Devices and Circuits for Phase-Locked Systems

Behzad Razavi

Abstract—This turtorial deals with the design of devices such as varactors and inductors and circuits such as ring and LC oscillators. First, MOS varactors are introduced as a means of frequency control for low-voltage circuits and their modeling issues are discussed. Next, spiral inductors are studied and various geometries targetting improved Q or higher self-resonance frequencies are presented. Noise-tolerant ring oscillator topologies are then described. Finally, a procedure for the design of LC oscillators is outlined.

The design of phase-locked systems requires a thorough understanding of devices, circuits, and architectures. Intended as a continuation of [1], this tutorial provides an overview of concepts in device and circuit design for phase-locking in digital, broadband, and RF systems.

I. Passive Devices

The demand for low-noise PLLs has encouraged extensive research on active and passive devices. In this section, we study varactors and inductors as essential components of LC oscillators.

A. Varactors

As supply voltages scale down, pn junctions become a less attractive choice for varactors. Specifically, two factors limit the dynamic range of pn-junction capacitances: (1) the weak dependence of the capacitance upon the reverse bias voltage, e.g., $C_j = C_{j0}/(1 + V_R/\phi_B)^m$, where $m \approx 0.3$.; and (2) the narrow control voltage range if forward-biasing the varactor must be avoided.

As an example, consider the LC oscillator shown in Fig. 1. It is desirable to maximize the voltage swings at nodes X and Y so as to both minimize the relative phase noise and ease the design of the stage(s) driven by the VCO. On the other hand, to avoid forward-biasing the varactors significantly, V_X and V_Y must remain above approximately $V_{cont} - 0.4$ V. Thus, the peak-to-peak swing at each node is limited to about 0.8 V. Note that the cathode terminals of the varactors also introduce substantial n-well capacitance at X and Y, further constraining the tuning range.

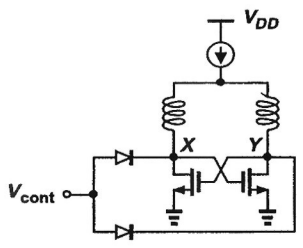


Fig. 1. LC oscillator using pn-junction varactors.

In contrast to pn junctions, MOS varactors are immune to forward biasing while exhibiting a sharper C-V characteristic and a wider dynamic range. If configured as a capacitor [Fig. 2(a)], a MOSFET suffers from both a nonmonotonic C-V behavior and a high channel resistance in the region between accumulation and strong inversion. To avoid these issues, an "accumulation-mode" MOS varactor is formed by placing an NMOS device inside an n-well [Fig. 2(b)]. Providing an ohmic connection between the source and drain for all gate voltages, the n-well experiences depletion of mobile charges under the oxide as the gate voltage becomes more negative. Thus, the varactor capacitance, C_{var}, (equal to the series combination of the oxide capacitance and the depletion region capacitance) varies as shown in Fig. 2(b). Note that for a sufficiently positive gate voltage, C_{var} approaches the oxide capacitance.

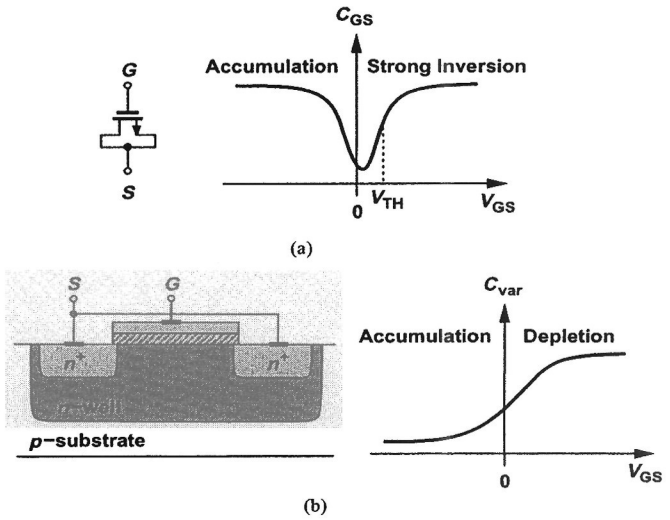


Fig. 2. (a) Simple MOSFET operating as capacitor, (b) MOS varactor.

The design of MOS varactors must deal with two important issues: (1) the trade-off between the dynamic range and the

channel resistance, and (2) proper modeling for circuit simulations. We now study each issue.

Dynamic Range Deep-submicron MOSFETs exhibit susbtantial overlap capacitance between the gate and source/drain terminals. For example, in a typical 0.13-μm technology, a transistor having minimum channel length, L_{min}, displays an overlap capacitance of 0.4 fF/μm and a gate-channel capacitance of 12 fF/μm^2. In other words, for an effective channel length of 0.12 μm and a given width, the overlap capacitance between the gate and source/drain terminals of a varactor constitutes 2×0.4 fF $/(0.12 \times 12$ fF$+2 \times 0.4$ fF$) \approx 36\%$ of the total capacitance. Thus, even if the gate-channel component varies by a factor of two across the allowable voltage range, the overall dynamic range of the capacitance is given by $(0.12 \times 12$ fF$+2 \times 0.4$ fF$)/(0.12 \times 6$ fF $+2 \times 0.4$ fF$) = 1.47$.

In order to widen the varactor dynamic range, the transistor length can be increased, thereby raising the voltage-dependent component while maintaining the overlap capacitance relatively constant. This remedy, however, leads to a greater resistance between the source and drain, lowering the Q. The resistance reaches a maximum for the most negative gate-source voltage, at which the depletion region's width is maximum and the path through the n-well the longest (Fig. 3).[1] Note that

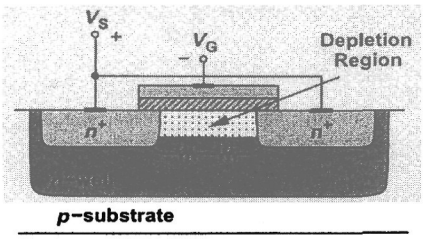


Fig. 3. Effect of n-well resistance in MOS varactor.

the total equivalent resistance that appears in series with the varactor is equal to $1/12$ of the drain-source resistance. This is because shorting the drain and source lowers the resistance by a factor of 4 and the distributed nature of the capacitance and resistance reduces it by another factor of 3 [2]. Depending on both the phase noise requirements and the Q limitations imposed by inductors, the varactor length is typically chosen between L_{min} and $3L_{min}$.

Modeling The C-V characteristics of MOS varactors can be approximated by a hyperbolic tangent function with reasonable accuracy. Using the characteristic shown in Fig. 4 and noting that $\tanh(\pm\infty) = \pm 1$, we can write

$$C_{var}(V_{GS}) = \frac{C_{max} - C_{min}}{2} \tanh(a + \frac{V_{GS}}{V_0}) + \frac{C_{max} + C_{min}}{2}. \quad (1)$$

Here, a and V_0 allow fitting for the intercept and the slope, respectively, and C_{min} includes the overlap capacitance.

The above model yields different characteristics in different circuit simulation programs! Simulation tools that analyze

[1] Fortunately, the capacitance reaches a minimum at this point, and the Q degrades only gradually.

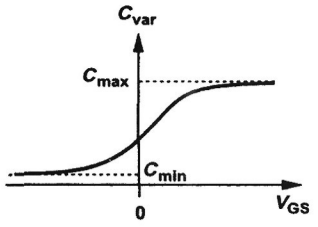


Fig. 4. Typical MOS varactor characteristic.

circuits in terms of voltages and currents (e.g., SPICE) interpret the nonlinear capacitance equation correctly. On the other hand, programs that represent the behavior of capacitors by charge equations (e.g., Cadence's Spectre) require that the model be transformed to a Q-V relationship [3]:

$$Q_{var} = \int C_{var} dV_{GS} \quad (2)$$

$$= \frac{C_{max} - C_{min}}{2} V_0 \ln\left[\cosh(a + \frac{V_{GS}}{V_0})\right] + \frac{C_{max} + C_{min}}{2} V_{GS}, \quad (3)$$

which is then used to compute

$$I_{var} = \frac{dQ_{var}}{dt}. \quad (4)$$

If used in charge-based analyses, Eq. (1) typically overestimates the tuning range of oscillators.

B. Inductors

The design of monolithic inductors has been studied extensively. The parameters of interest include the inductance, the Q, the parasitic capacitance (i.e., the self-resonance frequency, f_{SR}), and the area, all of which trade with each other to some extent. For a spiral structure such as that in Fig. 5, the line width, the line spacing, the number of turns, and the outer

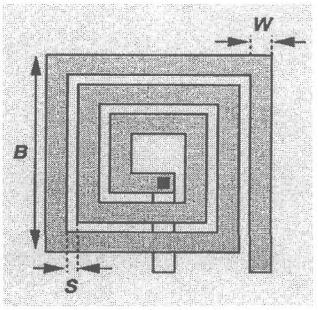


Fig. 5. Spiral inductor.

dimension are under the designer's control, chosen so as to obtain the required performance.

Quality Factor The quality factor of monolithic inductors has been the subject of many studies. Before considering the phenomena that limit the Q, it is important to select a useful

and clear definition for this quantity. For a simple inductor operating at low frequencies, the Q is defined as

$$Q = \frac{L_S \omega}{R_S}, \tag{5}$$

where R_S denotes the metal series resistance. In analogy with this expression, a more general definition is sometimes given as

$$Q = \frac{\mathrm{Im}(Z_L)}{\mathrm{Re}(Z_L)}, \tag{6}$$

where Z_L represents the overall impedance of the inductor at the frequency of interest. While reducing to Eq. (5) at low frequencies, this definition yields $Q = 0$ if the inductor resonates with its own capacitance and/or any other capacitance. This is because at resonance, the impedance is purely resistive. Since nearly all circuits employ inductors in a resonance mode,[2] this expression fails to provide a meaningful measure of inductor performance in circuit design. A more versatile definition assumes that a resonant tank can be represented by a *parallel* combination [Fig. 6(a)], yielding

$$Q = \frac{R_P}{L_P \omega_R}, \tag{7}$$

where ω_R is the resonance frequency. Note that the tank reduces to R_P at $\omega = \omega_R$, exhibiting a finite (rather than zero) Q. Hereafter, we consider the behavior of inductors at or near resonance.

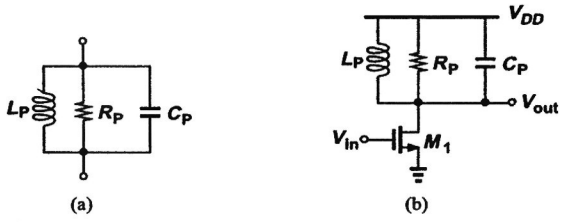


Fig. 6. (a) Parallel tank for definition of Q, (b) common-source stage using a tank.

The utility of Eq. (7) can be seen in the example illustrated in Fig. 6(b). Here, the knowledge of L_P and $Q = R_P/(L_P \omega_R)$ directly provides the voltage gain and the output swing, whereas the Q given by Eq. (6) serves no purpose.

The Q of inductors is limited by resistive losses: parasitic resistances dissipate a fraction of the energy that is reciprocated between the inductor and the capacitor in a tank. Note that the finite Q is also accompanied by generation of *noise*. For example, in the circuit of Fig. 6(b), R_P produces an output noise voltage of $\overline{V_n^2} = 4kTR_P = 4kTQL_P\omega_R$ per unit bandwidth if L_P resonates with C_P.

The losses in inductors arise from three mechanisms (Fig. 7): (1) the series resistance of the spiral, including both low-frequency resistance and current crowding due to skin effect;

[2]One exception is inductive degeneration in low-noise amplifiers.

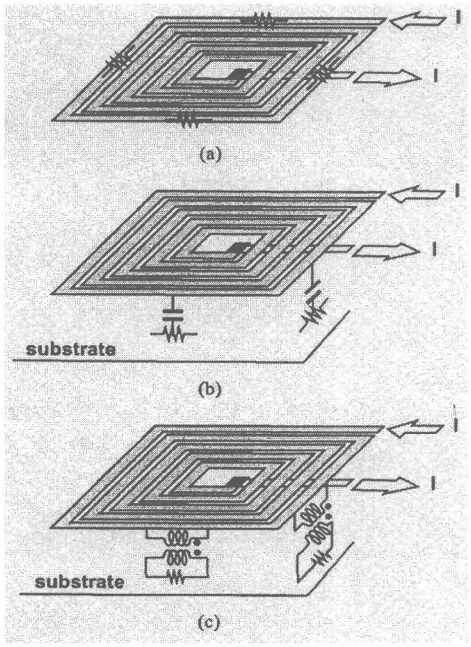


Fig. 7. Inductor loss mechanisms: (a) metal resistance, (b) substrate loss due to electric coupling, (c) substrate loss due to magnetic coupling.

(2) the flow of displacement current through the series combination of the inductor's parasitic capacitance and the substrate resistance; (3) the flow of magnetically-induced ("eddy") currents in the substrate resistance. At low frequencies, the dc resistance is dominant, and as the frequency rises, the other components begin to manifest themselves.

With the above observations in mind, let us construct a circuit model for inductors. Depicted in Fig. 8(a) is a simple model where R_S denotes the series resistance at the frequency

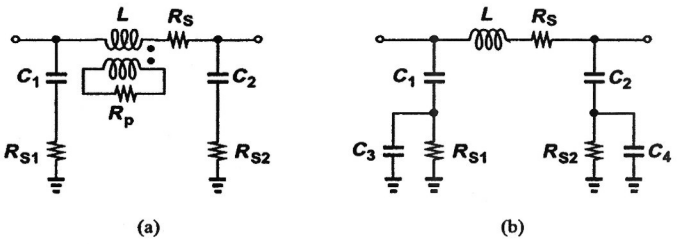


Fig. 8. (a) Inductor model including magnetic coupling to substrate, (b) simplified model.

of interest, R_{S1} and R_{S2} represent the substrate resistance through which the diplacement current flows, the transformer models magnetic coupling to the substrate, and R_P is the substrate resistance through which the eddy currents flow. This model reveals how the Q drops at high frequencies. As the impedance of C_1 and C_2 falls, R_{S1} and R_{S2} appear as a constant resistance in *parallel* with the inductor, lowering the Q as ω rises. Similarly, at high frequencies, the effect of R_P becomes relatively constant, shunting L_P and further reducing the Q.

In practice, the model of Fig. 8(a) is modified as shown in Fig. 8(b) to both allow an easier fit to measured data and

account for the substrate capacitance. The model is usually assumed to be symmetric, i.e., $C_1 = C_2, C_3 = C_4$, and $R_{S1} = R_{S2}$, implying that the equivalent parasitic capacitance, C_{eq}, is one-half of the total capacitance, C_{tot}, if one end of the inductor is grounded. This result, however, is not correct because the distributed nature of the structure yields $C_{eq} = C_{tot}/3$ in this case [5]. To avoid this inaccuracy, the inductor must be modeled as a distributed network [5].

Characterization Most inductor modeling programs provide limited capabilities in terms of the type of structure that they can analyze or the maximum frequency at which their results are valid. For this reason, it is often necessary to fabricate and characterize monolithic inductors and use the results to revise the simulated models, thereby obtaining a better fit.

Owing to the need for precise measurements at high frequencies, inductors are typically characterized by direct on-wafer probing. High-speed coaxial probes having a tightly-controlled 50-Ω characteristic impedance and a low loss are positioned on pads connected to the inductor. Figure 9(a) shows an example where one end of the spiral is tied to the

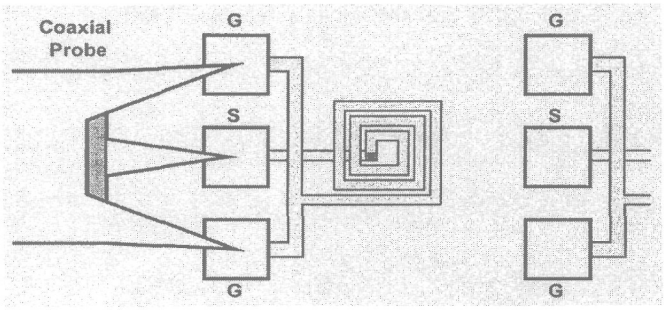


Fig. 9. (a) On-wafer measurement of inductor using coaxial probe, (b) calibration structure.

"signal" (S) pad and the other to the "ground" (G) pads. The signal pad is sensed by the center conductor and the ground pads by the outer shield of the coaxial probe.

Since the capacitance of the pads and the wires connecting to the spiral is typically significant, the test device is accompanied by a calibration structure [Fig. 9(b)], where the spiral itself is omitted. The scattering (S) parameters of both structures are measured by means of a network analyzer across the band of interest and subsequently converted to Y parameters. Subtraction of the Y parameters of the calibration geometry from those of the device under test yields the actual characteristics of the spiral.

An alternative method of measuring the Q of inductors is illustrated in Fig. 10. Here, inductors are incorporated in an oscillator and the tail current can be controlled externally. In the laboratory measurement, the output is monitored on a spectrum analyzer while I_{SS} is reduced so as to place the circuit at the edge of oscillation. Next, the value of I_{SS} thus obtained is used in the simulation of the oscillator and the equivalent parallel resistance of each tank, R_P, is lowered

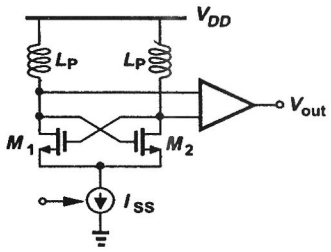


Fig. 10. Setup for "in-situ" measurement of Q.

until the circuit fails to oscillate. For such value of R_P, we have $Q = R_P/(L\omega)$. Of course, this technique assumes that the value of the inductor and the oscillation frequency are known.

The above method proves useful if (a) the frequency of interest is so high and/or the inductance so low that direct measurements are difficult, or (b) an oscillator has been fabricated but the inductors are not available individually, requiring "in-situ" measurement of the Q. Note that other oscillator parameters such as phase noise and ouput swing are also functions of Q, but it is much more straightforward to place the circuit at the edge of oscillation than to calculate the Q from phase noise or output swing measurements.

Choice of Geometry The design of inductors begins with the choice of the geometry. Shown in Fig. 11 are two commonly-used structures. The asymmetric spiral of Fig. 11(a) exhibits

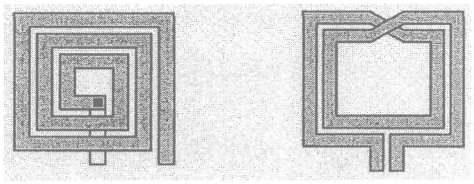


Fig. 11. (a) Asymmetric and (b) symmetric inductors.

a moderate Q, about 5 to 6 at 5 GHz, and its interwinding capacitance does not limit the self-resonance frequency because adjacent turns sustain a small potential difference. The line spacing is therefore set to the minimum allowed by the technology.

The symmetric geometry of Fig. 11(b) provides a greater Q if stimulated differentially [4], about 7 to 10 at 5 GHz, but its interwinding capacitance is typically quite significant because of the large voltage difference between adjacent turns. For this reason, the line spacing is chosen to be twice or three times the minimum allowable value, lowering the fringe capacitance considerably but degrading the Q slightly.

