 (more on the effect of the divide ratio on jitter in the next section). The `accJitter` param-

LISTING 6 *Fixed-frequency oscillator with accumulating and synchronous jitter.*

```

`include "disciplines.vams"
module osc (out);
output out; electrical out;

parameter real freq=1 from (0:inf);
parameter real ratio=1 from (0:inf);
parameter real Vlo=-1, Vhi=1;
parameter real tt=0.01*ratio/freq from (0:inf);
parameter real accJitter=0 from [0:0.1/freq]; // period jitter
parameter real syncJitter=0 from [0:0.1*ratio/freq]; // edge-to-edge jitter

integer n, accSeed, syncSeed;
real next, dT, dt, accSD, syncSD;

analog begin
    @(initial_step) begin
        accSeed = 286;
        syncSeed = -459;
        accSD = accJitter*sqrt(ratio/2);
        syncSD = syncJitter;
        next = 0.5/freq + $abstime;
    end

    @(timer(next + dt)) begin
        n = !n;
        dT = accSD*$rdist_normal(accSeed,0,1);
        dt = syncSD*$rdist_normal(syncSeed,0,1);
        next = next + 0.5*ratio/freq + dT;
    end

    V(out) <+ transition(n ? Vhi : Vlo, 0, tt);
end
endmodule

```

eter is used to model the accumulating jitter of the reference oscillator, and the `syncJitter` parameter is used to model the synchronous jitter of FD_M , FD_N and PFD/CP. Synchronous jitter is modeled in the oscillator without using a nonzero delay in the transition function. This is a more efficient approach because it avoids generating two unnecessary events per period. To get full benefit from this optimization, a modified PFD/CP given in Listing 7 is used. This model runs more efficiently by removing support for jitter and the `td` parameter.

LISTING 7 PFD/CP without jitter:

```

`include "disciplines.vams"
module pfd_cp (out, ref, vco);
input ref, vco; output out; electrical ref, vco, out;
parameter real lout=100u;
parameter integer dir=1 from [-1:1] exclude 0; // dir = 1 for positive edge trigger
// dir = -1 for negative edge trigger
parameter real tt=1n from (0:inf);
parameter real ttol=1p from (0:inf);
integer state;
analog begin
    @(cross(V(ref), dir, ttol)) begin
        if (state > -1) state = state - 1;
    end
    @(cross(V(vco), dir, ttol)) begin
        if (state < 1) state = state + 1;
    end
    I(out) <+ transition(lout * state, 0, tt);
end
endmodule

```

7.4.2 Merging the VCO and FD_N

If the output of the VCO is not used to drive circuitry external to the synthesizer, if the divider exhibits simple synchronous jitter, and if the VCO exhibits simple accumulating jitter, then it is possible to include the frequency division aspect of the FD_N as part of the VCO by simply adjusting the VCO gain and jitter. If the divide ratio of FD_N is large, the simulation runs much faster because the high VCO output frequency is never generated. The Verilog-A model for the merged VCO and FD_N is given in Listing 8. It also includes code for generating a logfile containing the length of each period. The logfile is used in Section 8 when determining S_{VCO} , the power spectral density of the phase of the VCO output.

Recall that the synchronous jitter of FD_M and FD_N has already been included as part of OSC, so the divider model incorporated into the VCO is noiseless and the jitter at the output of the noiseless divider results only from the VCO jitter. Since the divider outputs one pulse for every N pulses at its input, the variance in the output period is the sum of the variance in N input periods. Thus, the period jitter at the output, J_{FD} , is \sqrt{N} times larger than the period jitter at the input, J_{VCO} , or

$$J_{FD} = \sqrt{N}J_{VCO}. \quad (60)$$

Thus, to merge the divider into the VCO, the VCO gain must be reduced by a factor of N , the period jitter increased by a factor of \sqrt{N} , and the divider model removed.

