Predicting the Phase Noise and Jitter of PLL-Based Frequency Synthesizers

Ken Kundert
Designer's Guide Consulting, Inc.

Version 4i, 23 October 2015

Two methodologies are presented for predicting the phase noise and jitter of a PLL-based frequency synthesizer using simulation that are both accurate and efficient. The methodologies begin by characterizing the noise behavior of the blocks that make up the PLL using transistor-level RF simulation. For each block, the phase noise or jitter is extracted and applied to a model for the entire PLL.

Search Terms

Phase-locked loop, PLL simulation, PLL phase-domain modeling, frequency synthesizer, oscillator phase noise, jitter, cyclostationary noise, charge-pump noise, phase-detector noise, frequency divider noise, SpectreRF, Verilog-A.

This paper was written in August 2002. It was last updated on March 10, 2019. You can find the most recent version at www.designers-guide.org. Contact the author via e-mail at ken@designers-guide.com.

Contents

1 Introduction

Phase-locked loops (PLLs) are used to implement a variety of timing related functions, such as frequency synthesis, clock and data recovery, and clock de-skewing. Any jitter or phase noise in the output of the PLL used in these applications generally degrades the performance margins of the system in which it resides and so is of great concern to the designers of such systems. Jitter and phase noise are different ways of referring to an undesired variation in the timing of events at the output of the PLL. They are difficult to

predict with traditional circuit simulators because the PLL generates repetitive switching events as an essential part of its operation, and the noise performance must be evaluated in the presence of this large-signal behavior. SPICE is useless in this situation as it can only predict the noise in circuits that have a quiescent (time-invariant) operating point. In PLLs the operating point is at best periodic, and is sometimes chaotic. Recently a new class of circuit simulators has been introduced that are capable of predicting the noise behavior about a periodic operating point [18]. Spectre®RF[1] is the most popular of this class of simulators and, because of the algorithms used in its implementation, is likely to be the best suited for this application [1]. These simulators can be used to predict the noise performance of PLLs. The ideas presented in this paper allow those simulators to be applied even to those PLLs that have chaotic operating points.

The focus of this paper is frequency synthesis. Information on predicting the noise and jitter of clock and data recovery circuits can be found elsewhere [21,23].

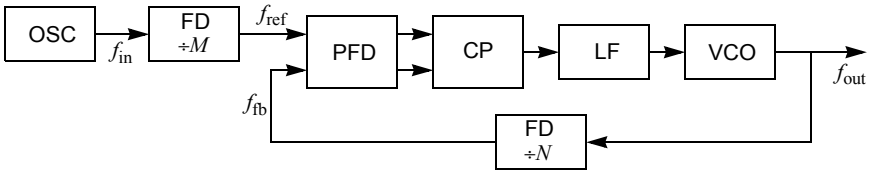
1.1 Frequency Synthesis

The block diagram of a PLL operating as a frequency synthesizer is shown in Figure 1 [8]. 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 an input frequency f_{in}, the frequency at the output of the PLL is

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

where M is the divide ratio of the input frequency divider, and N is the divide ratio of the feedback divider. By choosing the frequency divide ratios and the input frequency appropriately, the synthesizer generates an output signal at the desired frequency that inherits the long-term stability of the input oscillator. In RF transceivers, this architecture is commonly used to generate the local oscillator (LO) at a programmable frequency that tunes the transceiver to the desired channel by adjusting the value of N.

FIGURE 1 *The block diagram of a frequency synthesizer.*



1.2 Direct Simulation

In many circumstances, SpectreRF can be directly applied to predict the noise performance of a PLL. To make this possible, the PLL must at a minimum have a periodic steady state solution. This rules out systems such as bang-bang clock and data recovery circuits and fractional-N synthesizers because they behave in a chaotic way by design. It

1. Spectre is a registered trademark of Cadence Design Systems.

also rules out any PLL that is implemented with a phase detector that has a dead zone. A dead zone has the effect of opening the loop and letting the phase drift seemingly at random when the phase of the reference and the output of the voltage-controlled oscillator (VCO) are close. This gives these PLLs a chaotic nature.

To perform a noise analysis, SpectreRF must first compute the steady-state solution of the circuit with its periodic steady state (PSS) analysis. If the PLL does not have a periodic solution, as the cases described above do not, then it will not converge. There is an easy test that can be run to determine if a circuit has a periodic steady-state solution. Simply perform a transient analysis until the PLL approaches steady state and then observe the VCO control voltage. If this signal consists of frequency components at integer multiples of the reference frequency, then the PLL has a periodic solution. If there are other components, it does not. Sometimes it can be difficult to identify the undesirable components if the components associated with the reference frequency are large. In this case, use the strobing feature of Spectre's transient analysis to eliminate all components at frequencies that are multiples of the reference frequency. Do so by strobing at the reference frequency. In this case, if the strobed VCO control voltage varies in any significant way the PLL does not have a periodic solution.

If the PLL has a periodic solution, then in concept it is always possible to apply SpectreRF directly to perform a noise analysis. However, in some cases it may not be practical to do so. The time required for SpectreRF to compute the noise of a PLL is proportional to the number of circuit equations needed to represent the PLL in the simulator multiplied by both the number of time points needed to accurately render a single period of the solution and the number of frequencies at which the noise is desired. When applying SpectreRF to frequency synthesizers with large divide ratios, the number of time points needed to render a period can become problematic. Experience shows that divide ratios greater than ten are often not practical to simulate. Of course, this varies with the size of the PLL.

For PLLs that are candidates for direct simulation using SpectreRF, simply configure the simulator to perform a PSS analysis followed by a periodic noise (PNoise) analysis. The period of the PSS analysis should be set to be the same as the reference frequency as defined in Figure 1. The PSS stabilization time (tstab) should be set long enough to allow the PLL to reach lock. This process was successfully followed on a frequency synthesizer with a divide ratio of 40 that contained 2500 transistors, though it required several hours for the complete simulation [36].

1.3 When Direct Simulation Fails

The challenge still remains, how does one predict the phase noise and jitter of PLLs that do not fit the constraints that enable direct simulation? The remainder of this document attempts to answer that question for frequency synthesizers, though the techniques presented are general and can be applied to other types of PLLs by anyone who is sufficiently determined.

1.4 Monte Carlo-Based Methods

Demir proposed an approach for simulating PLLs whereby a PLL is described using behavioral models simulated at a high level [3,4]. The models are written such that they include jitter in an efficient way. He also devised a simulation algorithm based on solv-

ing a set of nonlinear stochastic differential equations that is capable of characterizing the circuit-level noise behavior of blocks that make up a PLL [4,5]. 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]. Once everything is ready, simulation of the PLL occurs with the blocks of the PLL being described with behavioral models that exhibit jitter. The actual jitter or phase noise statistics of the PLL are observed during this simulation. Generally tens to hundreds of thousands of cycles are simulated, but the models are efficient so the time required for the simulation is reasonable. This approach allows 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 to characterize the jitter of the blocks.

In an earlier series of papers [19,20], the relevant ideas of Demir were adapted to allow use of a commercial simulator, Spectre [16], and an industry standard modeling language, Verilog-A[2] [17,33]. These ideas are further refined in the later half of this manuscript.

1.5 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 in the first half of this manuscript.

The process of predicting the phase noise of a PLL using phase-domain models involves:

1. Using SpectreRF to predict the noise of the individual blocks that make up the PLL.
2. Building high-level behavioral models of each of the blocks that exhibit phase noise.
3. Assembling the blocks into a model of the PLL.
4. Simulating the PLL to find the phase noise of the overall system.

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 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

2. Verilog is a registered trademark of Cadence Design Systems licensed to Accellera.

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 with voltage-domain models 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 exhibit jitter.
4. Assembling the blocks into a model of the PLL.
5. Simulating the PLL to find the jitter of the overall system.

The simple linear phase-domain model described in the first part of this paper, and the nonlinear voltage-domain model described in the second part, 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 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 Phase-Domain Model

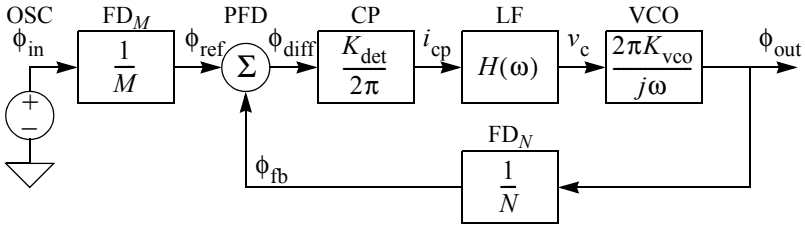
It is widely understood that simulating PLLs is expensive because the period of the VCO is almost always very short relative to the time required to reach lock. This is particularly true with frequency synthesizers, especially those with large multiplication factors. The problem is that a circuit simulator must use at least 10-20 time points for every period of the VCO for accurate rendering, and the lock process often involves hundreds or thousands of cycles at the input to the phase detector. With large divide ratios, this can translate to hundreds of thousands of cycles of the VCO. Thus, the number of time points needed for a single simulation could range into the millions.

This is all true when simulating the PLL in terms of voltages and currents. When doing so, one is said to be using *voltage-domain models*. However, that is not the only option available. It is also possible to formulate models based on the phase of the signals. In this case, one would be using *phase-domain models*. The high frequency variations associated with the voltage-domain models are not present in phase-domain models, and so simulations are considerably faster. In addition, when in lock the phase-domain-

based models generally have constant-valued operating points, which simplifies small-signal analysis, making it easier to study the closed-loop dynamics and noise performance of the PLL using either AC or noise analysis.

A linear phase-domain model of a frequency synthesizer is shown in Figure 2. Such a model is suitable for modeling the behavior of the PLL to small perturbations when the PLL is in lock as long as you do not need to know the exact waveforms and instead are interested in how small perturbations affect the phase of the output. This is exactly what is needed to predict the phase noise performance of the PLL.

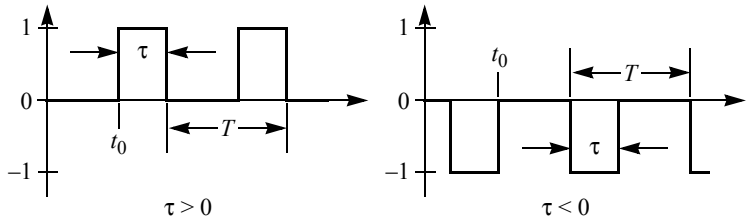
FIGURE 2 *Linear time-invariant phase-domain model of the synthesizer shown in Figure 1.*



The derivation of the model begins with the identification of those signals that are best represented by their phase. Many blocks have large repetitive input signals with their outputs being primarily sensitive to the phase of their inputs. It is the signals that drive these blocks that are represented as phase. They are identified using a ϕ variable in Figure 2. Notice that this includes all signals except those at the inputs of the LF and VCO.