In differential circuits, the use of symmetric inductors appears to save area as well. For example, two asymmetric 1-nH inductors can be replaced by a symmetric 2-nH structure, which occupies less area. However, a cascade of differential stages employing multiple symmetric inductors [Fig. 12(a)] faces routing difficulties. As illustrated in Fig. 12(b), the signal lines must travel across the spirals, impacting the

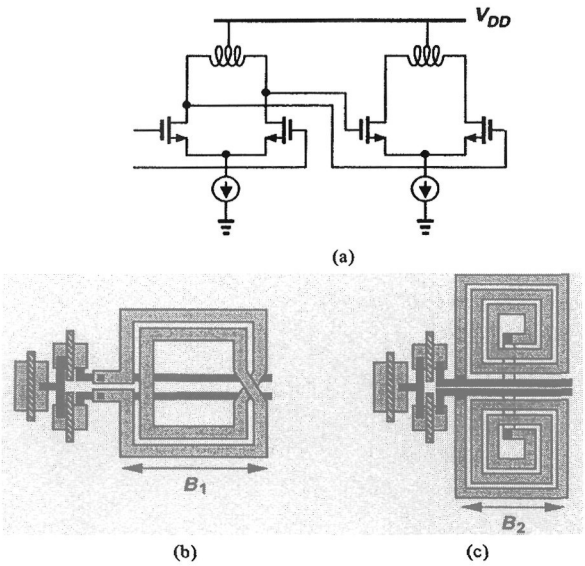


Fig. 12. (a) Cascade of inductively-loaded differential pairs, (b) layout of first stage using a symmetric inductor, (c) layout of first stage using asymmetric inductors.

performance of the inductors. Furthermore, the power and ground lines must either cross the spirals or go around with adequate spacing. With asymmetric inductors, on the other hand, the lines can be routed as shown in Fig. 12(c), leaving the inductors undisturbed. Note that B_1 is quite larger than B_2 because the symmetric structure must provide an inductance twice that of each asymmetric spiral. Thus, the signal lines in Fig. 12(b) are longer.

The two geometries of Fig. 11 can also be converted to stacked structures, wherein spirals in different metal layers are placed in series so as to achieve a greater inductance per unit area. Figure 13(a) depicts an example using metal 8 and metal

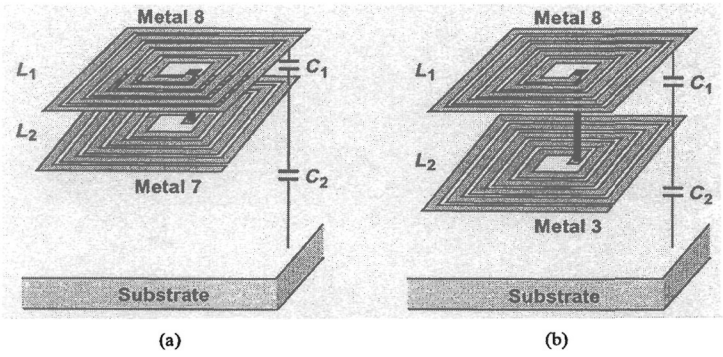


Fig. 13. Stack of (a) metal 8 and metal 7 spirals, (b) metal 8 and metal 3 spirals.

7 spirals. The total inductance is equal to L_1+L_2+2M, where M denotes the mutual coupling between L_1 and L_2. Owing to the strong magnetic coupling, the value of M is close to L_1 and L_2, suggesting a fourfold increase in the overall inductance as a result of stacking. In the general case, n stacked identical spirals raise the inductance by a factor of approximately n^2.

Stacking reduces the area occupied by inductors significantly. However, the capacitance between the spirals may limit the self-resonance frequency. For the two-layer structure of Fig. 13(a), the overall equivalent capacitance is given by [5]

$$C_{eq} = \frac{4C_1 + C_2}{12}. \tag{8}$$

Thus, if the bottom layer is moved down [Fig. 13(b)], then C_{eq} falls considerably. For example, in a typical 0.13-μm CMOS technology having eight metal layers, the geometry of Fig. 13(b) exhibits one-fifth as much as capacitance as the structure in Fig. 13(a) does.

Stacked structures use lower metal layers, which typically suffer from a greater sheet resistance than the topmost layer. As explained below, the resistance can be reduced by placing spirals in parallel.

Figure 14 illustrates three other configurations aiming to improve the quality factor. In Fig. 14(a), multiple spirals are

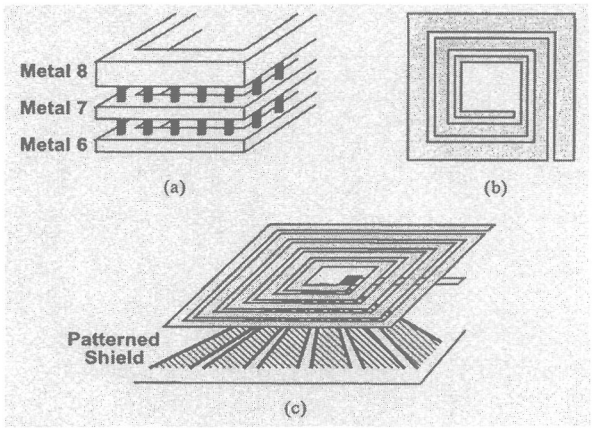


Fig. 14. (a) Parallel combination of spirals to reduce metal resistance, (b) tapered metal width, (c) patterned shield.

placed in parallel so as to reduce the series resistance, but at the cost of larger capacitance to the substrate. Nonetheless, in a typical process having eight metal layers, metal 6 capacitance is about 30% greater than that of metal 8. Since metal 8 is typically twice as thick as metal 6 or metal 7, this topology lowers the series resistance by twofold while raising the parasitic capacitance by 30%. By the same token, in the stacked structure of Fig. 13(b), addition of a metal 2 spiral in parallel with the metal 3 layer decreases the overall resistance by 30% while increasing the equivalent capacitance by about 15%.

At frequencies above 5 GHz, the skin depth of aluminum falls below 2 μm, making the parallel combination of spirals less effective. Electromagnetic field simulations may therefore be necessary to determine the optimum configuration.

The structure in Fig. 14(b) employs tapering of the line width to reduce the resistance of the outer turns. The idea is to maintain a relatively constant inductance-resistance product per turn, achieving a slightly higher Q for a given inductance and capacitance. Unfortunately, most inductor simulation programs cannot analyze such a geometry.

Shown in Fig. 14(c) is a method of lowering the loss due to the electric coupling to the substrate. A heavily-conductive

shield is placed under the spiral and connected to ground so that the displacement current flowing through the inductor's bottom-plate capacitance does not experience resistive loss. To stop the flow of magnetically-induced currents, the shield is broken regularly. Note that eddy currents still flow through the substrate, dissipating energy.

The conductive shield in Fig. 14(c) may be realized in n-well, n^+, or p^+ diffusion, polysilicon, or metal, thus bearing a trade-off between the parasitic capacitance and the Q enhancement. The resulting increase in the Q depends on the frequency of operation and the type of shield material, falling in the range of 5 to 10%.

Thus far, we have studied square spirals. However, for a given inductance value, a circular structure exhibits less series resistance. Since mask generation for circles is more difficult, some inductors are designed as octagonal geometries to benefit from a slightly higher Q.

C. MOS Transistors

The modeling of MOSFETs for analog and high-frequency design continues to pose challenging problems as sub-0.1-μm generations emerge. BSIM models provide reasonable accuracy for phase-locked system design, with the exception that their representation of thermal and flicker noise may err considerably. This issue becomes critical in the prediction of oscillator phase noise.

The thermal noise arising from the channel resistance is usually represented by a current source tied between the source and drain and having a spectral density $\overline{I_n^2} = 4kT\gamma g_m$, where γ is the excess noise coefficient. For long-channel devices, $\gamma = 2/3$, but for submicron transistors, γ may reach 2.5 to 3. Since some MOS models lack an explicit γ parameter that the user can set, it is often necessary to artificially raise the effective value of γ in circuit simulations. For linear, time-variant circuits, this can be accomplished using a noise copying technique [6]. However, the time variance of currents and voltages in oscillators make it difficult to apply this method. As a first-order approximation, the contribution of the transistors to the overall phase noise can be increased by a factor equal to $2.5/(2/3)$ before all of the noise components are summed.[3]

The flicker noise parameters are usually obtained by measurements. It is therefore important to check the validity of the device models by comparing measured and simulated results. Owing to their buried channel, PMOS transistors exhibit substantially less flicker noise than NMOS devices even in deep submicron technologies.

II. Ring Oscillators

Despite their relative high noise and poor drive capability, ring oscillators are used in many high-speed applications. Several reasons justify this popularity: (1) in some cases, the oscillator must be tuned over a wide frequency range (e.g., one decade) because the system must support different data rates or retain a low-frequency clock in the "sleep" mode; (2) ring oscillators occupy substantially less area than LC topologies do, an important issue if many oscillators are used; (3) the behavior of ring oscillators across process, supply, and temperature corners is predicted with reasonable accuracy by standard MOS models, whereas the design of LC oscillators heavily relies on inductor and varactor models.

In mostly-digital systems such as microprocessors, ring oscillators experience considerable supply and substrate noise, making differential topologies desirable. Figure 15(a) shows an example of a differential gain stage that allows several

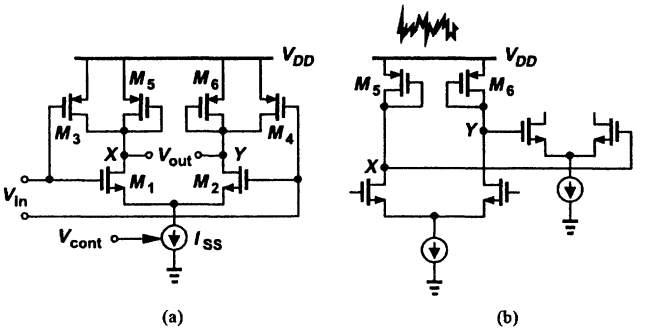


Fig. 15. (a) Differential stage for use in a ring oscillator, (b) effect of supply noise.

decades of frequency tuning with relatively constant voltage swings. Here, M_5 and M_6 define the output common-mode (CM) level while M_3 and M_4 pull nodes X and Y to V_{DD}, maintaining a constant voltage swing even at low current levels.

Unlike a simple differential pair, the stage of Fig. 15(a) does respond to input CM noise even with an ideal I_{SS}. This is because the gate voltages of M_3 and M_4 are referenced to V_{DD}, introducing a change in the drain currents if the input CM level varies. In the presence of asymmetries, such a change results in a differential component at the output. Nevertheless, since the input CM level of each stage in the ring is referenced to V_{DD} by the diode-connected PMOS devices in the preceding stage [Fig. 15(b)], the oscillator exhibits low sensitivity to supply voltage.

Figure 16(a) depicts another ring oscillator topology that has become popular in low-voltage digital systems. Here, the

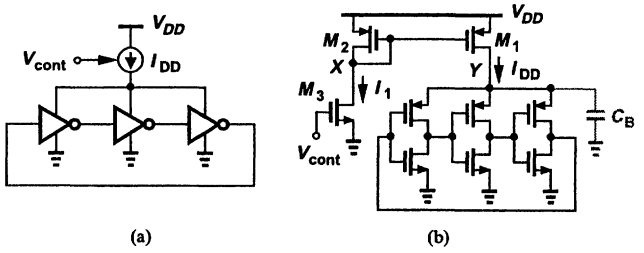


Fig. 16. (a) Constant-current ring oscillator, (b) transistor-level implementation of (a).

inverters in the ring are supplied by a current source, I_{DD},

[3] In reality, the effective value of γ also depends on the drain-source voltage to some extent, further complicating the matter.

rather than a voltage source, and frequency tuning is also accomplished through I_{DD}. If I_{DD} is designed for low sensitivity to V_{DD}, then the oscillator remains relatively immune to supply noise—the principal advantage of this configuration over standard inverter-based rings that are directly connected to the supply voltage.

In practice, the nonidealities associated with I_{DD} limit the supply rejection. Shown in Fig. 16(b) is a transistor implementation where M_1 operates as a contolled current source. If I_1 is constant, V_X tracks V_{DD} variations whereas V_Y does not, yielding a change in I_{DD} through channel-length modulation in M_1. Choosing long channels for M_1 and M_2 alleviates this issue while necessitating wide channels as well to allow a relatively small drain-source voltage for M_1. However, the resulting high drain junction capacitance of M_1 at Y creates a low-impedance path from V_{DD} to this node at high frequencies. To suppress both resistive and capacitive feedthrough of V_{DD} noise, a bypass capacitor, C_B, is tied from Y to ground. However, the pole associated with this node now enters the VCO transfer function, complicating the design of the PLL.

Let us now study the response of the circuit of Figs. 15(a) and 16 to substrate noise, V_{sub}. In the former, V_{sub} manifests itself through two mechanisms (Fig. 17): (1) by modulating the drain junction capacitance of M_1 and M_2 and hence

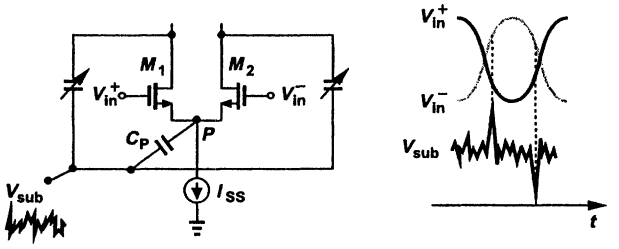


Fig. 17. Effect of substrate noise on a differential stage.

the delay of the stage (a static effect); and (2) by injecting a common-mode displacement current through C_P (a dynamic effect). If injected slightly before or after the zero crossings of the oscillation waveform, such a current gives rise to a differential component at the drains of M_1 and M_2 because these transistors display unequal transconductances as they depart from equilibrium.

In the circuit of Fig. 16(b), V_{sub} modulates both the drain junction capacitance of the NMOS devices and their threshold voltage (and hence the transition points of the waveform). Both effects are static, making the circuit susceptible even to low-frequency noise.

It is instructive to determine the minimum supply voltage for the above two circuits. At the midpoint of switching, where the input and output differential voltages are around zero, the stage of Fig. 15(a) requires that $V_{DD} \geq |V_{GSP}| + V_{GSN} + V_{ISS}$, where V_{GSP} abd V_{GSN} denote the gate-source voltages of M_3-M_4 and M_1-M_2, respectively, and V_{ISS} is the minimum voltage necessary for I_{SS}. Interestingly, the circuit of Fig. 16(b) imposes the same minimum supply voltage.

Another critical issue in the circuits of Figs. 15(a) and 16 relates to frequency tuning by means of current sources. The voltage-to-current (V/I) conversion required here presents difficulties at low supply voltages. In the example of Fig. 16(b), as V_{cont} rises and V_X falls, transistor M_3 eventually enters the triode region, thus making I_1 supply-dependent. The useful range of V_{cont} is therefore given by $V_{THN} < V_{cont} < V_{DD} - |V_{GSP}| - V_{THN}$, suggesting the use of a wide device for M_2 to minimize $|V_{GSP}|$.

III. LC Oscillators

LC oscillators have found wide usage in high-speed and/or low-noise systems. Extensive research on inductors, varactors, and oscillator topologies has provided the grounds for systematic design, helping to demystify the "black magic."

LC oscillators offer a number of advantages over ring structures: (a) lower phase noise for a given frequency and power dissipation; (b) greater output voltage swings, with peak levels that can exceed the supply voltage; and (c) ability to operate at higher frequencies.

However, LC VCO design requires precise device and circuit modeling because (a) the narrow tuning range calls for accurate prediction of the center frequency; (b) the phase noise is greatly affected by the quality of inductors and varactors and the noise of transistors. Also, occupying a large area, spiral inductors pick up noise from the substrate and make it difficult to incorporate many such oscillators on one chip.

The design of LC VCOs targets the following parameters: center frequency, phase noise, tuning range, power dissipation, voltage headroom, startup condition, output voltage swing, and drive capability. The last two have often received less attention, but they directly determine the design difficulty and power consumption of the stages following the oscillator. That is, a buffer placed after the VCO may consume more power than the VCO itself!

A. Design Example

As an example of VCO design, let us consider the topology shown in Fig. 18. Here, M_1 and M_2 present a small-signal negative resistance of $-2/g_{m1,2}$ between nodes X and Y,

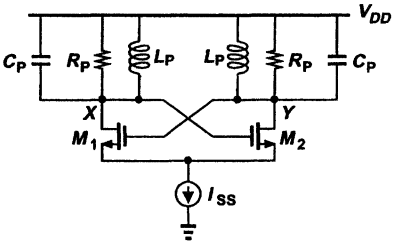


Fig. 18. LC oscillator.

compensating for the resistive loss in the tanks and sustaining oscillation. Each tank is modeled by a parallel RLC network, with all loss mechanisms lumped in R_P.[4]

[4]For a narrow frequency range, series resistances in the tank elements can be transformed to parallel components.

The design process begins with a power budget and hence a maximum value for I_{SS}. This is justified by the following observation. Once completed and optimized for a given power budget, the design can readily be scaled for different power levels, bearing a linear trade-off with phase noise while maintaining all other parameters constant. For example, if I_{SS}, the width of M_1 and M_2, and the total tank capacitance are doubled and the inductance value is halved, the phase noise power falls by a factor of two but the frequency of oscillation and the output voltage swings remain unchanged.[5]

Since subsequent stages typically require the VCO core to provide a minimum voltage swing, V_{min}, we assume M_1 and M_2 steer nearly all of I_{SS} to their correponding tanks and write $I_{SS}R_P = V_{min}$. Thus, the minimum inductance value is given by

$$L_P = \frac{R_P}{Q\omega} \quad (9)$$

$$= \frac{V_{min}}{I_{SS}Q\omega}, \quad (10)$$

where it is assumed the tank Q is limited by that of the inductor. Note that this calculation demands knowledge of the Q *before* the inductance is computed, a minor issue because for a given geometry and frequency of operation, the Q is relatively independent of the inductance.

We now determine the dimensions of M_1 and M_2. Increasing the channel length beyond the minimum value allowed by the technology does not significantly lower γ unless the length exceeds approximately 0.5 μm. For this reason, the minimum length is usually chosen to minimize the capacitance contributed by the transistors. The transistors must be wide enough to steer most of I_{SS} while experiencing a voltage swing of V_{min} at nodes X and Y. Viewing M_1 and M_2 as a differential pair, we note that M_1 must turn off as $V_X - V_Y$ reaches V_{min}. For square-law devices,

$$V_{min} = \sqrt{\frac{2I_{SS}}{\mu_n C_{ox} W/L}}, \quad (11)$$

and hence

$$W = \frac{2I_{SS}}{\mu_n C_{ox} V_{min}^2/L}, \quad (12)$$

but for short-channel devices, W must be obtained by simulations using proper device models. This choice of W typically guarantees a small-signal loop gain greater than unity, enabling the circuit to start at power-up.