After simulation, it is necessary to refer the computed results, which are from the output of the divider, to the output of VCO, which is the true output of the PLL. The period jitter at the output of the VCO, J_{VCO} , can be computed with (60).

To determine the effect of the divider on $S_{\phi}(\omega)$, square both sides of (60) and apply (47)

LISTING 8 *VCO with FD_N .*

```

`include "disciplines.vams"
module vco (out, in);
input in; output out; electrical out, in;

parameter real Vmin=0;
parameter real Vmax=Vmin+1 from (Vmin:inf);
parameter real Fmin=1 from (0:inf);
parameter real Fmax=2*Fmin from (Fmin:inf);
parameter real ratio=1 from (0:inf);
parameter real Vlo=-1, Vhi=1;
parameter real tt=0.01*ratio/Fmax from (0:inf);
parameter real jitter=0 from [0:0.25*ratio/Fmax]; // VCO period jitter
parameter real ttol=1u*ratio/Fmax from (0:ratio/Fmax);
parameter real outStart=inf from (1/Fmin:inf);

real freq, phase, dT, delta, prev, Vout;
integer n, seed, fp;

analog begin
    @(initial_step) begin
        seed = -561;
        delta = jitter * sqrt(2*ratio);
        fp = $fopen("periods.m");
        Vout = Vlo;
    end

    // compute the freq from the input voltage
    freq = (V(in) - Vmin)*(Fmax - Fmin) / (Vmax - Vmin) + Fmin;

    // bound the frequency (this is optional)
    if (freq > Fmax) freq = Fmax;
    if (freq < Fmin) freq = Fmin;

    // apply the frequency divider, add the phase noise
    freq = (freq / ratio)/(1 + dT * freq / ratio);

    // phase is the integral of the freq modulo 1
    phase = idtmod(freq, 0.0, 1.0, -0.5);

    // update jitter twice per period
    @(cross(phase - 0.25, +1, ttol)) begin
        dT = delta * $rdist_normal(seed, 0, 1);
        Vout = Vhi;
    end

    @(cross(phase + 0.25, +1, ttol)) begin
        dT = delta * $rdist_normal(seed, 0, 1);
        Vout = Vlo;
        if ($abstime >= outStart) $fstrobe( fp, "%0.10e", $abstime - prev);
        prev = $abstime;
    end

    end
    V(out) <+ transition(Vout, 0, tt);
end
endmodule

```

$$c_{\text{VCO}} T_{\text{VCO}} = \frac{c_{\text{FD}} T_{\text{FD}}}{N}. \quad (61)$$

$T_{\text{VCO}} = T_{\text{FD}} / N$, and so

$$c_{\text{VCO}} = c_{\text{FD}} \quad (62)$$

From (49),

$$2S_{\text{VCO}} \frac{f^2}{f_{\text{VCO}}^2} = 2S_{\text{FD}} \frac{f^2}{f_{\text{FD}}^2} \quad (63)$$

Finally, $f_{\text{VCO}} = N f_{\text{FD}}$, and so

$$S_{\text{VCO}} = N^2 S_{\text{FD}}. \quad (64)$$

Once FD_N is incorporated into the VCO, the VCO output signal is no longer observable, however the characteristics of the VCO output are easily derived from (60) and (64), which are summarized in Table 3.

TABLE 3 Characteristics of VCO output relative to the output of FD_N assuming the VCO exhibits simple accumulating jitter and the FD_N is noise free.

Frequency	Jitter	Phase Noise
$f_{\text{VCO}} = N f_{\text{FD}}$	$J_{\text{VCO}} = \frac{J_{\text{FD}}}{\sqrt{N}}$	$S_{\phi_{\text{VCO}}} = N^2 S_{\phi_{\text{FD}}}$

It is interesting to note that while the frequency at the output of FD_N is N times smaller than at the output of the VCO, except for scaling in the amplitude, the spectrum of the noise close to the fundamental is to a first degree unaffected by the presence of FD_N . In particular, the width of the noise spectrum is unaffected by FD_N . This is extremely fortuitous, because it means that the number of cycles we need to simulate is independent of the divide ratio N . Thus, large divide ratios do not affect the total simulation time.

