Modeling and Simulation of Jitter in Phase-Locked Loops

Ken Kundert

Cadence Design Systems
San Jose, California, USA

Abstract

A methodology is presented for predicting the jitter performance of a PLL using simulation that is both accurate and efficient. The methodology begins by characterizing the noise behavior of the blocks that make up the PLL using transistor-level simulation. For each block, the jitter is extracted and provided as a parameter to behavioral models for inclusion in a high-level simulation of the entire PLL. This approach is efficient enough to be applied to complex systems, such as frequency synthesizers with large divide ratios or fractional-N synthesizers.

1. Introduction

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

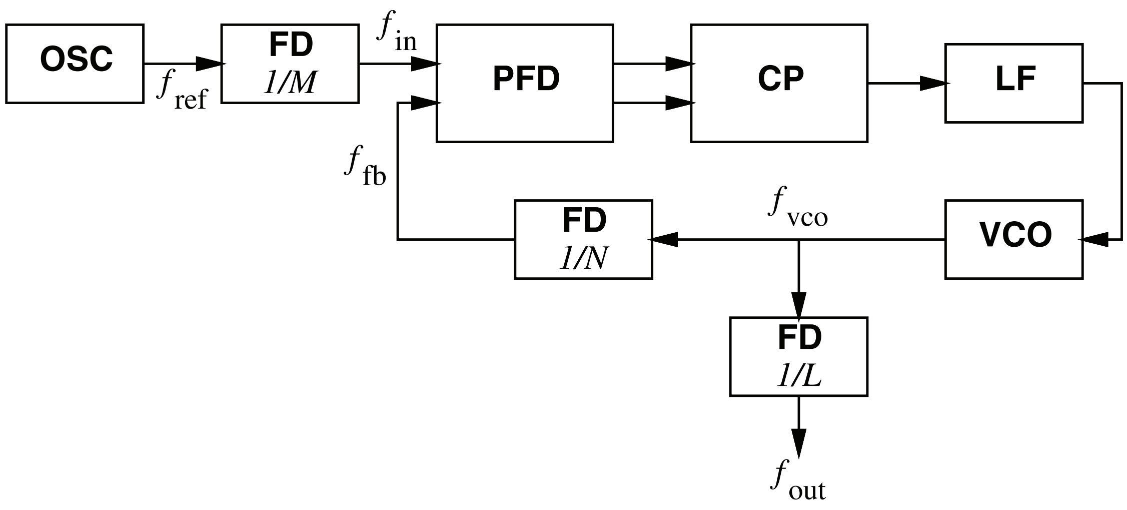
Demir proposed an approach for simulating PLLs whereby a PLL is described using behavioral models simulated at a high level and described an efficient way to include jitter in these models [demir94, chang97]. He devised a powerful new simulation algorithm that is capable of characterizing the circuit-level noise behavior of blocks that make up a PLL that is based on solving a set of nonlinear stochastic differential equations [demir96a, demir96b]. 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 [demir97]. This approach provides accurate and efficient prediction of PLL jitter behavior once the noise behavior of the blocks has been characterized. However, it requires the use of an experimental simulator that is not readily available.

This paper presents the relevant ideas of Demir, but bases the jitter extraction methodology on the commercially available SpectreRF simulator [telichevesky96b, telichevesky96b] and presents behavioral models for Verilog-A, a standard, non-proprietary analog behavioral modeling language [ovi96]. Both SpectreRF and Verilog-A are options to the Spectre circuit simulator, available from Cadence Design Systems.[1]

2. Frequency Synthesis

The block diagram of a PLL operating as a frequency synthesizer is shown in Figure

Figure 1 — The block diagram of a frequency synthesizer.



1 [gardner79].[2] 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 three frequency dividers (FDs). The PLL is a feedback loop that, when in lock, forces f_{fb} to be equal to f_{in}. Given a reference frequency f_{ref}, the frequency at the output of the PLL is

$$f_{\text{out}} = \frac{N}{LM} f_{\text{ref}} \, . \tag{1}$$

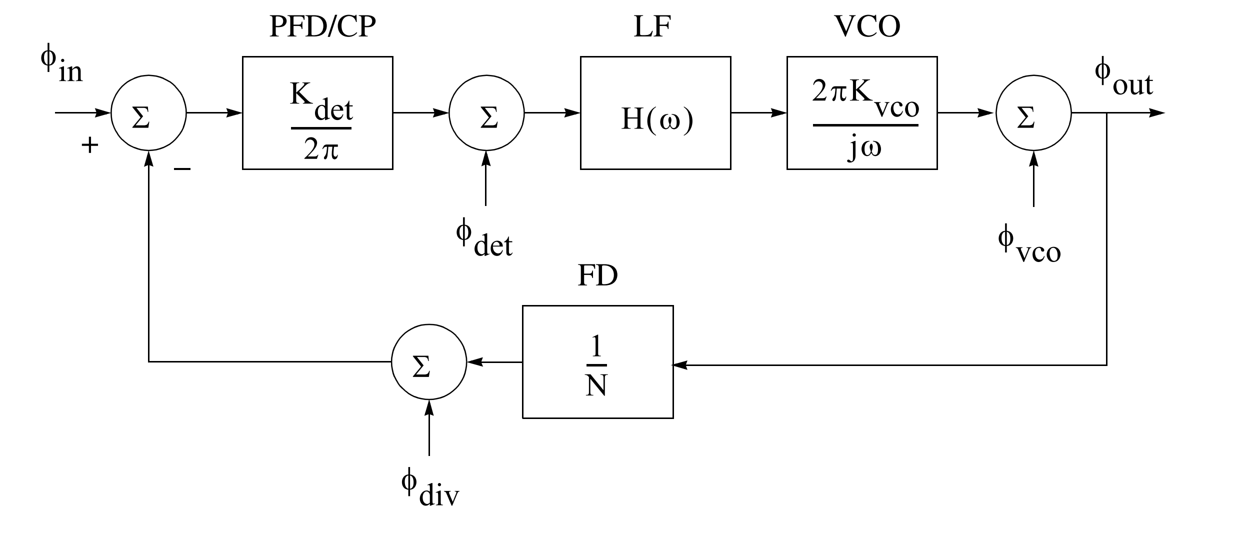
By choosing the frequency divide ratios and the reference frequency appropriately, the synthesizer generates an output signal at the desired frequency that inherits much of the stability of the reference frequency. In RF transceivers, this architecture is used to generate the local oscillator (LO) at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