The models of the individual blocks will be derived by assuming that the signals associated with each of the phase variables is a pulse train. Though generally the case, it is not a requirement. It simply serves to make it easier to extract the models. Define $\Pi(t_0, \tau, T)$ to be a periodic pulse train where one of the pulses starts at t_0 and the pulses have duration τ and period T as shown in Figure 3. This signal transitions between 0 and 1 if τ is positive, and between 0 and –1 if τ is negative. The phase of this signal is defined to be $\phi = 2\pi t_0/T$. In many cases, the duration of the pulses is of no interest, in which case $\Pi(t_0, T)$ is used as a short hand. This occurs because the input that the signal is driving is edge triggered. For simplicity, we assume that such inputs are sensitive to the rising edges of the signal, that t_0 specifies the time of a rising edge, and that the signal is transitioning between 0 and ±1.

FIGURE 3 *The pulse train waveform represented by* $\Pi(t_0, \tau, T)$.



The input source produces a signal $v_{in} = \Pi(t_0, T)$. Since this is the input, t_0 is arbitrary. As such, we are free to set its phase ϕ to any value we like.

Given a signal $v_i = \Pi(t_0, T)$ a frequency divider will produce an output signal $v_o = \Pi(t_0, NT)$ where N is the divide ratio. The phase of the input is $\phi_i = 2\pi t_0/T$ and the phase of the output is $\phi_o = 2\pi t_0/(NT)$ and so the phase transfer characteristic of a divider is

$$\phi_o = \phi_i/N. \tag{2}$$

There are many different types of phase detectors that can be used, each requiring a somewhat different model. Consider a simple phase-frequency detector combined with a charge pump [31]. In this case, the detector takes two inputs, $v_1 = \Pi(t_1, T)$ and $v_0 = \Pi(t_0, T)$ and produces an output $i_{cp} = I_{max}\Pi(t_0, t_1 - t_0, T)$ where I_{max} is the maximum output current of the charge pump. The output of the charge pump immediately passes through a low pass filter that is designed to suppress signals at frequencies of $1/T$ and above, so in most cases the pulse nature of this signal can be ignored in favor of its average value, $\langle i_{cp} \rangle$. Thus, the transfer characteristic of the combined PFD/CP is

$$\langle i_{cp} \rangle = I_{max}\frac{t_1 - t_0}{T} = I_{max}\frac{\phi_1 - \phi_0}{2\pi} = \frac{K_{det}}{2\pi}(\phi_1 - \phi_0) \tag{3}$$

where $K_{det} = I_{max}$. Of course, this is only valid for $|\phi_1 - \phi_2| < 2\pi$ at the most. The behavior outside this range depends strongly on the type of phase detector used [8]. Even within this range, the phase detector may be better modeled with a nonlinear transfer characteristic. For example, there can be a flat spot in the transfer characteristics near 0 if the detector has a dead zone. However it is generally not productive to model the dead zone in a phase-domain model.[3]

The model of (3) is a continuous-time approximation to what is inherently a discrete-time process. The phase detector does not continuously monitor the phase difference between its two input signals, rather it outputs one pulse per cycle whose width is proportional to the phase difference. Using a continuous time approximation is generally acceptable if the bandwidth of the loop filter is much less than f_{ref} (generally less than $f_{ref}/10$ is sufficient). In practical PLLs this is almost always the case. It is possible to develop a detailed phase-domain PFD model that includes the discrete-time effects, but it would run more slowly and the resulting phase-domain model of the PLL would not have a quiescent operating point, which makes it more difficult to analyze.

The voltage-controlled oscillator, or VCO, converts its input voltage to an output frequency, and the relationship between input voltage and output frequency can be represented as

$$f_{out} = F(v_c) \tag{4}$$

The mapping from voltage to frequency is designed to be linear, so a first-order model is often sufficient,

$$f_{out} = K_{vco}v_c. \tag{5}$$

It is the output phase that is needed in a phase-domain model,

3. This phase-domain model is a continuous-time model that ignores the sampling nature of the phase detector. A dead zone interacts with the sampling nature of the detector to create a chaotic limit cycle behavior that is not modeled with the phase-domain model. This chaotic behavior creates a substantial amount of jitter, and for this reason, most modern phase detectors are designed such that they do not exhibit dead zones.

$$\phi_{out}(t) = 2\pi\int K_{vco} v_c(t) dt \tag{6}$$

or in the frequency domain,

$$\phi_{out}(\omega) = \frac{2\pi K_{vco}}{j\omega} v_c(\omega) . \tag{7}$$

2.1 Small-Signal Stability

This completes the derivation of the phase-domain models for each of the blocks. Now the full model is used to help predict the small-signal behavior of the PLL. Start by using Figure 2 to write a relationship for its loop gain. Start by defining

$$G_{fwd} = \frac{\phi_{out}}{\phi_{diff}} = \frac{K_{det}}{2\pi} H(\omega) \frac{2\pi K_{vco}}{j\omega} = \frac{K_{det} K_{vco} H(\omega)}{j\omega} \tag{8}$$

to be the forward gain,

$$G_{rev} = \frac{\phi_{fb}}{\phi_{out}} = \frac{1}{N} \tag{9}$$

to be the feedback factor, and

$$T = G_{fwd} G_{rev} = \frac{K_{det} K_{vco} H(\omega)}{j\omega N} \tag{10}$$

to be the loop gain. The loop gain is used to explore the small-signal stability of the loop. In particular, the phase margin is an important stability metric. It is the negative of the difference between the phase shift of the loop at unity gain and 180°, the phase shift that makes the loop unstable. It should be no less than 45° [10]. When concerned about phase noise or jitter, the phase margin is typically 60° or more to reduce peaking in the closed-loop gain, which would result in excess phase noise.

2.2 Noise Transfer Functions

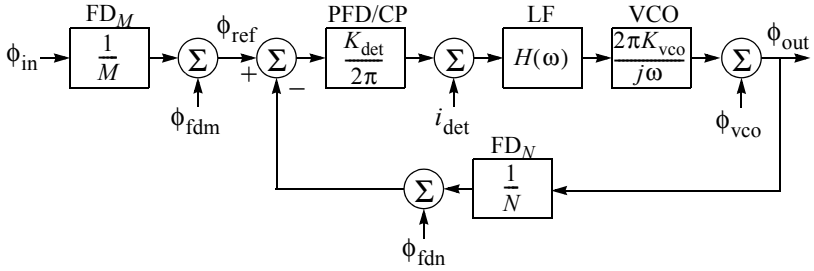
In Figure 4 various sources of noise have been added. These noise sources can represent either the noise created by the blocks due to intrinsic noise sources (thermal, shot, and flicker noise sources), or the noise coupled into the blocks from external sources, such as from the power supplies, the substrate, etc. Most are sources of phase noise, and denoted ϕ_{in}, ϕ_{fdm}, ϕ_{fdn}, and ϕ_{vco}, because the circuit is only sensitive to phase at the point where the noise is injected. The one exception is the noise produced by the PFD/CP, which in this case is considered to be a current, and denoted i_{det}.

Then the transfer functions from the various noise sources to the output are

$$G_{ref} = \frac{\phi_{out}}{\phi_{ref}} = \frac{G_{fwd}}{1+T} = \frac{N G_{fwd}}{N + G_{fwd}}, \tag{11}$$

$$G_{vco} = \frac{\phi_{out}}{\phi_{vco}} = \frac{1}{1+T} = \frac{N}{N + G_{fwd}}, \tag{12}$$

FIGURE 4 *Linear time-invariant phase-domain model of the synthesizer shown in Figure 2 with representative noise sources added. The ϕ's represent various sources of noise.*



$$G_{in} = \frac{\phi_{out}}{\phi_{in}} = \frac{1}{M}\frac{G_{fwd}}{1+T} = \frac{1}{M}\frac{NG_{fwd}}{N+G_{fwd}} = \frac{G_{ref}}{M}, \tag{13}$$

and by inspection,

$$G_{fdn} = \frac{\phi_{out}}{\phi_{fdn}} = -G_{ref}, \tag{14}$$

$$G_{fdm} = \frac{\phi_{out}}{\phi_{fdm}} = G_{ref}, \tag{15}$$

$$G_{det} = \frac{\phi_{out}}{i_{det}} = \frac{2\pi G_{ref}}{K_{det}}. \tag{16}$$

On this last transfer function, we have simply referred i_{det} to the input by dividing through by the gain of the phase detector.

These transfer functions allow certain overall characteristics of phase noise in PLLs to be identified. As $\omega \to \infty$, $G_{fwd} \to 0$ because of the VCO and the low-pass filter, and so G_{ref}, G_{det}, G_{fdm}, G_{fdn}, $G_{in} \to 0$ and $G_{vco} \to 1$. At high frequencies, the noise of the PLL is that of the VCO. Clearly this must be so because the low-pass LF blocks any feedback at high frequencies.

As $\omega \to 0$, $G_{fwd} \to \infty$ because of the $1/j\omega$ term from the VCO. So at DC, G_{ref}, G_{fdm}, $G_{fdn} \to N$, $G_{in} \to N/M$ and $G_{vco} \to 0$. At low frequencies, the noise of the PLL is contributed by the OSC, PFD/CP, FD$_M$ and FD$_N$, and the noise from the VCO is diminished by the gain of the loop.

Consider further the asymptotic behavior of the loop and the VCO noise at low offset frequencies $(\omega \to 0)$. Oscillator phase noise in the VCO results in the power spectral density $S_{\phi_{vco}}$ being proportional to $1/\omega^2$, or $S_{\phi_{vco}} \sim 1/\omega^2$ (neglecting flicker noise). If the LF is chosen such that $H(\omega) \sim 1$, then $G_{fwd} \sim 1/\omega$, and contribution from the VCO to the output noise power, $G_{vco}^2 S_{\phi_{vco}}$, is finite and nonzero. If the LF is chosen such that $H(\omega) \sim 1/\omega$, as it typically is when a true charge pump is employed, then $G_{fwd} \sim 1/\omega^2$ and the noise contribution to the output from the VCO goes to zero at low frequencies.

2.3 Noise Model

One predicts the phase noise exhibited by a PLL by building and applying the model shown in Figure 4. The first step in doing so is to find the various model parameters, including the level of the noise sources, which generally involves either direct measurement or simulating the various blocks with an RF simulator, such as SpectreRF. Use periodic noise (or PNoise) analysis to predict the output noise that results from stochastic noise sources contained within the blocks using simulation. Use a periodic AC or periodic transfer function (PAC or PXF) to compute the perturbation at the output of a block due to noise sources outside the block, such as on supplies.

Once the model parameters are known, it is simply a matter of computing the output phase noise of the PLL by applying the equations in Section 2.2 to compute the contributions to ϕ_{out} from every source and summing the results. Be careful to account for correlations in the noise sources. If the noise sources are perfectly correlated, as they might be if the ultimate source of noise is in the supplies or substrate, then use a direct sum. If the sources produce completely uncorrelated noise, as they would when the ultimate source of noise is random processes within the devices, use a root-mean-square sum.