To understand why FD_N does not affect the width of the noise spectrum, recall that while we started with a jitter that varied continuously with time, $j(t)$ in (2), for either efficiency or modeling reasons we eventually sampled it to end up with a discrete-time version. The act of sampling the jitter causes the spectrum of the jitter to be replicated at the multiples of the sampling frequency, which adds aliasing. This aliasing is visible, but not obvious, at high frequencies in Figure 9. However, especially with accumulating jitter, the phase noise amplitude at low frequencies is much larger than the aliased noise, and so the close-in noise spectrum is largely unaffected by the sampling. The effect of FD_N is to decimate the sampled jitter by a factor of N , which is equivalent to sampling the jitter signal, $j(t)$, at the original sample frequency divided by N . Thus, the replication is at a lower frequency, the amplitude is lower, and the aliasing is greater, but the spectrum is otherwise unaffected.

8 Simulation and Analysis

The synthesizer is simulated using the netlist from Listing 10 and the Verilog-A descriptions in Listings 6-8, modifying them as necessary to fit the actual circuit. The simulation should cover an interval long enough to allow accurate Fourier analysis at the lowest frequency of interest (F_{\min}). With deterministic signals, it is sufficient to simulate for K cycles after the PLL settles if $F_{\min} = 1/(TK)$. However, for these signals, which are stochastic, it is best to simulate for $10K$ to $100K$ cycles to allow for enough averaging to reduce the uncertainty in the result.

One should not simply apply an FFT to the output signal of the VCO/ FD_N to determine $L(\Delta f)$ for the PLL. The result would be quite inaccurate because the FFT samples the waveform at evenly spaced points, and so misses the jitter of the transitions. Instead, $L(\Delta f)$ can be measured with Spectre's Fourier Analyzer, which uses a unique algorithm that does accurately resolve the jitter [11]. However, it is slow if many frequencies are needed and so is not well suited to this application.

Unlike $L(\Delta f)$, $S_\phi(\Delta f)$ can be computed efficiently. The Verilog-A code for the VCO/ FD_N given in Listing 8 writes the length of each period to an output file named *periods.m*. Writing the periods to the file begins after an initial delay, specified using `outStart`, to allow the PLL to reach steady state. This file is then processed by Matlab from MathWorks using the script shown in Listing 9. This script computes $S_\phi(\Delta f)$, the power spectral density of ϕ , using Welch's method [20]. The frequency range is from $f_{\text{out}}/2$ to $f_{\text{out}}/nfft$. The script computes $S_\phi(\Delta f)$ with a resolution bandwidth of `rbw`.¹⁰ Normally, $S_\phi(\Delta f)$ is given with a unity resolution bandwidth. To compensate for a non-unity resolution bandwidth, broadband signals such as the noise should be divided by `rbw`. Signals with bandwidth less than `rbw`, such as the spurs generated by leakage in the CP, should not be scaled. The script processes the output of VCO/ FD_N . The results of the script must be further processed using the equations in Table 3 to remove the effect of FD_N .

9 Example

These ideas were applied to model and simulate a PLL acting as a frequency synthesizer. A synthesizer was chosen with $f_{\text{ref}} = 25$ MHz, $f_{\text{out}} = 2$ GHz, and a channel spacing of 200 kHz. As such, $M = 125$ and $N = 10,000$.

The noise of OSC is -95 dBc/Hz at 100 kHz. Applying (52) to compute c , where $L(\Delta f) = 316 \times 10^{-12}$, $\Delta f = 100$ kHz, and $f_o = 25$ MHz, gives $c = 5 \times 10^{-15}$. The period jitter J is then computed from (47) to be 14 ps.

