Fast and Accurate Behavioral Simulation of Fractional-N Frequency Synthesizers and other PLL/DLL Circuits

Michael H. Perrott
Microsystems Technology Laboratory, MIT, Rm 38-344B
Cambridge, MA 02139
perrott@mit.edu, 617.452.2889

ABSTRACT

Techniques for fast and accurate simulation of fractional-N synthesizers at a detailed behavioral level are presented. The techniques allow a uniform time step to be used for the simulator, and can be applied to a variety of phase locked loop (PLL) and delay locked loop (DLL) circuits beyond fractional-N synthesizers, as well as to a variety of simulation frameworks such as Verilog and Matlab. Implementation in code, as well as simulated results, are presented using a custom C++ simulator, and compared to calculated and measured results from a prototype fractional-N synthesizer using a Σ-Δ modulator to dither its divide value.

1. INTRODUCTION

Fractional-N frequency synthesizers provide high speed frequency sources that can be accurately set with very high resolution, which is of high value to many communication systems. Figure 1 illustrates a fractional-N synthesizer, which consists of a phase-frequency detector (PFD), charge pump, loop filter, voltage controlled oscillator (VCO), and a frequency divider that is dithered between integer values to achieve fractional divide ratios. As the figure reveals, this paper will focus on a class of fractional-N synthesizers known as Σ-Δ frequency synthesizers [11], for which the divide value is dithered according to the output of a Σ-Δ modulator [7].

Dithering of the divide value by the Σ-Δ modulator allows high frequency resolution to be achieved [11], but also has the negative side effect of introducing quantization noise that degrades the overall PLL noise performance. It is highly desirable to be able to simulate the effects of this quantization noise, along with other noise sources in the PLL shown in Figure 2, on the overall PLL performance. It is also desirable to simulate the dynamic response of the synthesizer in response to variations of the Σ-Δ input in order to evaluate stability and characterize the performance of the system when it is used as a transmitter [9].

Simulation of fractional-N synthesizers is particularly challenging for a variety of reasons. First, the high output frequency of the synthesizer (often in the GHz 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 in frequency than the output frequency (often 100 kHz to 1 MHz bandwidth compared to a GHz output frequency). Thus, traditional simulators take a long time to compute the dynamic response of the system since many simulation samples are required, which is the classical problem that is encountered with the simulation of PLL circuits. For noise simulation, the fractional-N synthesizer adds the additional constraint that its behavior is non-periodic in steady-state due to the dithering action of the divide value, which prevents the use of methods developed for periodic steady-

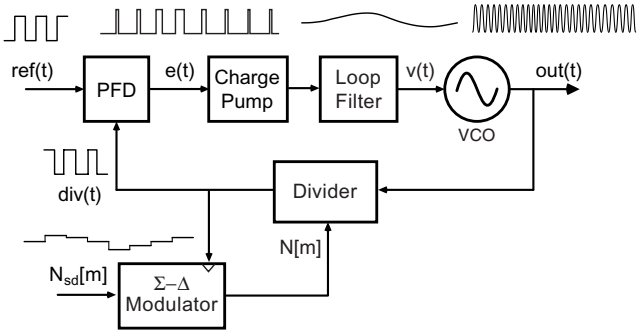

Figure 1: Σ-Δ synthesizer and associated signals.

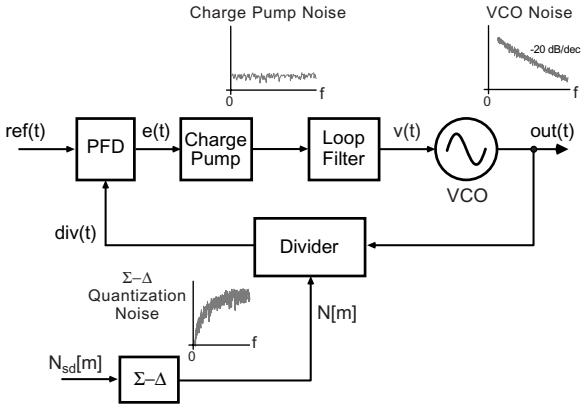

Figure 2: Spectral densities of PLL noise sources.

DAC 2002, June 10-14, New Orleans, LA USA

state conditions [6] as used with simulators such as SpectreRF. Thus far, the author is aware of no simulation tools that allow accurate simulation of the noise performance of fractional-N frequency synthesizers.

In contrast to the above approaches, two techniques are presented in this paper that allow fast and accurate simulation of both dynamic and noise performance of fractional-N synthesizers at a detailed behavioral level. The first provides accurate representation of the continuous-time (CT) PFD output with a discrete-time (DT) sequence using an area conservation principle. The second allows a dramatic reduction of the simulation sample frequency, and therefore a longer sample period, by including the divider implementation in the VCO simulation module. Both of these methods allow a uniform time sample period to be used, and also allow non-iterative computation of the sample values of the various signals within the system. The uniform time sample period allows the results of the simulator to be readily examined in the frequency domain without resampling, and the non-iterative computation allows the technique to be easily used in mainstream simulators such as Verilog, VHDL, Matlab, and custom C/C++ programs.

An outline of the paper is as follows. Section 2 provides an overview of the discretization technique for representation of the CT PFD output with a DT sequence, and presents the corresponding mathematical analysis. Section 3 describes, at a high level, how the discretization technique can be implemented with the PFD described in terms of basic building blocks such as registers and logic gates — this allows the designer to easily use the approach for a variety of PFD topologies. Section 4 focuses on the method of dramatically reducing the required simulation sample rate by combining the VCO and divider functions into one simulation block. Section 5 provides example code for a Σ-Δ synthesizer using a custom C++ simulator and compares the simulated noise performance to calculated and measured results of an actual circuit implementation described in [9]. Finally, Section 6 concludes.