Alternatively, one could build a Verilog-A model and use simulation to determine the result. The top-level of such a model is shown in Listing 1. It employs noisy phase-domain models for each of the blocks. These models are given in Listings 3-7 and are described in detail in the next few sections (3-6). In this example, the noise sources are coded into the models, but the noise parameters are not set at the top level to simplify the model. To predict the phase noise performance of the loop in lock, simply specify these parameters in the block models (given later) along with the parameters of Listing 1 and perform a noise analysis. To determine the effect of injected noise, first refer the noise to the output of one of the blocks, and then add a source into the netlist of Listing 1 at the appropriate place and perform an AC analysis.

Listings 1 and 3-7 have phase signals, and there is no phase discipline in the standard set of disciplines provided by Verilog-A or Verilog-AMS in *disciplines.vams*. There are several different resolutions for this problem. Probably the best solution is to simply add such a discipline, given in Listing 2, either to *disciplines.vams* as assumed here or to a separate file that is included as needed. Alternatively, one could use the *rotational* discipline. It is a conservative discipline that includes torque as a flow nature, and so is overkill in this situation. Finally, one could simply use either the electrical or the voltage discipline. Scaling for voltage in volts and phase in radians is similar, and so it will work fine except that the units will be reported incorrectly. Using the rotational discipline would require that all references to the phase discipline be changed to rotational in the appropriate listings. Using either the electrical or voltage discipline would require that both the name of the disciplines be changed from phase to either electrical or voltage, and the name of the access functions be changed from *Theta* to *V*.

3 Oscillators

Oscillators are responsible for most of the noise at the output of the majority of well-designed frequency synthesizers. This is because oscillators inherently tend to amplify noise found near their oscillation frequency and any of its harmonics. The reason for

LISTING 1 *Phase-domain model for a PLL configured as a frequency synthesizer.*

```
`include "disciplines.vams"

module pll(out);
output out;
phase out;
parameter integer m = 1 from [1:inf);   // input divide ratio
parameter real Kdet = 1 from (0:inf);   // phase detector gain
parameter real Kvco = 1 from (0:inf);   // VCO gain
parameter real c1 = 1n from (0:inf);    // Loop filter C1
parameter real c2 = 200p from (0:inf);  // Loop filter C2
parameter real r = 10K from (0:inf);    // Loop filter R
parameter integer n = 1 from [1:inf);   // feedback divide ratio
phase in, ref, fb;
electrical c;

oscillator OSC(in);
divider #(.ratio(m)) FDm(in, ref);
phaseDetector #(.gain(Kdet)) PD(ref, fb, c);
loopFilter #(.c1(c1), .c2(c2), .r(r)) LF(c);
vco #(.gain(Kvco)) VCO(c, out);
divider #(.ratio(n)) FDn(out, fb);

endmodule
```

LISTING 2 *Signal flow discipline definition for phase signals (the nature Angle is defined in disciplines.vams). This definition is assumed to reside in a file named "phase.vams".*

```
`include "disciplines.vams"

discipline phase
    potential Angle;
enddiscipline
```

this behavior is covered next, followed by a description of how to characterize and model the noise in an oscillator. The origins of oscillator phase noise are described in a conceptual way here. For a detailed description, see the papers by Käertner or Demir et al [7,14,15].

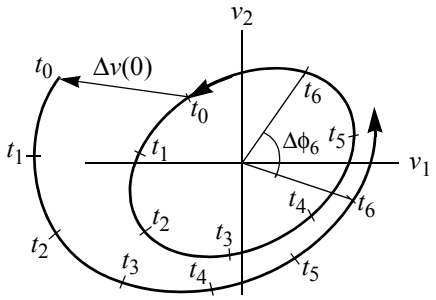
3.1 Oscillator Phase Noise

Nonlinear oscillators naturally produce high levels of phase noise. To see why, consider the trajectory of a fully autonomous oscillator's stable periodic orbit in state space. In steady state, the trajectory is a stable limit cycle, *v*. Now consider perturbing the oscillator with an impulse and assume that the deviation in the response due to the perturbation is Δv, as shown in Figure 5. Separate Δv into amplitude and phase variations,

$$\Delta v(t) = [1 + \alpha(t)]v\left(t + \frac{\phi(t)}{2\pi f_o}\right) - v(t)\,. \tag{17}$$

where *v* represents the unperturbed *T*-periodic output voltage of the oscillator, α represents the variation in amplitude, ϕ is the variation in phase, and $f_o = 1/T$ is the oscillation frequency.

FIGURE 5 *The trajectory of an oscillator shown in state space with and without a perturbation Δv. By observing the time stamps (t_0, ..., t_6) one can see that the deviation in amplitude dissipates while the deviation in phase does not.*



Since the oscillator is stable and the duration of the disturbance is finite, the deviation in amplitude eventually decays away and the oscillator returns to its stable orbit ($\alpha(t) \to 0$ as $t \to \infty$). In effect, there is a restoring force that tends to act against amplitude noise. This restoring force is a natural consequence of the nonlinear nature of the oscillator that acts to suppresses amplitude variations.

The oscillator is autonomous, and so any time-shifted version of the solution is also a solution. Once the phase has shifted due to a perturbation, the oscillator continues on as if never disturbed except for the shift in the phase of the oscillation. There is no restoring force on the phase and so phase deviations accumulate. A single perturbation causes the phase to permanently shift ($\phi(t) \to \Delta\phi$ as $t \to \infty$). If we neglect any short term time constants, it can be inferred that the impulse response of the phase deviation $\phi(t)$ can be approximated with a unit step $s(t)$. The phase shift over time for an arbitrary input disturbance u is

$$\phi(t) \sim \int_{-\infty}^{\infty} s(t-\tau)u(\tau)d\tau = \int_{-\infty}^{t} u(\tau)d\tau, \tag{18}$$

or the power spectral density (PSD) of the phase is

$$S_\phi(\Delta f) \sim \frac{S_u(\Delta f)}{(2\pi\Delta f)^2} \tag{19}$$

This shows that in all oscillators the response to any form of perturbation, including noise, is amplified and appears mainly in the phase. The amplification increases as the frequency of the perturbation approaches the frequency of oscillation in proportion to $1/\Delta f$ (or $1/\Delta f^2$ in power).

Notice that there is only one degree of freedom — the phase of the oscillator as a whole. There is no restoring force when the phase of all signals associated with the oscillator shift together, however there would be a restoring force if the phase of signals shifted relative to each other. This observation is significant in oscillators with multiple outputs, such as quadrature or ring oscillators. The dominant phase variations appear identically in all outputs, whereas relative phase variations between the outputs are naturally suppressed by the oscillator or added by subsequent circuitry and so tend to be much smaller [6].

3.2 Characterizing Oscillator Phase Noise

Above it was shown that oscillators tend to convert perturbations from any source into a phase variation at their output with an amplification that varies with $1/\Delta f$ (or $1/\Delta f^2$ in power). Now assume that the perturbation is from device noise in the form of white and flicker stochastic processes. The oscillator's response will be characterized first in terms of the phase noise S_ϕ, and then because phase noise is not easily measured, in terms of the normalized single-sideband noise power L. The result will be a small set of easily extracted parameters that completely describe the response of the oscillator to white and flicker noise sources. These parameters are used when modeling the oscillator.

Assume that the perturbation consists of white and flicker noise and so has the form

$$S_u(\Delta f) \sim 1 + \frac{f_c}{\Delta f}. \tag{20}$$

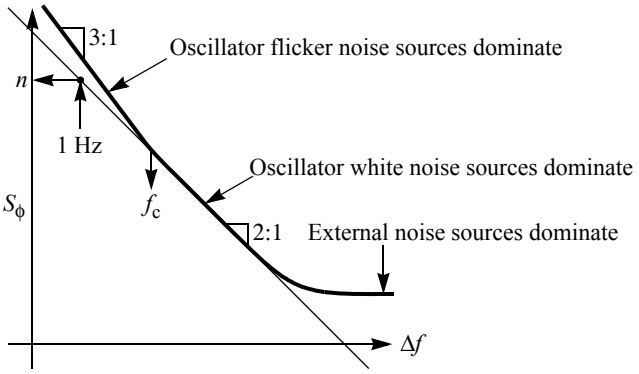
Then from (19) the response will take the form

$$S_\phi(\Delta f) = n\left(\frac{1}{\Delta f^2} + \frac{f_c}{\Delta f^3}\right), \tag{21}$$

where the factor of $(2\pi)^2$ in the denominator of (19) has been absorbed into the constant of proportionality n and S_ϕ is chosen to be the single-sided PSD[4]. Thus, the response of the oscillator to white and flicker noise sources is characterized using just two parameters, n and f_c, where n is the portion of S_ϕ attributable to the white noise sources alone at $\Delta f = 1$ Hz and f_c is the flicker noise corner frequency.

As shown in Figure 6, n is extracted by simply extrapolating to 1 Hz from a frequency where the noise from the white sources dominates.

FIGURE 6 *Extracting the noise parameters, n and f_c, for an oscillator from the single-sided power spectral density of its phase noise. The graph is plotted on a log-log scale.*



S_ϕ is not directly observable and often difficult to find. So instead oscillator phase noise is often characterized using L, the spot noise power of the output voltage S_v normalized

4. A single-sided PSD has a domain of $0 \le f < \infty$. The double-sided PSD is also commonly used and it has a domain of $-\infty < f < \infty$. They are related in that if S_{DS} and S_{SS} are the double- and single-sided PSDs of a signal, then $S_{DS}(0) = S_{SS}(0)$ and $S_{DS}(f) = \frac{1}{2}S_{SS}(f)$ for $f \neq 0$.

by the power in the fundamental tone. S_v is directly available from either measurement with a spectrum analyzer or from RF simulators, and L is defined as

$$L(\Delta f) = \frac{2}{a_1^2 + b_1^2} S_v(f_o + \Delta f), \tag{22}$$

where S_v is the single-sided PSD normalized to 1 V RMS[5] and a_1 and b_1 are the Fourier coefficients for the fundamental frequency components of v, the noise-free output signal. They satisfy

$$v(t) = \sum_{k=0}^{\infty} a_k \cos(2\pi k f_o t) + b_k \sin(2\pi k f_o t). \tag{23}$$

In (41) of [7], Demir et al shows that for a free-running oscillator perturbed only by white noise sources

$$L(\Delta f) = \frac{c f_o^2}{c^2 f_o^4 \pi^2 + \Delta f^2}, \tag{24}$$

which is a Lorentzian process with corner frequency of

$$f_\Delta = c f_o^2 \pi. \tag{25}$$

The corner frequency is also known as the linewidth of the oscillator. At frequencies well above f_Δ and well below f_o,

$$L(\Delta f) = \frac{c f_o^2}{\Delta f^2}. \tag{26}$$

Over this same range of frequencies, Vendelin [32] showed that

$$L(\Delta f) = \frac{1}{2} S_\phi(\Delta f). \tag{27}$$

The empirical constants c and n are related by using this equation to combine (21) and (26).

$$\frac{c f_o^2}{\Delta f^2} = \frac{n}{2}\left(\frac{1}{\Delta f^2} + \frac{f_c}{\Delta f^3}\right) \tag{28}$$

This equation is valid only for $\Delta f \gg f_c$, and so the flicker noise term can be ignored, giving

5. While S_v is a power-spectral density, it is important to understand that the units are not W/Hz. In this case S_v has been normalized to 1 V RMS, meaning that the units are V^2/Hz (dBV/Hz if given in decibels) and that

$$P_{\text{total}} = \int_0^\infty S_v(f) df$$

is the total RMS power that would be dissipated if the signal were applied to a 1Ω resistor.

$$n = 2cf_o^2 . \quad (29)$$

The parameters n, or alternatively c, and f_c are used to describe the noise behavior of an oscillator. To extract these parameters, start by measuring either L or S_ϕ for a range of frequencies offset from the center frequency. If flicker noise is present, there will be a range of low frequencies for which the noise power drops at a rate of 30 dB per decade. Above this, the rate of drop will be 20 dB per decade. As shown in Figure 6, the frequency at which the rate switches from 30 dB to 20 dB per decade is f_c. Choose a frequency Δf well above f_c in the region where the noise is dropping at a rate of 20 dB per decade and use either (21) or (26) and (29) to determine n.