The noise of VCO is -48 dBc/Hz at 100 kHz. Applying (52) and (47) with $L(\Delta f) = 1.59 \times 10^{-5}$, $\Delta f = 100$ kHz, and $f_o = 2$ GHz, gives $c = 4 \times 10^{-14}$ and a period jitter of $J = 4.5$ ps.

The period jitter of the PFD/CP and FDs was found to be 2 ns. The FDs were included into the oscillators, which suppresses the high frequency signals at the input and output

10. The Hanning window used in the `psd()` function has a resolution bandwidth of 1.5 bins [9]. Assuming broadband signals, Matlab divides by 1.5 inside `psd()` to compensate. In order to resolve narrowband signals, the factor of 1.5 is removed by the script, and instead included in the reported resolution bandwidth.

LISTING 9 *Matlab script used for computing $S_{\phi}(\Delta f)$. These results must be further processed using Table 3 to map them to the output of the VCO.*

```
% Process period data to compute  $S_{\phi}(\Delta f)$ 
echo off;
nfft=512; % should be power of two
winLength=nfft;
overlap=nfft/2;
winNBW=1.5; % Noise bandwidth given in bins

% Load the data from the file generated by the VCO
load periods.m;

% output estimates of period and jitter
T=mean(periods);
J=std(periods);
maxdT = max(abs(periods-T))/T;
fprintf('T = %.3gs, F = %.3GHz\n', T, 1/T);
fprintf('Jabs = %.3gs, Jrel = %.2g%%\n', J, 100*J/T);
fprintf('max dT = %.2g%%\n', 100*maxdT);
fprintf('periods = %d, nfft = %d\n', length(periods), nfft);

% compute the cumulative phase of each transition
phases=2*pi*cumsum(periods)/T;

% compute power spectral density of phase
[Sphi,f]=psd(phases,nfft,1/T,winLength,overlap,'linear');

% correct for scaling in PSD due to FFT and window
Sphi=winNBW*Sphi/nfft;

% plot the results (except at DC)
K = length(f);
semilogx(f(2:K),10*log10(Sphi(2:K)));
title('Power Spectral Density of VCO Phase');
xlabel('Frequency (Hz)');
ylabel('S phi (dB/Hz)');
rbw = winNBW/(T*nfft);
RBW=sprintf('Resolution Bandwidth = %.0f Hz (%.0f dB)', rbw, 10*log10(rbw));
imtext(0.5,0.07, RBW);
```

of the synthesizer. The netlist is shown in Listing 10. The results (compensated for non-unity resolution bandwidth (-28 dB) and for the suppression of the dividers (80 dB)) are shown in Figures 8-11. The simulation took 7.5 minutes for 450k time-points on a HP 9000/735. The use of a large number of time points was motivated by the desire to reduce the level of uncertainty in the results. The period jitter in the PLL was found to be 9.8 ps at the output of the VCO.

The low-pass filter LF blocks all high frequency signals from reaching the VCO, so the noise of the phase lock loop at high frequencies is the same as the noise generated by the open-loop VCO alone. At low frequencies, the loop gain acts to stabilize the phase of the VCO, and the noise of the PLL is dominated by the phase noise of the OSC. There is some contribution from the VCO, but it is diminished by the gain of the loop. In this example, noise at the middle frequencies is dominated by the synchronous jitter generated by the PFD/CO and FDs. The measured results agree qualitatively with the expected results. The predicted noise is higher than one would expect solely from the

LISTING 10 *Spectre netlist for PLL synthesizer.*

```
// PLL-based frequency synthesizer that models jitter
simulator lang=spectre

ahdl_include "osc.va" // Listing 6
ahdl_include "pfd_cp.va" // Listing 7
ahdl_include "vco.va" // Listing 8

Osc (in)oscfreq=25MHz ratio=125 accJitter=14ps syncJitter=2ns
PFD(err in fb)pfd_cpout=500ua
C1 (err c)capacitorc=3.125nF
R (c 0)resistorr=10k
C2 (c 0)capacitorc=625pF
VCO(fb err)vcoFmin=1GHz Fmax=3GHz Vmin=-4 Vmax=4 ratio=10000 \
jitter=4.5ps outStart=10ms

JitterSimtranstop=60ms
```

