Behavioral Simulation of Fractional-N Frequency Synthesizers and Other PLL Circuits

Michael H. Perrott
Massachusetts Institute of Technology

Two techniques are presented that allow fast and accurate simulation of fractional-N synthesizers. A uniform time step allows implementation of these techniques in various simulation frameworks, such as Verilog, Matlab, and C or C++ programs. The techniques are also applicable to the simulation of other PLL systems, such as clock and data recovery circuits.

FRACTIONAL-N FREQUENCY SYNTHESIZERS provide high-speed frequency sources that can be accurately set with very high resolution—a valuable feature for many communication systems. Figure 1a illustrates a fractional-N synthesizer, which includes a

- phase-frequency detector (PFD),
- charge pump,
- loop filter,
- voltage-controlled oscillator (VCO), and
- frequency divider dithered between integer values to achieve fractional divide ratios.

At the Microsystems Technology Laboratory at MIT, we have developed two techniques that allow fast and accurate simulation of both dynamic and noise performance of fractional-N synthesizers at a detailed behavioral level. We incorporated these techniques into a custom C++ program to simulate the dynamic and noise performance of a prototype Σ-Δ frequency synthesizer.

This class of fractional-N synthesizers dithers the divide value according to the output of a Σ-Δ modulator.[1] Such dithering ensures high frequency resolution,[1] but also has the negative effect of introducing quantization noise, which degrades the overall phase-locked loop (PLL) noise performance. It can be very helpful to simulate the effects of this quantization noise, along with other noise sources (shown in Figure 1b) in the PLL, on overall PLL performance. Simulating the synthesizer's dynamic response to variations of the Σ-Δ input can also help evaluate stability and characterize the system's performance when it is used as a transmitter.[2]

The problem

Fast behavioral simulation of fractional-N synthesizers is challenging for several reasons. First, the synthesizer's high output frequency (often in the gigahertz range) imposes a high simulation sample frequency for traditional simulators. Unfortunately, the overall PLL dynamics have a bandwidth that is typically three to four orders of magnitude lower than

0740-7475/02/$17.00 © 2002 IEEE

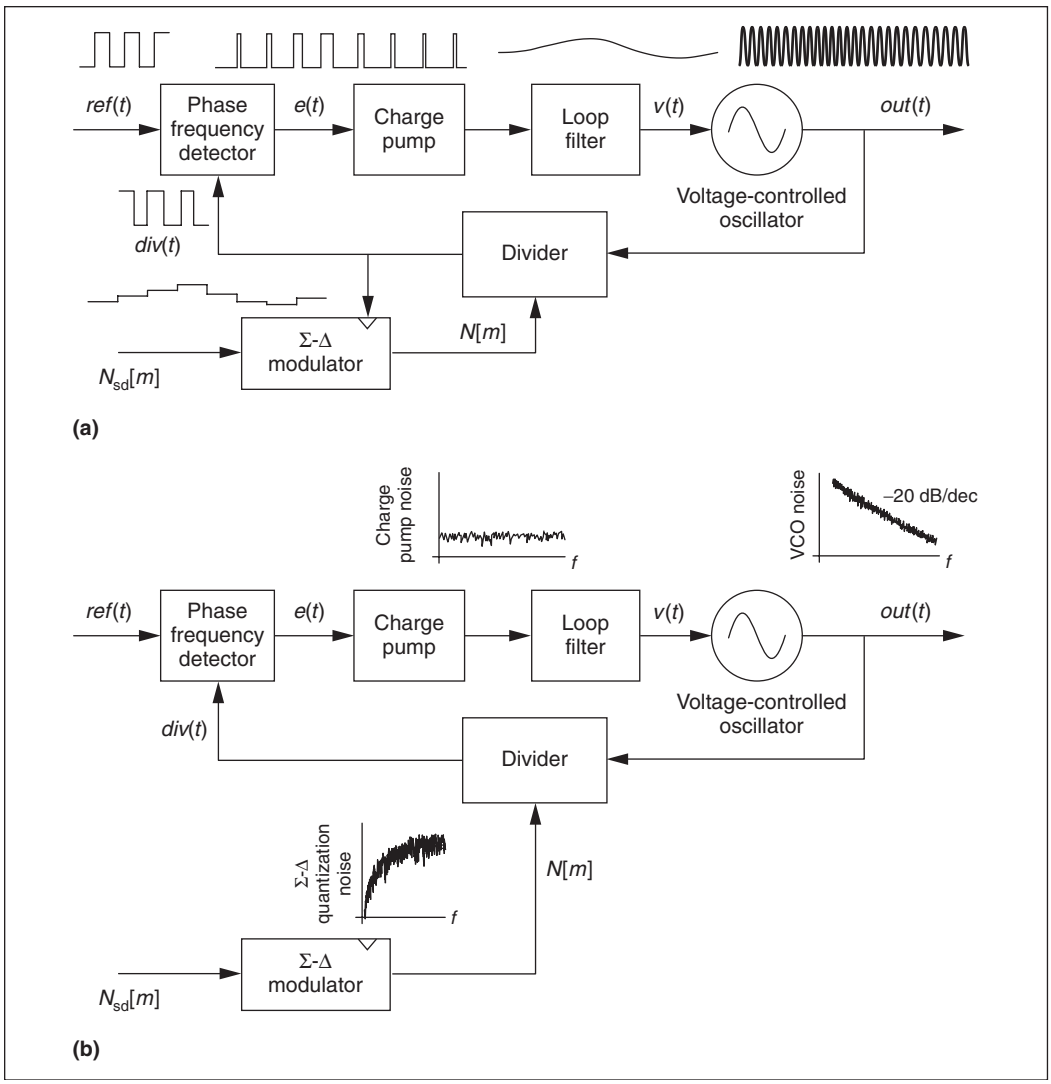

Figure 1. Σ-Δ synthesizer with associated signals (a) and spectral densities of phase-locked loop (PLL) noise sources (b). In the figure, *ref*(*t*) is the input frequency reference, *e*(*t*) is the phase-frequency detector (PFD) output, *v*(*t*) is the voltage-controlled oscillator (VCO) input voltage, *out*(*t*) is the VCO output, *div*(*t*) is the divided-down output frequency, $N_{sd}[m]$ is the input to the Σ–Δ modulator, and *N*[*m*] is the instantaneous divide value.