1. SpectreRF is currently the only commercial simulator that is suitable for characterizing the jitter of the blocks that make up a PLL. SPICE and its descendants are not suitable because they only perform noise analysis about a DC operating point and so do not take into account the time-varying nature of these circuits. Harmonic balance simulators do perform noise analysis about a periodic operating point, which is a critical prerequisite, but they have convergence, accuracy, and performance problems with blocks such as the PFD/CP and FD that are strongly nonlinear.
2. Frequency synthesis is used as an example, but the concepts presented are easily applied to other applications, such as clock recovery.

2.1 Phase-Domain Noise Model

If the signals around the loop are interpreted as phase, then the small-signal noise behavior of the loop can be explored by linearizing the components and evaluating the transfer functions. Figure 2 shows this *phase-domain* model.

Figure 2 — Linear time-invariant phase-domain model of the synthesizer shown in Figure 1. The out-board frequency dividers are removed for simplicity. The ϕ's represent various sources of noise.



Define

$$T_{fwd} = \frac{K_{det}}{2\pi} H(\omega) \frac{2\pi K_{vco}}{j\omega} = \frac{K_{det} K_{vco} H(\omega)}{j\omega} \tag{2}$$

as the forward gain of the loop. Then the transfer function from the various noise sources to the output are

$$T_{in} = \frac{\phi_{out}}{\phi_{in}} = \frac{T_{fwd}}{1 + T_{fwd}/N} = \frac{N T_{fwd}}{N + T_{fwd}} \tag{3}$$

$$T_{vco} = \frac{\phi_{out}}{\phi_{vco}} = \frac{N}{N + T_{fwd}} \, . \tag{4}$$

And by inspection,

$$T_{div} = \frac{\phi_{out}}{\phi_{div}} = -T_{in} \tag{5}$$

and

$$T_{\mathrm{det}} = \frac{\phi_{\mathrm{out}}}{\phi_{\mathrm{det}}} = \frac{2\pi T_{\mathrm{in}}}{K_{\mathrm{det}}} . \tag{6}$$

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

As $\omega \to \infty$, $T_{\mathrm{fwd}} \to 0$ because of the VCO and the low-pass filter, and so $T_{\mathrm{in}}, T_{\mathrm{det}}, T_{\mathrm{div}} \to 0$ and $T_{\mathrm{vco}} \to 1$. At high frequencies, the noise of the PLL is that of the VCO.

As $\omega \to 0$, $T_{\mathrm{fwd}} \to \infty$ because of the $1/j\omega$ term from the VCO. So at DC, $T_{\mathrm{in}}, T_{\mathrm{div}} \to N$ and $T_{\mathrm{vco}} \to 0$. At low frequencies, the noise of the PLL is contributed by the OSC, PFD/CP, and FD, 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 frequencies $(\omega \to 0)$. Because of phase noise in the VCO, $\phi_{\mathrm{vco}} \sim 1/\omega$ (neglecting flicker noise). If the LF is chosen such that $H(\omega) \sim 1$, then $T_{\mathrm{fwd}} \sim 1/\omega$, and noise contribution from the VCO to the output, $T_{\mathrm{fwd}}\phi_{\mathrm{vco}}$, is finite and nonzero. If the LF is chosen such that $H(f) \sim 1/\omega$, as it typically is when a true charge pump is employed, then $T_{\mathrm{fwd}} \sim 1/\omega^2$ and the noise contribution to the output from the VCO goes to zero at low frequencies.

3. Jitter

Jitter is an uncertainty or randomness in the timing of events. In the case of our synthesizer, the events of interest are the transitions in the output signal. One models jitter in a signal by starting with a noise-free signal $x(t)$ and displacing time with a stochastic process $j(t)$. The noisy signal becomes

$$x_n(t) = x(t + j(t)) . \tag{7}$$

For simplicity, j is assumed to be a zero-mean Gaussian process, but it may be nonstationary.

There are two types of jitter that are of interest in PLLs: phase modulation jitter*, or PM jitter,* and frequency modulation jitter, or *FM jitter*. Signals of the form (7) exhibit PM jitter if j is a stationary process and FM jitter if j is a Wiener process. However, it is not necessary or desirable to be so restrictive. These restrictions will be loosened somewhat as we go on.