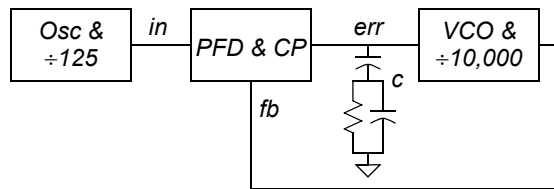
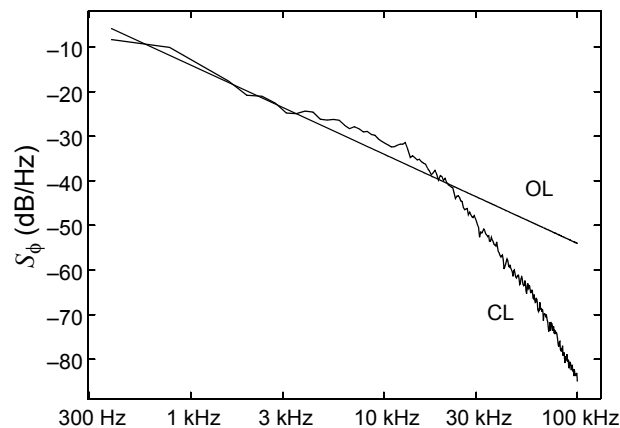


FIGURE 8 *Noise of the closed-loop PLL at the output of the VCO when only the reference oscillator exhibits jitter (CL) versus the noise of the reference oscillator mapped up to the VCO frequency when operated open loop (OL).*



open-loop behavior of each block because of peaking in the response of the PLL from 5 kHz to 50 kHz. For this reason, PLLs used in synthesizers where jitter is important are usually over damped.

10 Conclusion

A methodology for modeling and simulating the jitter performance of phase-locked loops was presented. The simulation is done at the behavioral level, and so is efficient

FIGURE 9 Noise of the closed-loop PLL at the output of the VCO when only the VCO exhibits jitter (CL) versus the noise of the VCO when operated open loop (OL).

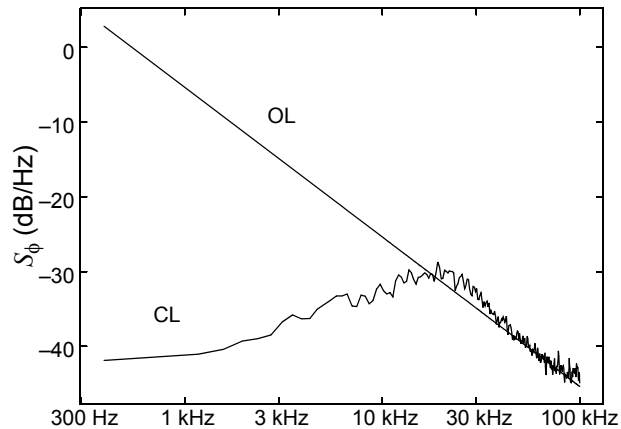
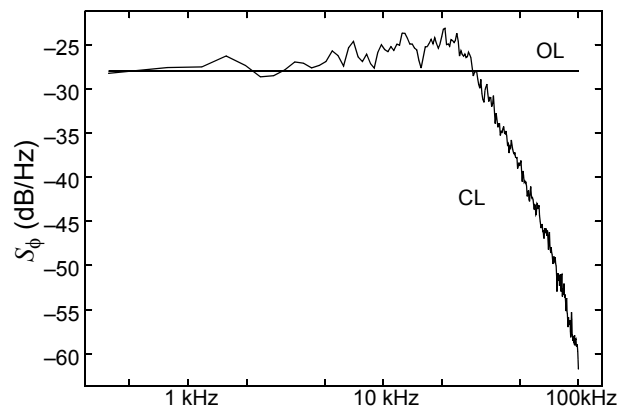