3.3 Phase-Domain Models for the Oscillators

The phase-domain models for the reference and voltage-controlled oscillators are given in Listings 3 and 4. The VCO model is based on (6). Perhaps the only thing that needs to be explained is the way that phase noise is modeled in the oscillators. Verilog-AMS provides the *flicker_noise* function for modeling flicker noise, which has a power spectral density proportional to $1/f^\alpha$ with α typically being close to 1. However, Verilog-AMS does not limit α to being close to one, making this function well suited to modeling oscillator phase noise, for which α is 2 in the white-phase noise region and close to 3 in the flicker-phase noise region (at frequencies below the flicker noise corner frequency). Alternatively, one could dispense with the noise parameters and use the *noise_table* function in lieu of the *flicker_noise* functions to use the measured noise results directly. The "wpn" and "fpn" strings passed to the noise functions are labels for the noise

LISTING 3 *Phase-domain oscillator noise model.*

```
`include "phase.vams"                    // from Listing 2, includes disciplines.vams.

module oscillator(out);
output out;
phase out;
parameter real n = 0 from [0:inf);       // white output phase noise at 1 Hz (rad^2/Hz)
parameter real fc = 0 from [0:inf);      // flicker noise corner frequency (Hz)

analog begin
    Theta(out) <+ flicker_noise(n, 2, "wpn") + flicker_noise(n*fc, 3, "fpn");
end
endmodule
```

sources. They are optional and can be chosen arbitrarily, though they should not contain any white space or special characters. *wpn* was chosen to represent white phase noise and *fpn* stands for flicker phase noise.

When interested in the effect of signals coupled into the oscillator through the supplies or the substrate, one would compute the transfer function from the interfering source to the phase output of the oscillator using either a PAC or PXF analysis. Again, one would simply assume that the perturbation in the output of the oscillator is completely in the phase, which is true except at very high offset frequencies. One then employs (12) and (13) to predict the response at the output of the PLL.

LISTING 4 *Phase-domain VCO noise model.*

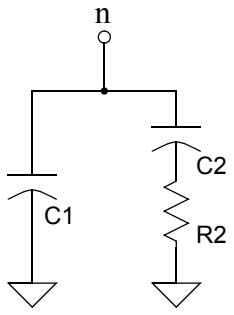
```
`include "phase.vams"                    // from Listing 2, includes disciplines.vams.
`include "constants.vams"

module vco(in, out);
input in; output out;
voltage in;
phase out;
parameter real gain = 1 from (0:inf);    // transfer gain, Kvco (Hz/V)
parameter real n = 0 from [0:inf);       // white output phase noise at 1 Hz (rad^2/Hz)
parameter real fc = 0 from [0:inf);      // flicker noise corner frequency (Hz)

analog begin
    Theta(out) <+ 2*`M_PI*gain*idt(V(in));
    Theta(out) <+ flicker_noise(n, 2, "wpn") + flicker_noise(n*fc, 3, "fpn");
end
endmodule
```

4 Loop Filter

Even in the phase-domain model for the PLL, the loop filter remains in the voltage domain and is represented with a full circuit-level model, as shown in Listing 5. As such, the noise behavior of the filter is naturally included in the phase-domain model without any special effort assuming that the noise is properly included in the resistor model.

LISTING 5 *Loop filter model.*

```
`include "disciplines.vams"

module loopFilter(n);
electrical n, gnd;
ground gnd;†
parameter real c1 = 1n from (0:inf);
parameter real c2 = 200p from (0:inf);
parameter real r = 10K from (0:inf);
electrical int;

capacitor #(.c(c1)) C1(n, gnd);
capacitor #(.c(c2)) C2(n, int);
resistor #(.r(r)) R(int, gnd);

endmodule
```



† The *ground* statement was not previously supported in Cadence's Verilog-A implementation, so instead ground should be explicitly passed into the module through a terminal.

5 Phase Detector and Charge Pump

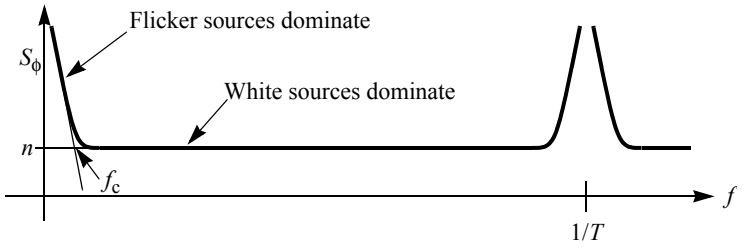
As with the VCO, the noise of the PFD/CP as needed by the phase-domain model is found directly with simulation. Simply drive the block with a representative periodic signal, perform a PNoise analysis, and measure the output noise current. In this case, a

representative signal would be one that produced periodic switching in the PFD and CP. This is necessary to capture the noise present during the switching process.

The noise in the PFD/CP comes from jitter in the PFD and noise in the output current of the CP. The total noise produced by the CP will be proportional to how long it is on, while the noise from the PFD will all be in the edges and so will be independent of how long the CP is on. The CP itself has two current sources connected to its output, one that pulls up and one that pulls down. Which one is activated depends on whether the edges on the reference input lead or lag those on the feedback input. The time for which the current source stays on is equal to the difference in arrival times for the edges. Most PLLs operate in steady-state with the edges on the reference and feedback input occurring almost simultaneously (because the loop gain of the PLL is infinite at DC). In addition, most phase detectors are what is known as 'live zone' phase detectors. With these, when edges occur simultaneously on the reference and feedback inputs, both the pull up and pull down current sources will turn on for a very short period of time. From an output current perspective the pull up and pull down currents will act to cancel each other and so the effective output current is zero, however both current sources will be contributing uncorrelated noise to the output while they are on. Thus it is best to characterize the noise of the PFD/CP with simultaneous edges occurring on both the reference and feedback inputs. The output should be connected to a current probe (often in the form of an ideal voltage source) that is biased to present the expected voltage to the output of the CP.

Generally the power spectral density of the output noise current appears as in Figure 7, in which case the noise is parameterized with n and f_c. n is the noise power density at frequencies above the flicker noise corner frequency, f_c, and below the noise bandwidth of the circuit.

FIGURE 7 *Extracting the noise parameters, n and f_c, for the PFD/CP from the power spectral density of its output noise current. The graph is plotted on a log-log scale.*



The phase-domain model for the PFD/CP is given in Listing 6. It is based on (3). Alternatively, as before one could use the *noise_table* function in lieu of the *white_noise* and *flicker_noise* functions to use the measured noise results directly.

6 Frequency Dividers

There are several reasons why the process of extracting the noise produced by the frequency dividers is more complicated than that needed for other blocks. First, the phase noise is needed and, as of the time when this document was written, SpectreRF reports on the total noise and does not yet make the phase noise available separately. Secondly,

LISTING 6 *Phase-domain phase detector noise model.*

```
`include "phase.vams"                    // from Listing 2, includes disciplines.vams.
`include "constants.vams"

module phaseDetector(pin, nin, out);
input pin, nin; output out;
phase pin, nin;
electrical out;
parameter real gain = 1 from (0:inf);    // transfer gain (A/cycle)
parameter real n = 0 from [0:inf);       // white output current noise (A^2/Hz)
parameter real fc = 0 from [0:inf);      // flicker noise corner frequency (Hz)

analog begin
    I(out) <+ –gain ∗ Theta(pin,nin) / (2∗`M_PI);
    I(out) <+ white_noise(n, "wpn") + flicker_noise(n∗fc, 1, "fpn");
end
endmodule
```

the frequency dividers are always followed by some form of edge-sensitive thresholding circuit, in this case the PFD, which implies that the overall noise behavior of the PLL is only influenced by the noise produced by the divider at the time when the threshold is being crossed in the proper direction. The noise produced by the frequency divider is cyclostationary, meaning that the noise power varies over time. Thus, it is important to analyze the noise behavior of the divider carefully. The second issue is discussed first.

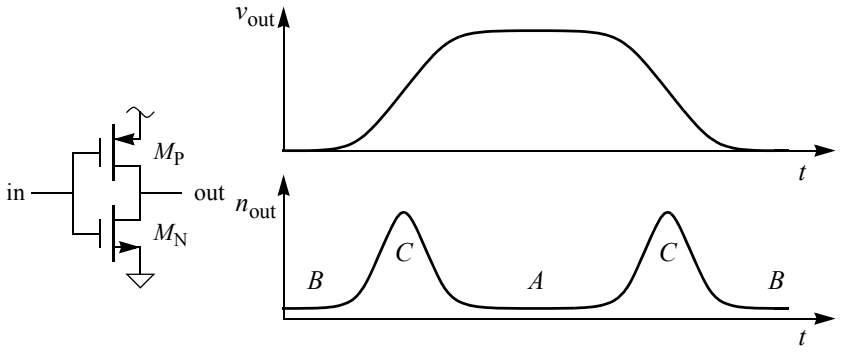
6.1 Cyclostationary Noise.