With L_P computed from Eq. (10), the total capacitance at nodes X and Y is calculated as $C_{tot} = (L_P\omega^2)^{-1}$. This capacitance includes the following *fixed* components: (1) the parasitic capacitance of L_P, C_{LP}; (2) the drain junction, gate-source, and gate-drain capacitances of M_1 and M_2, $C_{DB} + C_{GS} + 4C_{GD}$;[6] and (3) the input capacitance of the next state (typically a buffer), C_L. Thus, the allowable varactor capacitance is given by the difference between C_{tot} and the sum of these components:

$$C_{var} = (L_P\omega^2)^{-1} - C_{LP} - C_{DB} - C_{GS} - 4C_{GD} - C_L. \quad (13)$$

This expression gives the center value of the tolerable varactor capacitance. Of course, a negative C_{var} means the inductance is excessively large, calling for a lower L_P, a smaller R_P, and hence a larger I_{SS}. However, to steer a greater tail current, the circuit must employ wider MOS transistors, thus incurring a larger capacitance at nodes X and Y and approaching diminishing returns. This ultimately limits the frequency of oscillation in a given technology.

For a given supply voltage and oscillator topology, the varactor capacitance exhibits a known dynamic range $C_{var,min} \leq C_{var} \leq C_{var,max}$, yielding a tuning range of $\omega_{min} \leq \omega_{osc} \leq \omega_{max}$, where

$$\omega_{min} = \frac{1}{\sqrt{L_P(C_{var,max} + C_{fixed})}} \quad (14)$$

$$\omega_{max} = \frac{1}{\sqrt{L_P(C_{var,min} + C_{fixed})}}, \quad (15)$$

and $C_{fixed} = C_{LP} + C_{DB} + C_{GS} + 4C_{GD} + C_L$.

Figure 19(a) depicts the oscillator with MOS varactors directly tied to X and Y. Since the output common-mode level

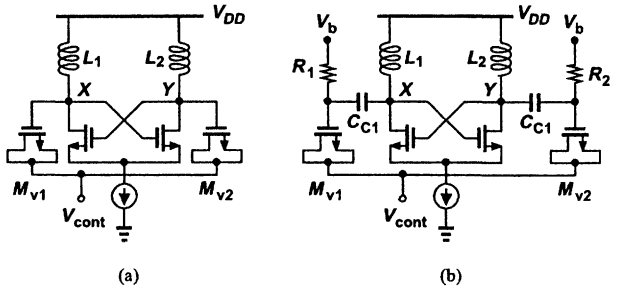


Fig. 19. LC oscillator with (a) direct coupling and (b) capacitive coupling of varactors to tanks.

is near V_{DD}, M_3 and M_4 sustain only a positive gate-source voltage (if $0 < V_{cont} < V_{DD}$). As seen from the C-V characteristic of Fig. 2(b), this limitation reduces the dynamic range of the capacitance by about a factor of two. As a remedy, the varactors can be capacitively coupled to X and Y, allowing independent choice of dc levels. Illustrated in Fig. 19(b), such an arrangement defines the gate voltage of M_{v1} and M_{v2} by $V_b \approx V_{DD}/2$ through large resistors R_1 and R_2.

The coupling capacitors, C_{C1} and C_{C2}, must be chosen much greater than the maximum value of C_{var} so as not to limit the tuning range. For example, if $C_{C1} = C_{C2} = 5C_{var,max}$, then the equivalent series capacitance reaches only $5C_{var,max}^2/(6C_{var,max}) = 0.83C_{var,max}$, suffering from a 17% reduction in dynamic range. On the other hand, large coupling capacitors display significant bottom-plate capacitance,

[5] We assume that, at a given frequency, the Q is relatively independent of the inductance value.

[6] Since C_{GD} experiences a total voltage swing of $2V_{min}$, its Miller effect translates to a factor of two for each transistor.

thereby loading the oscillator and limiting the tuning range.[7] It is possible to realize C_{C1} and C_{C2} as "fringe" capacitors (Fig. 20) [7] to exploit the lateral field between adjacent metal

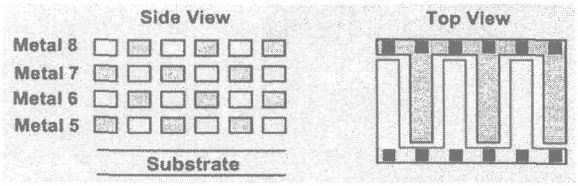


Fig. 20. Fringe capacitor.

lines. This structure exhibits a bottom-plate parasitic of a few percent, but its value must usually be calculated by means of field simulators.

The tuning range of LC VCOs must be wide enough to encompass (a) process and temperature variations, (b) uncertainties due to model inaccuracies; and (c) the frequency band of interest. In wireless communications, the last component makes the design particularly difficult, especially if a single VCO must cover more than one band. For example, in the Global System for Mobile Communication (GSM) standard, the transmit and receive bands span 890-915 MHz and 935-960 MHz, respectively. For one VCO to operate from 890 MHz to 960 MHz, the tuning range must exceed 7.8%. With another 7 to 10% required for variations and model inaccuracies, the overall tuning rang reaches 15 to 18%, a value difficult to achieve. In such cases, two or more oscillators may prove necessary, but at the cost of area and signal routing issues.

The phase noise of each oscillator topology must be quantified carefully. The reader is referred to the extensive literature on the subject.

B. Digital Tuning

Our study thus far implies that it is desirable to maximize the tuning range. However, for a given supply voltage, a wider tuning range inevitably translates to a greater VCO gain, K_{VCO}, thereby making the circuit more sensitive to disturbance ("ripple") on the control line. This effect leads to larger reference sidebands in RF synthesizers and higher jitter in timing applications. With the scaling of supply voltages, the problem of high K_{VCO} has become more serious, calling for alternative solutions.

A number of circuit and architecture techniques have been devised to lower the sensitivity of the VCO to ripple on the control line. For example, a *digital* tuning mechanism can be added to perform coarse adjustment of the frequency, allowing the analog (fine) control to cover a much narrower range. Illustrated in Fig. 21(a), the idea is to switch constant capacitors into or out of the tanks, thereby introducing discrete frequency steps. The varactors then tune the frequency within each step, leading to the characteristic shown in Fig. 21(b). Note that the switches are placed between the capacitors and ground - rather than between the tank and the capacitors. This permits

[7]This is relatively independent of whether the bottom plates are connected to nodes X and Y or to R_1 and R_2.

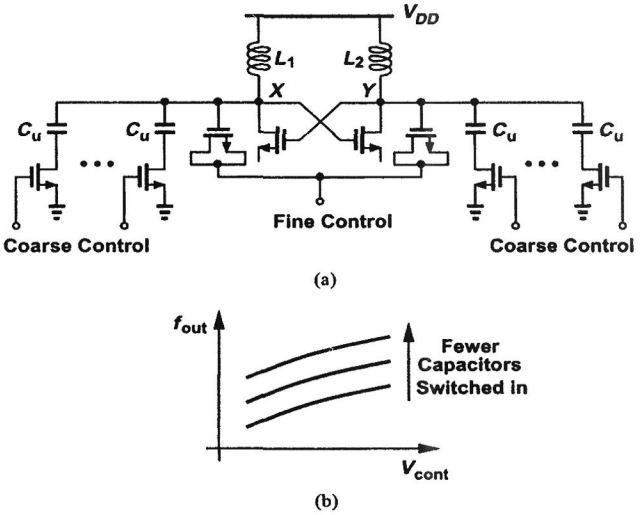


Fig. 21. (a) VCO with fine and coarse digital control, (b) resulting characteristics.

the use of NMOS devices with a gate-source voltage equal to V_{DD}, minimizing their on-resistance.

The above technique entails three critical issues. First, the trade-off between the on-resistance and junction capacitance of the MOS switches translates to another between the Q and the tuning range. When on, each switch limits the Q of its corresponding capacitor to $(R_{on}C_u\omega)^{-1}$. When off, each switch presents its drain junction and gate-drain capacitances, $C_{DB} + C_{GD}$, in series with C_u, constraining the lower bound of the capacitance to $C_u(C_{DB} + C_{GD})/(C_u + C_{DB}C_{GD})$ rather than zero. In other words, wider switches degrade the overall Q to a lesser extent but at the cost of narrowing the discrete frequency steps.

The second issue relates to potential "blind" zones in the characteristic of Fig. 21(b). As exemplified by Fig. 22, if the

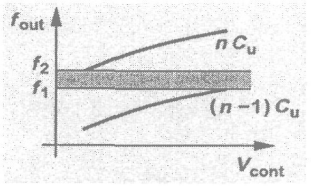


Fig. 22. Blind zone resulting from insufficient fine tuning range.

discrete step resulting from switching out one unit capacitor is greater than the range spanned continuously by the varactors, then the oscillator fails to assume the frequency values between f_1 and f_2 for any combination of the digital and analog controls. For this reason, the discrete steps must be sufficiently small to ensure overlap between consecutive bands.[8]

The third issue stems from the loop settling speed. As described below, the PLL takes a long time to determine how

[8]With a finite overlap, however, more than one combination of digital and analog controls may yield a given frequency. To avoid this ambiguity, the loop must begin with a minimum (or maximum) value of the digital control and adjust it monotonically.

many capacitors must be switched into the tanks. Thus, if a change in temperature or channel frequency requires a discrete frequency step, then the system using the PLL must remain idle while the loop settles.

When employed in a phase-locked loop, the oscillator of Fig. 21(a) requires additional mechanisms for setting the digital control. Figure 23 depicts an example for frequency synthesis.

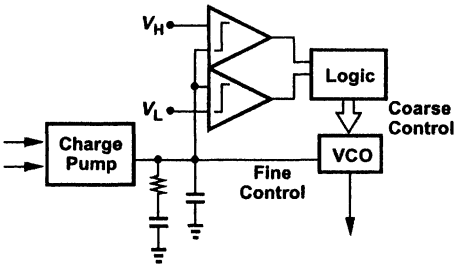


Fig. 23. Synthesizer using fine and coarse frequency control.

Here, the oscillator control voltage is monitored and compared with two low and high voltages, V_L and V_H, respectively. If V_{cont} falls below V_L, the oscillation frequency is excessively low[9], and one unit capacitor is switched out. Conversely, if V_{cont} exceeds V_H, one unit capacitor is switched in. After each switching, the loop settles and, if still unlocked, continues to undergo discrete frequency steps.

[9] We assume the frequency increases with V_{cont}.

REFERENCES

[1] B. Razavi, "Design of Monolithic Phase-Locked Loops and Clock Recovery Circuits - A Tutorial," in *Monolithic Phase-Locked Loops and Clock Recovery Circuits*, B. Razavi, Ed., Piscataway, NJ: IEEE Press, 1996.

[2] P. Larsson, "Parasitic Resistance in an MOS Transistor Used as On-Chip Decoupling Capacitor," *IEEE J. Solid-State Circuits*, vol. 32, pp. 574-576, April 1997.

[3] K. Kundert, Private Communication.

[4] M. Danesh et al., "A Q-Factor Enhancement Technique for MMIC Inductors," *Proc. IEEE Radio Frequency Integrated Circuits Symp.*, pp. 217-220, April 1998.

[5] A. Zolfaghari, A. Y. Chan, and B. Razavi, "Stacked Inductors and Transformers in CMOS Technology," *IEEE Journal of Solid-State Circuits,* vol. 36, pp. 620-628, April 2001.

[6] F. Behbahani, et al., "A 2.4-GHz Low-IF Receiver for Wideband WLAN in 0.6-μm CMOS," *IEEE Journal of Solid-State Circuits,* vol. 35, pp. 1908-1916, December 2000.

[7] O. E. Akcasu, "High-Capacity Structures in a Semiconductor Device," US Patent 5,208,725, May 1993.

Delay-Locked Loops - An Overview

Chih-Kong Ken Yang

Abstract — **Phase-locked loops have been used for a wide range of applications from synthesizing a desired phase or frequency to recovering the phase and frequency of an input signal. Delay-locked loops (DLLs) have emerged as a viable alternative to the traditional oscillator-based phase-locked loops. With its first-order loop characteristic, a DLL both is easier to stabilize and has no jitter accumulation. The paper describes design considerations and techniques to achieve high performance in a wide range of applications. Issues such as avoiding false lock, maintaining 50% clock duty cycle, building unlimited phase range for frequency synthesis, and multiplying the reference frequency are discussed.**

I. Introduction

Many applications require accurate placement of the phase of a clock or data signal. Although simply delaying the signal could shift the phase, the phase shift is not robust to variations in processing, voltage, or temperature. For more precise control, designers incorporate the phase shift into a feedback loop that locks the output phase with an input reference signal that indicates the desired phase shift. In essence, the loop is identical to a phase-locked loop (PLL) except that phase is the only state variable and that a variable-delay line replaces the oscillator. Such a loop is commonly referred to as a delay-line phase-locked loop or delay-locked loop (DLL). As with a PLL, the goals are (1) accurate phase position or low static-phase offset, and (2) low phase noise or jitter.

Because a DLL does not contain an element of variable frequency, it historically has fewer applications than PLLs. Bazes in [1] demonstrated an example of precisely delaying a signal in generating the timing of the row and column access strobe signals for a DRAM. Another common application uses a DLL to generate a buffered clock that has the same phase as a weakly-driven input clock. Johnson in [2] synchronizes the timing of the buffered clock of a floating-point unit with the clock of a microprocessor. A similar application recovers the data of a parallel bus by generating a properly positioned sampling clock. Typically, these systems provide a sampling clock with the same sampling rate but with an arbitrary phase as compared to the data (i.e. a "mesochronous" system [4]). A clocked DRAM data bus is an example of such a system. A clock propagates with the data as one of the signals in the bus and therefore has a nominally known phase relationship with the data. However, in order to receive and buffer the clock to sample the data bus, the actual sampling clock is no longer properly aligned with the data. A DLL is commonly used to lock the phase of the buffered clock to that of the input data. The phase locking significantly reduces timing uncertainty in sampling the data, which then enables higher data rates as in [3].

Although aperiodic signals can also be delayed by the delay line in a DLL, the inputs to delay lines are typically clock signals. By using a periodic signal, the delay lines do not need arbitrarily long delays and typically only need to span the period of the clock to generate all possible phases. A data signal can be delayed by sampling the data with the appropriately delayed clock.

The motivation for using DLLs is that the design of the control loop is simplified by having only phase as the state variable. Section II reviews how such a loop is unconditionally stable and has better jitter characteristics. However, a DLL is not without its own limitations. The variable delay line has a finite delay range and finite bandwidth. Section II also discusses these design considerations. Section III describes different implementations of the variable delay line. Within the past ten years, modifications to the basic DLL architecture have enabled clock and data recovery applications in "plesiochronous" systems [4] where the sampling rates for clock and data differ by a few hundred parts-per-million in frequency. Delay lines with effectively infinite delay are also addressed in Section III.

More recently, several researchers such as [5] and [6] have introduced architectures that permit frequency multiplication based on delay lines which further extends their use in clock generation and frequency synthesis. Section IV describes these architectures.

II. DLL Characteristics

The basic loop building blocks are similar to that of a PLL: a phase detector, a filter, and a variable-delay line. Figure 1 illustrates the three main functional blocks. Since phase is the only state variable, a control loop higher than first-order is not needed to compensate a fixed phase error. The resulting transient impulse response is a simple exponential. Although the simple loop characteristics are an advantage that DLLs have over PLLs, the design is complicated by the additional circuitry that is needed to overcome having a limited delay range and not producing its own frequency.

A. First-order Loop

A phase detector compares the phase of the reference input and the delay-line output. The comparison yields a signal proportional to the phase error. The error is low-pass

C.K. Ken Yang is with University of California at Los Angeles, yang@ee.ucla.edu.

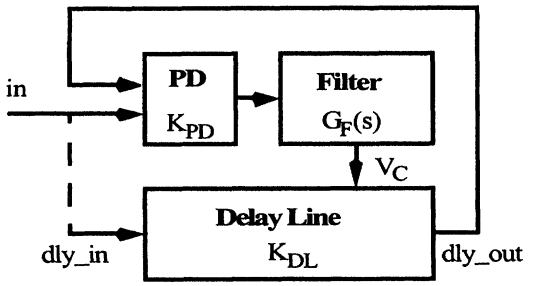


Figure 1: DLL architecture.

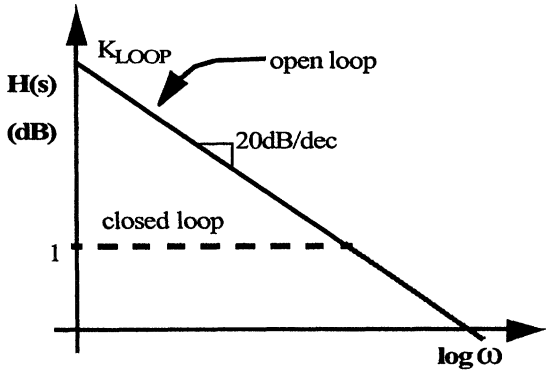


Figure 2: Open- and closed-loop transfer characteristics.

filtered to produce a control voltage or current that adjusts the delay of the delay line. The delay-line input can be either the reference input or a clean clock signal.

The s-domain representation of each loop element is depicted within each block in Fig. 1. The open-loop transfer function can be written as $T(s) = K_{PD}K_{DL}G_F(s)$ where K_{PD} is the phase-detector gain, $G_F(s)$ is the filter transfer function, and K_{DL} is the delay-line gain. If the loop has finite gain at dc, the resulting output signal will exhibit a static phase error as shown in the following equation.

$$H(s)\big|_{s=0} = \frac{1}{1 + 1/(K_{PD}K_{DL}G_F(s))}\bigg|_{s=0} \qquad (1)$$

To eliminate the static phase error, the filter is often an integrator to store the phase variable. This results in a first-order closed-loop transfer function.

$$H(s) = \frac{1}{1 + (s/K_{PD}K_{DL}G_F)} \qquad (2)$$

The equation assumes that the delay-line input is a clean reference as opposed to the reference input. Higher-order loop filters have not commonly been used but can enable better tracking of a phase ramp (i.e. a frequency difference).

Figure 2 shows the open-loop and closed-loop transfer functions. With only a single integrator, the open-loop phase margin is 90°. The loop is unconditionally stable as long as the delay in the loop does not degrade the phase margin excessively. The closed-loop transfer function illustrates that

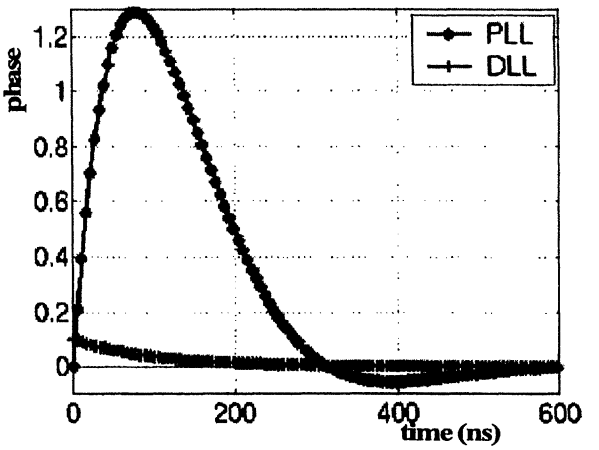


Figure 3: Step response of PLL and DLL (with same loop characteristics).

the tracking of the phase of the input clock changes at different frequencies. Based on the transfer function, the loop bandwidth is $\omega_{bw} = K_{PD}K_{DL}G_F$. For frequencies within the loop bandwidth the phase of the output clock will track that of the reference input and reject noise within the loop. The phase characteristics of the output clock above the bandwidth of the loop depend on the phase behavior of the delay-line input and the noise from the delay line. The noise transfer function from a noise source lumped at the delay-line output is a high-pass response.