the output frequency (often 100 kHz to 1 MHz of bandwidth compared to 1 to 10 GHz for the output frequency). Thus, traditional simulators take a long time to compute the system's dynamic response, because they must process many simulation samples. This is a classical problem in the simulation of PLL circuits. Second, for noise simulation, the fractional-N synthesizer adds an additional constraint: Its behavior is nonperiodic in steady state because of the divide value's dithering action. Dithering prevents the use of fast methods developed for periodic steady-state conditions,[3] as used with simulators such as Cadence's SpectreRF.

Accurate simulation of fractional-N synthesizers is difficult because the synthesizer's continuous-time (CT) signals must be simulated in discrete time (DT). CT-to-DT conversion introduces numerical noise for classical simulation techniques with a constant time step. The key signal for introducing such noise is the PFD output. The standard approach for converting the CT PFD output to a DT sequence is to apply the sampling operation shown in Figure 2a.[4]

Unfortunately, this approach effectively quantizes the location of the PFD edges, according to simulation sample period T_s. You can reasonably assess the PLL's dynamic performance by making T_s sufficiently small. However, the resulting quantization noise overpowers the signals' true noise characteristics and prevents proper noise analysis of the overall PLL.

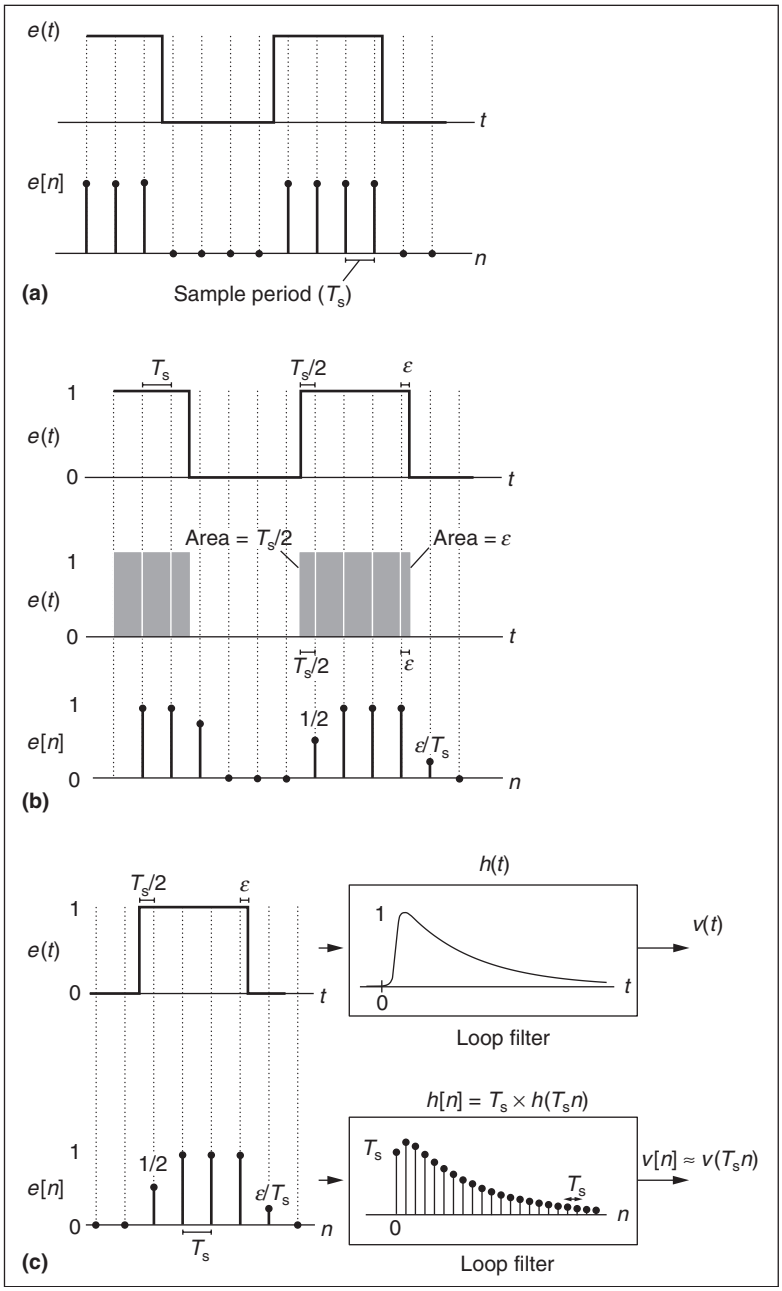

Figure 2. Approaches for converting continuous-time (CT) PFD outputs to a discrete-time (DT) sequence: classical uniform-time-sample method (a), details of the PFD discretization technique (b), and example DT loop-filter impulse response with a corresponding PFD signal (c). In the figure, *e*[*n*] is the DT PFD signal, *n* is the simulator time index, T_s is the simulation sample period, ε is the timing parameter, *h*(*t*) is the loop filter's CT impulse response, *h*[*n*] is a DT version of the loop filter impulse response, and *v*[*n*] is a DT version of the VCO input.