2. PFD DISCRETIZATION TECHNIQUE

For simulation based on uniform time sampling, a straightforward approach of converting the CT PFD output to a DT sequence is to apply a simple sampling operation as shown in Figure 3. Unfortunately, this approach effectively quantizes the location of the PFD edges according to the simulation sample period, T_s. A reasonable assessment of the dynamic performance of the PLL can be achieved if T_s is made sufficiently small. However, the resulting quantization noise overpowers the true noise characteristics of the signals, and prevents proper noise analysis of the overall PLL.

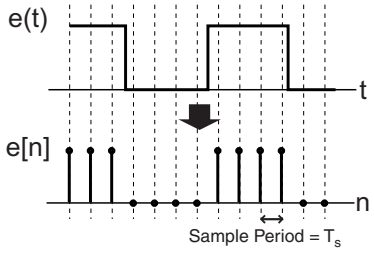

Figure 3: Classical uniform time sample method.

To solve the quantization noise issue, event-driven simulation methods have been developed for classical frequency synthesizers that align simulation samples precisely to the edges of the PFD output [2, 5, 1]. Although higher accuracy can be achieved with such methods, they are generally more complicated than uniform time sampling methods. Either closed-form calculation of the loop filter step response must be developed and then inserted into the simulation, or iterative methods, as used in SPICE or Verilog-A, must be incorporated into the simulator to calculate the loop filter response with varying time steps. The former approach is tedious and typically restricted to a low loop filter order, so that most of the recent methods focus on the latter approach [2, 5, 1]. In this case, the up-front work of the designer is minimized, but the simulation time is often longer due to the iterative calculations that are performed at each time step. Unfortunately, for either case, event-driven simulators have not yet been applied successfully to the noise analysis of fractional-N frequency synthesizers in which the divide value is dynamically varied.

In contrast to the above approaches, a constant time step method is proposed in this paper that applies an area conservation principle when converting the CT PFD output to the DT domain. This approach allows non-iterative computation of the loop filter dynamics by allowing them to be converted from CT to DT using either impulse invariance or bilinear transform methods [8]. Figure 4 illustrates an example of the resulting DT PFD signal, along with the corresponding DT loop filter impulse response. The charge pump is ignored in this analysis for simplicity; its effect can be included by simply scaling the PFD output by the value of the charge pump current. In the example, we see that the DT PFD output takes on values at its transitions that vary between 0 and 1 depending on the location of the transition edge. The DT version of the loop filter simply consists of a DT filter whose impulse response corresponds to samples of the CT impulse response of the loop filter, $h(t)$.

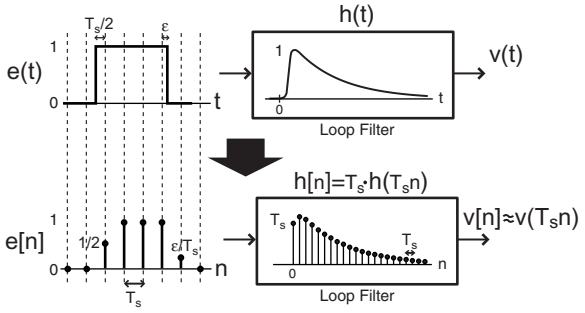

Figure 4: Proposed discretization method.

The discretization procedure is now discussed in more detail. As illustrated by Figure 5, we can view the CT PFD output, $e(t)$, as a series of rectangular pulses with height of one or zero and a width and time offset that varies according to the location of PFD edges. For rectangular pulses not associated with edges, the width corresponds to the sample period of the simulation, T_s. For rectangular pulses at edge boundaries, the width of the pulse varies between 0 and T_s as shown in the figure. In either case, these pulses look like impulses to the loop filter so that, from an intuitive standpoint, their influence can be characterized by two parame-

ters — their area and time offset. Therefore, in line with this intuition, the corresponding discrete-time PFD signal, $e[n]$, is chosen as samples that have amplitude proportional to the *area* of the respective rectangular pulse in that time sample interval. The area of each pulse corresponds to its associated timing parameter ϵ shown in the figure — the method of calculating ϵ for each pulse will be discussed in Section 4. It will be shown that the proposed discretization procedure yields highly accurate results, fast computation, and a simple implementation framework.

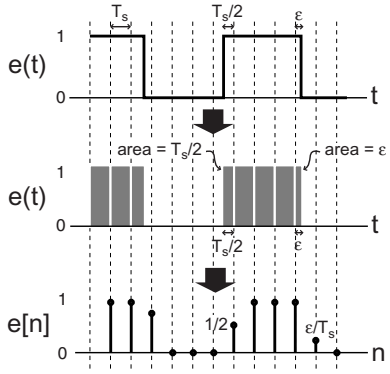

Figure 5: Details of PFD discretization technique.

To mathematically justify the technique, let us begin by specifying a notation for the rectangular pulses composing $e(t)$ that are shown in Figure 5, namely

$$\text{rect}(t, \epsilon_k) = 1 \text{ for } -\epsilon_k/2 \le t \le \epsilon_k/2, \quad 0 \text{ elsewhere.}$$

Given the above notation, we can describe the loop filter output as

$$v(t) = \sum_{k=-\infty}^{\infty} \text{rect}(t - kT_s - \Delta t_k, \epsilon_k) * h(t), \qquad (1)$$

where $*$ denotes convolution, and ϵ_k and Δt_k are constrained to $0 \le \epsilon_k \le T_s$ and $-T_s < \Delta t_k < 0$.