3.1 PM Jitter

PM jitter is a non-accumulating jitter exhibited by driven systems. In the PLL, the PFD, CP, and FDs all exhibit PM jitter. In these components, an output event occurs as a direct result of, and some time after, an input event. PM jitter is a random fluctuation in the delay between the input and the output events. It is modeled using (7), except j is replaced by n_t,

$$x_n(t) = x(t + n_t(t)) \tag{8}$$

where n_t differs from j in that it is constrained to be either a stationary process or a process with bounded variation. Thus, the variance is allowed to change with time, but that there is an upper bound V such that $|\mathrm{var}(n_t(t), t)| < V$ for all t. This generality is needed because we want to model systems where n_t is cyclostationary. A stochastic process is cyclostationary if its statistical moments, such as mean and variance, are periodic. If n_t is white, then $\delta x(t) = x_n(t) - x(t)$ is also white.

PM jitter is so named because it is a modulation of the phase of the signal by a random process with zero mean and bounded variation. As a result, PM jitter is simply another way of describing PM noise.

3.2 FM Jitter

FM jitter is exhibited by systems, such as autonomous circuits, that generate a stream of spontaneous output transitions. In the PLL, the OSC and VCO exhibit FM jitter. FM jitter is characterized by a randomness in the time since the last output transition, thus the uncertainty of when a transition occurs accumulates with every transition. One can construct a signal that exhibits FM jitter from a process n_t with bounded variation using

$$x_n(t) = x\left(t + \int_0^t n_t(\tau)d\tau\right). \tag{9}$$

Thus, the phase difference between $x_n(t)$ and $x(t)$ is a random walk that is not bounded. If n_t is white and if $x(t)$ is periodic with frequency f_c, then $\delta x(t) = x_n(t) - x(t)$ is red in $f_m = f - f_c$. That is, it has a power spectral density that is proportional to $1/f_m^2$.

FM jitter is so named because it is the modulation of the frequency of a signal by a random process with zero mean and bounded variation. As a result, FM jitter is simply another way of describing FM noise.

FM jitter is strongly related to phase noise. Both are different ways of describing the same underlying phenomenon. In other words, all free running oscillators exhibit a behavior that is generically referred to as phase noise. Any noise in an autonomous system will cause the phase to drift freely because there is no reference signal with which to lock. When the phase fluctuations are measured in terms time deviation over a period, it is referred to as FM jitter. If it is measured in terms of noise signal amplitude as a function of frequency, it is referred to as phase noise.

3.3 A Metric for Jitter

Define J to be the standard deviation of the period of a single cycle,

$$J = \sigma(T) = \sqrt{\mathrm{var}(T)}\,. \tag{10}$$

Then J is a measure of jitter with units of time. This definition is valid for both PM and FM jitter, but does not distinguish between them.

3.4 Thresholds and Jitter

In systems where signals are continuous valued, an event is usually defined as a signal crossing a threshold. 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 PM 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, $\dot{v}(t)$, as the threshold is crossed. If the noise n_v is stationary, the jitter is

$$J = \frac{\sqrt{\mathrm{var}(n_v)}}{\dot{v}(t_c)} \tag{11}$$

where t_c is the time of a threshold crossing.

Generally n is not stationary because it is being generated by a nonlinear circuit undergoing periodic oscillations.[3] In this case, n is cyclostationary. It is only impor-

3. Another reason why n would not be stationary is if it is derived from a flicker noise source. Flicker noise is not explicitly considered in this paper. It is not conceptually difficult to include flicker noise, but flicker noise models tend to be expensive to implement in the time domain [demir96a].

tant 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,

$$J = \frac{\sqrt{\mathrm{var}(n_v, t_c)}}{\dot{v}(t_c)}. \tag{12}$$

4. Model of PLL

The basic behavioral models for the blocks that make up a PLL are well known and so will not be discussed here in any depth [demir94, chang97]. Instead, only the techniques for adding jitter to the models is discussed.

4.1 Jitter Models

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.

4.2 Modeling PM jitter

A feature of Verilog-A allows especially simple modeling of PM 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 generates discrete-valued outputs. It is used in the Verilog-A divider module shown in Listing 1, which models PM jitter with (8) where n_t is a wide-sense stationary white discrete-time Gaussian random process. It is also used in Listing 2, which models a simple PFD/CP.

4.3 Modeling FM jitter

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

A VCO is modeled by modifying a conventional VCO model. The input voltage is scaled by the VCO gain constant to convert it into frequency, and is then integrated to convert it to phase. Output transitions are generated when the phase passes $-\pi/2$

Listing 1 — Verilog-A description of a frequency divider that models jitter. PM jitter is modeled by randomly dithering the *delay* argument of the *transition*() function on every output transition.

```
// Frequency Divider with Jitter

‘include "discipline.h"
‘include "constants.h"

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);
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;
        // add jitter
        dT = jitter*$dist_normal(seed,0,1);
    end

    // generate the output
    n = (2*count >= ratio);
    V(out) <+ transition(n ? Vhi : Vlo, td+dT,tt);
end
endmodule
```

and $\pi/2$. Jitter is added as a random perturbation of the frequency, with a correction applied to accurately convert the jitter specification from time to frequency. The dither is updated once per output transition. The final model given in Listing 4. This model can be easily modified to fit other needs. Converting it to a model that generates sine-waves rather than square waves simply requires replacing the last two lines with one that computes and outputs the sine of the phase. When doing so, you might 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 2 — Verilog-A description of a simple PFD/CP that models jitter.

```
// Phase-Frequency Detector & Charge Pump

`include "discipline.h"
`include "constants.h"

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);
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
        state = state – 1;
        dt = jitter ∗ $dist_normal(seed,0,1);
    end

    @(cross(V(vco), dir, ttol)) begin
        state = state + 1;
        dt = jitter ∗ $dist_normal(seed,0,1);
    end

    if (state > 1) state = 1;
    if (state < –1) state = –1;

      I(out) <+ transition(Iout ∗ state, td + dt, tt);
end
endmodule
```

This model does not model the finite response time of a real VCO. Those dynamics would be separated out and included as part of the model for the LF.