To solve the quantization noise issue, researchers have developed event-driven simulation methods for classical frequency synthesizers that align simulation samples precisely to the PFD output's edges.[5-7] Although such methods can achieve higher accuracy, they are generally more complicated than uniform-time-sample methods. Designers must either develop closed-form calculations of the loop-filter step response and insert them into the simulation or incorporate iterative methods, such as those used in Spice or Verilog-A, into the simulator to calculate the loop-filter response with varying time steps. The former is tedious and typically restricted to a low loop-filter order, so most recent methods focus on the latter.[5-7] This approach minimizes the designers' up-front work, but the simulation time is often longer due to the iterative calculations performed at each time step. Unfortunately, designers have not yet successfully applied event-driven simulators to the noise analysis of fractional-N frequency synthesizers, where the divide value varies dynamically.

Our approach

Our techniques use a uniform sample period and a noniterative computation of the sample values for the signals in the system. A uniform sample period lets designers readily examine the simulator results in the frequency domain without resampling. The noniterative computation makes the techniques suitable in mainstream simulators such as Verilog, VHDL, Matlab, and custom C or C++ programs.

The first technique uses area conservation to accurately represent the CT PFD output with a DT sequence. The second dramatically reduces simulation sample frequency, thus allowing a longer sample period, by including the divider implementation in the VCO simulation module.

Accurate PFD discretization

To accurately discretize PFD output $e(t)$, we view this signal as a series of rectangular pulses

with a height of 1 or 0, with width and time offsets that vary according to the location of PFD edges, as Figure 2b shows. For rectangular pulses not associated with edges, the width corresponds to simulation sample period T_s. For rectangular pulses at edge boundaries, the pulse width varies between 0 and T_s. In either case, these pulses look like impulses to the loop filter. Therefore, we represent the corresponding DT PFD signals, $e[n]$, as samples having an amplitude proportional to the *area* of the respective rectangular pulse in that time sample interval.[8] Each pulse's area corresponds to its associated timing parameter, ε. A later section explains how to calculate ε for each pulse.

Given the PFD output discretization in Figure 2b, we can represent the CT loop filter dynamics with a DT filter using either the impulse invariance or bilinear transform methods.[9] These representations allow noniterative computation of the loop filter dynamics. Figure 2c illustrates an example DT loop-filter impulse response, along with a corresponding DT PFD signal. To simplify the analysis, we ignored the charge pump; however, you can include its effect by scaling the PFD output by the value of the charge pump current. In the example, the loop filter is implemented as a DT filter with an impulse response corresponding to samples of the loop filter's CT impulse response, $h(t)$.

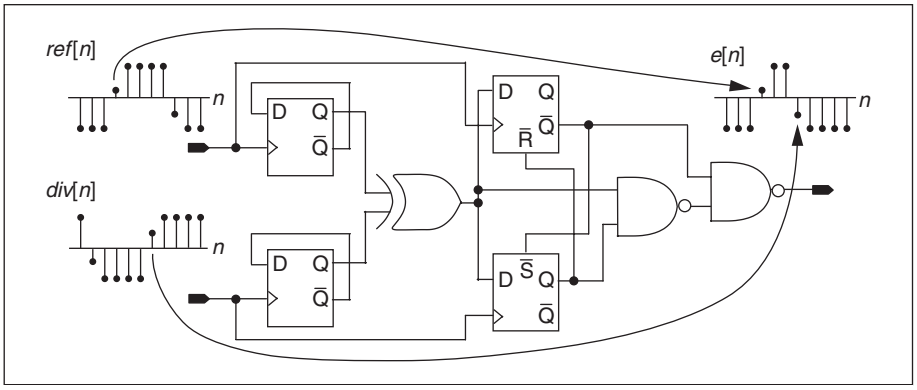

Figure 3. XOR-based PFD. In the figure, *ref*[*n*] is a DT version of the input frequency reference, *div*[*n*] is a DT version of the divided-down VCO output, D is the register input, Q is the register output, and R is the register reset input.

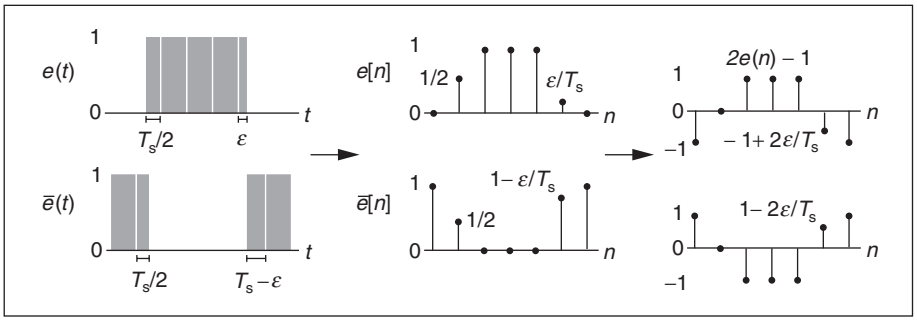

Figure 4. Achieving a better signal representation.

Implementation of PFD

Now that I've shown how to use area conservation to accurately represent the PFD output signal as a DT sequence, let's examine the practical issue of implementing this technique in simulation code. Of particular concern is the ability to encompass a wide variety of PFD topologies. As Figure 3 shows, computing $e[n]$ for a given PFD topology requires passing the transition information along, and using primitive elements such as registers and logic gates to process this information. This computation must also support basic operations such as calculating a signal's complement.

As Figure 4 shows, we can achieve the complement operation as a sign change by modifying the representation of $e[k]$ such that it alternates between –1 and 1, as opposed to 0 and 1. We accomplish this modification through the transformation, $e[n] \rightarrow 2e[n] - 1$.