Formally, the term cyclostationary implies that the autocorrelation function of a stochastic process varies with t in a periodic fashion [9,28], which in practice is associated with a periodic variation in the noise power of a signal. In general, the noise produced by all of the nonlinear blocks in a PLL is strongly cyclostationary. To understand why, consider the noise produced by a logic circuit, such as the inverter shown in Figure 8. The noise at the output of the inverter, n_{out}, comes from different sources depending on the phase of the output signal, v_{out}. When the output is high, the output is insensitive to small changes on the input. The transistor M_P is on and the noise at the output is predominantly due to the thermal noise from its channel. This is region A in the figure. When the output is low, the situation is reversed and most of the output noise is due to the thermal noise from the channel of M_N. This is region B. When the output is transitioning, thermal noise from both M_P and M_N contribute to the output. In addition, the output is sensitive to small changes in the input. In fact, any noise at the input is amplified before reaching the output. Thus, noise from the input tends to dominate over the thermal noise from the channels of M_P and M_N in this region. Noise at the input includes noise from the previous stage and noise from both devices in the form of flicker noise and thermal noise from gate resistance. This is region C in the figure.

The challenge in estimating the effect of noise passing through a threshold is the difficulty in estimating the noise at the point where the threshold is crossed. There are several different ways of estimating the effect of this noise, but the simplest is to use the strobed noise feature of SpectreRF.[6] When the strobed noise feature is active, the noise

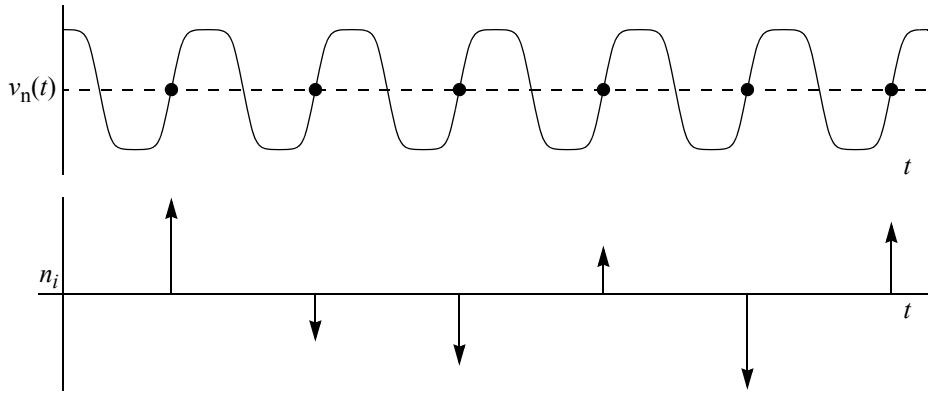
6. The strobed-noise feature of SpectreRF is also referred to as its time-domain noise feature.

FIGURE 8 *Noise produced by an inverter (n_{out}) as a function of the output signal (v_{out}). In region A the noise is dominated by the thermal noise of M_P, in region B its dominated by the thermal noise of M_N, and in region C the output noise includes the thermal noise from both devices as well as the amplified noise from the input.*



produced by the circuit is periodically sampled to create a discrete-time random sequence, as shown in Figure 9. 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 9 *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.*



6.2 Converting to Phase Noise

The act of converting the noise from a continuous-time process to a discrete-time process by sampling at the threshold crossings makes the conversion into phase noise easier. If v_n is the continuous-time noisy response, and v is the noise-free response (response with the noise sources turned off), then[7]

$$n_i = v_n(iT) - v(iT). \tag{30}$$

Then if v_n is noisy because it is corrupted with a phase noise process ϕ, then

$$v_{\mathrm{n}}(t) = v\left(t + \frac{\phi(t)}{2\pi f_0}\right). \tag{31}$$

Assume the phase noise ϕ is small and linearize v using a Taylor series expansion

$$v_{\mathrm{n}}(t) \cong v(t) + \frac{dv(t)}{dt}\frac{\phi(t)}{2\pi f_0} \tag{32}$$

and

$$n_i \cong v(iT) + \frac{dv(iT)}{dt}\frac{\phi(iT)}{2\pi f_0} - v(iT) = \frac{dv(iT)}{dt}\frac{\phi(iT)}{2\pi f_0}. \tag{33}$$

Finally, ϕ_i can be found from n_i using

$$\phi_i = 2\pi f_0 n_i \Big/ \frac{dv(iT)}{dt}. \tag{34}$$

v is T periodic, which makes $dv(iT)/dt$ a constant, and so

$$S_\phi(f) = \left[2\pi f_0 \Big/ \frac{dv(iT)}{dt}\right]^2 S_n(f). \tag{35}$$

where $S_n(f)$ and $S_\phi(f)$ are the power spectral densities of the n_i and ϕ_i sequences.

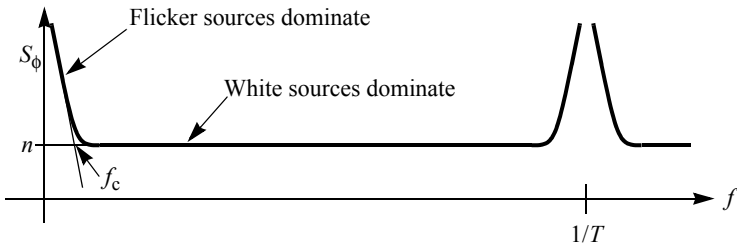
6.3 Phase-Domain Model for Dividers

To extract the phase noise 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 $S_n(f)$. 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. $S_\phi(f)$ is then computed from (35). Figure 10 shows the various attributes of the phase noise at the output of the divider. Notice that the noise is periodic in f with period $1/T$ because n is a discrete-time sequence with period T. The parameters n and f_{c} for the divider are extracted as illustrated.

With ripple counters, one usually only characterizes one stage at a time and combines the phase noise from each stage by assuming that the noise in each stage is independent (true for device noise, would not be true for noise coupling into the divider from external sources). The variation due to phase noise accumulates, however it is necessary to account for the increasing period of the signals at each stage along the ripple counter. Consider an intermediate stage of a K-stage ripple counter. The total phase noise at the output of the ripple counter that results due to the phase noise S_{ϕ_k} at the output of stage k is $(T_k/T_K)^2 S_{\phi_k}$. So the total phase noise at the output of the ripple counter is

7. It is assumed that the sequence n_i is formed by sampling the noise at iT, which implies that the threshold crossings also occur at iT. In practice, the crossings will occur at some time offset from iT. That offset is ignored. It is done without loss of generality with the understanding that the functions v and v_{n} can always be reformulated to account for the offset.

FIGURE 10 *Extracting the noise parameters, n and f_c, for the divider from the power spectral density of its phase noise. The graph is plotted on a log-log scale.*



$$S_{\phi_{\text{out}}} = \frac{1}{T_K^2} \sum_{k=0}^{K} T_k^2 S_{\phi_k} \tag{36}$$

where S_{ϕ_0} and T_0 are the phase noise and signal period at the input to the first stage of the ripple counter.

With undesired variations in the supplies or in the substrate the resulting phase noise in each stage would be correlated, so one would need to compute the transfer function from the signal source to the phase noise of each stage and combine in a vector sum.

Unlike in ripple counters, phase noise does not accumulate with each stage in synchronous counters. Phase noise at the output of a synchronous counter is independent of the number of stages and consists only of the noise of its clock along with the noise of the last stage.

The phase-domain model for the divider, based on (2), is given in Listing 7. As before, one could use the *noise_table* function in lieu of the *white_noise* and *flicker_noise* functions to use the measured noise results directly.

LISTING 7 *Phase-domain divider noise model.*

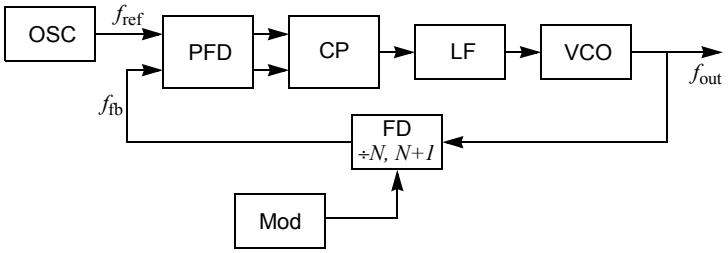
```
`include "phase.vams"                    // from Listing 2, includes disciplines.vams.

module divider(in, out);
input in; output out;
phase in, out;
parameter real ratio = 1 from (0:inf);   // divide ratio
parameter real n = 0 from [0:inf);       // white output phase noise (rads^2/Hz)
parameter real fc = 0 from [0:inf);      // flicker noise corner frequency (Hz)

analog begin
    Theta(out) <+ Theta(in) / ratio;
    Theta(out) <+ white_noise(n, "wpn") + flicker_noise(n*fc, 1, "fpn");
end
endmodule
```

7 Fractional-*N* Synthesis

One of the drawbacks of a traditional frequency synthesizer, also known as an integer-*N* frequency synthesizer, is that the output frequency is constrained to be *N* times the reference frequency. If the output frequency is to be adjusted by changing *N*, which is constrained by the divider to be an integer, then the output frequency resolution is equal to the reference frequency. If fine frequency resolution is desired, then the reference frequency must be small. This in turn limits the loop bandwidth as set by the loop filter, which must be at least 10 times smaller than the reference frequency to prevent signal components at the reference frequency from reaching the input of the VCO and modulating the output frequency, creating spurs or sidebands at an offset equal to the reference frequency and its harmonics. A low loop bandwidth is undesirable because it limits the response time of the synthesizer to changes in *N*. In addition, the loop acts to suppress the phase noise in the VCO at offset frequencies within its bandwidth, so reducing the loop bandwidth acts to increase the total phase noise at the output of the VCO.

The constraint on the loop bandwidth imposed by the required frequency resolution is eliminated if the divide ratio *N* is not limited to be an integer. This is the idea behind fractional-*N* synthesis. In practice, one cannot directly implement a frequency divider that implements non-integer divide ratio except in a few very restrictive cases, so instead a divider that is capable of switching between two integer divide ratios is used, and one rapidly alternates between the two values in such a way that the time-average is equal to the desired non-integer divide ratio [31]. A block diagram for a fractional-*N* synthesizer is shown in Figure 11. Divide ratios of *N* and *N* + 1 are used, where *N* is the first integer below the desired divide ratio, and *N* + 1 is the first integer above. For example, if the desired divide ratio is 16.25, then one would alternate between the ratios of 16 and 17, with the ratio of 16 being used 75% of the time. Early attempts at fractional-*N* synthesis alternated between integer divide ratios in a repetitive manner, which resulted in noticeable spurs in the VCO output spectrum. More recently, ΔΣ modulators have been used to generate a random sequence with the desired duty cycle to control the multi-modulus dividers [30]. This has the effect of trading off the spurs for an increased noise floor, however the ΔΣ modulator can be designed so that most of the power in its output sequence is at frequencies that are above the loop bandwidth, and so are largely rejected by the loop.

FIGURE 11 *The block diagram of a fractional-N frequency synthesizer.*



The phase-domain small-signal model for the combination of a fractional-*N* divider and a ΔΣ modulator is given in Listing 8. It uses the *noise_table* function to construct a simple piece-wise linear approximation of the noise produced in an n^{th} order ΔΣ modulator

that is parameterized with the low frequency noise generated by the modulator, along with the corner frequency and the order.