Taking the Fourier Transform of both sides of Equation 1, we obtain

$$V(jw) = \sum_{k=-\infty}^{\infty} e^{-jw(kT_s+\Delta t_k)} \frac{2\sin((\epsilon_k/2)w)}{w} H(jw), \qquad (2)$$

where $H(jw)$ is the loop filter frequency response.

One might be bothered that the Fourier Transform is being taken with sequences that are stochastic in nature, namely ϵ_k and Δt_k. This step is justified by noting that these sequences will be defined for a specific simulation run, and have finite duration (i.e., $\epsilon_k = 0$ for $|k| > P$, where the number of simulation samples is $2P + 1$). Given these conditions, Equation 2 will be valid, though we cannot infer statistical properties from its formulation or the analysis that follows.

Equation 2 can be simplified in light of the following assumptions:

- $H(jw)$ is a lowpass filter such that $|H(jw)| \approx 0$ for $|w| > w_o$,
- The sample period, T_s, is much smaller than the time constant of $|H(jw)|$, so that $w_o T_s \ll 1$. Since $|\epsilon_k| < T_s$, we have $\sin((\epsilon_k/2)w_o) \approx (\epsilon_k/2)w_o$.

The second assumption is well justified in practice since it is typical for $w_o T_s \ll 1/100$. For instance, the author recommends sampling at a rate that is greater than a factor of 10 above the reference frequency, which is, in turn, at least a factor of 10 higher in frequency than the loop filter bandwidth in Hz, f_o, to achieve stable PLL dynamics [10]. In this case, $f_o T_s < 1/100$, so that $w_o T_s < 1/(2\pi 100)$.

Based on the above assumptions, Equation 2 is approximated as

$$V(jw) \approx \sum_{k=-\infty}^{\infty} \epsilon_k e^{-jw(kT_s+\Delta t_k)} H(jw).$$

The inverse Fourier Transform of the above expression is

$$v(t) = \sum_{k=-\infty}^{\infty} \epsilon_k h(t - kT_s - \Delta t_k). \qquad (3)$$

We are now ready to develop the discrete-time model of the PFD/loop filter section that we are seeking. We begin by sampling Equation 3:

$$v(nT_s) = \sum_{k=-\infty}^{\infty} \epsilon_k h((n-k)T_s - \Delta t_k). \qquad (4)$$

The above formulation requires nonconsistent sampling of $h(t)$ due to the inclusion of Δt_k — it is preferable to remove this parameter if it can be shown that its influence is negligible. We will examine this issue using a specific form for $h(t)$, and then comment on the extension of the analysis for more general forms of $h(t)$.

Let us assume that the loop filter corresponds to a lead/lag network with transfer function of the form:

$$H(jw) = K\frac{jw + w_z}{jw(jw + w_o)}.$$

Using the method of partial fractions [8], it is straightforward to show that the corresponding loop filter impulse response is of the form

$$h(t) = K_1 e^{-w_o t}u(t) + K_2 u(t) = (K_1 e^{-w_o t} + K_2)u(t),$$

where $u(t)$ is the unit step (0 for $t < 0$, 1 for $t \ge 0$), and K_1 and K_2 are constant scale factors. Plugging the above expression into Equation 4, we obtain

$$v(nT_s) = \sum_{k=-\infty}^{\infty} \epsilon_k (K_1 e^{-w_o((n-k)T_s-\Delta t_k)} + K_2)u((n-k)T_s - \Delta t_k). \qquad (5)$$

We note that:

$$\begin{aligned} u((n-k)T_s - \Delta t_k) &= u((n-k)T_s) \\ &\quad \text{since } -T_s < \Delta t_k < 0, \end{aligned}$$

and

$$\begin{aligned} e^{-w_o((n-k)T_s-\Delta t_k)} &= e^{w_o \Delta t_k} e^{-w_o(n-k)T_s} \\ &\approx (1 + w_o \Delta t_k) e^{-w_o(n-k)T_s}. \end{aligned}$$

Therefore, the effect of the time shift operation by Δt_k has no influence on samples of $u(t)$, and only slightly modulates the amplitude of samples of the exponential response $e^{-w_o t}$. Although its effect could be incorporated into the numerical model, the author has found that it can be safely ignored given that two conditions are met. The first condition is that $w_o T_s$ be much less than 1 so that $1 + w_o \Delta t_k \approx 1$. This condition is satisfied in practice; it was argued earlier in

this section that we can typically expect that $w_o T_s \ll 1/100$. The second condition is that the sample rate of the simulator, $1/T_s$, be chosen as an integer multiple of the nominal frequency of the pulses associated with the CT PFD output. The effect of violating either of these conditions is the introduction of false spurs in the output phase noise of the synthesizer, as will be demonstrated in Section 5.

Given that the above conditions are satisfied, we can simplify Equation 5 as

$$v(nT_s) = \sum_{k=-\infty}^{\infty} \epsilon_k (K_1 e^{-w_o(n-k)T_s} + K_2) u((n-k)T_s),$$

so that, for this case, we have

$$v(nT_s) = \sum_{k=-\infty}^{\infty} \frac{\epsilon_k}{T_s} T_s h((n-k)T_s). \quad (6)$$

Equation 6 is the conclusion of our effort, and matches the picture representation of the method illustrated in Figure 4 when $e[n] = \epsilon_n / T_s$.