Transferring transition information through primitives is straightforward, as Figure 5 (next page) shows for a register and representative AND logic gate. For the register, the clock signal contains the relevant timing information. Specifically, whenever a transition occurs at the register's output, the location of the clock's rising edge sets that transition's location in time. As Figure 5 shows, transferring this information to the register output involves simply passing along the clock transition value when the output transitions in the same direction. When the output transitions in the opposite direction, the complement of the clock transition value should be

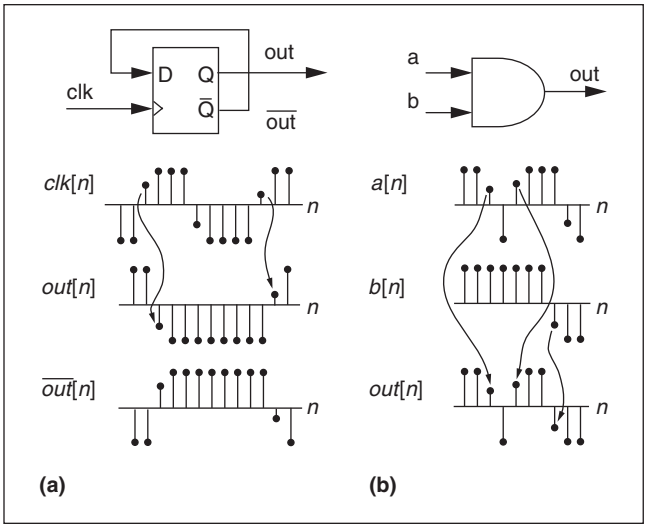

Figure 5. Example of register (a) and logic gate (b) signals.

passed on. For the AND gate, either input can cause the output to transition. As the figure illustrates, it is straightforward to determine which input is causing the transition and then appropriately pass this input's edge location value to the AND gate's output. Similar arguments apply to more complicated registers that include set and reset functions, and other primitives such as OR and XOR gates.

Fast computation

To improve computation speed, we set simulation sample period T_s according to the reference frequency rather than the much higher VCO frequency. Doing so usually achieves more than two orders of magnitude speedup in the PLL simulation time because the VCO frequency is typically higher than the reference frequency by this amount. We achieve this goal by combining the VCO and divider into one computation block.

To explain, let VCO phase $\Phi_{vco}(t)$ be the integral of the VCO's output frequency. The output frequency is adjusted about its nominal value by a function that depends on its input voltage, so the VCO phase becomes

$$\Phi_{vco}(t) = \int_{-\infty}^{t} 2\pi\left(K_v v(\tau) + f_c\right)d\tau + \Phi_{vn}(t) \quad (1)$$

where $v(t)$ is the VCO input voltage, K_v is the VCO gain (in hertz per volt), f_c is the nominal VCO frequency when $v(t) = 0$, and $\Phi_{vn}(t)$ is the VCO noise (illustrated in Figure 1b). Note that modeling a nonlinear relationship from the input voltage to the VCO frequency would require multiplying the VCO input by a polynomial gain expression rather than just K_v. In general, $\Phi_{vco}(t)$ looks like a ramp in time, and the VCO output's rising edges occur for each 2π increment of this output.

VCO simulation simply involves discretizing equation 1 as

$$\Phi_{vco}(nT_s) = \sum_{k=-\infty}^{n} 2\pi T_s\left(K_v v(kT_s) + f_c\right) + \Phi_{vn}(nT_s) \quad (2)$$

where n is the time index of the simulator. To prevent loss of information in the CT-to-DT conversion, $1/T_s$ must be higher than twice the highest frequency content of $v(t)$ and $\Phi_{vn}(t)$, according to the Nyquist theorem.[9] Practically speaking, meeting the sampling requirements for the PFD output will often satisfy this condition.

Rising edges of the divider output occur whenever the VCO output completes $N[m]$ rising edges, where $N[m]$ corresponds to the instantaneous divide value. Therefore, as Figure 6a shows, VCO phase $\Phi_{vco}(t)$ completely specifies the location of the divider edges. Hence, the value of ε_k at the divider output's transition points can be determined entirely by the computed VCO phase, as Figure 6b shows for first-order interpolation.[6] It suffices to choose a sample rate for the VCO phase computation according to the divider frequency, which equals the reference frequency, rather than the far higher VCO frequency.

Results

Using our techniques, I now discuss the results of simulating the dynamic behavior and noise performance of a prototype synthesizer.[2] The simulator consists of a custom C++ program employing the techniques. To verify its accuracy, simulated noise is compared to measured plots.

Figure 1a gives a block diagram of the prototype system. Both dynamic and noise simulations included the noise sources shown in Figure 1b. We used a white-noise source referenced to the VCO input to represent the VCO

noise.[2] Figure 7 shows the noise sources included in the simulation. We computed the variances of the charge-pump noise sources from Hspice simulations, and the variance of the VCO noise source from VCO measurements.

Relevant characteristics of the prototype include

- a 20-MHz reference frequency,
- 84-kHz PLL bandwidth,
- VCO with f_c = 1.84 GHz and K_v = 30 MHz/V,
- second-order Σ-Δ modulator,
- charge pump that outputs ± 1.5 μA, and
- nominal divide value of 92.3.

The prototype also includes the PFD topology shown in Figure 3,[10] and a lead/lag filter with transfer function

$$H(j\omega) = \frac{1 + j\omega / (2\pi f_z)}{C_3 j\omega \left(1 + j\omega / (2\pi f_p)\right)},$$

where j is $\sqrt{-1}$, and ω is frequency in radians. The loop-filter zero frequency, f_z, is 11.6 kHz; its pole frequency, f_p, is 127.2 kHz; and its integrator capacitance, C_3, is 30 pf.

The simulation sample frequency was $1/T_s$ = 400 MHz, a factor of 20 higher than the reference frequency. The bilinear transform was used to convert the CT loop filter to DT.[9] All simulations were run on a 650-MHz Pentium III laptop.