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

4.4 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 tim-

Listing 3 — Verilog-A description of a free-running fixed frequency oscillator. FM jitter is modeled by adding a random perturbation to the length of each period. This model also supports PM noise, which is useful if the PM noise from the PFD/CP and FD is included as part of the reference oscillator model rather than modeled separately.

```
// Fixed-Frequency Oscillator with Jitter

‘include "discipline.h"
‘include "constants.h"

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 fmJitter=0 from [0:0.1/freq);
parameter real pmJitter=0 from [0:0.1/freq);

integer n, fmSeed, pmSeed;
real next, dT, dt;

analog begin
    @(initial_step) begin
        fmSeed = 286;
        pmSeed = –459;
        next = 0.5/freq + $realtime;
    end

    @(timer(next + dt)) begin
        n = !n;
        dT = fmJitter*$dist_normal(fmSeed,0,1);
        dt = pmJitter*$dist_normal(pmSeed,0,1);
        next = next + 0.5/freq + 0.707*dT;
    end

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

ing 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.

From the discussion of the phase-domain model of the synthesizer, it is clear that one can easily combine the output-referred noise of the FD and the input-referred noise of the PFD/CP with the output noise of OSC. The fixed-frequency oscillator model given in Listing 3 supports two jitter parameters. The fmJitter parameter is used to model the FM jitter of the reference oscillator, and the pmJitter parameter is

Listing 4 — Verilog-A description of a VCO model that includes jitter.

```
// Voltage Controlled Oscillator with Jitter

‘include "discipline.h"
‘include "constants.h"

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);
parameter real ttol=1u/Fmax from (0:1/Fmax);

real freq, phase, dT, delta;
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
    delta = dT∗freq;
    delta = delta/(1 – delta);
    freq = freq∗(1 + delta);

    // phase is the integral of the freq modulo 2π
    phase = 2∗‘M_PI∗idtmod(freq, 0.0, 1, –0.5);

    // update jitter twice per period
    @(cross(phase + ‘M_PI/2, +1, ttol) or cross(phase – ‘M_PI/2, +1, ttol)) begin
        dT = 1.414∗jitter∗$dist_normal(seed,0, 1);
                // 1.414=sqrt(N) where N=2 jitter updates/period
    end

    // generate the output
    n = (phase >= –‘M_PI/2) && (phase < ‘M_PI/2);
    V(out) <+ transition(n ? Vhi : Vlo, 0, tt);
end
endmodule
```

used to model the PM jitter of the FD and PFD/CP. PM jitter is modeled in the oscillator without using a nonzero delay in the transition function. This is a more effi-

Listing 5 — Verilog-A description of a quadrature VCO model that includes jitter.

```
// Quadrature Differential VCO with Jitter

'include "discipline.h"
'include "constants.h"

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 Vlo=–1, Vhi=1;
parameter real Fmin=1 from (0:inf);
parameter real Fmax=2*Fmin from (Fmin:inf);
parameter real jitter=0 from [0:0.25/Fmax), ttol=1u/Fmax from (0:1/Fmax);
parameter real tt=0.01/Fmax;

real freq, phase, dT, delta;
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
    delta = dT∗freq;
    delta = delta/(1 – delta);
    freq = freq∗(1 + delta);

    // phase is the integral of the freq modulo 2π
    phase = 2*'M_PI*idtmod(freq, 0.0, 1, –0.5);

    // update jitter where phase crosses π/2
    @(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))
        dT = 2*jitter*$dist_normal(seed,0,1);  // 2=sqrt(N), N=4 jitter updates per period