Although the above analysis was performed for a simple lead/lag loop filter, higher order filters can be analyzed in similar fashion using the partial fraction expansion method. Specifically, high order filters have impulse responses that consist of a sum of exponentials, with each exponential corresponding to a distinct pole in the loop filter. The impact of e_k and Δt_k on each of these exponentials can be assessed in the same manner as derived above.

3. IMPLEMENTATION OF PFD

Now that it has been established that the PFD output signal can be accurately represented as a discrete-time sequence using a principle of area conservation, let us examine the practical issue of representing the PFD topology in simulation code. It will be shown that the technique accommodates a wide variety of PFD topologies by allowing their 'construction' in the simulation code using primitive elements such as registers and logic gates. Although the framework will be explained in terms of a custom C++ simulator developed by the author, the method can be implemented in a variety of frameworks including Verilog and Matlab.

Figure 6 illustrates an XOR-based PFD [4] that is implemented in Section 5 using the custom C++ simulator. In this case, the registers, nand gates, and xor gate are implemented as objects reg1, reg2, and1, xor1, etc. Each object has associated output signals, which are designated as reg1.out, reg2.out, and1.out, etc. Note that the nand gates are implemented as 'and' gates whose outputs are complemented with a sign change.

One should notice in Figure 6 that the signals vary between -1 and 1 as opposed to 0 and 1 as assumed in the preceding sections. The reason for the change is that the area conservation principle is easier to explain when signals vary between 0 and 1, but the preferred practical implementation is to specify signals that vary between -1 and 1 since it allows straightforward complement operations. This point is demonstrated by Figure 7, which illustrates an example $e(t)$ and its complement $\overline{e}(t)$ along with their discrete-time counterparts, $e[n]$ and $\overline{e}[n]$. By performing the linear operation $e[n] \Longrightarrow 2e[n] - 1$, we see that the transformed signal $x[n]$ is now related to its complement through a simple sign change.

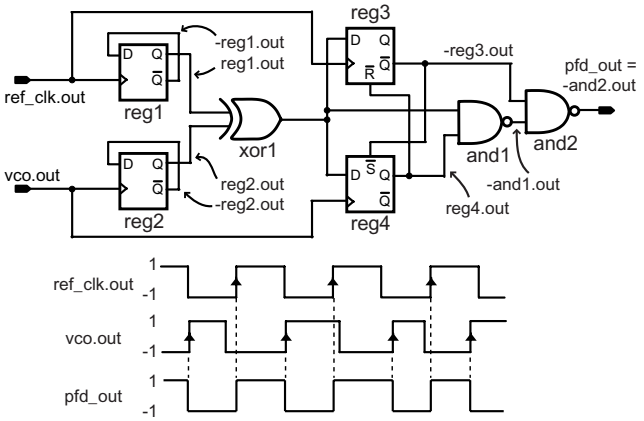

Figure 6: XOR-based PFD.

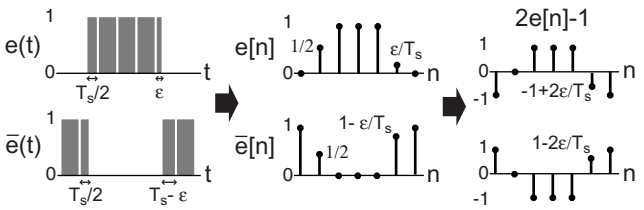

Figure 7: A signal representation allowing sign inversion to implement the complement operation.

Implementation of the PFD as shown in Figure 6 requires that computation of the PFD output be separable into subcomputations associated with primitive elements such as registers and logic gates. It will now be demonstrated that this is indeed possible by showing that primitive elements can process relevant information pertaining to the location of edges of the reference clock and divided down VCO clock and transfer that information to other primitives.

Figure 8 illustrates example input and output signals associated with a register and a representative logic gate, namely the 'and' gate. In the case of the register, the relevant timing information is contained in the clock signal. Specifically, whenever there is a transition at the output of the register, the location of that transition in time is set by the location of the rising (or falling) edge of the clock. The transition location is uniquely specified by the clk transition value (a real number in the range of -1 to 1), and by the prior clk sample value to determine whether the clk edge is rising (prior clk sample = -1) or falling (prior clk sample = 1). As shown in the figure, this information is transfered to the register output by simply passing on the clk transition value when the output transitions in the same direction, and passing on the complement of the clk transition value when the output transitions in the opposite direction. In the case of the 'and' gate, either input can cause the output to transition. As gleaned from the figure, it is straightforward to determine which input is causing the transition, and appropriately pass its edge location value to the output of the 'and' gate. Similar arguments can be made for more complicated registers that include set and reset functions, and other primitives such as 'or' and 'xor' gates.

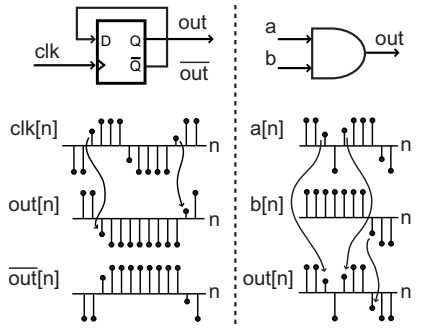

Figure 8: Example of register and logic gate signals.

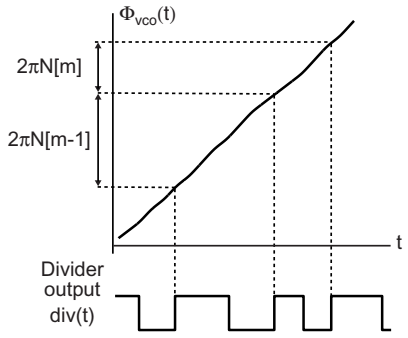

Figure 9: Relationship of divider transitions to unwrapped VCO phase with varying divide value.