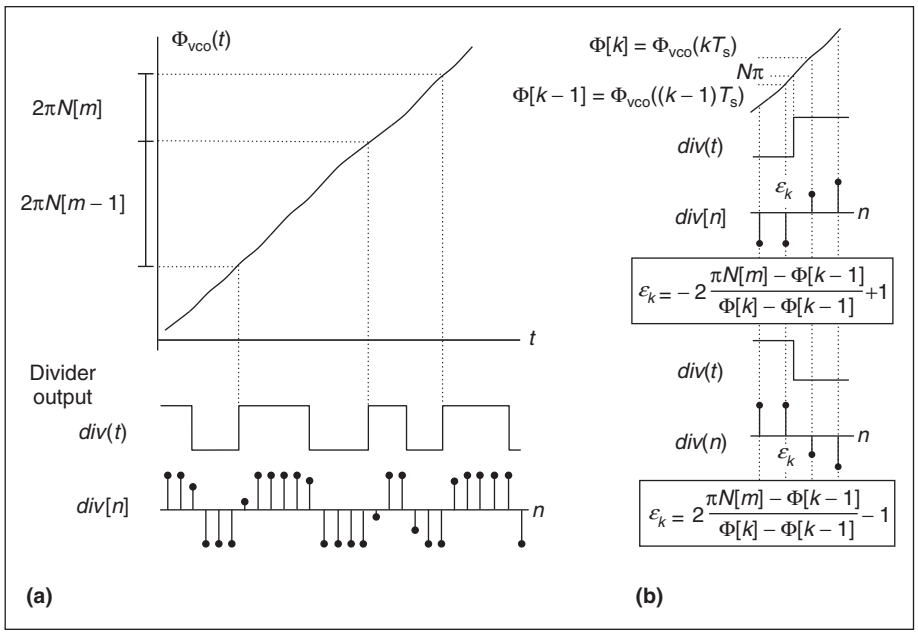

Figure 6. VCO and divider signals: relationship of divider edges to VCO phase $\Phi_{vco}(t)$ (a) and calculation of timing parameter ε_k (b), where $N[m]$ is the instantaneous divide value.

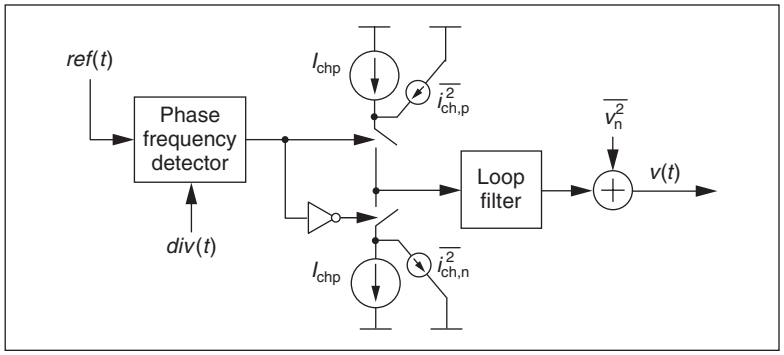

Figure 7. Model of a charge pump and the corresponding VCO noise, where I_{chp} is the current of the charge pump, $\overline{i^2_{ch,p}}$ and $\overline{i^2_{ch,n}}$ are the charge-pump noise sources, and $\overline{v_n^2}$ is the input-referred VCO noise source.

First, we consider dynamic behavior. Figure 8a (next page) shows the simulated VCO output frequency constructed from the simulated VCO input. The outputs are responses to variations at the Σ-Δ modulator's inputs (step and ramp functions). The step size was chosen large enough to knock the synthesizer out of frequency lock, to illustrate the nonlinear reaction of the PLL. The corresponding oscillations in the VCO output frequency were due to cycle slipping before the VCO became frequency locked again. The subsequent ramp in divide value illustrates the synthesizer's high resolution as we varied its output frequency over a 40-MHz range. For this simulation, the custom C++ simulator computed 260,000 time steps in less than 5 seconds.

Next, we consider how noise affects PLL behavior. Figure 8b shows a noise simulation of the prototype, constructed from the simulated VCO input, with the Σ-Δ modulator input held constant. The plot shows the simulated output