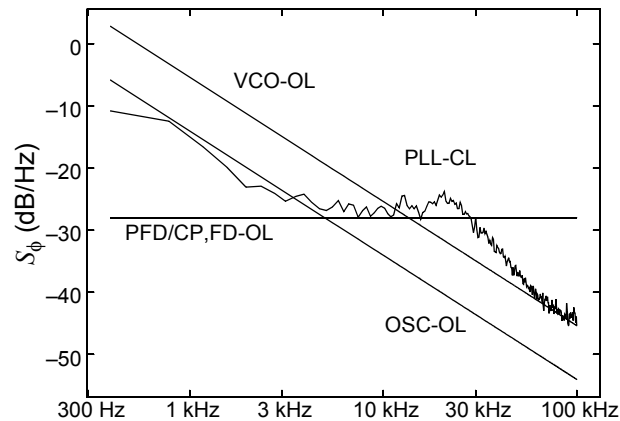
FIGURE 10 Noise of the closed-loop PLL at the output of the VCO when only the PFD/CP, FD_M , and FD_N exhibit jitter (CL) versus the noise of these components mapped up to the VCO frequency when operated open loop (OL).



enough to be applied in a wide variety of applications. The behavioral models are calibrated from circuit-level noise simulations, and so the high-level simulations are accurate. Behavioral models were presented in the Verilog-A language, however these same ideas can be used to develop behavioral models in purely event-driven languages such as Verilog-HDL and Verilog-AMS.

This methodology is flexible enough to be used in a broad range of applications where jitter is important. Examples include, clock generation and recovery, sampling systems, over-sampled ADCs, digital modulation and demodulation systems, and fractional- N frequency synthesis (though it is not possible to merge the VCO and divider in this case).

FIGURE 11 Closed-loop PLL noise performance compared to the open-loop noise performance of the individual components that make up the PLL. The achieved noise is slightly larger than what is expected from the components due to peaking in the response of the PLL.



10.1 If You Have Questions

If you have questions about what you have just read, feel free to post them on the *Forum* section of *The Designer's Guide Community* website. Do so by going to www.designers-guide.org/Forum. For more in depth questions, feel free to contact me in my role as a consultant at ken@designers-guide.com.

Acknowledgement

I would like to recognize the collective contributions of the readers of www.designers-guide.org, who have pointed out and helped correct many errors. I would also like to thank Alper Demir and Manolis Terrovitis of the University of California in Berkeley for many enlightening conversations about noise and jitter. Furthermore, I would like to thank Mark Chapman, Masayuki Takahashi, and Kimihiro Ogawa of Sony Semiconductor, Rich Davis, Frank Hellmich and Randeep Sooin of Cadence Design Systems, Jess Chen of RF Micro Devices, Frank Herzel of IHP Microelectronics and Frank Wiedmann of Infineon for their probing questions and insightful comments, as well as their help in validating these ideas on real frequency synthesizers.

Thanks to Srinivasa Rao Madala for pointing out an error in (25), the cycle-to-cycle jitter of synchronous jitter.

References

- [1] H. Chang, E. Charbon, U. Choudhury, A. Demir, E. Felt, E. Liu, E. Malavasi, A. Sangiovanni-Vincentelli, and I. Vassiliou. *A Top-Down Constraint-Driven Methodology for Analog Integrated Circuits*. Kluwer Academic Publishers, 1997.