LISTING 8 *Phase-domain fractional-N divider model.*

```
`include "phase.vams"                    // from Listing 2, includes disciplines.vams.

module divider(in, out);
input in; output out;
phase in, out;
parameter real ratio = 1 from (0:inf);          // divide ratio
parameter real n = 0 from [0:inf);              // white output phase noise (rads^2/Hz)
parameter real bw = 1 from (0:inf);             // ΔΣ modulator bandwidth
parameter integer order = 1 from (0:9);         // ΔΣ modulator order
parameter real fmax = 10*bw from (bw:inf);// maximum frequency of concern

analog begin
    Theta(out) <+ Theta(in) / ratio;
    Theta(out) <+ noise_table({
                    0,      n,
                    bw,     n,
                    fmax,   n*pow((fmax/bw), 2*order)
                }, "dsn"));
end
endmodule
```

8 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

$$v_{\mathrm{n}}(t) = v(t + j(t)) \tag{37}$$

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 is phase, or phase noise, using

$$\phi(t) = 2\pi f_{\mathrm{o}} j(t), \tag{38}$$

where $f_{\mathrm{o}} = 1/T$ and

$$v_{\mathrm{n}}(t) = v\left(t + \frac{\phi(t)}{2\pi f_{\mathrm{o}}}\right). \tag{39}$$

8.1 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.[8]

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 RMS jitter J_{ee} is

$$J_{ee}(i) = \sqrt{\text{var}(t_i)}\,. \tag{40}$$

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 $t_{i+k} - t_i$,

$$J_k(i) = \sqrt{\text{var}(t_{i+k} - t_i)}\,. \tag{41}$$

$J_k(i)$ is referred to as *k-cycle jitter* or *long-term jitter*[9]. 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

$$J_{cc}(i) = \sqrt{\text{var}(T_{i+1} - T_i)}\,. \tag{42}$$

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 [13]. 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.

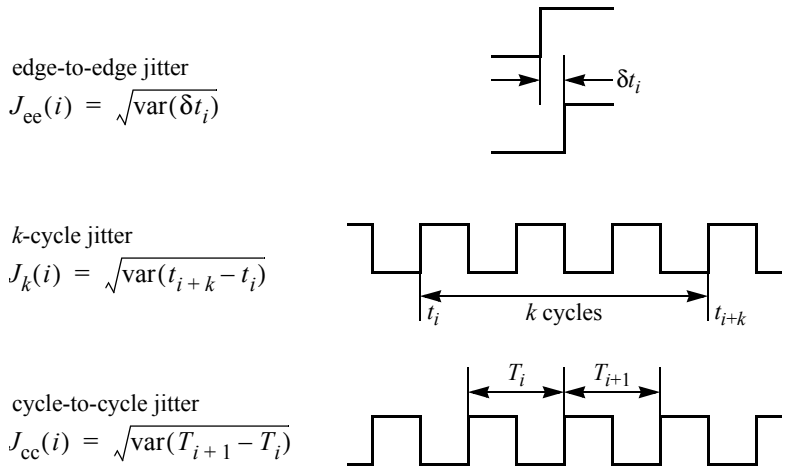
If $j(t)$ is either stationary or T-cyclostationary, then the sequence $\{t_i\}$ is stationary, meaning that these metrics do not vary with i, and so $J_{ee}(i)$, $J_k(i)$, and $J_{cc}(i)$ can be shortened to J_{ee}, J_k, and J_{cc}.

8. 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 [22].

9. 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 $k \to \infty$.

These jitter metrics are illustrated in Figure 12.

FIGURE 12 *The various jitter metrics.*



8.1.1 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* [24]. 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

$$J_{PP} = \alpha J_{RMS}. \tag{43}$$

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

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^{-13}	14.698
10^{-14}	15.301
10^{-15}	15.883
10^{-16}	16.444

8.2 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)	$J_{ee} = \frac{\sqrt{\mathrm{var}(n_v(t_c))}}{dv(t_c)/dt}$
accumulating	autonomous (OSC, VCO)	$J = \sqrt{cT}$

9 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

$$j_{\mathrm{sync}}(t) = \eta(t) \tag{44}$$

$$v_{\mathrm{n}}(t) = v(t + j_{\mathrm{sync}}(t)) \tag{45}$$

exhibits synchronous jitter. If η is further restricted to be a white Gaussian stationary or *T*-cyclostationary process, then $v_{\mathrm{n}}(t)$ 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 (38) 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 (40),

$$J_{\mathrm{ee}}(i) = \sqrt{\mathrm{var}(j_{\mathrm{sync}}(t_i))}\,. \tag{46}$$

Similarly, from (41), *k* cycle jitter is

$$J_k(i) = \sqrt{\mathrm{var}(t_{i+k} - t_i)}\,, \tag{47}$$

$$J_k(i) = \sqrt{\mathrm{var}([(i+k)T + j_{\mathrm{sync}}(t_{i+k})] - [iT + j_{\mathrm{sync}}(t_i)])}\,, \tag{48}$$

$$J_k(i) = \sqrt{\mathrm{var}(j_{\mathrm{sync}}(t_{i+k})) + \mathrm{var}(j_{\mathrm{sync}}(t_i))}\,. \tag{49}$$

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

$$J_k(i) = \sqrt{2\mathrm{var}(j_{\mathrm{sync}})}\,, \text{ and} \tag{50}$$

$$J_k(i) = \sqrt{2}J_{\mathrm{ee}}\,. \tag{51}$$

The factor of $\sqrt{2}$ in (51) stems from the length of an interval including the independent variation from two transitions. From (51), J_k is independent of *k*, and so

$$J_k = J \text{ for } k = 1, 2, \ldots\,. \tag{52}$$

The approach is similar for cycle-to-cycle jitter. From (42),

$$J_{\mathrm{cc}}(i) = \sqrt{\mathrm{var}(T_{i+1} - T_i)} \tag{53}$$

$$J_{\mathrm{cc}}(i) = \sqrt{\mathrm{var}([t_{i+1} - t_i] - [t_i - t_{i-1}])} \tag{54}$$

$$J_{\mathrm{cc}}(i) = \sqrt{\mathrm{var}(t_{i+1} - 2t_i + t_{i-1})} \tag{55}$$

$$J_{\mathrm{cc}}(i) = \sqrt{\mathrm{var}([(i+k)T + j_{\mathrm{sync}}(t_{i+k})] - 2[iT + j_{\mathrm{sync}}(t_i)] + [(i-k)T + j_{\mathrm{sync}}(t_{i-k})])} \tag{56}$$

$$J_{\mathrm{cc}}(i) = \sqrt{\mathrm{var}(j_{\mathrm{sync}}(t_{i+k})) + \mathrm{var}(2j_{\mathrm{sync}}(t_i)) + \mathrm{var}(j_{\mathrm{sync}}(t_{i-k}))} \tag{57}$$

$$J_{\mathrm{cc}}(i) = \sqrt{\mathrm{var}(j_{\mathrm{sync}}(t_{i+k})) + 4\mathrm{var}(j_{\mathrm{sync}}(t_i)) + \mathrm{var}(j_{\mathrm{sync}}(t_{i-k}))} \tag{58}$$

$$J_{\mathrm{cc}}(i) = \sqrt{6\mathrm{var}(j_{\mathrm{sync}})} \tag{59}$$

$$J_{\mathrm{cc}}(i) = \sqrt{6}J_{\mathrm{ee}} \tag{60}$$

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

9.1 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.

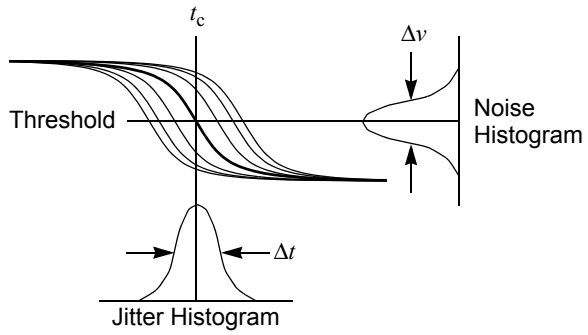
In systems where signals are continuous valued, an event is usually defined as a signal crossing a threshold in a particular direction. The threshold crossings of a noiseless periodic signal, $v(t)$, are precisely evenly spaced. However, when noise is added to the signal, $v_n(t) = v(t) + n_v(t)$, each 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 13 [35]. If the noise n_v is stationary, then

$$\mathrm{var}(j_{\mathrm{sync}}(t_c)) \cong \frac{\mathrm{var}(n_v)}{[dv(t_c)/dt]^2} \tag{61}$$

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

FIGURE 13 *How a threshold converts noise into jitter.*



Generally n_v is not stationary, but cyclostationary (refer back to Section 6.1). 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,

$$\mathrm{var}(j_{\mathrm{sync}}(t_c)) = \frac{\mathrm{var}(n_v(t_c))}{[dv(t_c)/dt]^2}. \tag{62}$$

The jitter is computed from (46) using (61) or (62),

$$J_{\mathrm{ee}} = \frac{\sqrt{\mathrm{var}(n_v(t_c))}}{dv(t_c)/dt}. \tag{63}$$

To compute $\mathrm{var}(n_v(t_c))$, 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 $f = -f_o/2$ to $f_o/2$. Recall that the noise is periodic in f with period $f_o = 1/T$ because n is a discrete-time sequence with rate T.

In practice, this is done by using the strobed noise capability of SpectreRF[10] 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 9. 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.

9.1.1 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,

$$\mathrm{var}(n_v(t_c)) = \int_0^{f_o/2} S_{n_v}(f, t_c)\,df \,. \tag{64}$$

Then J_{ee} is computed from (63).

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.

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.

9.1.2 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.

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

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 [27] to determine

$$\mathrm{var}(n) = \int_0^\infty S_n(f)df, \tag{65}$$

the total output noise current squared [9]. 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,

$$J_{\mathrm{ee}_{\mathrm{PFD/CP}}} = \frac{T}{K_{\mathrm{det}}}\sqrt{\frac{\mathrm{var}(n)}{2}}. \tag{66}$$

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.