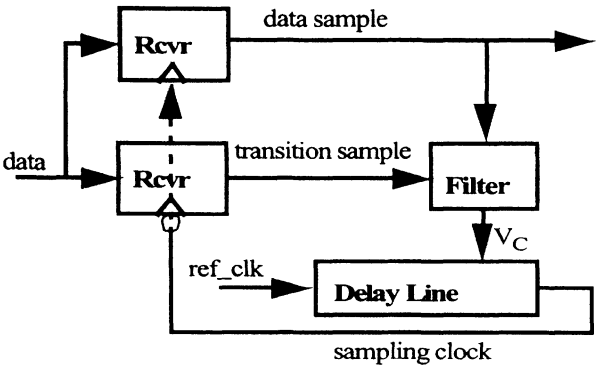
$$H(s) = \frac{(s/K_{PD}K_{DL}G_F)}{1 + (s/K_{PD}K_{DL}G_F)} \qquad (3)$$

In some degenerate cases, the delay-line input is also the reference input. The feedback loop would guarantee a fixed phase relationship between the delay-line output and the reference so any phase variations in the reference would directly appear at the delay-line output in an all-pass response. However, noise due to the delay line is still high-pass filtered.

B. Advantages over a PLL

The loop characteristics are considerably simpler than those of a PLL. A PLL would contain at least two states to store both the frequency and phase information. In order to maintain loop stability, an additional zero is needed. A DLL is less constrained with only a single pole. The loop gain directly determines the desired bandwidth. The only stability consideration is when the loop bandwidth is very near the reference frequency. The periodic sampling nature of the phase detection and the delay in the feedback loop degrade the phase margin. For instance, if the feedback delay is one reference cycle, the loop bandwidth should not exceed 1/4 of the reference frequency.

Figure 3 illustrates the response to a noise step applied to the control voltage for both a PLL and a DLL. A PLL accumulates phase error due to its higher-order loop characteristic. In response to a phase error, the control



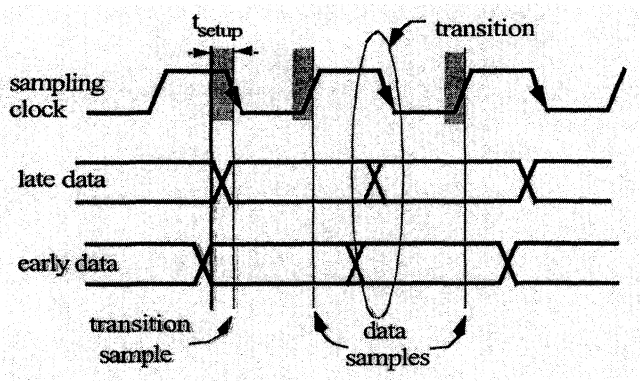


Figure 4: Early-late receiver architecture using the receiver as the phase detector. Timing diagram showing early and late data.

voltage alters the frequency of an oscillator. The output phase is an integration of the frequency change. In response to a noise perturbation, the loop accumulates a phase error before correcting. In contrast, a DLL attenuates the phase error by the time constant of the loop. In the figure, both loops are designed with the same 3-dB bandwidth, the same delay elements, and the PLL is a 2nd-order loop with a damping factor of unity. Clearly, the PLL suffers from larger phase errors due to the phase accumulation.

A second advantage relates to clock and data recovery applications. An effective way to recover the timing for sampling a data input is to use the data receiver as a phase detector. The architecture, depicted in Fig. 4, uses the 180°-shifted clock to sample the data transitions in addition to sampling the data values [7]. Whenever data changes values, the sampled transition and the data values can be combined to indicate whether the sampling clock edge is earlier or later than the data transition. Phase information is only present with data transitions. The feedback loop locks when the transition sampling clock samples a metastable value. This commonly used design is known as an early-late or bang-bang architecture. The timing diagram in Fig. 4 illustrates examples of the data being early and late. Due to the inherent setup time of the data receiver, the transition sampling clock may not occur at the same time as the data transition. The phase shift compensates for the receiver setup time and maximizes the margin of error for the data sampling.

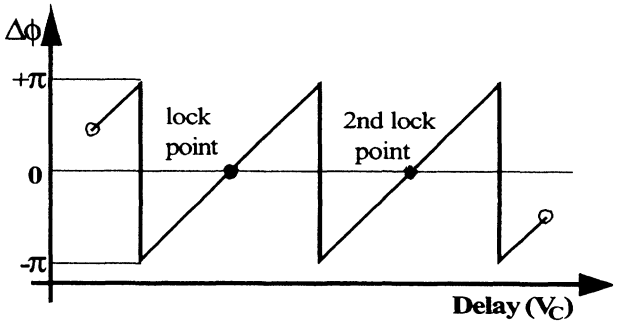


Figure 5: Delay line phase/delay characteristic.

However, the data receiver is ultimately a binary comparator and the phase detector does not indicate an error that is proportional to the phase difference. Hence, the timing-recovery loop is nonlinear. Although a higher-order PLL using early-late control can be made conditionally stable [8], the resulting phase dithers with a limit cycle translating into jitter. The oscillation depends on the loop parameters and can be considerable for high bandwidth loops. With an early-late DLL, the phase of the clock output also dithers. But because the stability only depends on the delay within the loop, the dithering would only be a few cycles and can be significantly less than the dithering of a PLL.

C. Design Considerations in a DLL

A typical DLL involves several design considerations. First, the delay line usually has a finite delay range. If the desired phase of the output signal is beyond the delay range, the loop will not lock properly. Second, the output of the DLL also depends greatly on the input to the delay line. Since the delay-line input propagates to the DLL output, tracking jitter and the output's duty cycle depend not only on the delay-line design but also on the delay-line input. Third, the basic DLL cannot generate new frequencies different from that of the delay-line input.

A variable-delay line adjusts the delay by varying the RC time constant of a buffer and often has limited adjustment range. Section III will describe several techniques in greater detail. Even though the delay range is limited, DLLs for a periodic clock signal only need the range to exceed 2π in phase across process and systematic variations to cover all possible phases. For systems with a range of operating frequencies, the delay line must span 2π for the lowest input frequency.

An issue known as false-locking occurs when the delay range exceeds 2π. There can be several secondary lock points repeating every 2π. Figure 5 depicts an example of the characteristic of a delay line with two lock points. Since phase detectors must be periodic, if the delay line initializes within π of the second lock point, the phase detector will push the delay line toward lock with a longer than necessary delay. Long delays require large RC time constants for a given variable-delay buffer element. The bandlimiting by the

filter would significantly attenuate a high-frequency input clock. The attenuation increases the jitter and may even prohibit the input from reaching the output.

Even if the delay line is constrained to span only one lock point but greater than 2π, a second similar issue exists. It is difficult to design a delay line such that the adjustable range is exactly $-\pi$ to $+\pi$ across different operating and processing conditions. If initialized at the minimum or maximum delay, the phase detector may push the loop toward either the maximum or minimum delay limit and "false-lock" to an incorrect phase.

To address false-locking, designers employ several techniques depending on the application. For systems that require a delay line with a known fixed delay, operating condition variations may be small enough such that the delay line only needs a small variable range that is less than $+\pi$ and $-\pi$. For systems that lock to a fixed phase over a wide range of frequencies, one design [9] uses an auxiliary frequency-sensing loop that generates a voltage to coarsely set the delay for the given input frequency. Then DLL only fine tunes the delay for the desired phase. For data recovery applications where the clock phase can be arbitrary with respect to the data, a common design uses a startup circuit for the DLL that initializes the delay line at its minimum delay to avoid any secondary lock points. However, as mentioned earlier, the phase detector may keep the delay line at the minimum delay. A sensing circuit or a state machine detects when the delay line is at its limit and optionally inverts the feedback clock. The phase would flip by 180° and the loop would lock properly. As will be discussed in Section III, a more robust alternative reconfigures the delay line such that the delay only spans 2π and wraps back to 0° when the delay exceeds 360°.

The jitter and duty cycle of the delay-line output clock depend on the input, the coupling of the input to the delay line, and the delay line itself. Often the input is from off-chip and, therefore, it must be carefully received to prevent supply and substrate noise from coupling onto the signal as jitter. In contrast, the high-frequency phase noise of the clock output of an oscillator-based PLL depends primarily on the oscillator design. An improperly received input clock can often result in worse jitter performance in a DLL as compared to a PLL. Similarly, while the duty cycle from an oscillator is only modestly distorted (by the difference between the rising edge and falling edge delays), the duty cycle of the DLL's input clock can be significantly distorted as it propagates to the output. Since duty cycle is a systematic error, a good design corrects duty cycle using an explicit block instead of compounding the difficulty of the delay-line design.

A duty-cycle corrector (DCC) is commonly added to either the DLL input or output. Figure 6 illustrates the basic components of the feedback loop: an input with finite slew rate, a buffer element with adjustable threshold, a comparator, and an integrator. The comparator determines the threshold crossing of the clock waveform. The result is integrated and used to skew the threshold of the buffer stage.

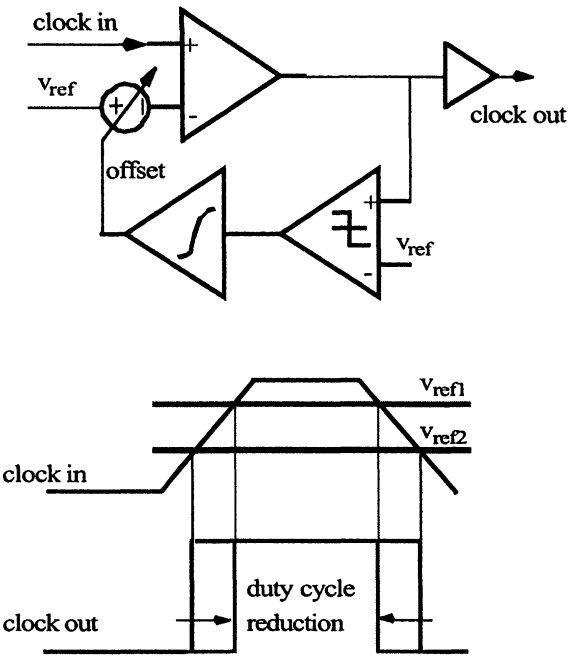


Figure 6: Duty-cycle corrector block diagram. Timing diagram shows change in duty cycle with changing offset.

Because the buffer input has finite slew rate, changing the threshold effectively adjusts the output high and low half-periods. The loop settles when the high and low half-periods are equal. Figure 6 illustrates the reduction in duty cycle as the threshold shifts from V_{ref1} to V_{ref2}. Since random variation of the duty cycle effectively appears as jitter, single-ended implementations such as that shown in the figure can be very sensitive to common-mode noise. For this reason, differential architectures are preferred [3].

For low jitter on the output clock, the loop components must be carefully designed. Many of the loop components are very similar to that of a PLL and are well described in [10]. For a charge-pump based loop filter, since the filter is only first-order, a simple capacitor replaces the RC filter. As in a PLL, noise on the control voltage directly translates into jitter. Designers may use additional filtering to suppress the noise. The loop element that has deviated the most from PLL design and is critical for functionality and performance is the design of the delay line.

III. Delay-Line Architectures

The primary characteristics of a delay line are (1) gain (i.e. change in delay for a given change in voltage), and (2) delay range. For most applications using periodic inputs, the absolute delay is not critical as long as the range spans 2π. Because delay lines are relatively short, they do not contribute significant thermal or 1/f phase noise. However, for large digital systems, low supply/substrate sensitivity is needed to reject the on-chip switching noise.

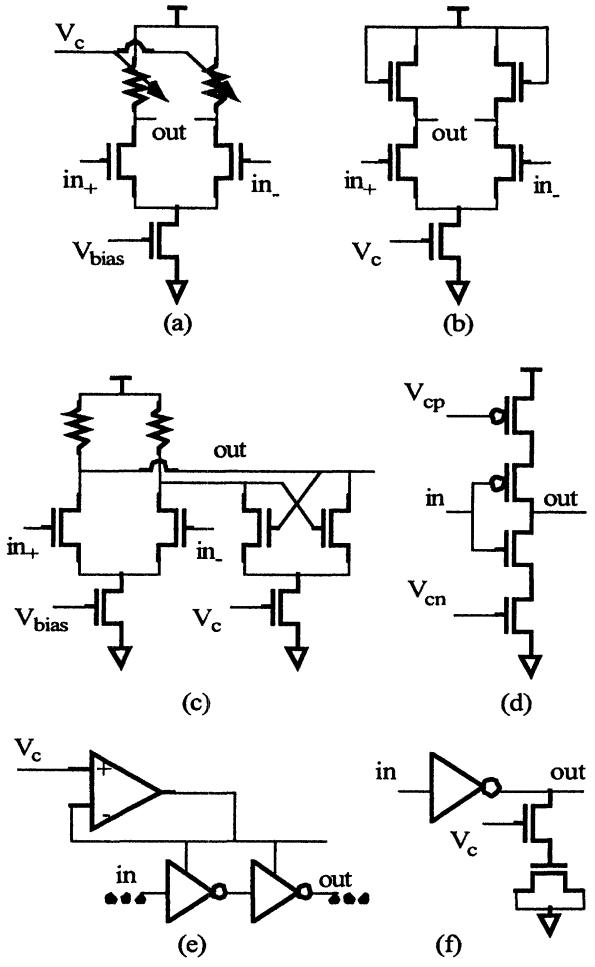


Figure 7: Six different delay elements.

A. Basic Delay Line

A delay line comprises of a chain of variable-delay elements. Each element is controllable by either a voltage or a current. The delay of each element is proportional to its RC time constant and changing the effective resistance or capacitance adjusts the delay.

Figure 7 depicts several examples of buffer elements. For a differential buffer, the load resistance can be an MOS transistor in the triode region [Fig. 7-(a)] where the resistance is proportional to V_{GS}-V_{th}. Varying the gate voltage adjusts the delay of the element. A non-linear device such as a diode can also serve as a load resistance [Fig. 7-(b)]. Since the resistance varies with the current, varying the bias current of the buffer would adjust the delay. Similarly, a negative transconductance that changes with the bias current can be placed in parallel with a fixed load resistance [Fig. 7-(c)]. The varying negative transconductance changes the effective load resistance and hence varies the delay. Because nonlinear elements have resistances that depend on both voltage and current, they can be more sensitive to supply noise.

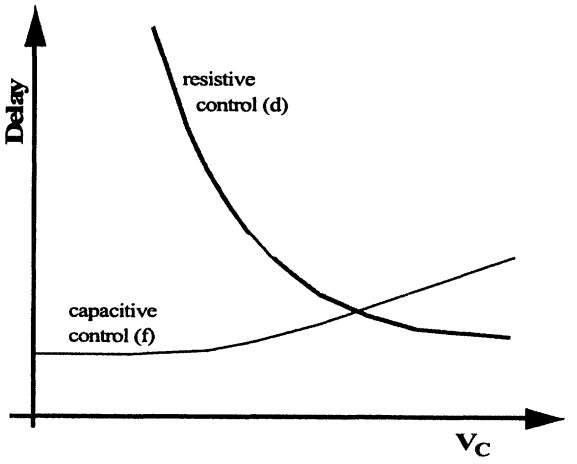


Figure 8: Delay versus voltage for two different delay buffer elements: types (d) and (f) of Fig. 7.

For push-pull type elements such as inverters, the delay can be changed by changing the rate at which the output capacitance is charged [Fig. 7-(d)]. An adjustable current source limits the peak current of an inverter and varies the delay. An alternative method regulates the supply voltage of the inverters and uses the control voltage to set the supply voltage [Fig. 7-(e)]. The effective switching resistance varies with the supply voltage. Instead of changing the resistance, the effective capacitance can also be made adjustable [Fig. 7-(f)]. A transistor that behaves as an adjustable resistance can be used to decouple an explicit output capacitance. The larger the resistance the less capacitance is seen at the output.

Figure 8 illustrates the delay versus control voltage for a resistively-controlled delay element. For the element of Fig. 7-(d), either V_{GS}-V_{th} or the bias current can be zero and, therefore, a single element's delay can span from the minimum buffer delay to infinite. However, since the time constant is proportional to the delay, a long delay setting would significantly attenuate a high-frequency clock. Delay lines with a wide range for high clock frequencies require a large number of broadband delay elements.

Unlike resistive control, the maximum delay in a capacitively-controlled element [Fig. 7-(f)] is proportional to $R(C_{int}+C_{exp})$ and the minimum delay is proportional to RC_{int} where C_{int} is the intrinsic capacitance of the buffer and the load of the subsequent stage, and C_{exp} is the explicit capacitance added to the circuit. Because of the limited range per buffer, obtaining a wide delay range involves a large number of buffers. The maximum delay of each buffer is chosen to avoid attenuating the signal. In designs where the clock has a large voltage swing, the transistor in series with the explicit capacitance no longer appears as a variable resistor because the device enters saturation and cut-off. For these buffers, the control voltage determines the fraction of current and period of time in which the buffer's current charges the explicit capacitance.

An example of the delay versus control voltage for a capacitively-controlled element is overlaid in Fig. 8. Most

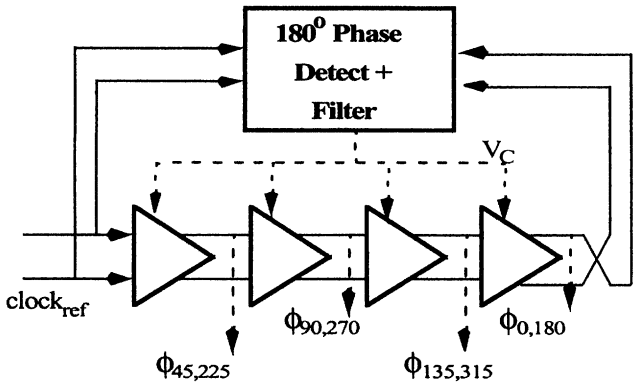


Figure 9: 180°-locked DLL to generate intermediate phases that are a fraction of a cycle.

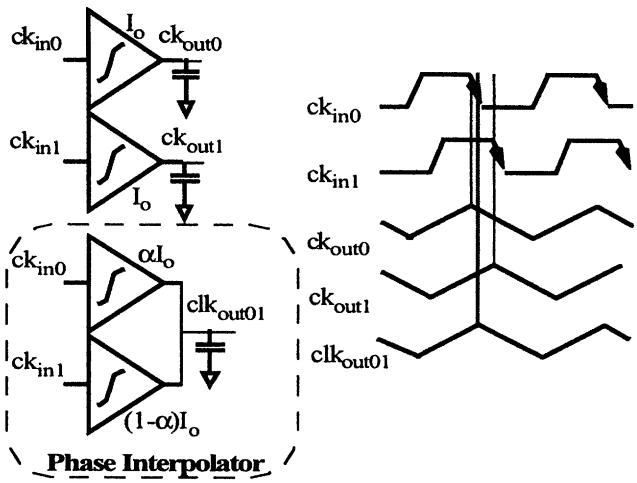


Figure 10: Phase interpolator design by shorting of the output of two integrators/buffers. .

delay elements exhibit some nonlinearity. As a result, the delay-line gain, K_{DL}, is a function of the delay. Because a DLL is unconditionally stable, the loop still functions with the varying loop parameter. However, more linear elements are better for designs that require a constant loop bandwidth. To compensate for the variable K_{DL}, designers add programmability to the loop-filter capacitor.