    // generate the I and Q outputs
    i = (phase >= –3*'M_PI/4) && (phase < 'M_PI/4);
    q = (phase >= –'M_PI/4) && (phase < 3*'M_PI/4);
    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
```

cient approach because it avoids generating two unnecessary events per period. To get full benefit from this optimization, set td in the FD and PFD/CP to zero.

If the output of the VCO is not used to drive circuitry external to the synthesizer, then it is usually possible to include the frequency division aspect of the FD as part of the VCO by simply adjusting the VCO gain. Recall that the PM jitter of the FD has already been included as part of OSC. The jitter at the output of the modified VCO is the same as jitter at the output of the unmodified VCO. If the divide ratio of FD is large, the simulation runs much faster because the high VCO output frequency is never generated.

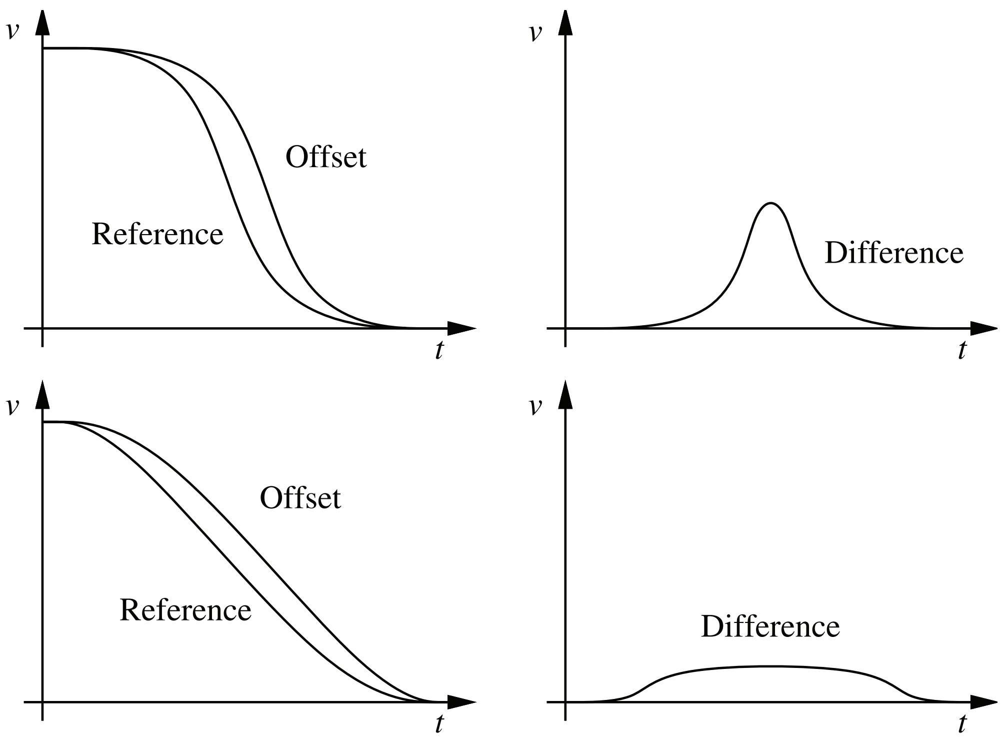
To make the HDL models even faster, rewrite them in either Verilog-HDL or VHDL. Be sure to set the time resolution to be sufficiently small to prevent the discrete nature of time in these simulators from adding an appreciable amount of jitter.

5. Characterizing Jitter

The switching nature of the blocks in a PLL prevents use of the conventional noise analysis available from SPICE to characterize the noise of any of the blocks in a PLL with the possible exception of the LF. The SPICE noise analysis operates by linearizing the block about a DC operating point, which is not sufficient when the block exhibits switching behavior. SpectreRF and the simulator developed by Demir both linearize the circuit about a time-varying operating point and compute the noise at the output of the block while taking into account both the effect of the time-varying operating point on the bias-dependent noise sources and the time-varying nature of the transfer function from the noise source to the output. They differ in that SpectreRF is constrained to operate on periodic circuits. In addition, SpectreRF outputs noise as a function of frequency averaged over a period, while Demir's simulator computes the output noise as a function of time and integrated over all frequencies.

Both simulators linearize about the operating point and compute the noise as a post processing step. Thus, the noise does not affect the operating point calculation and so the simulation will not be accurate if the noise is large enough to affect the large-signal behavior of the circuit. Generally, the amplitude of the noise sources is quite small and so this is not a concern. However, in thresholding circuits, the noise present when the signal crosses the threshold gets amplified tremendously. When cascading several thresholding stages, the noise can be amplified to such a great degree that it does change the large signal behavior, making the simulation inaccurate. This occurs in a FD implemented as a ripple counter with a large number of stages. In such cases it is necessary to break the circuit down and only characterize the jitter of one or two stages at a time. The maximum number of stages that can be characterized together is greater if the jitter is small relative to the transition time of the circuit, as shown in Figure 3.