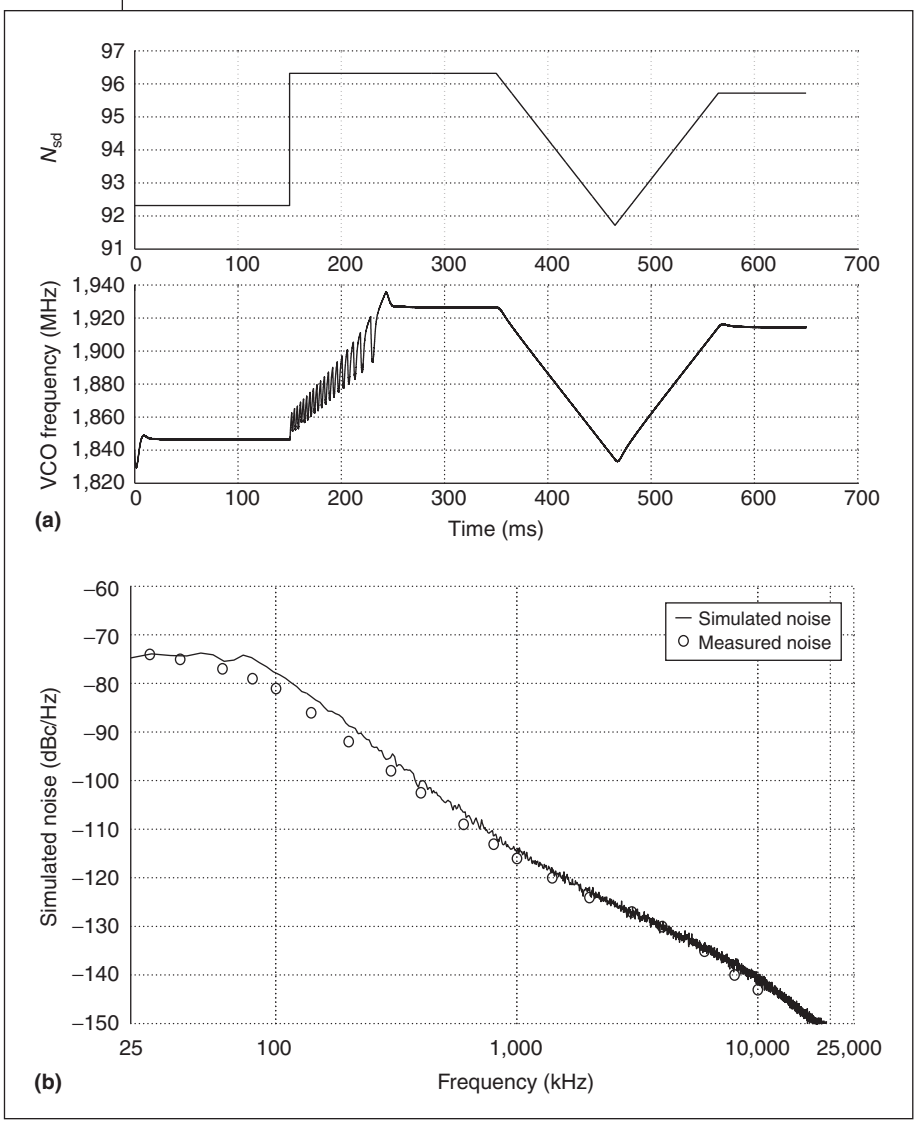

Figure 8. Simulated dynamic response of the synthesizer output to the Σ-Δ modulator input (a), and simulated synthesizer phase noise compared with measured synthesizer noise (b).

noise spectral density, with simulation sample frequency $1/T_s$ set to 20 times the reference frequency. The simulated phase noise agreed quite well with the measured results, which are from an earlier work.[2] The larger discrepancy at frequencies close to 100 kHz probably comes from not modeling the charge pump's nonideal characteristics, such as duty cycle offset and transient dynamics. You could, of course, include such effects in the given framework. However, even though the simulation ignored these effects, its results were still quite accurate. For this simulation, the custom C++ simulator computed 5 million time steps in 80 seconds.

Application to other circuits

We can apply the presented transition framework for discretizing the PFD output of other PLL circuits to achieve accurate simulation results. For example, consider the clock-and-data-recovery (CDR) circuit shown in Figure 9a.[11] This circuit reconstructs the clock associated with input data by appropriately adjusting a VCO's phase and frequency. The structure is essentially a PLL with a phase detector designed to handle the random transitions of the input data sequence.

Figures 9b and 9c illustrate two popular phase detectors used in CDR structures: Hogge and bang-bang.[11] Applying the Hogge detector leads to linear PLL dynamics, whereas the bang-bang detector leads to nonlinear PLL dynamics. In both cases, the phase detector consists of latches, registers, and XOR gates, and encodes phase error information using pulse width modulation (the bang-bang pulse width is quantized to one clock period). Therefore, issues associated with discretizing the CDR phase detector output are essentially the same as those for frequency synthesizers. We can, in fact, achieve accurate simulation results for CDR systems by applying the discretization technique presented here.

Using this technique, Figure 10 plots the simulated dynamic response of two different CDR structures: one with the Hogge detector and the other with the bang-bang detector. Table 1 (page 82) summarizes each CDR structure's relevant characteristics. We chose the bang-bang step size to achieve steady-state jitter comparable to the Hogge-based CDR structure. The Hogge-based CDR's dynamic response is expo-