4. VCO AND DIVIDER SIMULATION

Simulation of the VCO and divider portions of the PLL is now discussed, and a technique illustrated whereby the simulation sample period, T_s, can be set according to the reference frequency rather than the much higher VCO frequency. This technique typically allows more than two orders of magnitude speedup in simulation time of the PLL since the VCO frequency is typically more than two orders of magnitude higher than the reference frequency. Unlike a previous method that provided speedup for a PLL with no divider and a memoryless phase detector by modeling the VCO entirely in the phase domain [12], the presented technique accommodates fractional-N synthesizers that have dividers with dynamically varying value and digital PFD topologies as described in the previous sections. The key idea behind the technique is to combine the VCO and divider into one computation block.

To begin, let us define the phase of the VCO, $\Phi_{vco}(t)$, as the integral of its output frequency. Since the output frequency of the VCO is varied about its nominal frequency by its input voltage, we have:

$$\Phi_{vco}(t) = \int_{-\infty}^{t} 2\pi(K_v v(\tau) + f_c)d\tau + \Phi_{vn}(t), \qquad (7)$$

where $v(t)$ is the VCO input voltage, K_v is the VCO gain (Hz/V), f_c corresponds to the nominal VCO frequency when $v(t) = 0$, and $\Phi_{vn}(t)$ is VCO noise as illustrated in Figure 2. In general, $\Phi_{vco}(t)$ looks like a ramp in time, and rising edges of the VCO output occur every time this signal increments by 2π radians. Rising edges of the divider output occur every $N[m]$ rising edges of the VCO output, where $N[m]$ corresponds to the instantaneous divide value. Therefore, as illustrated in Figure 9, the VCO phase, $\Phi_{vco}(t)$, completely specifies the location of the divider edges.

Since $\Phi_{vco}(t)$ completely characterizes the VCO and divider, simulation of these two blocks can be performed by simply discretizing Equation 7 as

$$\Phi_{vco}(nT_s) = \sum_{k=-\infty}^{n} 2\pi T_s(K_v v(kT_s) + f_c) + \Phi_{vn}(nT_s). \quad (8)$$

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 $\Phi_{vn}(t)$, as stated by the Nyquist theorem [8]. From a practical perspective, this condition will often be satisfied by meeting the sampling requirements for the PFD output. Choosing a sample rate such that $w_o T_s \ll 1/100$ is obviously sufficient for $v(t)$ since it is the output of the loop filter with bandwidth w_o rad/s. Practically speaking, $\Phi_{vn}(t)$ is also bandlimited since it rolls off at -20 dB/decade, or more, before eventually hitting a low valued noise floor [3]. To examine the effect of VCO noise, the value of T_s should be chosen to allow evaluation of the phase noise performance over the frequency offsets of interest.

Assuming all information of relevance related to the VCO and divider is computed using Equation 8, the question that remains is how one would model the divider output. In this paper, the approach taken is to represent the divider output in a consistent fashion as discussed for the PFD output. Namely, the divider output has value either 1 or -1 for samples not coincident with edges, and takes on a real value in the range of -1 to 1 for samples that are coincident with edges, as discussed in Section 2 for the PFD output representation. Note that this method is also applied to the reference frequency output.

The most expedient way to explain the technique is to present its implementation in code. As such, a simplified version of the input function associated with class Vco in the custom C++ simulator is shown below. The first part of the code wraps the VCO phase to maintain numerical accuracy throughout the simulation — leaving the phase unwrapped will cause the phase value to increase over time, which swamps out variations due to noise and, in turns, leads to the introduction of numerical 'noise'. The second part of the code flips the state of the divider output depending on the range of the VCO phase. The divider output is 1 between 0 and πN radians of VCO phase, and -1 between πN and $2\pi N$ radians of VCO phase. Samples at transition boundaries are calculated by interpolation of the VCO phase to determine the precise edge location. Note that phase noise associated with the VCO, $\Phi_{vn}(t)$, is not included in the code. Rather, this noise is referred to the input of the VCO [9] as discussed in the following section. Also, while a linear relationship from input voltage to output frequency has been assumed for the VCO for simplicity, a nonlinear relationship is easily accommodated by multiplying the VCO input by a polynomial gain expression rather than just K_v.

```
double Vco::inp(double in, double fc, int divide_val)
{
phase = prev_phase + sample_period*2.0*PI*(Kv*in+fc);
if (phase >= 2*PI*N) // wrap VCO phase for accuracy
  {
   phase -= 2*PI*N;
   if (prev_phase >= phase)
```

```
      prev_phase −= 2∗PI∗N;
   N = (double) divide_val;
  }
if (phase >= 0.0 && phase < PI∗N) // state = 1
  {
    if (clk_state == 1) // no transition
       out = 1.0;
    else  // compute sample value for transition
       out = (phase+prev_phase)/(phase−prev_phase);
    clk_state = 1;
  }
else // state = 0
  {
    if (clk_state == 0) // no transition
       out = −1.0;
    else  // compute sample value for transition
       out = (2∗PI∗N−(phase+prev_phase))/
                 (phase−prev_phase);
    clk_state = 0;
  }
prev_phase = phase;
return(out);
}
```

5. RESULTS

To test the validity of the proposed simulation techniques, the results of simulating the dynamic behavior and noise performance of a prototype synthesizer described in [9] are now compared to corresponding calculated and measured results. Figure 1 provides a block diagram of the prototype system; the reader is referred to [9] for more details. Noise analysis will include the noise sources depicted in Figure 2, with VCO noise being input referred as a white noise source as described in [9]. Parameters associated with the noise sources are shown in Figure 10, which were computed from Hspice simulations and VCO measurements.