Figure 3 — The figures on the left show a reference signal, and one offset due to jitter. In both the top and bottom figures, the offset is the same (they have the same amount of jitter). The top signals have a smaller transition time than those on the bottom. The figures on the right show the difference between the reference signal and the offset signal. For the same amount of jitter, the maximum difference is larger when the transition time is smaller. If the difference is large enough to cause a nonlinear response, the noise simulations will not be accurate. This problem becomes significant when the jitter is about the same size of the transition time or larger. It limits the number of cascaded thresholding stages that can be characterized at one time.



5.1 Characterizing PM Jitter

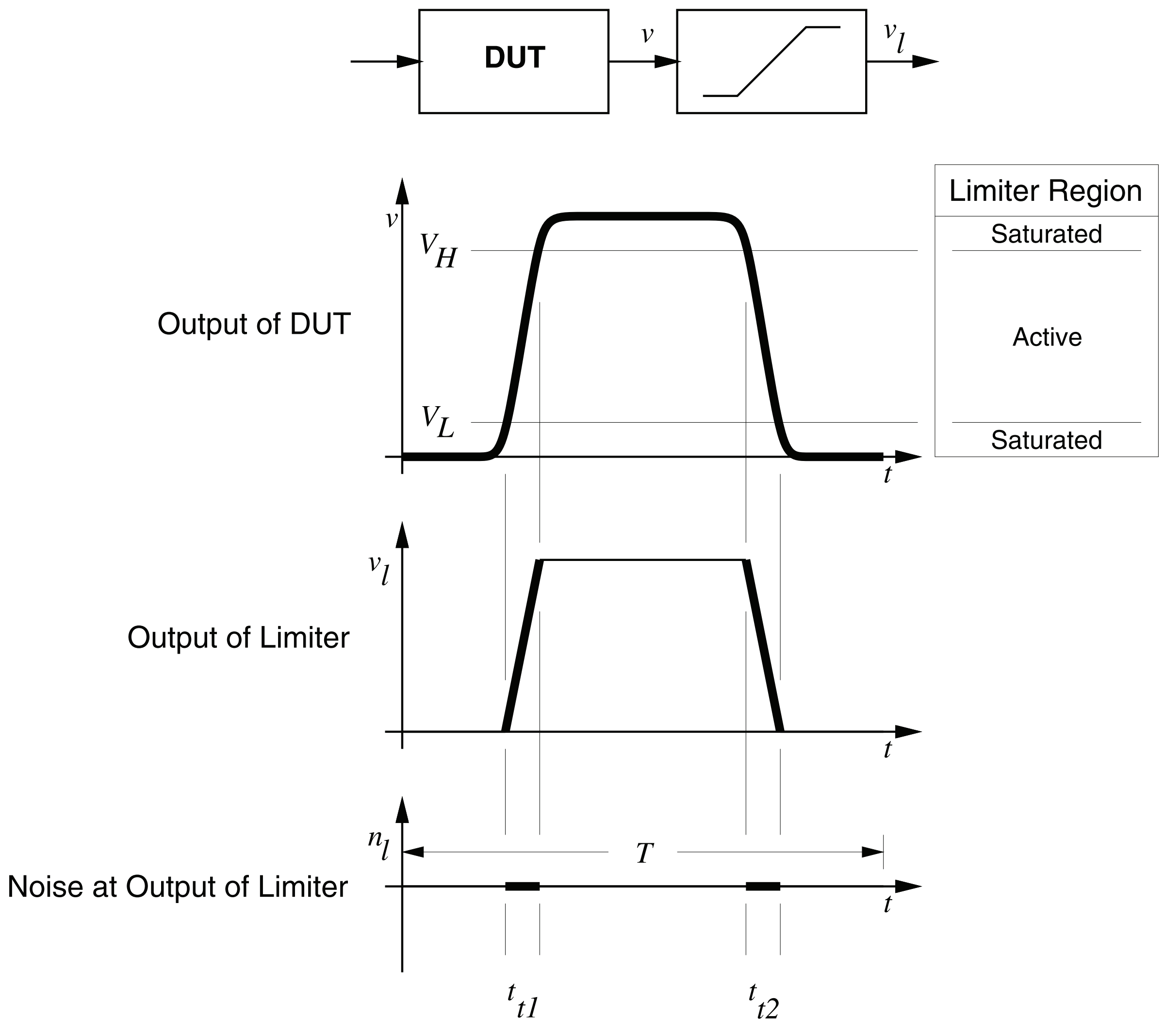
SpectreRF's PNoise analysis computes the power spectral density of the noise at the output of the block. If this noise is stationary (as opposed to cyclostationary), it is a simple matter to apply (11) to calculate the jitter. Simply choose a representative set of periodic inputs to the block and use SpectreRF's Periodic Steady State (PSS) analysis to compute the steady-state response. This computes the periodic operating point about which the noise analysis is performed. It also gives $\dot{v}(t_c)$, the slew rate of the output at threshold crossing. Apply SpectreRF's PNoise analysis to compute the noise power at the output as a function of frequency. 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 at the highest frequency simulated. Finally, integrate the noise power over frequency and apply Parseval's Theorem to determine

$$\mathrm{var}(n_v) = \int n_v^2(f)df\,, \tag{13}$$

the total noise power [gray93], and apply (11).

When the noise is cyclostationary the same procedure is used, except a gating function is applied to the output so that only the noise that occurs near the threshold crossing is considered in the jitter calculation. SpectreRF's PNoise analysis computes the time-average of the noise at the output, so it is not possible to post-process the PNoise results to determine the noise at the time of the threshold crossing. Rather, a limiter is added to the output of the block and SpectreRF computes the noise at the output of the limiter. The limiter, shown in Figure 4, is designed to saturate when the output of the block is outside a certain range to prevent any noise at the output from being considered except the noise present near when the signal

Figure 4 — When the device-under-test (DUT) exhibits cyclostationary noise, a limiter is applied to output of driven block to suppress noise present outside of the active transitions. *v(t)* is the output of the block, $v_l(t)$ is the output of the limiter, and $n_l(t)$ is the noise at the output of the limiter. Noise on a waveform is denoted by using a thick trace. The output of the limiter is only noisy when it is inside its active region (when it is not limiting).



crosses the threshold. The limiter is shown in Listing 6. The range of the limiter, V_L

Listing 6 — Verilog-A limiter used for characterizing PM jitter in circuits with binary outputs.