The control signal for either type of delay elements can be digital. In a digital implementation [11], the current source is binary weighted and switched by a digital word. For capacitively-controlled elements, the capacitance can be binary weighted and switched. A nearly all-digital DLL is then possible by using a simple counter to replace the analog integrating filter.

B. Phase Interpolation

Instead of only using the clock phase at the end of a delay line, an earlier clock phase can be tapped from the middle of a delay line. Some applications require the delay line to produce a delay that is a fixed *fraction* of the input-clock period. Figure 9 shows one implementation that uses a DLL to lock the input clock to the output. An 180° phase detector would guarantee the absolute delay of a delay line to be a half-cycle. Tapping from different points on the delay line provides different phases. As shown in Fig. 9, for a 45° phase shift, the clock can be tapped from the first delay stage of a 4-stage differential delay line. If an arbitrary phase is needed, each delay stage can be tapped and multiplexers can select the nearest desired phase. The number of delay elements quantizes the phase step and limits the resolution [12]. Fine phase resolution requires longer delay lines. Yet, the resolution is limited at high clock frequencies because the maximum number of delay elements needed to span 180° is limited.

An arbitrary intermediate phase can be obtained by "interpolating" between two clock phases that are tapped from a delay line. Depending on the weighting, an interpolator produces a clock that has a programmable output phase in between the input clock phases. As long as discrete clock phases that span the entire cycle are available as inputs, any phase for the interpolator's output is possible. Multiplexers are needed to select the phases to interpolate between. For example, with phases tapped from a 4-stage delay line, if the desired output clock phase is 120°, the interpolator inputs would be from the second and third delay elements.

Interpolators essentially perform a weighted average of the input phases. As shown in Fig. 10, ideally, the two input phases drive two integrators which charge a single output. The weighting of the average is by the relative currents of the two integrators. When α=1, the output clock phase depends only on ck_{in0}. When α=0.5, i.e. the current is split equally between the two integrators, the output phase is additionally delayed by half the phase difference. As illustrated in Fig. 10, the phase of the interpolated output (ck_{out01}) falls between the phases of the non-interpolated outputs (ck_{out0} and ck_{out1}).

With ideal integrators, the interpolation is linear, resulting in a constant K_{DL}. Alternatively, an interpolator can effectively be formed with buffer elements instead of integrators. By weighting the drive strength or current of two buffer elements whose outputs are shorted together, one can adjust the output phase. Because the output is not integrated, the resulting interpolation is slightly nonlinear and depends on (1) the phase difference between the inputs and (2) the slew rate (or time constant) of the input and output signals [13]. Figure 11 depicts the linearity of the interpolation for two different input phase separations, s=τ and s=2τ where τ is the buffer's time constant. The larger phase spacing results in greater nonlinearity. Similar to RC delay elements, the interpolation can be digitally controlled. Since the weighting of the interpolation depends on the proportional current, the current sources of the integrators or buffers can be digitally weighted and programmed.

In a design for clock and data recovery by [3], quadrature clocks are interpolated to generate an intermediate clock phase within a quadrant. Figure 12 illustrates the mostly analog architecture. An analog control

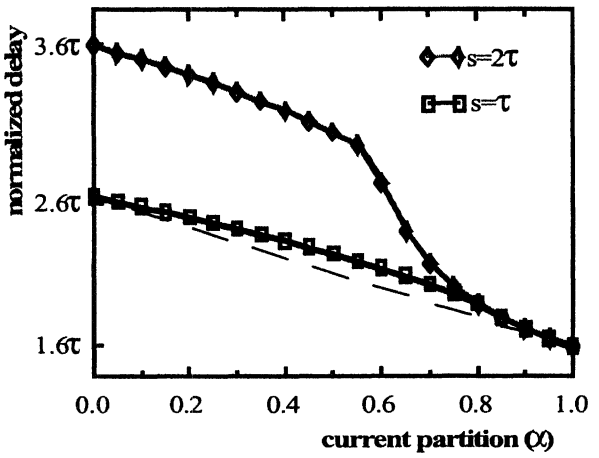


Figure 11: Buffer based phase interpolator linearity.

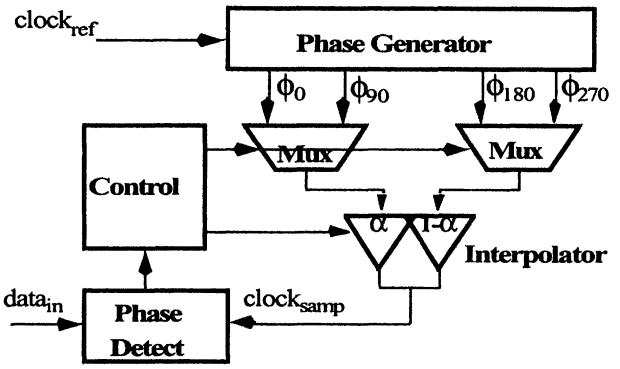


Figure 12: Infinite-range delay line based on phase rotation.

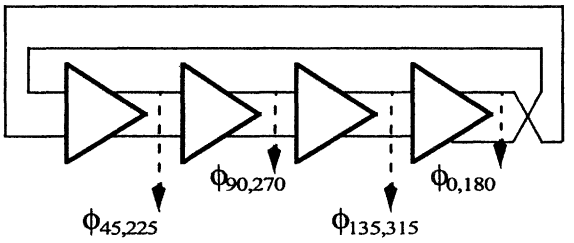


Figure 13: Oscillator with tapped outputs for multiple phases.

voltage produced by the phase detector and filter determines the interpolator currents. Comparators indicate when the current is fully steered to one integrator. A finite state machine driven by the comparators selects the appropriate quadrant by switching the interpolator inputs such that all 360° phases are possible. The quadrature input clocks is generated from an external reference clock through the use of a divide-by-two circuit.

Interestingly, because the phase rotates from one quadrant to the next, the architecture effectively has an unlimited delay range. If the input data rate and the reference clock frequency are slightly different, a DLL would continually increase or decrease the delay in order to track the accumulating input phase. A typical DLL with a finite delay range would run out of delay or lose lock. On the other hand, plesiochronous operation is possible with an interpolator-based delay line since the phase smoothly rotates between quadrants.

Interpolating between clocks with large phase spacings such as quadrature clocks results in an output clock with slow slew rate. Such waveforms are more susceptible to noise and result in higher jitter. An enhancement uses more closely-spaced phases that span the cycle. The finer phases spacing is possible using a multi-stage ring oscillator. As shown in Fig. 13, a 4-stage differential oscillator would generate 8 phases 45° apart. To guarantee a correct period for each clock phase, the ring oscillator is locked to the external reference clock using a PLL. The role of the PLL is solely for generating the phases. A purely DLL-based architecture is also possible by replacing by using the DLL in Fig. 9 that locks the delay-line output with a 180° phase shift [13].

The architecture is commonly known as a dual-loop design because the first loop, a PLL or DLL, generates the phases and the second loop, the interpolation-based DLL, recovers the data and phase. Since the first loop is not in the feedback of the second loop (or vice versa), the overall system is stable as long as each loop is individually stable. A dual-loop design is possible with the second loop within the feedback of the first loop [15] as long as the stability of the loop is carefully considered.

The data recovery portion of a dual-loop design is conducive to a digital implementation. The binary output of the receiver-replica phase detector can be accumulated using a digital counter. The counter output selects the appropriate phase from the oscillator and controls the digitally programmable interpolators [13],[14]. As long as the quantized phase step is small, the small error only minimally impacts the data recovery.

C. Oversampled Implementation

An alternative purely digital approach to clock and data recovery can be implemented by oversampling the data. Figure 14 illustrates an example of a digital architecture. Multiple finely-spaced clock phases oversample the data input. The sampled results are digitally processed to determine both the correct data value and the optimal phase of the data sample. The digital processing can vary in complexity. Simple implementations use the optimal data sample as the received data [18] or take a majority vote from the samples of a single bit [17]. The bit boundaries determine the samples associated with a bit. Transitions that are detected in the samples from the prior or current bits indicate the bit boundaries.

The sampling rate limits the timing error margin. Greater amount of oversampling reduces the data-recovery timing error, but increases the number of clock phases. Low data rate UARTs [16] typically use 8 to 16 times oversampling. For high data rates, generating accurate clock phases separated by sub-100ps is very challenging. More

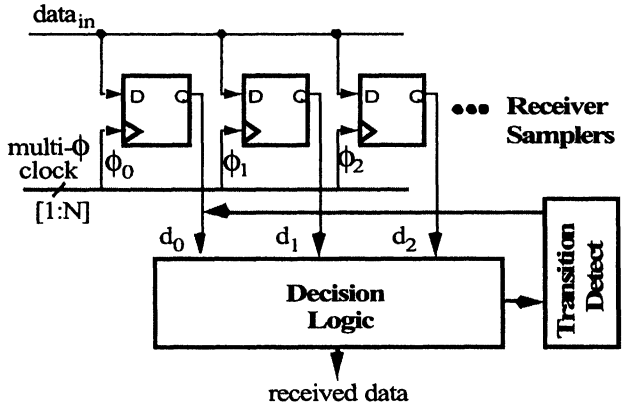


Figure 14: Oversampled data recovery architecture.

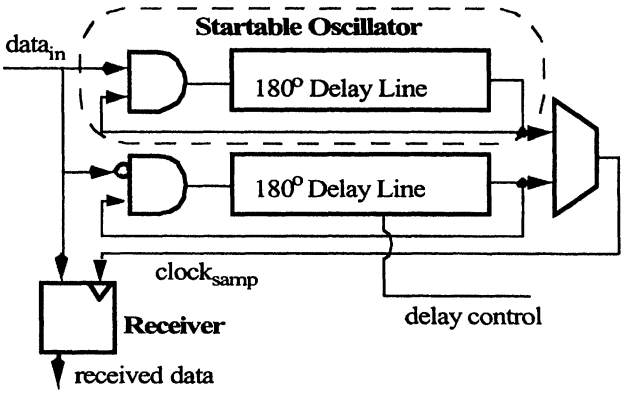


Figure 15: Clock/data recovery using startable oscillator.

aggressive designs with the least amount of clocking overhead and high data rates use a minimum of 3x oversampling [17], [18].

Even though phase spacing scales with the gate delay of a technology, so does the bit time in each generation of applications. For oversampling of the data bits, finely-spaced clock phases are needed. Tapping from a delay line produces phases separated by a buffer delay. For even finer phases, several techniques are commonly used. For example, several interpolators can be used where each interpolator has slightly different weighting to generate intermediate phases with spacing less than a buffer delay [19]. An alternative method uses a chain or array of coupled oscillators [20]. By taking a chain of oscillators and coupling them such that the output and input of the chain are separated by only one gate delay, sub-gate-delay phase spacings result from the outputs of each oscillator. Lastly, if the data can be delayed with a chain of delay buffers along with the clock, the clock at each delay stage can be used to sample the data of the corresponding stage. As long as the data and the clock delay lines have slightly different delays, the sub-sampled outputs are effectively an oversampling of the data. The effective phase spacing depends only on the difference between the data delay and clock delay [21]. The architecture has a drawback in that it requires delaying the data and clock by long delays of several cycles, which can significantly increase jitter.

IV. Clock Multiplication

With a dual-loop architecture, a DLL can produce a frequency plesiochronous to the delay-line input. However, the rate at which the interpolator weight changes limits the frequency difference. Generating a significantly different or multiplied frequency from a low-frequency input reference is not possible with the architecture.

Recently designers have explored several methods of using DLLs for frequency multiplication. One method uses a delay line that is locked to 180°. With the phases that span an entire cycle, the tapped clock edges are combined to form a clock with multiplied frequency. The most direct method uses logical AND-ORs to combine the multiple phased clocks into a single high-frequency clock [23]. Alternatively, the method in [24] converts each phase into a small pulse and ORs the pulses together to form the output clock. In cases where the output capacitance of the logic gates limits the output frequency, one design [6] uses phases to excite a tuned LC tank to combine the clock phases.

Instead of edge combining, the multiplied clock can be the direct output of a delay line. The architecture is similar to a technique for clock and data recovery that uses a startable oscillator [22]. As shown in Fig. 15, the architecture uses data transitions to trigger startable oscillators: high-value data triggers one oscillator and low-value data triggers another. Each startable oscillator comprises of a delay line and an AND gate. The data value enables the AND gate and the triggered oscillator propagates an edge through the delay elements and produces a clock edge delayed by a half-cycle. The edge is used to sample the data. In the absence of input transitions, the delay line is configured an oscillator and generates a sampling edge every cycle. Whenever a new data transition occurs, the oscillator resynchronizes its phase to that of the input. In the implementation by [22], the natural oscillation frequency of the oscillator is determined by an external plesiochronous clock reference. The architecture has not been widely applied to higher data rate designs because the sampling phase is directly derived from the input data without any filtering. The deterministic and random jitter inherent in the data are effectively doubled and can be considerable.

If the input is a low-jitter reference clock, a similar architecture can be used for clock multiplication [5]. As illustrated in Fig. 16, a lower frequency but clean reference clock is one input to a multiplexer that feeds into a delay line. The output of the delay line is fed back to the multiplexer as the second input. When a reference clock edge is available, the multiplexer selects the reference input. Otherwise, the multiplexer configures the delay line as an oscillator with the output frequency controlled by the delay. The multiplexer inputs are selected by a counter circuit that determines the number of cycles to oscillate before accepting the next reference clock edge. A phase detector compares the

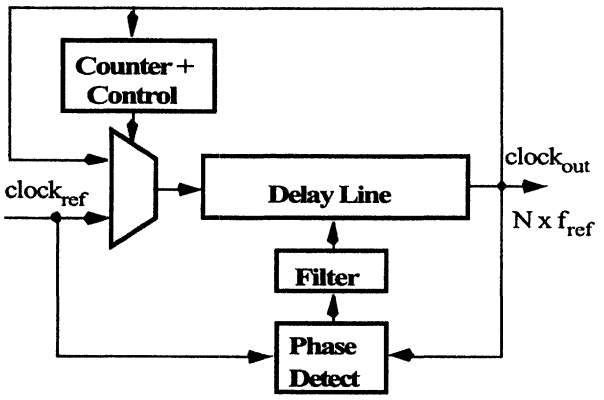


Figure 16: DLL-based clock multiplication.

reference input with the oscillator output and tunes the delay of the delay elements. Once locked, the resulting output clock frequency is a multiple of the input reference frequency. Recent designs [26] extend the frequency range and use an interpolator instead of a multiplexer to blend the delay-line feedback and the low-frequency reference clocks.

Both edge-combining multiplication and delay-line multiplication reduce the phase noise of the output clock because the core DLL does not have an oscillator that accumulates phase error. After N cycles, where N is the divide ratio, a new clean reference clock edge arrives and resets any accumulated phase error to zero. The architecture potentially lowers jitter by eliminating the peaking in the transfer function and allows a high tracking bandwidth. However, matching is critical in these designs. Mismatches in the phase detector or charge pump result in a static phase error that modulates the output frequency at the input reference frequency. Similarly, in the edge combining implementations, if the delay line is mismatched, the output clock would contain significant reference tones. Designers either choose the reference frequency carefully so that the tones do not impact the system performance or employ additional circuitry to compensate for the mismatches.

V. Conclusion

DLLs have been commonly used for generating precise phase delays of a signal and have been increasingly popular in clock generation and data recovery applications. Most importantly, because of the first-order loop characteristics that controls the phase directly, DLLs can be designed with high tracking bandwidths and do not exhibit the phase accumulation of an oscillator-based PLL.

The more simple loop characteristics belie many subtleties in DLL design. The delay-line input clock must have low-jitter and good duty-cycle. Furthermore, it must be carefully received and coupled to the input of the delay line to maintain good jitter performance. This source of jitter counter-balances the jitter accumulation of PLLs and results in less jitter improvement. Additional circuitry is often needed to prevent false-locking. Since a delay line does not restore a clock's duty cycle, the output clock requires correction circuitry. To use DLLs in plesiochronous systems, the delay line must have even more circuitry to achieve an unlimited delay range. In clock multiplication applications, very careful matching in the DLL components is critical to eliminate reference tones. In the many designs that have addressed these subtleties, DLLs have demonstrated low-jitter clock outputs for a variety of clock generation and data recovery applications.