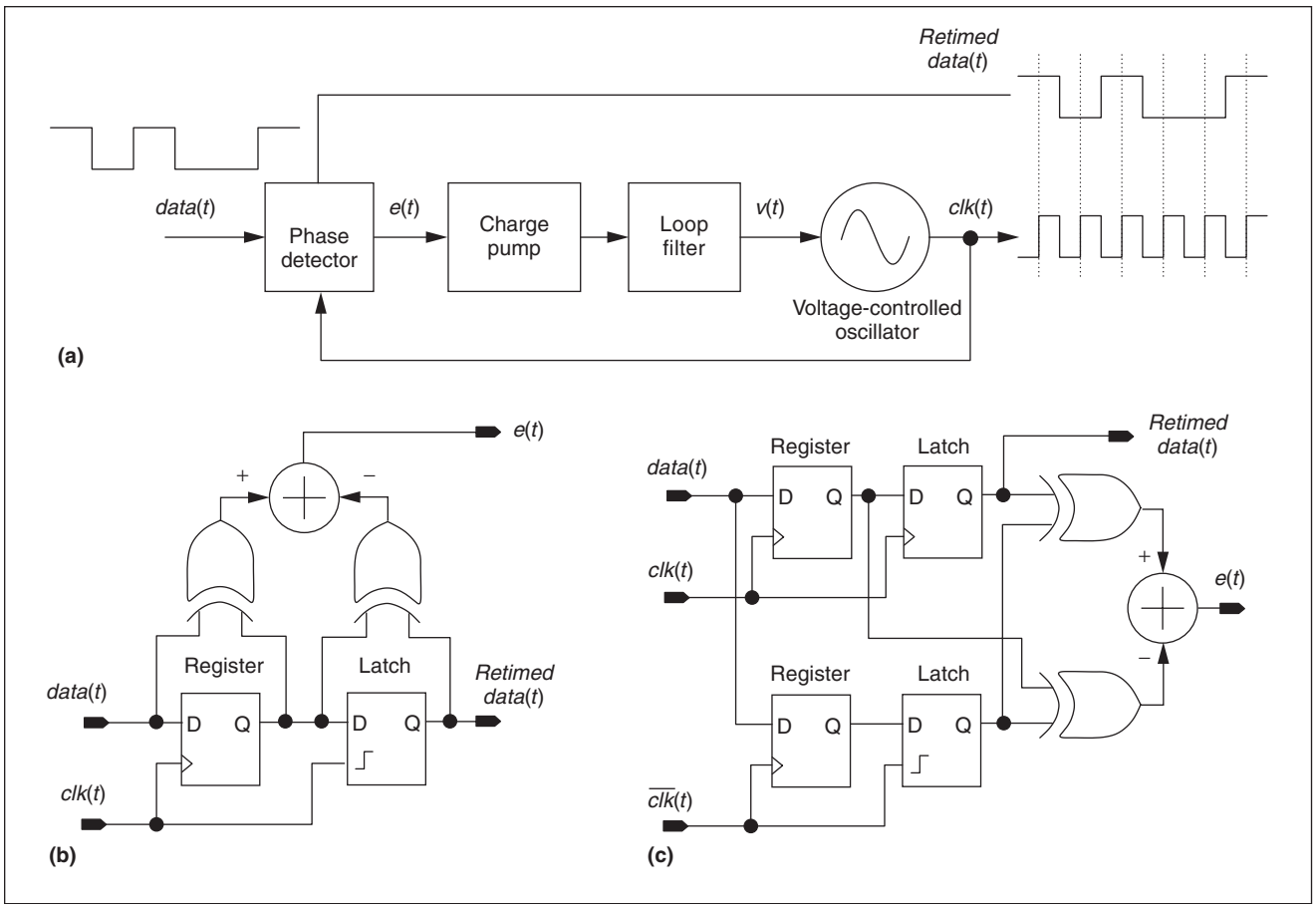

Figure 9. Clock-and-data-recovery (CDR) circuit (a); phase detectors for CDR applications: (linear) Hogge detector (b) and (nonlinear) bang-bang detector (c).

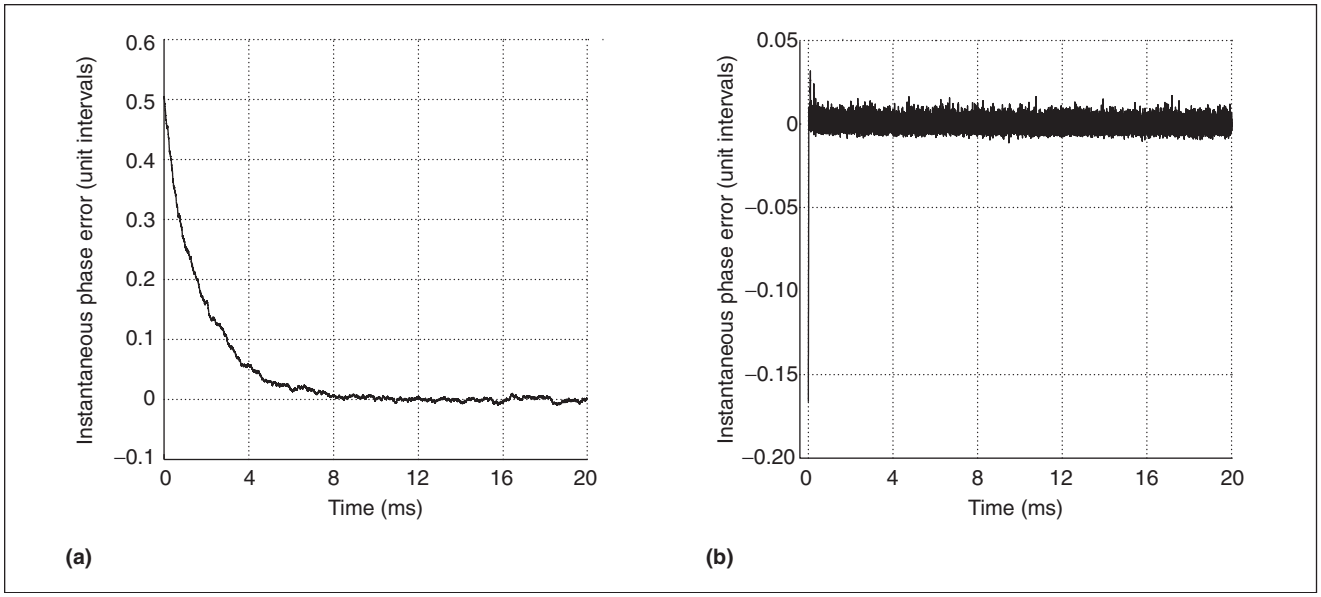

Figure 10. Simulated dynamic response of instantaneous phase error for the CDR structure with Hogge (a) and bang-bang (b) detectors. The calculated steady-state RMS jitter was 3.0756 mUI for the former and 3.4598 mUI for the latter.

Table 1. Properties of two simulated CDR structures.

Property	CDR with Hogge detector	CDR with bang-bang detector
Data rate (Gbits/s)	2.5	2.5
Locked VCO frequency (GHz)	2.5	2.5
PLL bandwidth (MHz)	1	NA
VCO noise at 1 MHz (dBc/Hz)	–100	–90
Sample rate, $1/T_s$ (GHz)	15	15
No. of time steps	500,000	300,000
Computation time (s)	5	3

nential, as expected for a PLL with linear dynamics. The CDR structure with the bang-bang detector is nonlinear, allowing fast elimination of the initial phase error. In both cases, the steady-state jitter was around 3×10^{-3} unit intervals (that is, 3 mUI). This jitter level is far smaller than the simulator's 16.7-mUI time step (determined as the ratio of 2.5 GHz to 15 GHz). Thus, our discretization technique lets designers use a relatively coarse simulation period T_s while still achieving fine resolution in calculating the CDR jitter performance.

THESE RESULTS SHOW that the presented techniques appear very promising in estimating the noise and dynamic behavior of phase-locked loops at an ideal behavioral level. Future work will focus on representing nonideal characteristics in circuit components in the presented simulation framework, and implementing the technique in the Verilog language. We are also investigating methods of using this simulation framework to estimate the noise impact of digital circuits on phase-locked loops embedded in ICs.