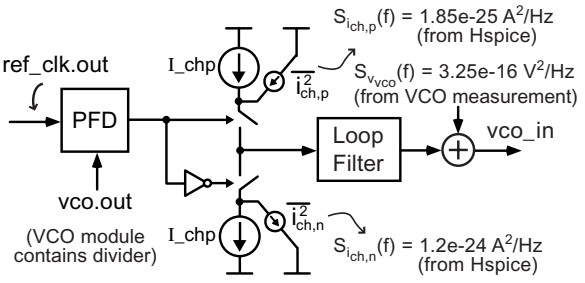

Figure 10: Model of charge pump and VCO noise.

Relevant characteristics of the prototype include a reference frequency of 20 MHz, a VCO with $f_c = 1.84$ GHz and $Kv = 30$ MHz/V, a second order Σ-Δ modulator, a charge pump that outputs ±1.5 μamps, a nominal divide value of 92.3, a PFD topology as shown in Figure 6, and a lead/lag filter with transfer function

$$H(jw) = \frac{1 + jw/(2\pi f_z)}{C_3 jw(1 + jw/(2\pi f_p))},$$

where

$$f_z = 11.6 \text{ kHz}, \quad f_p = 127.2 \text{ kHz}, \quad C_3 = 30\text{e-}12.$$

Simulation code for the prototype system is shown below. The simulator consists of a set of custom written C++ classes that have been placed in the library 'com_blocks.a' and are linked to the shown simulation code when it is compiled — the classes enable simulation of the various blocks based on the techniques discussed in this paper. The simulation sample frequency was chosen as $1/T_s = 400$ MHz, which is a factor of 20 higher than the reference frequency. The loop filter was implemented using the Filter class, which internally converts an input s-domain transfer function to the z-domain using the bilinear transform [8]. Also, note that inclusion of the noise sources, as depicted in Figure 10, in the code is quite straightforward.

```
#include "com_blocks.h"
main()
{
double Ts = 1/400e6; // simulation sample period
SdMbitMod sd_mod("1 - 2z^-1 + z^-2"); // 2nd order
Probe probe("test.tr0",Ts); // output file
Vco vco("fc + Kv*x","fc,Kv,Ts",1.84e9,30e6,Ts);
SigGen ref_clk("square",20e6,Ts); // 20 MHz ref clock
Reg reg1,reg2,reg3,reg4;
And and1,and2;
Xor xor1;
Rand randg("gauss");
Filter lfilt("1+1/(2*pi*fz)s","C3*s+C3/(2*pi*fp)*s^2",
          "fp,fz,C3,Ts", 127.2e3,11.6e3,30e−12,Ts);
double chp_out,vco_in,Nsd,prev_vco_out,I_chp,pfd_out;
int i;
Nsd = 92.31793713; // choose value to avoid S-D spurs
sd_mod.inp(Nsd); // initialize Sigma-Delta
prev_vco_out = vco.out;
I_chp = 1.5e−6;

// 260e3 samples for dynamics, 5e6 samples for noise
for (i = 0; i < 260000; i++)
  {
// Vary Nsd to sim dynamics (remove for noise sim)
   if (i == 60000)
      Nsd += 4;
   if (i > 140000 && i < 186000)
      Nsd −= 1e−4;
   if (i > 186000 && i < 226000)
      Nsd += 1e−4;
// SD modulator - update on rising edge of VCO
   if (prev_vco_out == −1.0 && vco.out != −1.0)
      sd_mod.inp(Nsd);
// reference oscillator - specify zero phase deviation
   ref_clk.inp(0.0);
// PFD - XOR-based topology
   reg1.inp(−reg1.out,ref_clk.out);
   reg2.inp(−reg2.out,vco.out);
   xor1.inp(reg1.out,reg2.out);
   reg3.inp(xor1.out,ref_clk.out,−1.0,−reg4.out);
   reg4.inp(xor1.out,vco.out,reg3.out,−1.0);
   and1.inp(xor1.out,reg4.out);
   and2.inp(−reg3.out,−and1.out);
   pfd_out = −and2.out;
// Charge Pump - include charge pump noise
   chp_out = pfd_out∗I_chp;
   if (pfd_out > 0.0)
      chp_out += sqrt(1.85e−25/Ts)∗randg.inp();
   else
      chp_out += sqrt(1.2e−24/Ts)∗randg.inp();
// Loop Filter
   lfilt.inp(chp_out);
// VCO and divider - include VCO noise
```

```
    prev_vco_out = vco.out; // save value for edge detect
    vco_in = lfilt.out + sqrt((3.25e−16)/Ts)∗randg.inp();
    vco.inp(vco_in,sd_mod.out); // VCO input is vco_in,
                        // divide value set by sd_mod.out
// Save signals to file
    probe.inp(Nsd,"Nsd");
    probe.inp(vco_in,"vco");
  }
}
```

To view the results of the simulation code, the data is output to a binary file called test.tr0 using a C++ class called Probe in the simulator. This file is binary compatible with Hspice output; a custom written Hspice toolbox for Matlab is used that allows direct loading of such files into Matlab.