10 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

$$j_{\mathrm{acc}}(t) = \int_0^t \eta(\tau)d\tau \tag{67}$$

$$v_{\mathrm{n}}(t) = v(t + j_{\mathrm{acc}}(t)) \tag{68}$$

exhibits accumulating jitter. While η is cyclostationary and so has bounded variance, (67) shows that the variance of j_{acc}, and hence the phase difference between $v(t)$ and $v_{\mathrm{n}}(t)$, 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 $\{j_{\mathrm{acc}}(iT)\}$ 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 [9]. 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.[11]

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,

$$J_k = \sqrt{k}J \text{ for } k = 0, 1, 2, \ldots, \tag{69}$$

where

$$J = \sqrt{\text{var}(j_{\text{acc}}(t_i + T) - j_{\text{acc}}(t_i))}\,. \tag{70}$$

Similarly,

$$J_{\text{cc}} = \sqrt{2}J\,. \tag{71}$$

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

10.1 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 (67) is a white Gaussian T-cyclostationary noise process (this excludes flicker noise) with a single-sided PSD of

$$S_\eta(f) = 2c\,, \tag{72}$$

and an autocorrelation function of

$$R_\eta(t_1, t_2) = c\delta(t_1 - t_2)\,, \tag{73}$$

where δ is a Dirac delta function. Then

11. 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.

$$j_{\text{acc}}(t) = \int_0^t \eta_T(\tau)d\tau \tag{74}$$

is a Wiener process [9], which has an autocorrelation function of

$$R_{j_{\text{acc}}}(t_1, t_2) = c\min(t_1, t_2)\,. \tag{75}$$

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

$$J^2 = \text{var}(j_{\text{acc}}(t+T) - j_{\text{acc}}(t))\,. \tag{76}$$

$$J^2 = \text{E}[(j_{\text{acc}}(t+T) - j_{\text{acc}}(t))^2] \tag{77}$$

$$J^2 = \text{E}[j_{\text{acc}}(t+T)^2 - 2j_{\text{acc}}(t+T)j_{\text{acc}}(t) + j_{\text{acc}}(t)^2] \tag{78}$$

$$J^2 = \text{E}[j_{\text{acc}}(t+T)^2] - 2\text{E}[j_{\text{acc}}(t+T)j_{\text{acc}}(t)] + \text{E}[j_{\text{acc}}(t)^2] \tag{79}$$

$$J^2 = R_{j_{\text{acc}}}(t+T, t+T) - 2R_{j_{\text{acc}}}(t+T, t) + R_{j_{\text{acc}}}(t, t) \tag{80}$$

$$J^2 = c(t+T) - 2ct + ct \tag{81}$$

$$J = \sqrt{cT}, \tag{82}$$

which agrees with Demir [7]. 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,

$$\phi_{\text{acc}}(t) = 2\pi f_{\text{o}} j_{\text{acc}}(t) = 2\pi f_{\text{o}} \int_0^t \eta(\tau)d\tau\,, \tag{83}$$

where $f_{\text{o}} = 1/T$. From (72) and (83) the PSD of ϕ_{acc} is

$$S_{\phi_{\text{acc}}}(\Delta f) = 2c\frac{(2\pi f_{\text{o}})^2}{(2\pi\Delta f)^2} = \frac{2cf_{\text{o}}^2}{\Delta f^2}\,. \tag{84}$$

From (27)

$$L(\Delta f) = \frac{1}{2}S_{\phi_{\text{acc}}}(\Delta f) = \frac{cf_{\text{o}}^2}{\Delta f^2}\,, \tag{85}$$

$$c = L(\Delta f)\frac{\Delta f^2}{f_{\text{o}}^2}\,, \tag{86}$$

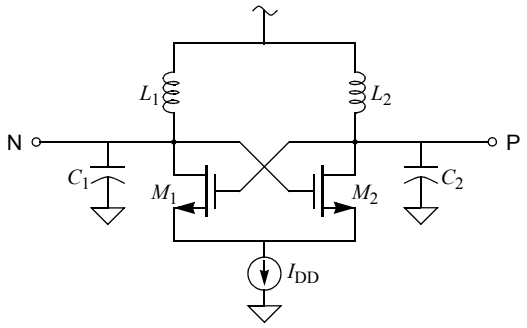
which, of course, is consistent with (26). Determine c by choosing Δf well above the corner frequency (f_{c}) to avoid ambiguity and well below f_{o} to avoid the noise from other sources that occur at these frequencies.

10.1.1 Example

To compute the jitter of an oscillator, an RF simulator such as SpectreRF is used to find L and f_{o} of the oscillator. Given these, c is found with (86), J is found with (82) and J_k is found with (69). This procedure is demonstrated for the oscillator shown in Figure 14.

This is a very low noise oscillator designed in 0.35µ CMOS by Rael and Abidi [29]. The frequency of oscillation is 1.1 GHz and the resonator has a loaded Q of 6.

FIGURE 14 *Differential LC oscillator.*



The procedure starts by using an RF simulator such as SpectreRF to compute the normalized phase noise L. Its PNoise analysis is used, with the *maxsideband* parameter set to at least 10 to adequately account for noise folding within the oscillator.[12] In this case, $L = -110$ dBc at 100 kHz offset from the carrier. Apply (86) to compute c from L,

$$c = \mathrm{L}(\Delta f)\frac{\Delta f^2}{f_\mathrm{o}^2} \tag{87}$$

where $L(\Delta f) = 10^{-11}$, $\Delta f = 100$ kHz, and $f_\mathrm{o} = 1.1$ GHz, which gives $c = 82.6 \times 10^{-21}$. The period jitter J is then computed from (82),

$$J = \sqrt{cT} = \sqrt{\frac{c}{f_\mathrm{o}}} = \sqrt{\frac{82.6 \times 10^{-21}}{1.1\ \mathrm{GHz}}} = 8.7\ \mathrm{fs}\,. \tag{88}$$

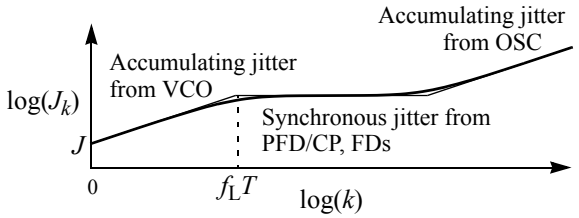
In this example, the noise was extracted for the VCO alone. In practice, the LF is generally combined with the VCO before extracting the noise so that the noise of the LF is accounted for.

11 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 [25]. Assume that the PLL has a closed-loop bandwidth of f_L, and that $\tau_\mathrm{L} = 1/2\pi f_\mathrm{L}$, then for k such that $kT \ll \tau_\mathrm{L}$, 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 15.

12. At one point it was mistakenly suggested in the documentation for SpectreRF that *maxsideband* should be set to 0 for oscillators. This causes SpectreRF to ignore all noise folding and results in a significant underestimation of the total noise.

FIGURE 15 *Long-term jitter (J_k) for an idealized PLL as a function of the number of cycles.*



12 Modeling a PLL 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 [2,3]. 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.

12.1 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 9, which models synchronous jitter with (45) where j_{sync} is a stationary white discrete-time Gaussian random process. It is also used in Listing 10, which models a simple PFD/CP.

12.1.1 Frequency Divider Model

The model, given in Listing 9, 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 Vhi if n is 1 and Vlo 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 delays, td must always be larger than dt. This model expects jitter to be specified as J_{ee}, as computed with (63).

LISTING 9 *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
```

12.1.2 PFD/CP Model

The model for a phase/frequency detector combined with a charge pump is given in Listing 10. 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 [8]. 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.

12.2 Modeling Accumulating Jitter

12.2.1 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 fre-

LISTING 10 *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 Iout=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(Iout*state, td + dt, tt);
end
endmodule
```

quency oscillator, the *timer*() function is used as shown in Listing 11. 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, $K = 2$.

12.3 VCO Model

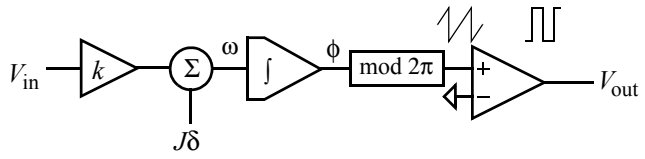
A VCO generates a sine or square wave whose frequency is proportional to the input signal level. VCO models, given in Listings 12 and 13, are constructed using three serial operations, as shown in Figure 16. First, the input signal is scaled to compute the desired output frequency. Then, the frequency 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 $\pi/2$.

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 $\tau = T/K = 1/Kf_o$. The frequency deviation

LISTING 11 *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
```

FIGURE 16 *Block diagram of VCO behavioral model that includes jitter.*



will be updated every interval and held constant during the intervals. With jitter, the duration of an interval is

$$\tau_i = \tau + \Delta\tau_i . \tag{89}$$

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

$$\mathrm{var}(\Delta\tau) = \frac{\mathrm{var}(T)}{K} = \frac{J^2}{K} . \tag{90}$$

Therefore,

$$\Delta\tau_i = \frac{J\delta_i}{\sqrt{K}} \tag{91}$$

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

$$f_i = \frac{1}{K}\left(\frac{1}{\tau + \Delta\tau_i}\right) = \frac{\frac{1}{K\tau}}{1 + \frac{\Delta\tau_i}{\tau}} = \frac{f_c}{1 + K\Delta\tau_i f_c} \tag{92}$$

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

$$f_i = \frac{f_c}{1 + \Delta T_i f_c}. \tag{93}$$

Finally, $\mathrm{var}(\tau_i) = J^2/K$, 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 12. 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 13 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.

12.4 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 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.

12.4.1 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 14 (more on the effect of the divide ratio on jitter in the next section). The accJitter parameter 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 15 is used. This model runs more efficiently by removing support for jitter and the td parameter.

LISTING 12 *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
```

12.4.2 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 16. It also includes code for generating a logfile containing the length of each period. The logfile is

LISTING 13 *Quadrature Differential VCO model that includes accumulating jitter.*

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

module quadVco (PIout,NIout, PQout,NQout, Pin,Nin);

electrical PIout, NIout, PQout, NQout, Pin, Nin;
output PIout, NIout, 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(PIout) <+ transition(i ? Vhi : Vlo, 0, tt);
    V(NIout) <+ transition(i ? Vlo : Vhi, 0, tt);
    V(PQout) <+ transition(q ? Vhi : Vlo, 0, tt);
    V(NQout) <+ transition(q ? Vlo : Vhi, 0, tt);
end
endmodule
```

used in Section 13 when determining S_{VCO}, the power spectral density of the phase of the VCO output.

LISTING 14 *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
```

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

$$J_{FD} = \sqrt{N} J_{VCO} . \tag{94}$$

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 (94).

To determine the effect of the divider on $S_\phi(\omega)$, square both sides of (94) and apply (82)

$$c_{VCO} T_{VCO} = \frac{c_{FD} T_{FD}}{N} . \tag{95}$$

LISTING 15 *PFD/CP without jitter.*

```
`include "disciplines.vams"

module pfd_cp (out, ref, vco);

input ref, vco; output out; electrical ref, vco, out;

parameter real Iout=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(Iout * state, 0, tt);
end
endmodule
```

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

$$c_{VCO} = c_{FD} \tag{96}$$

From (84),

$$2S_{VCO}\frac{f^2}{f_{VCO}^2} = 2S_{FD}\frac{f^2}{f_{FD}^2} \tag{97}$$

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

$$S_{VCO} = N^2 S_{FD}. \tag{98}$$

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 (94) and (98), which are summarized in Table 3.