References

[1] Bazes, M., "A Novel Precision MOS Synchronous Delay Line," *IEEE Journal of Solid-State Circuits*, vol sc-20, no 6, Dec. 1985, pp. 1265-71

[2] Johnson, M.G., E.L. Hudson, "A Variable Delay Line PLL for CPU-Coprocessor Synchronization," *IEEE Journal of Solid-State Circuits*, vol 23, no 5, Oct. 1988, pp. 1218-23

[3] Lee, T.H., et. al., "A 2.5V CMOS Delay-Locked Loop for an 18 Mbit, 500Megabytes/s DRAM," *IEEE Journal of Solid-State Circuits*, vol 29, no 12, Dec. 1994, pp. 1491-6

[4] Messerschmitt, D.G., "Synchronization in Digital System Design," *IEEE Journal on Selected Areas in Communications*, Oct. 1990, pp. 1404-1420

[5] Waizman, A., "A Delay Line Loop for Frequency Synthesis of De-Skewed Clock," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 1994, San Francisco, Session 18.5

[6] Chien, G., P.R. Gray, "A 900-MHz Local Oscillator Using a DLL-Based Frequency Multiplier Technique for PCS Applications," *IEEE Journal of Solid-State Circuits*, vol 35, no 12, Dec. 2000, pp. 1996-9

[7] Alexander, J.D., "Clock Recovery from Random Binary Data," *Electronic Letters*, vol 11, Oct. 1975, pp 541-2

[8] D'Andrea, N.A., F. Russo, "A binary quantized digital phase locked loop: a graphical analysis," *IEEE Transactions on Communications*, vol.COM-26, (no.9), Sept. 1978. p.1355-64

[9] Moon, Y., "An All-Analog Multiphase Delay-Locked Loop Using A Replica Delay Line for Wide-Range Operation and Low-Jitter Performance," *IEEE Journal of Solid-State Circuits*, vol 35, no 3, Mar. 2000, pp. 377-84

[10] Razavi, B., "Design of Monolithic Phase-Locked Loops and Clock Recovery Circuits - A Tutorial," *Monolithic Phase-locked Loops and Clock Recovery Circuits*, IEEE Press 1996 New Jersey, pp. 1-28

[11] Dunning, J., et. al. "An All-Digital Phase-Locked Loop with 50-Cycle Lock Time Suitable for High-Performance microprocessors," *IEEE Journal of Solid-State Circuits*, vol 30, no 4, Apr. 1995, pp. 412-22

[12] Efendovich, A., et. al., "Multifrequency Zero-Jitter Delay-Locked Loop," *IEEE Journal of Solid-State Circuits*, vol 29, no 1, Jan. 1994, pp. 67-70

[13] Sidiropoulos, S., M.A. Horowitz, "A Semidigital Dual Delay-Locked Loop," *IEEE Journal of Solid-State Circuits*, vol 32, no 11, Nov. 1997, pp. 1683-92

[14] Garlepp, B., et. al., "A Portable Digital DLL for High-Speed CMOS Interface Circuits," *IEEE Journal of Solid-State Circuits*, vol 34, no 5, May 1996, pp. 632-44

[15] Larsson, P., "A 2-1600-MHz CMOS Clock Recovery PLL with Low-Vdd Capability," *IEEE Journal of*

Solid-State Circuits, vol 34, no 12, Dec. 1999, pp. 1951-60

[16] Cordell, R., "A 45-Mbit/s CMOS VLSI Digital Phase Aligner," *IEEE Journal of Solid-State Circuits*, vol 23, no 2, Apr. 1988, pp. 323-28

[17] Lee, K., et. al., "A CMOS Serial Link For Fully Duplexed Data Communication," *IEEE Journal of Solid-State Circuits*, vol 30, no 4, Apr. 1995, pp. 353-64

[18] Yang, C.K., et al., "A 0.5-μm CMOS 4.0-Gb/s Serial Link Transceiver with Data Recovery Using Oversampling," *IEEE Journal of Solid-State Circuits*, vol 33, no 5, May 1998, pp. 713-22

[19] Weinlader, D., et al., "An Eight Channel 36-GS/s CMOS Timing Analyzer," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2000, San Francisco, pp. 170-1

[20] Maneatis, J., M. Horowitz, "Precise Delay Generation Using Coupled Oscillators," *IEEE Journal of Solid-State Circuits*, vol 28, no 12, Dec. 1993, pp. 1273-82

[21] Gray, C., et. al., "A Sampling Technique and Its CMOS Implementation with 1Gb/s Bandwidth and 25ps Resolution", *IEEE Journal of Solid-State Circuits*, vol 29, no 3, Mar. 1994, pp. 340

[22] Ota, Y. et. al., "High-Speed, Burst-Mode, Packet Capable Optical Receiver and Instantaneous Clock Recovery for Optical Bus Operation," *IEEE Journal of Lightwave Technology*, vol 12, no 2, Feb. 1994, pp. 325-330

[23] Foley, D., M.P. Flynn, "CMOS DLL-Based 2-V 3.2ps Jitter 1-GHz Clock Synthesizer and Temperature-Compensated Tunable Oscillaor," *IEEE Journal of Solid-State Circuits*, vol 36, no 3, Mar. 2001, pp. 417-23

[24] Kim, C., I. Hwang, S.M. Kang, "Low-Power Small-Area +/-7.28ps Jitter 1GHz DLL-Based Clock Generator," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2002, San Francisco, Session 8.3

[25] Farjad-rad, R., et. al., "A 0.2-2GHz 12mW Multiplying DLL for Low-Jitter Clock Synthesis in Highly-Integrated Data Communication Chips," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2002, San Francisco, Session 4.5

[26] Ye, S., L. Jansson, I. Galton, "A Multiple-Crystal Interface PLL with VCO Realignment to Reduce Phase Noise," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2002, San Francisco, Session 4.6

[27] Kim, J., et. al., "A Low-Jitter Mixed-Mode DLL for High-Speed DRAM Applications," *IEEE Journal of Solid-State Circuits*, vol 35, no 10, Oct. 2000, pp. 1430-3

Delta-Sigma Fractional-*N* Phase-Locked Loops

Ian Galton

Abstract—This paper presents a tutorial on delta-sigma fractional-*N* PLLs for frequency synthesis. The presentation assumes the reader has a working knowledge of integer-*N* PLLs. It builds on this knowledge by introducing the additional concepts required to understand ΔΣ fractional-*N* PLLs. After explaining the limitations of integer-*N* PLLs with respect to tuning resolution, the paper introduces the delta-sigma fractional-*N* PLL as a means of avoiding these limitations. It then presents a self-contained explanation of the relevant aspects of delta-sigma modulation, an extension of the well known integer-*N* PLL linearized model to delta-sigma fractional-*N* PLLs, a design example, and techniques for wideband digital modulation of the VCO within a delta-sigma fractional-*N* PLL.

I. Introduction

Over the last decade, delta-sigma (ΔΣ) fractional-*N* phase locked loops (PLLs) have become widely used for frequency synthesis in consumer-oriented electronic communications products such as cellular phones and wireless LANs. Unlike an integer-*N* PLL, the output frequency of a ΔΣ fractional-*N* PLL is not limited to integer multiples of a reference frequency. The core of a ΔΣ fractional-*N* PLL is similar to an integer-*N* PLL, but it incorporates additional digital circuitry that allows it to accurately interpolate between integer multiples of the reference frequency. The tuning resolution depends only on the complexity of the digital circuitry, so considerable flexibility and programmability is achieved. A single ΔΣ fractional-*N* PLL often can be used for local oscillator generation in applications that would otherwise require a cascade of two or more integer-*N* PLLs. Moreover, the fine tuning resolution makes it possible to perform digitally-controlled frequency modulation for generation of continuous-phase (e.g., FSK and MSK) transmit signals, thereby simplifying wireless transmitters. These benefits come at the expense of increased digital complexity and somewhat increased phase noise relative to integer-*N* PLLs. However, with the relentless progress in silicon VLSI technology optimized for digital circuitry, this tradeoff is increasingly attractive, especially in consumer products which tend to favor cost reduction over performance.

This paper presents a tutorial on ΔΣ fractional-*N* PLLs. It is assumed that the reader has a working knowledge of integer-*N* PLLs. The paper builds on this knowledge by presenting the additional concepts required to understand ΔΣ fractional-*N* PLLs. The limitations of integer-*N* PLLs with respect to tuning resolution are described in Section II. The key ideas

The author is with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA, USA.

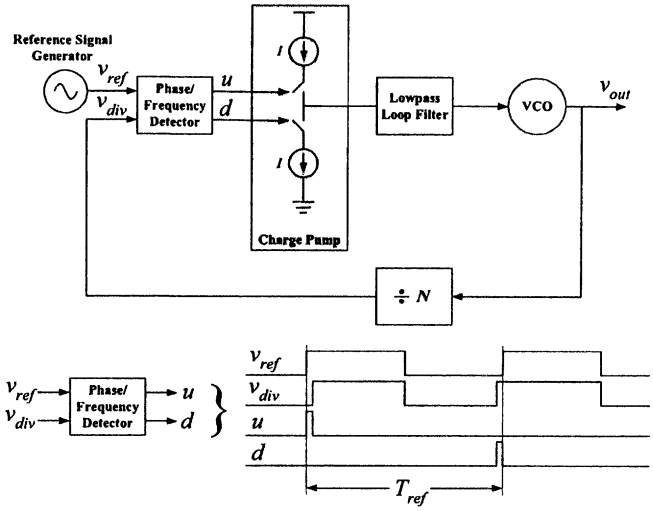


Figure 1: A typical integer-*N* PLL.

underlying fractional-*N* PLLs in general and ΔΣ fractional-*N* PLLs in particular are presented in Section III. The primary innovation in ΔΣ fractional-*N* PLLs relative to other types of fractional-*N* PLLs is the use of ΔΣ modulation. Therefore, a self-contained introduction to ΔΣ modulation as it relates to ΔΣ fractional-*N* PLLs is presented in Section IV. A ΔΣ fractional-*N* PLL linearized model is derived in Section V and compared to the corresponding model for integer-*N* PLLs. A design example is presented to demonstrate how the model is used in practice. Design issues that arise in ΔΣ fractional-*N* PLLs but not integer-*N* PLLs are presented in Section VI, and recently developed enhancements to ΔΣ fractional-*N* PLLs that allow wideband digital modulation of the VCO are presented in Section VII.

II. Integer-*N* PLL Limitations

An example of a typical integer-*N* PLL for frequency synthesis is shown in Figure 1 [1], [2]. Its purpose is to generate a spectrally pure periodic output signal with a frequency of $N f_{ref}$, where N is an integer, and f_{ref} is the frequency of the reference signal. The example PLL consists of a phase-frequency detector (PFD), a charge pump, a lowpass loop filter, a voltage controlled oscillator (VCO), and an N-fold digital divider. The PFD compares the positive-going edges of the reference signal to those from the divider and causes the charge pump to drive the loop filter with current pulses whose widths are proportional to the phase difference between the two signals. The pulses are lowpass filtered by the loop filter and the resulting waveform drives the VCO. Within the loop bandwidth phase noise from the VCO is suppressed and outside the loop bandwidth most of the other noise sources are suppressed, so the

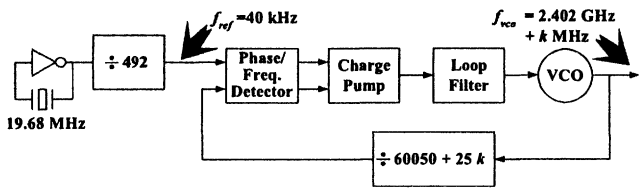


Figure 2: An example integer-N PLL for generation of the Bluetooth wireless LAN RF channel frequencies.

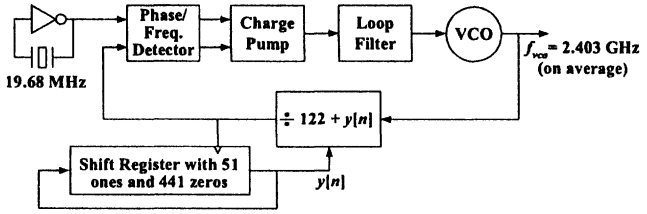


Figure 3: A fractional-N PLL that generates non-integer multiples of the reference frequency, but has phase noise consisting of large spurious tones.

PLL can be designed to generate a spectrally pure output signal at any integer multiple of the reference frequency, f_{ref}.

As indicated by the timing diagram in Figure 1, the loop filter is updated by the charge pump once every reference period. This discrete-time behavior places an upper limit on the loop bandwidth of approximately $f_{ref}/10$ above which the PLL tends to be unstable [1]. In integrated circuit PLLs, it is common to further limit the bandwidth to approximately $f_{ref}/20$ to allow for process and temperature variations.

The output frequency can be changed by changing N, but N must be an integer, so the output frequency can be changed only by integer multiples of the reference frequency. If finer tuning resolution is required the only option is to reduce the reference frequency. Unfortunately, this tends to reduce the maximum practical loop bandwidth, thereby increasing the settling time of the PLL, the noise contributed by the VCO, and the in-band portions of the noise contributed by the reference source, the PFD, the charge pump, and the divider.

This fundamental tradeoff between bandwidth and tuning resolution in integer-N PLLs creates problems in many applications. For example, a PLL that can be tuned from 2.402 GHz to 2.480 GHz in steps of 1 MHz is required to generate the local oscillator signal in a direct conversion Bluetooth transceiver [3]. An integer-N PLL capable of generating the local oscillator signal from a commonly used crystal oscillator frequency, 19.68 MHz, is shown in Figure 2. A reference frequency of f_{ref} = 40 kHz—the greatest common divisor of the crystal frequency and the set of desired output frequencies—is obtained by dividing the crystal oscillator signal by 492. The resulting PLL output frequency is 60050 + 25k times the reference frequency, where k is an integer used to select the desired frequency step.

The PLL achieves the desired output frequencies, but its bandwidth is limited to approximately 2 kHz, i.e., $f_{ref}/20$. Unfortunately, with such a low bandwidth the settling time exceeds the 200 μS limit specified in the Bluetooth standard, and the phase noise contributed by the VCO would be unacceptably high if it were implemented in present-day CMOS technology. One solution is to use a 1 MHz reference signal, but this requires the crystal frequency to be an integer multiple of 1 MHz, or another PLL to generate a 1 MHz reference frequency. Unfortunately, in low cost consumer electronics applications such as Bluetooth, it is often desirable to be compatible with all of the popular crystal frequencies, so restricting the crystal frequencies to multiples of 1 MHz is not always an option. In such cases, an additional PLL capable of generating the 1 MHz reference signal with very little phase noise from any of the crystal frequencies is required, or, as described in the next section, a single fractional-N PLL can be used.

III. The Idea Behind ΔΣ Fractional-N PLLs

In this section, the example problem of generating the second Bluetooth channel frequency, 2.403 GHz, with a reference frequency of 19.68 MHz is used as a vehicle with which to explain the idea behind ΔΣ fractional-N PLLs. First, a pair of "bad" fractional-N PLLs are presented that achieve the desired frequency but have poor phase noise performance. Then the ΔΣ fractional-N PLL technique is presented as a means of improving the phase noise performance.

The output frequency of an integer-N PLL with a reference frequency of 19.68 MHz is 2.40096 GHz when the divider modulus, N, is set to 122 and 2.42064 GHz when N is set to 123. The problem is that to achieve the desired frequency of 2.403 GHz, N would have to be set to the non-integer value of 122 + 51/492. This cannot be implemented directly because the divider modulus must be an integer value. However the divider modulus can be updated each reference period, so one option is to switch between N = 122 and N = 123 such that the average modulus over many reference periods converges to 122 + 51/492. In this case, the resulting average PLL output frequency is 2.403 GHz as desired. This is the fundamental idea behind most fractional-N PLLs [4].

While dynamically switching the divider modulus solves the problem of achieving non-integer multiples of the reference frequency, a price is paid in the form of increased phase noise. During each reference period the difference between the actual divider modulus and the average, i.e., ideal, divider modulus represents error that gets injected into the PLL and results in increased phase noise. As described below, the amount by which the phase noise is increased depends upon the characteristics of the sequence of divider moduli.

For example, in the fractional-N PLL shown in Figure 3, the divider modulus is set each reference period to 122 or 123 such that over each set of 492 consecutive reference periods it is set to 122 a total of 441 times and 123 a total of 51 times. Thus, the average modulus is 122 + 51/492 as required. The sequence of moduli is periodic with a period of 492, so it repeats at a rate of 40 kHz. Consequently, the difference between the actual divider moduli and their average is a periodic sequence with a repeat rate of 40 kHz, so the resulting phase noise is periodic and is comprised of spurious tones at integer multiples of 40 kHz. Many of the spurious tones occur at low frequencies, and they can be very large. Unfortunately, the

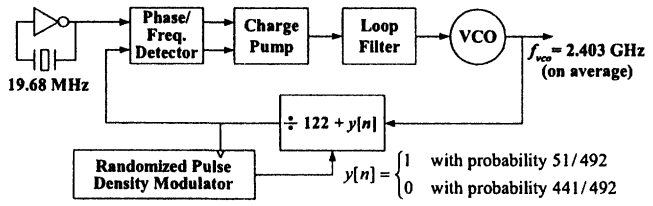


Figure 4: A fractional-N PLL that generates non-integer multiples of the reference frequency, but has a large amount of in-band phase noise.

only way to suppress the tones is have a very small PLL bandwidth, which negates the potential benefit of the fractional-N technique.

One way to eliminate spurious tones is to introduce randomness to break up the periodicity in the sequence of moduli while still achieving the desired average modulus. For example, as shown in Figure 4, a digital block can be used to generate a sequence, $y[n]$, that approximates a sampled sequence of independent random variables that take on values of 0 and 1 with probabilities 441/492 and 51/492, respectively. During the n^{th} reference period the divider modulus is set to 122 + $y[n]$, so the sequence of moduli has the desired average yet its power spectral density (PSD) is that of white noise. Thus, instead of contributing spurious tones, the modified technique introduces white noise. Unfortunately, the portion of the white noise within the PLL's bandwidth is integrated by the PLL transfer function, so the overall phase noise contribution again can be significant unless the PLL bandwidth is small.

In each fractional-N PLL example presented above, the sequence, $y[n]$, can be written as $y[n] = x + e_m[n]$, where x is the desired fractional part of the modulus, i.e., x = 51/492, and $e_m[n]$ is undesired zero-mean *quantization noise* caused by using integer moduli in place of the ideal fractional value. In the first example, $e_m[n]$ is periodic and therefore consists of spurious tones at multiples of 40 kHz. In the second example, $e_m[n]$ is white noise. Each PLL attenuates the portion of $e_m[n]$ outside its bandwidth, but the portion within its bandwidth is not significantly attenuated. Unfortunately, in each example $e_m[n]$ contains significant power at low frequencies, so it contributes substantial phase noise unless the PLL bandwidth is very low.

A ΔΣ fractional-N PLL avoids this problem by generating the sequence of moduli such that the quantization noise has most of its power in a frequency band well above the desired bandwidth of the PLL [5], [6], [7]. An example ΔΣ fractional-N PLL is shown in Figure 5. The PLL core is similar to those of the previous fractional-N PLL examples, but in this case $y[n]$ is generated by a digital ΔΣ modulator. The details of how the ΔΣ modulator works are presented in the next section, but its purpose is to coarsely quantize its input sequence, $x[n]$, such that $y[n]$ is integer-valued and has the form: $y[n] = x[n-2] + e_m[n]$, where $e_m[n]$ is dc-free quantization noise with most of its power outside the PLL bandwidth. In this example, $x[n]$ consists of the desired fractional modulus value, 51/492, plus a small, pseudo-random, 1-bit sequence. As described in the next section, the pseudo-random sequence is necessary to avoid spurious tones in the ΔΣ modulator's quantization noise, but its amplitude is very small so it does not appreciably increase the phase noise of the PLL.

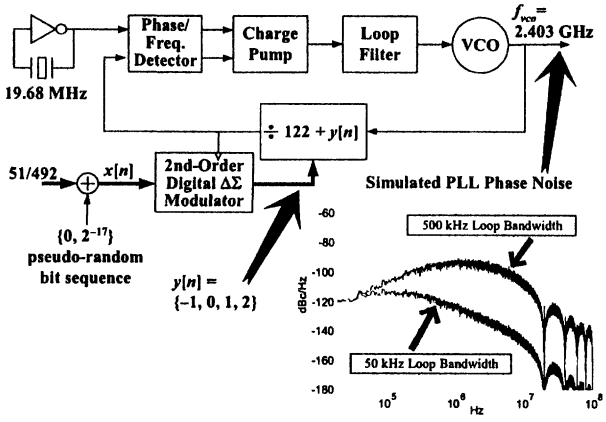


Figure 5: A ΔΣ fractional-N PLL example.

Also shown in Figure 5 are PSD plots of the output phase noise arising from ΔΣ modulator quantization noise, $e_m[n]$, in two computer simulated versions of the example ΔΣ fractional-N PLL, one with a 50 kHz loop bandwidth and the other with a 500 kHz loop bandwidth. As shown in the next section, the PSD of $e_m[n]$ increases with frequency, so the phase noise PSD corresponding to the 50 kHz bandwidth PLL is significantly smaller than that corresponding to the 500 kHz bandwidth PLL. For example, the former easily meets the requirements for a local oscillator in a direct conversion Bluetooth transceiver, but the latter falls short of the requirements by at least 23 dB.