Figure 11 shows the simulated VCO output frequency (constructed from the VCO input) in response to variations at the input of the Σ-Δ modulator that include a step function and a ramp in divide value. The step size is chosen to be large enough to knock the synthesizer out of frequency lock — the corresponding oscillations in the VCO output frequency are a result of cycle slipping before the VCO becomes frequency locked again. The subsequent ramp in divide value illustrates the high resolution of the synthesizer as its output frequency is varied over a 40 MHz range. The simulation code was compiled with the GNU C++ compiler and was performed on a Dell Latitude laptop running Windows 2000 with a 650 MHz Pentium III processor and 256 MBytes of DRAM. The simulator computed 260 thousand samples of the PLL signals in less than 5 seconds.

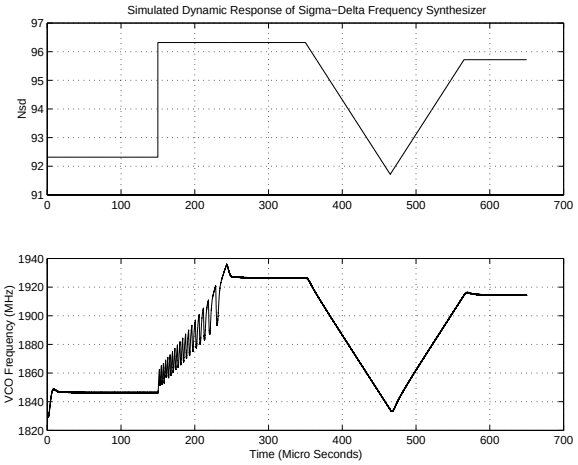

Figure 11: Simulated synthesizer dynamics.

As for examining the noise performance of the synthesizer, let us begin by showing calculated and measured noise plots taken from [9]. Figure 12 shows the calculated phase noise based on an analytical model of the prototype; the contribution of the individual noise sources depicted in Figure 2 to the overall phase noise is also shown. Figure 13 shows the measured synthesizer phase noise, and also includes a plot of the measured open loop VCO noise from which the input referred VCO noise variance in Figure 10 was computed. Note that the measured noise plot is not valid at frequency offsets higher than 10 MHz due to the fact that the noise floor of the measurement instrument dominates in that region.

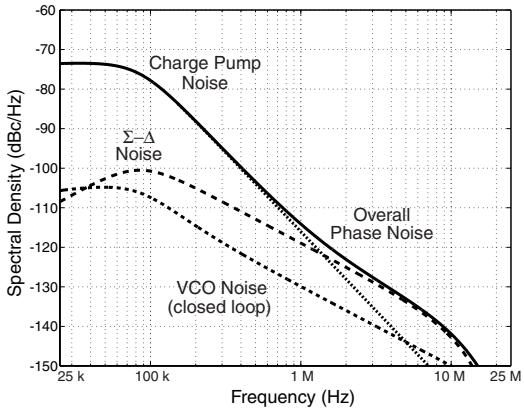

Figure 12: Calculated synthesizer phase noise.

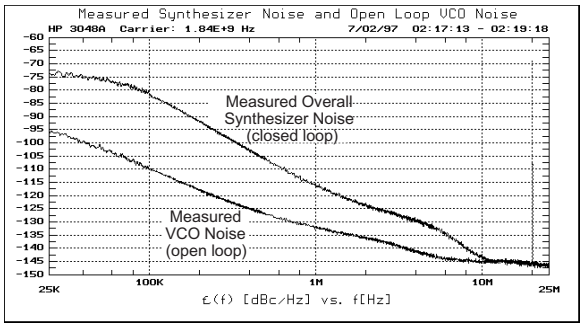

Figure 13: Measured synthesizer phase noise and open loop VCO noise.

The C++ simulation code shown earlier in this section was slightly modified to numerically compute the phase noise at the synthesizer output. In particular, the input to the Σ-Δ modulator was held constant and the number of simulation samples increased to 5 million in order to allow computation of the phase noise at frequency offsets less than 100 kHz. The entire simulation took 80 seconds on the same laptop computer, and the results were again saved in the binary file test.tr0 and then loaded into Matlab. The output phase noise of the synthesizer was then constructed from the VCO input — this procedure is valid since the VCO input includes the influence of all the PLL noise sources, as shown in Figure 10.

The following Matlab code describes the construction of the phase noise plot from the simulated VCO input — the resulting simulated phase noise plot is shown in Figure 14. Comparison of Figure 14 to Figures 12 and 13 reveal extremely close agreement between simulated, calculated, and measured results.

```
% loadsig and evalsig are part of Hspice Toolbox
x = loadsig('test.tr0'); % load file from C++ sim
vin = evalsig(x,'vco'); % input to VCO
t = evalsig(x,'TIME'); % simulation time samples
% set parameters and cut out initial transient
Ts = t(2)−t(1); % simulation sample period
Kv = 30e6; % VCO gain (Hz/V)
```

```
vin = vin(40000:length(vin)); % cut out initial transient
% create VCO output phase based on integration of vin
phase = filter(Ts*2*pi*Kv,[1 -1],vin-mean(vin));
% calculate L(f)
[Pxx,f] = psd(sqrt(Ts)*phase,2^16,1/Ts,2^16);
semilogx(f,10*log10(Pxx));
axis([25e3 25e6 -150 -60]);
title('Simulated Phase Noise of S-D Freq. Synth.');
xlabel('Frequency Offset from Carrier (Hz)');
ylabel('L(f) (dBc/Hz)');
grid on;
```

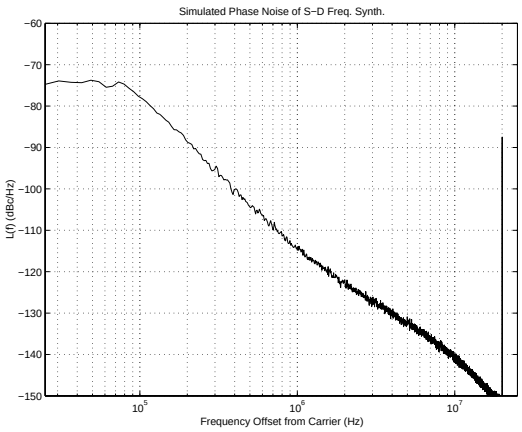

Figure 14: Simulated synthesizer phase noise (1/Ts = 20*ref freq).