The "Progress in analog-circuit modeling and simulation" sidebar gives another perspective on the challenges involved in simulating mixed-signal circuits. ■

■ References

1. T.A. Riley, M.A. Copeland, and T.A. Kwasniewski, "Delta-Sigma Modulation in Fractional-N Frequency Synthesis," *IEEE J. Solid-State Circuits*, vol. 28, no. 5, May 1993, pp. 553-559.
2. M. Perrott, T. Tewksbury, and C. Sodini, "A 27 mW CMOS Fractional-N Synthesizer Using Digital Compensation for 2.5 Mb/s GFSK Modulation," *IEEE J. Solid-State Circuits*, vol. 32, no. 12, Dec. 1997, pp. 2048-2060.
3. K. Kundert, J. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*, Kluwer Academic, Boston, 1990.
4. D. Johns and K. Martin, *Analog Integrated Circuit Design*, John Wiley & Sons, New York, 1997.
5. B. De Smedt and G. Gielen, "Nonlinear Behavioral Modeling and Phase Noise Evaluation in Phase Locked Loops," *Proc. IEEE Custom Integrated Circuits Conf.* (CICC 98), IEEE Press, Piscataway, N.J., 1998, pp. 53-56.
6. A. Demir et al., "Behavioral Simulation Techniques for Phase/Delay-Locked Systems," *Proc. IEEE Custom Integrated Circuits Conf.* (CICC 94), IEEE Press, Piscataway, N.J., 1994, pp. 453-456.
7. M. Hinz, I. Konenkamp, and E.-H. Horneber, "Behavioral Modeling and Simulation of Phase-Locked Loops for RF Front Ends," *Proc. 43rd IEEE Midwest Symp. Circuits and Systems*, vol. 1, IEEE Press, Piscataway, N.J., 2000, pp. 194-197.
8. M.H. Perrott, "Fast and Accurate Behavioral Simulation of Fractional-N Frequency Synthesizers and Other PLL/DLL Circuits," *Proc. 39th Design Automation Conf.* (DAC 2002), ACM Press, New York, 2002, pp.498-503.
9. A.V. Oppenheim and R.W. Schafer, *Discrete Time Signal Processing*, Prentice Hall, Upper Saddle River, N.J., 1999.
10. A. Hill and A. Surber, "The PLL Dead Zone and How to Avoid It," *RF Design*, Mar. 1992, pp. 131-134.
11. B. Razavi, *Monolithic Phase-Locked Loops and Clock Recovery Circuits: Theory and Design*, IEEE Press, Piscataway, N.J., 1996.

Progress in analog-circuit modeling and simulation

Georges Gielen,
Katholieke Universiteit Leuven

Numerical simulation of mixed-signal circuits—whether analog or digital, RF or base band—is challenging. Difficult issues include circuit complexity, large differences in the simulation time scale, and mixing of time and frequency domain behaviors. Recently, researchers have made considerable progress in analog, mixed analog-digital, and multilevel analog simulation. Several commercial solutions are available that simulate analog and mixed-signal circuits at the device level, sometimes with additional RF functionality.

Simulators have also begun allowing simulation at higher abstraction levels, such as macromodel, behavioral, and functional. Top-down design methodologies need higher-level models to describe the circuits' pin-to-pin behavior rather than its internal structural implementation. Moreover, verifying entire integrated systems is too computationally complex to allow a full device-level simulation. Finally, when providing analog IP macrocells for, or using them in, a system on a chip, designers must accompany the virtual component with an executable model that efficiently models its pin-to-pin behavior. Such a model is necessary for system-level design and verification. Standardized mixed-signal hardware description languages (HDLs), such as VHDL-AMS and Verilog-AMS, can help. But generating models for analog designs is still a problem because today's designers are not skilled at this and no systematic modeling approaches have been developed yet. In fact, it may prove the biggest hurdle for the adoption of these high-level modeling methodologies and analog HDLs in industrial design practice. Automatic generation of analog models is thus a very active research topic.

One of the most difficult analog blocks to simulate is the phase-locked loop (PLL), or frequency synthesizer, because of its combination of high-frequency oscillator signals (imposing small time steps in Spice) and relatively long-term phenomena such as acquisition and locking.[1-2] For several years, researchers have applied faster, simplified simulation techniques—for example, by using linearized models—and more general nonlinear behavioral modeling methods to synthesizers. In recent years, fractional-N synthesizer architectures have been introduced. These use a Σ-Δ modulator to dither the divide value. The article by Perrott presents techniques for fast and accurate simulation of dynamic performance and noise effects for fractional-N synthesizers and other PLL circuits at a detailed behavioral level. These techniques allow the use of a uniform time step. Perrott and his colleagues at MIT's Microsystems Technology Laboratory have incorporated them into an efficient, custom C++ simulator.

References

1. B. De Smedt and G. Gielen, "Models for Systematic Design and Verification of Frequency Synthesizers," *Proc. IEEE Trans. Circuits and Systems II*, vol. 46, no. 10, Oct. 1999, pp. 1301-1308.
2. F.M. Gardner, *Phaselock Techniques*, 2nd ed., John Wiley & Sons, New York, 1979.

Georges Gielen is a professor at Katholieke Universiteit Leuven. Contact him at georges.gielen@esat.kuleuven.ac.be.

Michael H. Perrott is an assistant professor in the Electrical Engineering and Computer Science Department at the Massachusetts Institute of Technology. His research interests include high-speed circuit and signal-processing techniques for data links and wireless applications. Perrott has a BS in electrical engineering from New Mexico State University, and an MS and PhD in electrical engineering and computer science from MIT. He is a member of the IEEE.

■ Direct questions and comments about this article to Michael H. Perrott, Assistant Professor, EECS, MIT 38-344B, 50 Vassar St., Cambridge, MA 02139; perrott@mit.edu.