IV. Delta-Sigma Modulation Overview

As mentioned above, a digital ΔΣ modulator performs coarse quantization in such a way that the inevitable error introduced by the quantization process, i.e., the quantization noise, is attenuated in a specific frequency band of interest. There are many different ΔΣ modulator architectures. Most use coarse uniform quantizers to perform the quantization with feedback around the quantizers to suppress the quantization noise in particular frequency bands. Therefore, to illustrate the ΔΣ modulator concept, first a specific uniform quantizer example is considered in isolation, and then a specific ΔΣ modulator architecture that incorporates the uniform quantizer is presented.

A. An Example Uniform Quantizer

The input-output characteristic of the example uniform quantizer is shown in Figure 6. It is a 9-level quantizer with integer valued output levels. For each input value with a magnitude less than 4.5, the quantizer generates the corresponding output sample by rounding the input value to the nearest integer. For each input value greater than 4.5 or less than –4.5, the quantizer sets its output to 4 or –4, respectively; such values are said to *overload* the quantizer. By defining the quantization noise as $e_q[n] = y[n] - r[n]$, the quantizer can be viewed without approximation as an additive noise source as illustrated in the figure.

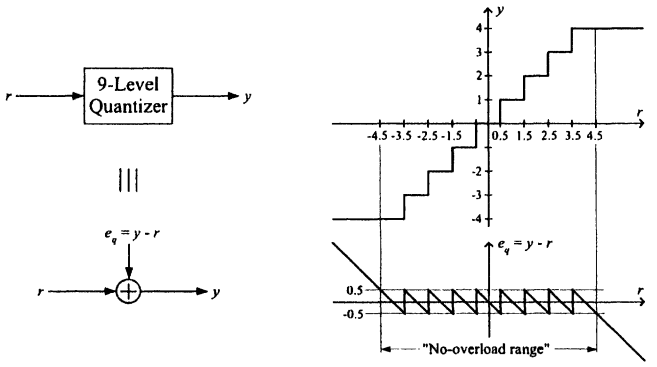


Figure 6: A 9-level quantizer example.

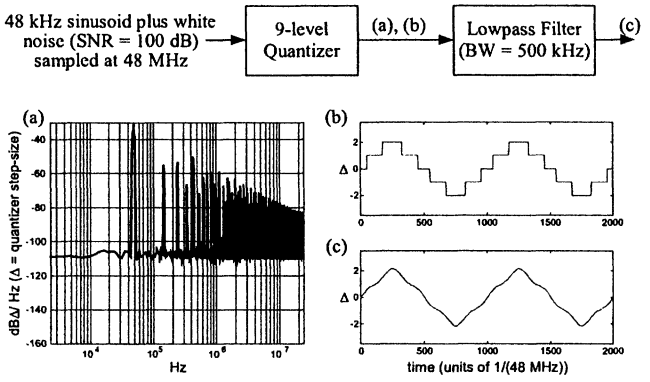


Figure 7: (a) A power spectral density plot of the quantizer output in dB, relative to the quantization step-size of $\Delta = 1$, per Hz, (b) a time domain plot of the quantizer output, and (c) a time domain plot of the quantizer output filtered by a sharp lowpass filter with a cutoff frequency of 500 kHz.

To illustrate some properties of the example quantizer, consider a 48 Msample/s input sequence, $x[n]$, consisting of a 48 kHz sinusoid with an amplitude of 1.7 plus a small amount of white noise such that the input signal-to-noise ratio (SNR) is 100 dB. Figure 7(a) shows the PSD plot of the resulting quantizer output sequence, and Figure 7(b) shows a time domain plot of the quantizer output sequence over two periods of the sinusoid. Given the coarseness of the quantization, it is not surprising that the quantizer output sequence is not a precise representation of the quantizer input sequence. As evident in Figure 7(a), the quantization noise for this input sequence consists primarily of harmonic distortion as represented by the numerous spurious tones distributed over the entire discrete-time frequency band. Even in the relatively narrow frequency band below 500 kHz, significant harmonic distortion corrupts the desired signal. To illustrate this in the time domain, Figure 7(c) shows the sequence obtained by passing the quantizer output sequence through a sharp lowpass discrete-time filter with a cutoff frequency of 500 kHz. The significant quantization noise power in the zero to 500 kHz frequency band causes the sequence shown in Figure 7(c) to deviate significantly from the sinusoidal quantizer input sequence.

B. An Example ΔΣ Modulator

The example ΔΣ modulator architecture shown in Figure 8

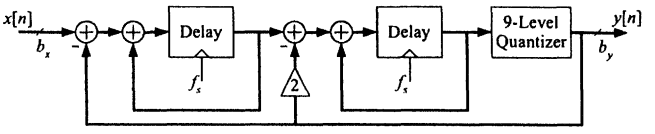


Figure 8: A ΔΣ modulator example.

can be used to circumvent this problem. The structure incorporates the same 9-level quantizer presented above, but in this case the quantizer is preceded by two delaying discrete-time integrators (i.e., accumulators), and surrounded by two feedback loops [8], [9]. Each discrete-time integrator has a transfer function of $z^{-1}/(1-z^{-1})$ which implies that its n^{th} output sample is the sum of all its input samples for times $k < n$. With the quantizer represented as an additive noise source as depicted in Figure 6, the ΔΣ modulator can be viewed as a two-input, single-output, linear time-invariant, discrete-time system. It is straightforward to verify that

$$y[n] = x[n-2] + e_m[n], \tag{1}$$

where $e_m[n]$ is the overall quantization noise of the ΔΣ modulator and is given by

$$e_m[n] = e_q[n] - 2e_q[n-1] + e_q[n-2]. \tag{2}$$

To illustrate the behavior of the ΔΣ modulator, suppose that the same 48 Msample/s input sequence considered above is applied to the input of the ΔΣ modulator, and that the discrete-time integrators in the ΔΣ modulator are clocked at 48 MHz. Figure 9(a) shows the PSD plot of the resulting ΔΣ modulator output sequence, $y[n]$, and Figure 9(b) shows a time domain plot of $y[n]$ over two periods of the sinusoid. Two important differences with respect to the uniform quantization example shown in Figure 7 are apparent: the quantization noise PSD is significantly attenuated at low frequencies, and no spurious tones are visible anywhere in the discrete-time spectrum. For instance, the SNR in the zero to 500 kHz frequency band is approximately 84 dB for this example as opposed to 14 dB for the uniform quantization example of Figure 7. Consequently, subjecting the ΔΣ modulator output sequence to a lowpass filter with a cutoff frequency of 500 kHz results in a sequence that is very nearly equal to the ΔΣ modulator input sequence as demonstrated in Figure 9(c).

Below about 120 kHz, the PSD shown in Figure 9(a) is dominated by the two components of the ΔΣ modulator input sequence: the 48 kHz sinusoid component, and the input noise component. Above 120 kHz, the PSD is dominated by the ΔΣ modulator quantization noise, $e_m[n]$, and rises with a slope of 40 dB per decade. It follows from (2) that $e_m[n]$ can be viewed as the result of passing the additive noise from the quantizer, $e_q[n]$, through a discrete-time filter with transfer function $(1-z^{-1})^2$. Since this filter has two zeros at dc, the smooth 40 dB per decade increase of the PSD of $e_m[n]$ indicates that $e_q[n]$ is very nearly white noise, at least for the example shown in Figure 9.

It can be proven that $e_q[n]$ is indeed white noise; it has a variance of 1/12 and is uncorrelated with the ΔΣ modulator input sequence [10]. Moreover, this situation holds in general for the example ΔΣ modulator architecture provided that the

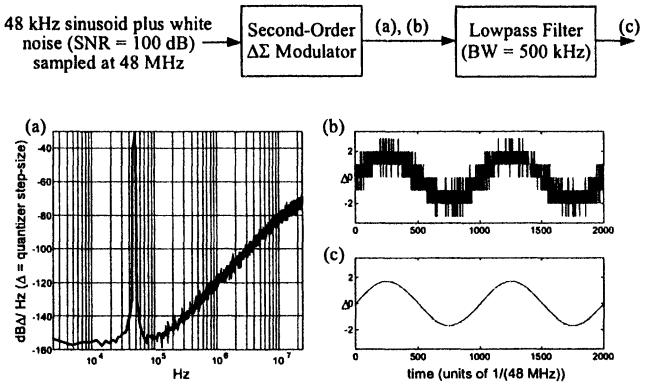


Figure 9: (a) A power spectral density plot of the ΔΣ modulator output in dB, relative to the quantization step-size of Δ = 1, per Hz, (b) a time domain plot of the ΔΣ modulator output, and (c) a time domain plot of the ΔΣ modulator output filtered by a sharp lowpass filter with a cutoff frequency of 500 kHz.

input sequence satisfies two conditions: 1) its magnitude is sufficiently small that the quantizer within the ΔΣ modulator never overloads, and 2) it consists of a signal component plus a small amount of independent white noise. It can be shown that the first condition is satisfied if the input signal is bounded in magnitude by 3Δ where Δ is the step-size of the quantizer (for this example, Δ = 1) [11]. Input sequences with values even slightly exceeding 3Δ in magnitude generally cause the quantizer to overload with the result that $e_q[n]$ contains spurious tones and the SNR in the frequency band of interest is degraded. For this reason, the range between –3Δ and 3Δ is said to be the *input no-overload range* of the ΔΣ modulator. For the second condition to be satisfied, the power of the ΔΣ modulator input sequence's white noise component may be arbitrarily small, but if it is absent altogether, $e_q[n]$, is not guaranteed to be white. For instance, in the example shown in Figure 9 the input sequence contains a white noise component with 100dB less power than the signal component. If this tiny noise component were not present, the resulting ΔΣ modulator output PSD would contain numerous spurious tones. Since the ΔΣ modulators used in ΔΣ fractional-*N* PLLs are all-digital devices, the noise must be added digitally. As shown in [12], it is sufficient to add a 1-bit, sub-LSB, independent, white noise *dither sequence* with zero mean at the input node. In practice, a 1-bit pseudo-random dither sequence is typically used in place of a truly random dither sequence. Such a sequence can be generated easily using a linear feedback shift register, and has the desired result with respect to the quantization noise despite not being truly random [13], [14].

C. Other ΔΣ Modulator Options

To this point, the ΔΣ modulation concept has been illustrated via the particular example ΔΣ modulator architecture shown in Figure 8, namely a second-order multi-bit ΔΣ modulator. While this type of ΔΣ modulator is widely used in ΔΣ fractional-N PLLs, there exist other types of ΔΣ modulators that can be applied to ΔΣ fractional-N PLLs. Most of the other architectures are higher-order ΔΣ modulators that perform higher than second-order quantization noise shaping,

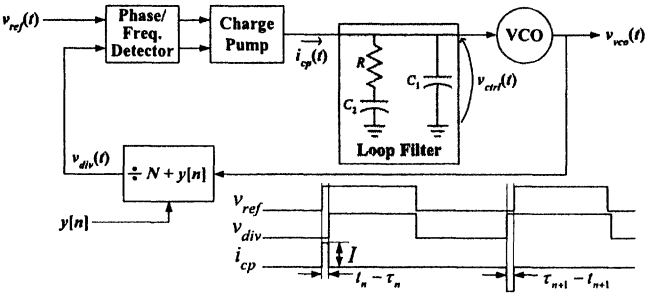


Figure 10: The ΔΣ fractional-*N* PLL with the details of a commonly used loop filter and a timing diagram relating to the charge pump output.

thereby more aggressively suppressing quantization noise in particular frequency bands relative to the example second-order ΔΣ modulator. Some of these higher-order ΔΣ modulators incorporate a higher than second-order loop filter (e.g., more than two discrete-time integrators) and a single quantizer surrounded by one or more feedback loops [15], [16]. In many cases, these ΔΣ modulators are designed specifically to allow one-bit quantization [7], [17], [18]. This simplifies the design of the divider in that only two moduli are required, but such ΔΣ modulators tend to have spurious tones in their quantization noise that cannot be completely suppressed even with elaborate dithering techniques. Others of these higher-order ΔΣ modulators, often referred to as MASH, cascaded, or multistage ΔΣ modulators, are comprised of multiple lower-order ΔΣ modulators, such as the second-order ΔΣ modulator presented above, cascaded to obtain the equivalent of a single higher-order ΔΣ modulator [5], [19], [20].

V. ΔΣ Fractional-*N* PLL Dynamics

A ΔΣ fractional-*N* PLL linearized model is derived in this section in the form of a block diagram that describes the output phase noise in terms of the component parameters and noise sources in the PLL. As in the case of an integer-*N* PLL the model provides an accurate tool with which to predict the total phase noise, bandwidth, and stability of the PLL.

A. Derivation of a ΔΣ fractional-N PLL Linearized Model

In PLL analyses it is common to assume that each periodic signal within the PLL has the form $v(t) = A(t)\sin(\omega t + \theta(t))$, where $A(t)$ is a positive *amplitude* function, ω is a constant *center frequency* in radians/sec, and $\theta(t)$ is zero-mean *phase noise* in radians. In most cases of interest for PLL analysis, the amplitude is well modeled as a constant value, and the phase noise is very small relative to π with a bandwidth that is much lower than the center frequency. Solving for the time of the n^{th} positive-going zero crossing, γ_n, of $v(t)$ gives $\gamma_n = [n - \theta(\gamma_n)/(2\pi)]\cdot T$, where $T = 2\pi/\omega$ is the period of the signal. Therefore, the sequence, γ_n, is a sampled version of the phase noise with very little aliasing, so knowing the sequence and T is approximately equivalent to knowing the phase noise. This approximation is made throughout the following analysis.

The relationship between the charge pump output current and the PFD input signals is shown in Figure 10. Ideally, dur-

ing the n^{th} reference period the charge pump output is a current pulse of amplitude I or $-I$ and duration $|t_n - \tau_n|$, where t_n and τ_n are the times of the charge pump output transitions triggered by the positive-going edges of the divider output and reference signal, respectively. Therefore, the average current sourced or sunk by the charge pump during the n^{th} reference period is $I\cdot(t_n - \tau_n)/T_{ref}$. In practice, the PFD is usually designed such that, except for a possible constant offset, this result holds even though the current sources have finite rise and fall times [2].

The first step in deriving the model is to develop an expression for $t_n - \tau_n$. Ideally, $\tau_n = nT_{ref}$, but phase noise introduced by the reference source and PFD cause it to have the form

$$\tau_n = nT_{ref} - \frac{T_{ref}}{2\pi}\left[\theta_{ref}(\tau_n) + \theta_{PFD}(\tau_n)\right], \tag{3}$$

where $\theta_{ref}(t)$ and $\theta_{PFD}(t)$ are the reference source and PFD phase noise functions, respectively. If the VCO output were ideal its positive-going edges would be spaced at uniform intervals of $T_{ref} / (N + \alpha)$, where α is the fractional part of the modulus (e.g., $\alpha = 51/492$ in Figure 5). Therefore, ideally,

$$t_n = \frac{T_{ref}}{N+\alpha}\sum_{k=0}^{n-1}\left(N + y[k]\right),$$

but in practice it deviates because of VCO phase noise, $\theta_{VCO}(t)$, divider phase noise, $\theta_{div}(t)$, and instantaneous deviations of the VCO control voltage from its ideal average value of $\overline{v}_{ctrl} = (N+\alpha)/(T_{ref}K_{VCO})$, where K_{VCO} is the VCO gain in units of Hz/Volt. As a result,

$$t_n = \frac{T_{ref}}{N+\alpha}\left[\sum_{k=0}^{n-1}\left(N + y[k]\right) - k_{VCO}\int_0^{t_n}\left(v_{ctrl}(t) - \overline{v}_{ctrl}\right)dt - \frac{\theta_{VCO}(t_n)}{2\pi}\right]$$

$$-\frac{T_{ref}}{2\pi}\theta_{div}(t_n),$$

which reduces to

$$t_n = nT_{ref} + \frac{T_{ref}}{N+\alpha}\left[\sum_{k=0}^{n-1}\left(y[k] - \alpha\right)\right.$$

$$\left. -k_{VCO}\int_0^{t_n}\left(v_{ctrl}(t) - \overline{v}_{ctrl}\right)dt - \frac{\theta_{VCO}(t_n)}{2\pi}\right] \tag{4}$$

$$-\frac{T_{ref}}{2\pi}\theta_{div}(t_n).$$

Subtracting (3) from (4) yields an expression for the average current sourced or sunk by the charge pump during the n^{th} reference period:

$$I(t_n - \tau_n)/T_{ref} =$$

$$I\left[\frac{\sum_{k=0}^{n}\left(y[k] - \alpha\right) - k_{VCO}\int_0^{t_n}\left(v_{ctrl}(t) - \overline{v}_{ctrl}\right)dt - \frac{\theta_{VCO}(t_n)}{2\pi}}{N+\alpha}\right. \tag{5}$$

$$\left. -\frac{\theta_{div}(t_n)}{2\pi} + \frac{\theta_{ref}(\tau_n)}{2\pi} + \frac{\theta_{PFD}(\tau_n)}{2\pi}\right].$$

As mentioned above, the phase noise terms are assumed to have bandwidths that are much smaller than the reference frequency. Consequently, the sampling of the phase noise functions in (5) can be neglected, and the charge pump output can be modeled as a smoothly varying function of time with an average value over each reference period equal to that of (5). With these approximations, (5) implies that

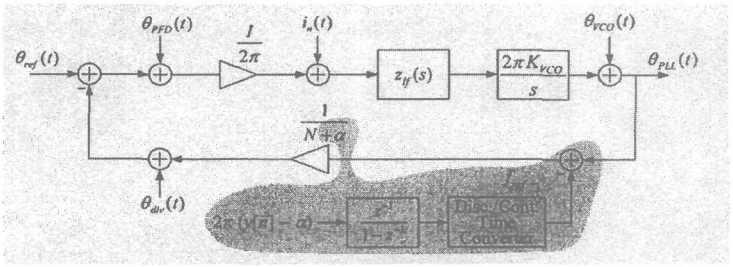


Figure 11: The ΔΣ fractional-N PLL linearized model. Except for the shaded region the model is identical to the corresponding integer-N PLL model.

$$i_{cp}(t) = I\left[\frac{u_m(t) - k_{VCO}\int_0^t\left(v_{ctrl}(t) - \overline{v}_{ctrl}\right)dt - \frac{\theta_{VCO}(t)}{2\pi}}{N+\alpha}\right. \tag{6}$$

$$\left. -\frac{\theta_{div}(t)}{2\pi} + \frac{\theta_{ref}(t)}{2\pi} + \frac{\theta_{PFD}(t)}{2\pi}\right],$$

where $u_m(t)$ is the result of discrete-time integrating and converting to continuous-time the quantity, $y[n] - \alpha$.