```
// Simple Limiter

‘include "discipline.h"
‘include "constants.h"

module limiter (out, in);

output out; input in; electrical out, in;

parameter real Vlo=–1, Vhi=1;

analog begin
    // Place time-point at threshold crossings
    @(cross(V(in) – Vlo) or cross(V(in) – Vhi));

    // Determine the output
    if (V(in) < Vlo)
        V(out) <+ Vlo;
    else if (V(in) > Vhi)
        V(out) <+ Vhi;
    else
        V(out) <+ V(in);
end
endmodule
```

and V_H, is chosen such that the noise and the slew rate is approximately constant while the limiter is active. When running PNoise analysis, assure that the maxsidebands parameter is at least ten times larger than T/t_{ti} for any i. This assures that the narrow noise pulses are adequately resolved by the PNoise analysis. Jitter is independent of T, so to reduce the number of sidebands needed, use T as small as possible. If maxsidebands is not set sufficiently large, then the extracted value of jitter will increase as T decreases.

If we assume that the amplitude of the noise during each transition is the same, then

$$\mathrm{var}(n, t_c) = \frac{\langle n_l^2 \rangle T}{t_{t1} + t_{t2}} \tag{14}$$

where $\langle n_l^2 \rangle$ is the time average of the noise power at the output of the limiter. SpectreRF computes the power spectral density of the noise at the output of the limiter, and by Parseval’s theorem,

$$\langle n_l^2 \rangle = \int n_l^2(f) df\,. \tag{15}$$

If we further assume that $t_t = t_{t1} = t_{t2}$, then

$$\dot{v}(t_c) \approx \frac{V_H - V_L}{t_t} . \tag{16}$$

And from (12)

$$J = \frac{\sqrt{\mathrm{var}(n, t_c)}}{\dot{v}(t_c)} \approx \sqrt{\frac{\langle n_l^2 \rangle T}{N t_t}} \frac{t_t}{V_H - V_L} , \tag{17}$$

$$J \approx \frac{\sqrt{\langle n_l^2 \rangle T t_t / N}}{V_H - V_L} , \tag{18}$$

where $N = 2$ is the number of transitions that occurred during the period.

This general methodology for characterizing the PM jitter of driven blocks with binary outputs is extended or clarified for important special cases in the next few sections.

5.1.1 PM jitter of the PFD/CP

The PFD and CP work together to generate a three-level discrete-valued signal (it takes the values –1, 0, and +1) whose time average is used as the loop error signal. The average of this signal controls the VCO after it has been extracted by the LF.

There are two aspects of the PFD/CP that differ from the assumptions made above. First, the output of the CP is a current, so the limiter and the equations given in the previous section need to be adapted. Second, the output of the CP has three distinct levels rather than the two assumed above. Thus, the CP has a ternary output rather than a binary output.

If it is necessary to apply a gating function, care must be taken because of the ternary nature of the output. A simple limiter would allow the noise associated with the middle value to pass. So the simple limiter should be replaced with a dead-band limiter. This is a limiter with a dead band in the center of its input range. The dead band rejects noise about the equilibrium point associated with the middle of the three values.

5.1.2 PM Jitter of a FD

With ripple counters, one can only characterize a few stages at a time because of the issue shown in Figure 3. Thus, a long ripple counter chain has to be broken into smaller chains, and characterized individually. The total jitter for the ripple counter is then computed by 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. Rather, jitter of a synchronous counter is the jitter of its clock along with the jitter of the last stage.

The input to counters are generally edge sensitive, and so are only affected by jitter on either the positive-going or the negative-going clock transitions, but not both. However, this should not affect the way in which blocks are characterized as long as the behavioral models for the dividers also have edge-sensitive inputs. Then the behavioral model of the dividers only react to jitter on the proper edge and ignore jitter on the other edge.

5.1.3 Differential Outputs

When a circuit has multiple outputs, such as in the case where a circuit has differential outputs, there may be correlation between the jitter on the various outputs. Currently SpectreRF does not compute the correlation in noise between two outputs, however in some cases it is possible to use physical reasoning to determine how to model the correlation. Consider a synchronous counter with differential outputs. Clearly, the jitter that derives from the clock will be perfectly correlated between the two outputs. The jitter from the last stage will be partially correlated, but again, some estimate of the degree of correlation is often possible if you understand the implementation of the last stage.

5.2 Characterizing FM Jitter