- [2] A. Demir, E. Liu, A. Sangiovanni-Vincentelli, and I. Vassiliou. Behavioral simulation techniques for phase/delay-locked systems. *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 453-456, May 1994.
- [3] A. Demir, E. Liu, and A. Sangiovanni-Vincentelli. Time-domain non-Monte-Carlo noise simulation for nonlinear dynamic circuits with arbitrary excitations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 5, pp. 493-505, May 1996.
- [4] A. Demir, A. Mehrotra, and J. Roychowdhury. Phase noise in oscillators: a unifying theory and numerical methods for characterization. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 5, May 2000, pp. 655 -674.
- [5] A. Demir, A. Sangiovanni-Vincentelli. Simulation and modeling of phase noise in open-loop oscillators. *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 445-456, May 1996.
- [6] A. Demir, A. Sangiovanni-Vincentelli. *Analysis and Simulation of Noise in Nonlinear Electronic Circuits and Systems*. Kluwer Academic Publishers, 1997.
- [7] F. Gardner. *Phaselock Techniques*. John Wiley & Sons, 1979.
- [8] W. Gardner. *Introduction to Random Processes: With Applications to Signals and Systems*. McGraw-Hill, 1989.
- [9] F. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, vol. 66, no. 1, January 1978.
- [10] Frank Herzel and Behzad Razavi. A study of oscillator jitter due to supply and substrate noise. *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 46. no. 1, Jan. 1999, pp. 56-62.
- [11] K. Kundert. *The Designer's Guide to SPICE and Spectre*. Kluwer Academic Publishers, 1995.
- [12] Kenneth S. Kundert. *The Designer's Guide to Verilog-AMS*. Kluwer Academic Publishers, 2004.
- [13] Ken Kundert. Introduction to RF simulation and its application. *Journal of Solid-State Circuits*, vol. 34, no. 9, September 1999. Available from www.designers-guide.org/Analysis.
- [14] Ken Kundert. Modeling and simulation of jitter in phase-locked loops. In *Analog Circuit Design: RF Analog-to-Digital Converters; Sensor and Actuator Interfaces; Low-Noise Oscillators, PLLs and Synthesizers*, Rudy J. van de Plassche, Johan H. Huijsing, Willy M.C. Sansen, Kluwer Academic Publishers, November 1997.
- [15] Ken Kundert. Modeling and simulation of jitter in PLL frequency synthesizers. Available from www.designers-guide.org/Analysis.
- [16] K. Kundert. Predicting the phase noise of PLL-based frequency synthesizers. Available from www.designers-guide.org/Analysis.
- [17] David C Lee, Analysis of jitter in phase-locked loops. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 11, pp. 704 -711, November 2002.

- [18] Maxim Integrated Products. *Converting between RMS and Peak-to-Peak Jitter at a Specified BER*. Application note HFAN-4.0.2, December 2000. Available from pdfserv.maxim-ic.com/arpdf/AppNotes/3hfan402.pdf.
- [19] J. McNeill. Jitter in Ring Oscillators. *IEEE Journal of Solid-State Circuits*, vol. 32, no. 6, June 1997.
- [20] A. Oppenheim, R. Schaffer. *Digital Signal Processing*. Prentice-Hall, 1975.
- [21] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 1991.
- [22] J. Phillips and K. Kundert. Noise in mixers, oscillators, samplers, and logic: an introduction to cyclostationary noise. The paper and presentation are both available from www.designers-guide.org/Theory.
- [23] J. J. Rael and A. A. Abidi. Physical processes of phase noise in differential LC oscillators. *Proceedings of the IEEE Custom Integrated Circuits Conference, CICC 2000*.
- [24] R. Telichevesky, K. Kundert, J. White. Receiver characterization using periodic small-signal analysis. *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1996.
- [25] R. Telichevesky, K. Kundert, J. White. Efficient AC and noise analysis of two-tone RF circuits. *Proceedings of the 33rd Design Automation Conference*, June 1996.
- [26] G. Vendelin, A. Pavio, U. Rohde. *Microwave Circuit Design*. J. Wiley & Sons, 1990.
- [27] *Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL*, version 2.1. Accellera, January 20, 2003. Available from www.accelera.org. An abridged version is available from www.verilog-ams.com or www.designers-guide.org.
- [28] T. C. Weigandt, B. Kim, and P. R. Gray. Jitter in ring oscillators. *1994 IEEE International Symposium on Circuits and Systems (ISCAS-94)*, vol. 4, 1994, pp. 27-30.