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 (37), 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 18. However, especially with accumulat-

LISTING 16 *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
    V(out) <+ transition(Vout, 0, tt);
end
endmodule
```

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*
$f_{\mathrm{VCO}} = Nf_{\mathrm{FD}}$	$J_{\mathrm{VCO}} = \frac{J_{\mathrm{FD}}}{\sqrt{N}}$	$S_{\phi_{\mathrm{VCO}}} = N^2 S_{\phi_{FD}}$

ing 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.

13 Simulation and Analysis

The synthesizer is simulated using the netlist from Listing 18 and the Verilog-A descriptions in Listings 14-16, 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 $F_{\mathrm{min}} = 1/(TK)$. 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 [16]. 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 16 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 17. This script computes $S_\phi(\Delta f)$, the power spectral density of ϕ, using Welch's method [26]. The frequency range is from $f_{\mathrm{out}}/2$ to f_{out}/nfft. The script computes $S_\phi(\Delta f)$ with a resolution bandwidth of rbw.[13] 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

13. The Hanning window used in the psd() function has a resolution bandwidth of 1.5 bins [12]. 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 17 *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_φ(Δf)
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);
```

script must be further processed using the equations in Table 3 to remove the effect of FD_N.

14 Example

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

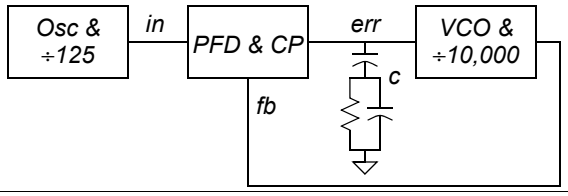
The noise of OSC is –95 dBc/Hz at 100 kHz. Applying (86) to compute c, where $L(\Delta f) = 316 \times 10^{-12}$, $\Delta f = 100$ kHz, and $f_o = 25$ MHz, gives $c = 5 \times 10^{-15}$. The period jitter J is then computed from (82) to be 14 ps.

The noise of VCO is –48 dBc/Hz at 100 kHz. Applying (86) and (82) with $L(\Delta f) = 1.59 \times 10^{-5}$, Δf = 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 of the synthesizer. The netlist is shown in Listing 18. The results (compensated for non-unity resolution bandwidth (–28 dB) and for the suppression of the dividers (80 dB)) are shown in Figures 17-20. 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.

LISTING 18 *Spectre netlist for PLL synthesizer.*

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

ahdl_include "osc.va"      // Listing 14
ahdl_include "pfd_cp.va"   // Listing 15
ahdl_include "vco.va"      // Listing 16

Osc   (in)        osc       freq=25MHz ratio=125 accJitter=14ps syncJitter=2ns
PFD   (err in fb) pfd_cp    Iout=500ua
C1    (err c)     capacitor c=3.125nF
R     (c  0)      resistor  r=10k
C2    (c  0)      capacitor c=625pF
VCO   (fb err)    vco       Fmin=1GHz Fmax=3GHz Vmin=–4 Vmax=4 ratio=10000 \
                            jitter=4.5ps outStart=10ms

JitterSim         tran      stop=60ms
```



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 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 overdamped.

FIGURE 17 *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).*

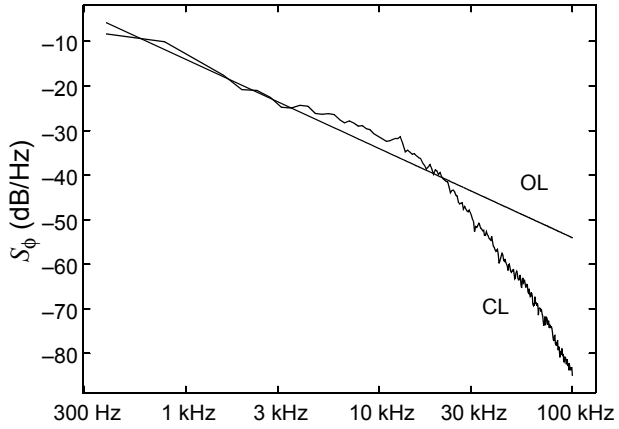
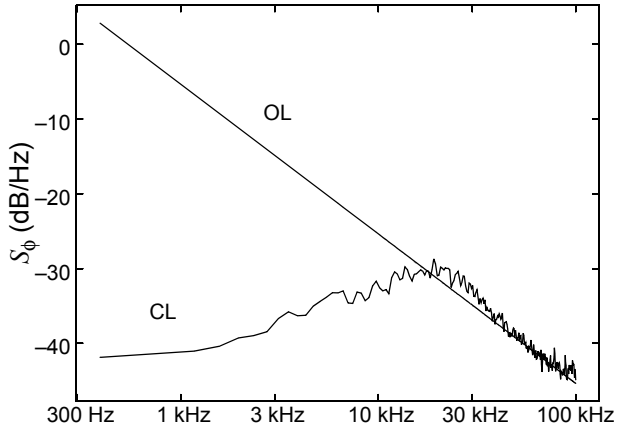


FIGURE 18 *Noise of the closed-loop PLL at the output of the VCO when only the VCO exhibits jitter (CL) versus the noise of the VCO when operated open loop (OL).*



15 Conclusion

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

FIGURE 19 *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).*

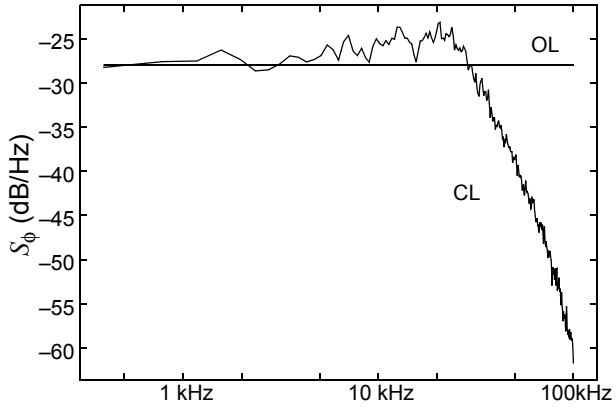
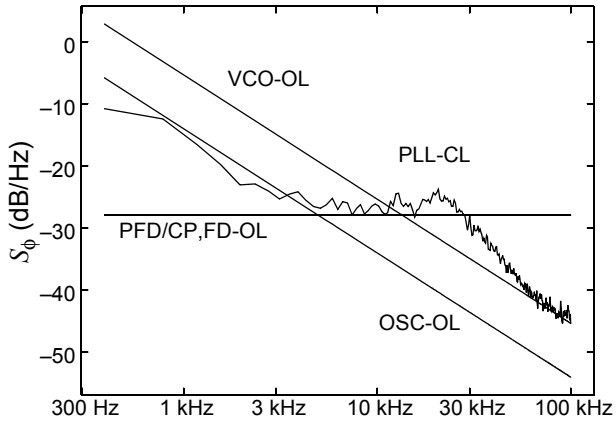


FIGURE 20 *Closed-loop PLL noise performance compared to the open-loop noise performance of the individual components that make up the PLL. The achieved noise is slightly larger than what is expected from the components due to peaking in the response of the PLL.*



15.1 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 Soin 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 (60), the cycle-to-cycle jitter of synchronous jitter, and to Prasun Raha for pointing out issues with the noise model of the charge pump. Thanks to "cheap_salary" from The Designer's Guide Community Forum for pointing out problems in Listing 8.

References

[1] Cadence Design Systems. SpectreRF simulation option. *www.cadence.com*.

[2] 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.

[3] 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.

[4] 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.

[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] 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.

[8] F. Gardner. *Phaselock Techniques*. John Wiley & Sons, 1979.

[9] W. Gardner. *Introduction to Random Processes: With Applications to Signals and Systems*. McGraw-Hill, 1989.

[10] Paul R. Gray and Robert G. Meyer. *Analysis and Design of Analog Integrated Circuits*. John Wiley & Sons, 1992.

[11] Emad Hegazi, Jacob Rael & Asad Abidi. *The Designer's Guide to High-Purity Oscillators*. Springer, 2004.

[12] 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.

[13] 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.

[14] F. Käertner. Determination of the correlation spectrum of oscillators with low noise. *IEEE Transactions on Microwave Theory and Techniqu*es, vol. 37, no. 1, pp. 90-101, Jan. 1989.

[15] F. X. Käertner. Analysis of white and $f^{-\alpha}$ noise in oscillators. *International Journal of Circuit Theory and Applications*, vol. 18, pp. 485–519, 1990.

[16] Kenneth S. Kundert. *The Designer's Guide to SPICE and Spectre*. Kluwer Academic Publishers, 1995.

[17] Kenneth S. Kundert. *The Designer's Guide to Verilog-AMS*. Kluwer Academic Publishers, 2004.

[18] 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*.

[19] 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.

[20] Ken Kundert. Modeling and simulation of jitter in PLL frequency synthesizers. Available from *www.designers-guide.org/analysis*.

[21] Ken Kundert. Verification of bit-error rate in bang-bang clock and data recovery circuits. Available from *www.designers-guide.org/analysis*.

[22] 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.

[23] Jri Lee, Kenneth S. Kundert, and Behzad Razavi. Analysis and modeling of bang-bang clock and data recovery circuits. *IEEE Journal of Solid-State Circuits*, vol. 39, no 9, September 2004, pp. 1571-1580.

[24] 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*.

[25] J. McNeill. Jitter in Ring Oscillators. *IEEE Journal of Solid-State Circuits*, vol. 32, no. 6, June 1997.

[26] A. Oppenheim, R. Schafer. *Digital Signal Processing*. Prentice-Hall, 1975.

[27] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 1991.

[28] Joel Phillips and Ken Kundert. Noise in mixers, oscillators, samplers, and logic: an introduction to cyclostationary noise. *Proceedings of the IEEE Custom Integrated Circuits Conference*, CICC 2000. The paper and presentation are both available from *www.designers-guide.org/theory.*

[29] 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.

[30] T. A. D. Riley, M. A. Copeland, and T. A. Kwasniewski. Delta-sigma modulation in fractional-*N* frequency synthesis. *IEEE Journal of Solid-State Circuits*, vol. 28 no. 5, May 1993, pp. 553 -559

[31] Ulrich L. Rohde. *Digital PLL Frequency Synthesizers*. Prentice-Hall, Inc., 1983.

[32] G. Vendelin, A. Pavio, U. Rohde. *Microwave Circuit Design.* J. Wiley & Sons, 1990.

[33] *Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL*, version 2.1. Accellera, January 20, 2003. Available from *www.accellera.com*. An abridged version is available from *www.verilogams.com* or *www.designers-guide.org/verilogams*.

[34] *Verilog-A/MS*, *verilogams.com*.

[35] 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.

[36] D. Yee, C. Doan, D. Sobel, B. Limketkai, S. Alalusi, and R. Brodersen. A 2-GHz low-power single-chip CMOS receiver for WCDMA applications. *Proceedings of the European Solid-State Circuits Conference*, Sept. 2000.