The noise of a free-running oscillator is dominated by phase noise, which is a random shifting of the frequency, and hence the phase, of the oscillation signal over time. The phase of an oscillator is subject to this variation because it is free running: there is no drive signal with which to lock and so no synchronization between the signals generated by the oscillator and any reference signal. Indeed, it is the function of the feedback in the PLL to provide this synchronization in order to reduce the phase noise of the VCO.

Phase noise and FM jitter are different ways of describing the same underlying phenomenon, and so there is a direct conversion between phase noise and FM jitter. There is no need to invoke the use of thresholds or gating functions in order to make the conversion.

SpectreRF's PNoise analysis is used to compute the phase noise of either the OSC or the combination of the LF and VCO. The power spectral density of the phase of phase noise is proportional to $1/f_m^2$ where $f_m = f - f_c$ is the frequency offset from the carrier (this ignores any flicker or *1/f* noise). Define a such that

$$S_\phi(f_m) = a\frac{(2\pi f_c)^2}{(2\pi f_m)^2} . \tag{19}$$

Then, the phase noise is completely characterized once a and f_c are known. Typically, phase noise is measured as $\mathcal{L}(f_m)$, the ratio of the noise power in a 1 Hz bandwidth at f_m to the carrier signal power. In the region where the noise at the $f_c \pm f_m$ sidebands are correlated [vendelin90],

$$S_\phi(f_m) = 2\mathcal{L}(f_m) . \tag{20}$$

This occurs at frequencies close to f_c where the phase noise dominates over additive noise. Thus,

$$a = 2\mathcal{L}(f_m)\frac{f_m^2}{f_c^2} . \tag{21}$$

a is the noise added per period T, and so the jitter is [demir97]

$$J = \sqrt{aT}. \tag{22}$$

6. Example

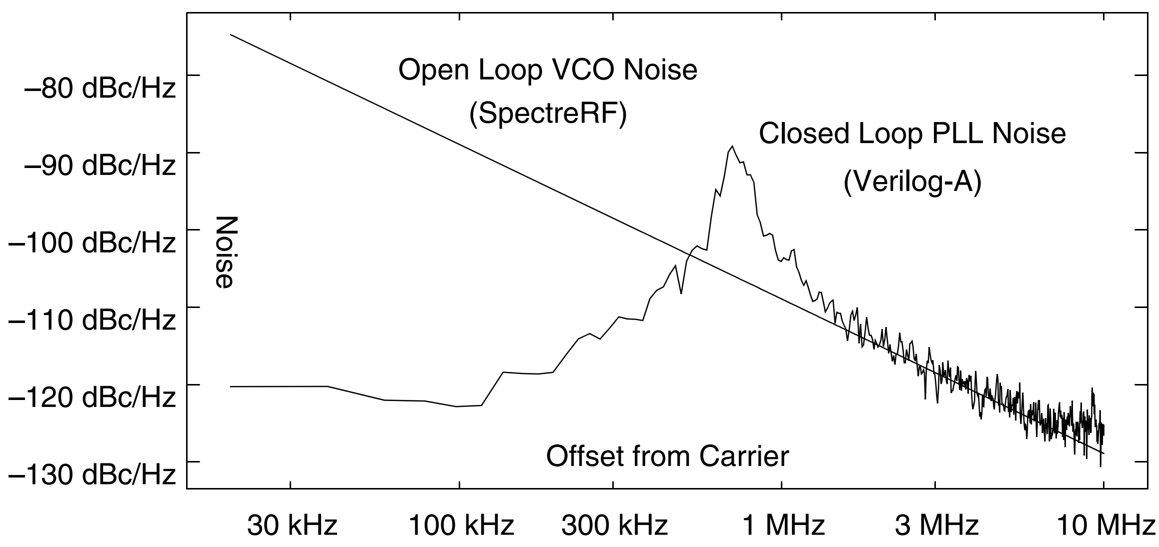
These ideas were applied to model and simulate a PLL acting as a frequency synthesizer that multiplies the reference frequency by a factor of roughly 5000 (N = 5000). The FD was included into the VCO, which suppressed the high frequency signal at the output of the VCO. The results are shown in Figure 5. The simulation took 15 minutes for 250k time-points on a Sun Sparc 5. The use of a large number was motivated by the desire to have good resolution of the noise at low frequencies.

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 act to stabilize the phase of the VCO, and the noise of the PLL is contributed by OSC, PFD/CP, FD. There is some contribution from the VCO, but it is diminished by the gain of the loop. The measured results agree qualitatively with the these expected results.

7. Conclusion

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

Figure 5 — SpectreRF results for the phase noise of the open-loop VCO, and Spectre results for the closed-loop PLL modeled in Verilog-A. In this case, the phase noise of the VCO dominates as a source of jitter in the PLL, but that jitter is reduced at low frequencies by the feedback of the loop.



cient 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 VHDL.

This methodology is flexible enough to be used in a broad range of applications where jitter is important. Examples include, clock generation and recovery, sampling systems, over-sampled ADCs, digital modulation and demodulation systems, fractional-*N* frequency synthesis, and disk read channels.

Acknowledgments

I would like to thank Alper Demir for many enlightening conversations about noise and jitter. I would also like to thank Mark Chapman, Masayuki Takahashi, and Kimihiro Ogawa of Sony Semiconductor and Frank Hellmich and Randeep Soin of Cadence Design Systems for their probing questions and insightful comments, as well as their help in validating these ideas on real frequency synthesizers.

References

[chang97] 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.

[demir94] 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.

[demir96a] 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.

[demir96b] A. Demir, A. Sangiovanni-Vincentelli. Simulation and modeling or phase noise in open-loop oscillators. *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 445-456, May 1996.

[demir97] Alper Demir. *Analysis and Simulation of Noise in Nonlinear Electronic Circuits and Systems*. Ph. D. Dissertation. University of California, Berkeley. Spring 1997.

[gardner79] Floyd M. Gardner. *Phaselock Techniques*. John Wiley & Sons, 1979.

[gray93] P. R. Gray and R. G. Meyer. *Analysis and Design of Analog Integrated Circuits.* J. Wiley & Sons, Third Edition, 1993.

[ovi96] *Verilog-A Language Reference Manual: Analog Extensions to Verilog-HDL*, version 1.0. Open Verilog International, 1996. Available from www.ovi.org.

[telichevesky96a]R. Telichevesky, K. Kundert, J. White. Receiver characterization using periodic small-signal analysis. *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1996.

[telichevesky96b]R. Telichevesky, K. Kundert, J. White. Efficient AC and noise analysis of two-tone RF circuits. *Proceedings of the 33rd Design Automation Conference*, June 1996.

[vendelin90] George D. Vendelin, Anthony M. Pavio, Ulrich L. Rohde. *Microwave Circuit Design.* J. Wiley & Sons, 1990.