It was mentioned in Section 2 that the simulation sample frequency, $1/T_s$, should be chosen as an integer multiple of the reference frequency in order to ignore the effects of Δt_k in Equation 4. Figure 15 shows the impact of choosing a non-integer multiple for T_s. We see there is no impact on the wideband phase noise, but the reference spur at 20 MHz offset is aliased to other frequency values as seen in the plot. A detailed explanation of this effect is beyond the scope of this paper, but a quick synopsis is that the aliasing occurs due to the presence of harmonics above 402 MHz of the 20 MHz reference spur in the CT PFD output.

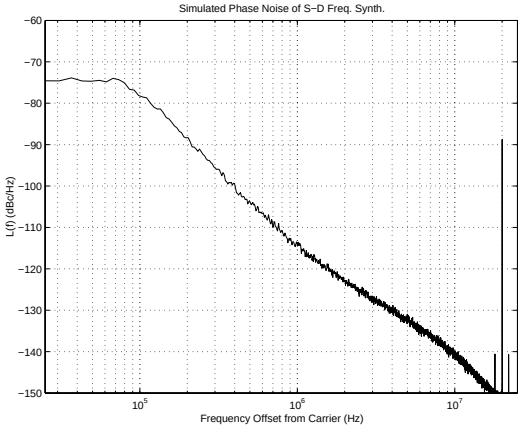

Figure 15: Simulated synthesizer phase noise (1/Ts = 20.1*ref freq).

6. CONCLUSION

Two techniques were presented in this paper that allow fast and accurate simulation of fractional-N synthesizers at a detailed behavioral level using a uniform time sample period. The first provides accurate representation of the CT PFD output with a DT sequence using an area conservation principle. The second allows a dramatic reduction of the simulation sample frequency by including the divider implementation in the VCO simulation module.

The techniques were incorporated into a custom C++ simulator, which was used to simulate the dynamic and noise performance of a prototype Σ-Δ frequency synthesizer. The simulations discussed in this paper took only 80 seconds to compute 5 million samples on a laptop computer, thus showing that the simulator is quite fast. The simulated noise performance was shown to agree quite well with calculated and measured results, thus showing that the simulator is also accurate. The technique can also be applied to other phase locked loop circuits, and be implemented in other simulation frameworks such as Verilog and Matlab.

7. REFERENCES

[1] B. De Smedt and G. Gielen. Nonlinear Behavioral Modeling and Phase Noise Evaluation in Phase Locked Loops. In *Custom Integrated Circuits Conference*, pages 53–56, 1998.

[2] A. Demir, E. Liu, A. L. Sangiovanni-Vincentelli, and I. Vassiliou. Behavioral Simulation Techniques for Phase/Delay-Locked Systems. In *Custom Integrated Circuits Conference*, pages 453–456, 1994.

[3] A. Hajimiri and T. Lee. A General Theory of Phase Noise in Electrical Oscillators. *IEEE Journal of Solid State Circuits (JSSC)*, 33(2):179–194, Feb. 1998.

[4] A. Hill and A. Surber. The PLL Dead Zone and How to Avoid It. In *RF Design*, pages 131–134, Mar. 1992.

[5] M. Hinz, I. Konenkamp, and E.-H. Horneber. Behavioral Modeling and Simulation of Phase-locked Loops for RF Front Ends. In *43rd Midwest Symp. on Circuits and Systems*, pages 194–197, 2000.

[6] K. Kundert, J. White, and A. Sangiovanni-Vincentelli. *Steady-State Methods for Simulating Analog and Microwave Circuits*. Kluwer, Boston, 1990.

[7] S. Norsworthy, R. Schreier, and G. Temes. *Delta-Sigma Data Converters: Theory, Design, and Simulation*. IEEE Press, New York, 1997.

[8] A. V. Oppenheim and R. W. Schafer. *Discrete Time Signal Processing*. Prentice Hall, N.J., 1999.

[9] M. Perrott, T. Tewksbury, and C. Sodini. A 27 mW CMOS Fractional-N Synthesizer using Digital Compensation for 2.5 Mb/s GFSM Modulation. *JSSC*, 32(12):2048–2060, Dec. 1997.

[10] B. Razavi. *Monolithic Phase-Locked Loops and Clock Recovery Circuits: Theory and Design*. IEEE Press, New York, 1996.

[11] T. A. Riley, M. A. Copeland, and T. A. Kwasniewski. Delta-Sigma Modulation in Fractional-N Frequency Synthesis. *JSSC*, 28(5):553–559, May 1993.

[12] P. Van Halen and G. Boyle. SPICE-Compatible Behavioral Phase-Space Simulation Techniques for Phase-Locked Systems. In *38th Symposium on Circuits and Systems Conference*, volume 1, pages 53–56, 1996.