The ΔΣ fractional-N PLL linearized model follows directly from (6) and Figure 10. It is shown in Figure 11, where $i_n(t)$ represents the noise contributed by the charge pump current sources and the loop filter, and $z_{lf}(s)$ is the transfer function of the loop filter. The model specifies the phase noise transfer functions and loop dynamics of the PLL. For example, the model implies that

$$\frac{\theta_{PLL}(s)}{\theta_{ref}(s)} = (N+\alpha)\frac{T(s)}{1+T(s)}, \quad \text{and} \quad \frac{\theta_{PLL}(s)}{\theta_{VCO}(s)} = \frac{1}{1+T(s)} \tag{7}$$

where

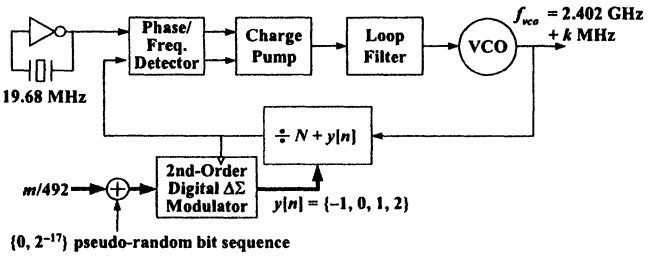
$$T(s) = \frac{IK_{VCO}z_{lf}(s)}{s(N+\alpha)} \tag{8}$$

is the *loop gain* of the PLL. For the loop filter shown in Figure 10, the transfer function is

$$z_{lf}(s) = \frac{1}{C_1 + C_2}\frac{1 + sRC_2}{s\left[1 + sRC_1C_2/(C_1 + C_2)\right]}. \tag{9}$$

B. Differences Between the ΔΣ Fractional-N and Integer-N PLL Models

The shaded region in Figure 11 indicates the part of the model that is specific to ΔΣ fractional-N PLLs; except for the shaded region the model is identical to the corresponding model for integer-N PLLs. Therefore, each phase noise transfer function in an integer-N PLL is identical to the corresponding phase noise transfer function in a ΔΣ fractional-N PLL, except every occurrence of N in the former is replaced by $N+\alpha$ in the latter. In most cases, $N >> 1$ and $\alpha < 1$, so $N + \alpha \approx N$ and the corresponding transfer functions in integer-N and ΔΣ fractional-N PLLs are nearly identical in practice. Similarly, the loop dynamics and stability issues are nearly the same in ΔΣ fractional-N PLLs and integer-N PLLs.



Frequency Plan:

- To get k = 0, 1, ..., or 18: set N = 122, $m = k \cdot 25 + 26$
- To get k = 19, 21, ..., or 38: set N = 123, $m = (k - 19) \cdot 25 + 9$
- To get k = 39, 41, ..., or 57: set N = 124, $m = (k - 39) \cdot 25 + 17$
- To get k = 58, 60, ..., or 79: set N = 125, $m = (k - 58) \cdot 25$

Figure12: The example ΔΣ fractional-N PLL and frequency plan for generation of the Bluetooth wireless LAN RF channel frequencies.

The primary difference between the ΔΣ fractional-N and integer-N PLL models is the signal path corresponding to the ΔΣ modulator shown in the shaded region of Figure 11. The sequence, $y[n] - \alpha$, consists of ΔΣ modulator quantization noise, $e_m[n]$, which, as described previously, gives rise to phase error in the PLL output. For the example second-order ΔΣ modulator it follows from the results presented in Section IV and the ΔΣ fractional-N PLL model equations presented above that the PLL phase noise component resulting from $e_m[n]$ has a PSD given by

$$S_{\theta_{PLL}}(f)\Big|_{\Delta\Sigma\ \text{only}} = 10 \cdot \log\left[\frac{\pi}{12 f_{ref}}\left[2 \cdot \sin\left(\frac{\pi f}{f_{ref}}\right)\right]^2 \left|\frac{1}{N+\alpha} \cdot \frac{\theta_{PLL}(j2\pi f)}{\theta_{ref}(j2\pi f)}\right|^2\right] \text{ dBc/Hz.} \quad (10)$$

The argument of the log function has the form of a highpass function times a lowpass function, which is consistent with the claim in Section III that the PLL lowpass filters the primarily high frequency quantization noise from the ΔΣ modulator. It follows from (10) that the phase noise resulting from $e_m[n]$ can be decreased by reducing the PLL bandwidth or increasing the reference frequency. If a higher-order ΔΣ modulator is used, an equation similar to (10) results except that the exponent of the sinusoid is greater than two. This reduces the in-band portion of the quantization noise, but increases the out-of-band portion, which, depending upon the loop parameters of the PLL, can result in a somewhat lower overall phase noise. However, the PLL loop filter is highly constrained to maintain PLL stability, so the phase noise reduction that can be achieved by increasing the order of the ΔΣ modulator is limited in most applications [16].

C. A System Design Example

The PLL bandwidth and the phase margin both depend upon the loop gain, $T(s)$, which, for the loop filter shown in Figure 10, depends upon the parameters f_{ref}, N, I, K_{VCO}, R, C_1, and C_2. Usually, f_{ref} and N are dictated by the application, and I and K_{VCO} are, at least partially, dictated by circuit design choices. This leaves the loop filter components as the main variables with which to set the desired PLL bandwidth, phase margin, and ΔΣ modulator quantization noise suppression.

The process is demonstrated below for the ΔΣ fractional-N PLL presented in Section III to generate the local oscillator frequencies in a direct conversion Bluetooth wireless LAN transceiver. The PLL is shown in Figure 12 with additional detail regarding the frequency plan. As described previously, the desired output frequencies are f_{VCO} = 2.402 GHz + k MHz for k = 0, ..., 78, and the crystal reference frequency is 19.68 MHz. Each of the 79 possible output frequencies is chosen by selecting m and N as indicated in the figure. In each case, the divider modulus is restricted to the set of four integers $\{N-1, N, N+1, N+2\}$. The combinations of m and N were chosen to achieve the desired output frequencies yet keep the signals at the input of the ΔΣ modulator sufficiently small so as not to overload the ΔΣ modulator [11].

Typical requirements for such a PLL are that the loop bandwidth must be greater than 40 kHz, the phase margin must be greater than 60°, and the PLL phase noise be less than −120 dBc/Hz at offsets from the carrier of 3 MHz and above. Assume that the VCO, divider, PFD, and charge pump circuits have been designed such that the overall PLL phase noise specification can be met provided the phase noise contributed by the ΔΣ modulator and loop filter are each less than −130 dBc/Hz at offsets from the carrier of 3 MHz and above. Furthermore, assume that the VCO and charge pump circuits are such that K_{VCO} and I are 200 MHz/V and 200 μA, respectively, and that the loop filter has the form shown in Figure 10. Thus, the remaining design task is to choose the loop filter components such that the bandwidth, phase margin, and phase noise specifications are met.

The PLL phase margin, bandwidth, and phase noise arising from ΔΣ modulator quantization noise can be derived from the linearized model equations, (7) through (10). While this can be done directly, it involves the solution of third order equations which can be messy. Alternatively, approximate solutions of the equations can be derived that provide better intuition [21]. A particularly convenient set of approximate solutions are

$$PM = \tan^{-1}\left(\frac{b-1}{2\sqrt{b}}\right), \quad (11)$$

$$f_{BW} = \frac{IK_{VCO}R}{2\pi N} \cdot \frac{b-1}{b}, \quad (12)$$

$$RC_2 = \frac{\sqrt{b}}{2\pi f_{BW}}, \quad (13)$$

and

$$S_{\theta_{PLL}}(f)\Big|_{\Delta\Sigma\ \text{only}} \approx 10 \cdot \log\left[\frac{2\pi b}{3 f_{ref}} \sin^2\left(\frac{\pi f}{f_{ref}}\right)\left(\frac{f_{BW}}{f}\right)^4\right] \text{ dBc/Hz,} \quad (14)$$

where PM is the phase margin of the PLL, f_{BW} is the 3 dB bandwidth of the PLL, and $b = 1 + C_2/C_1$ is a measure of the separation between the two loop filter capacitors [22]. The derivations assume that b is greater than about 10, and (14) is valid for frequencies greater than $(C_2+C_1)/(2\pi RC_2C_1)$.

These equations are sufficient to determine appropriate loop filter component values. For example, suppose b is set to

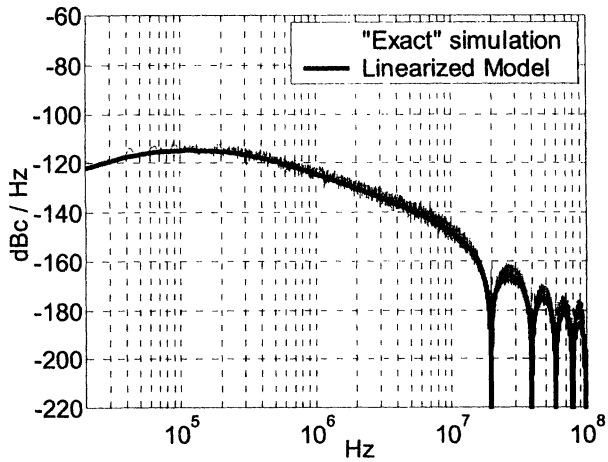


Figure13: Simulated and calculated PSD plots of the phase noise arising from ΔΣ modulator quantization noise for the example ΔΣ fractional-*N* PLL.

49, so, as indicated by (11), the phase margin is approximately 70°. Solving (14) with the phase noise set to −130 dBc/Hz at f = 3 MHz indicates that $f_{BW} \approx 50$ kHz. Therefore, the phase noise resulting from ΔΣ modulator quantization noise is sufficiently suppressed with a 50 kHz bandwidth and a phase margin of 70°. With this information (12) can be solved to find R = 960 Ω with which (13) and the definition of b can be used to calculate C_2 = 23 nF and C_1 = 480 pF. It is straightforward to verify that the phase noise introduced by the loop filter resistor (the only noise source in the loop filter) is well below −130 dBc/Hz at offsets from the carrier of 3 MHz and above as required.

Figure 13 shows PSD plots of the phase noise arising from ΔΣ modulator quantization noise for the example PLL with the loop filter component values derived above. The heavy curve was calculated directly from the linearized model equations (7) through (10). The light curve was obtained through a behavioral computer simulation of the PLL. As is evident from the figure, the two curves agree very well which suggests that the approximations made in obtaining the linearized model are reasonable.

An effect that does not have a counterpart in integer-*N* PLLs is the presence of zeros in the PSD of the phase noise arising from ΔΣ modulator quantization noise at multiples of the reference frequency. These zeros are a result of the discrete-to-continuous-time conversion of the ΔΣ modulator quantization noise; each zero is a sampling image of the dc zero imposed on the quantization noise by the ΔΣ modulator.

VI. ΔΣ Fractional-*N* PLL Specific Problems

One of the most significant problems specific to ΔΣ fractional-*N* PLLs is that they can be sensitive to modulus-dependent divider delays. In practice, each positive-going divider edge is separated from the VCO edge that triggered it by a propagation delay. Ideally, this propagation delay is independent of the corresponding divider modulus, in which case it introduces a constant phase offset but does not otherwise contribute to the phase noise. However, if the propagation delay depends upon the divider modulus and the number of ΔΣ modulator output levels is greater than two, the effect is that of a hard non-linearity applied to the ΔΣ modulator quantization noise. This tends to fold out-of-band ΔΣ modulator quantization noise to low frequencies and introduce spurious tones, which can significantly increase the PLL phase noise. The problem is analogous to that of multi-bit digital-to-analog converter step-size mismatches in analog ΔΣ data converters [23]. Unfortunately, circuit simulations are required to evaluate the severity of the problem on a case by case basis as both the extent of any modulus-dependent delays and their affect on the PLL phase noise are difficult to predict using hand analysis.

There are two well-known solutions to this problem. One solution is to resynchronize the divider output to the nearest VCO edge or at least a higher-frequency edge obtained from within the divider circuitry [22], [24]. The resynchronization erases memory of modulus-dependent delays and noise introduced within the divider circuitry, but care must be taken to ensure that the signal used for resynchronization is itself free of modulus dependent delays. The primary drawback of the approach is that it increases power consumption.

The other solution is to use a ΔΣ modulator with single-bit (i.e., two level) quantization. In this case, modulus-dependent delays give rise to phase error at the output of the divider that consists of a constant offset plus a scaled version of the ΔΣ modulator quantization noise. Since, by design, the ΔΣ modulator quantization noise has most of its power outside the PLL bandwidth, the modulus-dependent delays increase the phase noise only slightly. Unfortunately, ΔΣ modulators with single-bit quantization tend not to perform as well as ΔΣ modulators with multi-bit (i.e., more than two-level) quantization. For example, if the 9-level quantizer in the 48 Msample/s ΔΣ modulator example presented in Section IV were replaced by a one-bit quantizer, the dynamic range of the ΔΣ modulator in the zero to 500 kHz band would be reduced from 88.5 dB to approximately 65 dB. Moreover, unlike the 9-level quantizer case, the additive noise from the single-bit quantizer would not be white and would be correlated with the input sequence. Its variance would be input dependent and it would contain spurious tones.

These problems can be mitigated by using a higher-order ΔΣ modulator architecture to more aggressively suppress the in-band portion of the additive noise from the two-level quantizer. However, to maintain stability in a higher-order ΔΣ modulator with single-bit quantization, the useful input range of the ΔΣ modulator input signal must be reduced and more poles and zeros must be introduced within the feedback loop as compared to a multi-bit design with a comparable dynamic range. Even then, the problem of spurious tones persists, and it is difficult to predict where they will appear except through extensive simulation. Furthermore, to compensate for the restricted input range of the ΔΣ modulator the reference frequency must be large enough that all of the desired PLL output frequencies can be achieved. This can severely limit design flexibility. For example, if the magnitude of the ΔΣ modulator input signal were limited to less than 0.5 in the case

of the Bluetooth local oscillator application considered above, the reference frequency would have to be greater than 79 MHz. Otherwise, it would not be possible to generate all the Bluetooth channel frequencies.

Another issue specific to ΔΣ fractional-N PLLs is that modulus switching increases the average duration over which the charge pump current sources are turned on each period relative to integer-N PLLs. For comparison, consider a ΔΣ fractional-N PLL and an integer-N PLL with the same N (where $N >> \alpha$), the same f_{ref}, and identical loop components. It follows from (5) that

$$(t_n - \tau_n)\big|_{Fractional-N} = (t_n - \tau_n)\big|_{Integer-N} + \frac{T_{ref}}{N+\alpha}\sum_{k=0}^{n-1}(y[k]-\alpha). \quad (15)$$

The last term in (15), which is caused by having the ΔΣ modulator switch the divider modulus, represents a significant increase in the time during which the charge pump current sources are turned on each reference period. Consequently, the phase noise arising just from charge pump current source noise is larger in the ΔΣ fractional-N PLL by

$$A \cdot \log\left[\frac{\text{Average fractional-}N\text{ PLL charge pump "on time"}}{\text{Average integer-}N\text{ PLL charge pump "on time"}}\right]$$

where A is a constant between 10 and 20. The value of A depends upon the autocorrelation of the charge pump current source noise. For example, if the current source noise in successive charge pump pulses is completely uncorrelated, then A is 10. Near the other extreme, A is close to 20.

VII. Techniques to Widen ΔΣ Fractional-N PLL Loop Bandwidths

A transmitter with virtually any modulation format can be implemented using D/A conversion to generate analog baseband or IF signals and upconversion to generate the final RF signal. However, many of the commonly used modulation formats in wireless communication systems such as MSK and FSK involve only frequency or phase modulation of a single carrier [25]. In such cases, the transmitted signal can be generated by modulating a radio frequency (RF) VCO, thereby eliminating the need for conventional upconversion stages and much of the attendant analog filtering. At least two approaches have been successfully implemented in commercial wireless transmitters to date. One is based on open-loop VCO modulation, and the other is based on ΔΣ fractional-N synthesis.

An example of a commercial transmitter that uses the open-loop VCO modulation technique is presented in [26] and [27], in this case for a DECT cordless telephone. Between transmit bursts, the desired center frequency is set relative to a reference frequency by enclosing the VCO within a conventional PLL. During each transmit burst the VCO is switched out of the PLL and the desired frequency modulation is applied directly to its input. The primary limitation of the approach is that it tends to be highly sensitive to noise and interference from other circuits. For example, in [27], the required level of isolation precluded the implementation of a single-chip transmitter. Furthermore, the modulation index of the transmitted signal depends upon the absolute tolerances of the VCO components which are often difficult to control in low-cost VLSI technologies and can also drift rapidly over time.

In principle, ΔΣ fractional-N PLLs can avoid these problems by modulating the VCO within the PLL. This can be done by driving the input of the digital ΔΣ modulator with the desired frequency modulation of the transmitted signal. The primary limitation is that bandwidth of the PLL must be narrow enough that the quantization noise from the ΔΣ modulator is sufficiently attenuated, but sufficiently high to allow for the modulation. For instance, the phase noise PSD of the example ΔΣ fractional-N PLL shown in Figure 5 with a 50 kHz loop bandwidth meets the necessary phase noise specifications when used as a local oscillator in a conventional upconversion stage within a Bluetooth wireless LAN transmitter. However, if the Bluetooth transmitter is to be implemented by modulating the VCO through the digital ΔΣ modulator, then the loop bandwidth of the PLL must be approximately 500 kHz. Unfortunately, when the loop bandwidth of the fractional-N PLL shown in Figure 5 is widened to 500 kHz, the resulting phase noise becomes too large to meet the Bluetooth transmit requirements.

Nevertheless, commercial transmitters with VCO modulation through ΔΣ fractional-N synthesizers are beginning to be deployed, especially in low-performance, low-cost wireless systems such as Bluetooth wireless LANs [28]. Facilitating this trend are various solutions that have been devised in recent years to allow for wideband VCO modulation in ΔΣ fractional-N PLLs without incurring the phase noise penalty mentioned above. One of the solutions is to keep the loop bandwidth relatively low, but pre-emphasize (i.e., highpass filter) the digital phase modulation signal prior to the digital ΔΣ modulator [29]. Unfortunately, this approach requires the highpass response of the digital pre-emphasis filter to be a reasonably close match to the inverse of the closed-loop filtering imposed by the largely analog PLL. Another of the solutions is to use a high-order loop filter in the PLL with a sharp lowpass response [30]. Increasing the order of the loop filter increases the attenuation of out-of-band quantization noise which allows for higher-order ΔΣ modulation to reduce in-band quantization noise thereby allowing the loop bandwidth to be increased without increasing the total phase noise. However, as described in [30], this necessitates the use of a Type 1 PLL which significantly complicates the design of the phase detector. Yet another solution is to use a narrow loop bandwidth but modulate the VCO both through the digital ΔΣ modulator and through an auxiliary modulation port at the VCO input [28]. The idea is to apply the low-frequency modulation components at the ΔΣ modulator input and the high frequency modulation components directly to the VCO. Again, matching is an issue, but it has proven to be manageable at least for low-end applications such as Bluetooth transceivers.

VIII. Conclusion

The additional concepts and issues associated with ΔΣ

fractional-*N* PLLs for frequency synthesis relative to integer-*N* PLLs have been presented. It has been shown that ΔΣ fractional-*N* PLLs provide tuning resolution limited only by digital logic complexity, and, in contrast to integer-*N* PLLs, increased tuning resolution does not come at the expense of reduced bandwidth. Since one of the main innovations in a ΔΣ fractional-*N* PLL is the use of a ΔΣ modulator to control the divider modulus, the relevant concepts underlying ΔΣ modulation have been described in detail. A linearized model has been derived from first principles and a design example has been presented to illustrate how the model is used in practice. Techniques for wideband digital modulation of the VCO within a delta-sigma fractional-*N* PLL have also been presented.

ACKNOWLEDGEMENTS

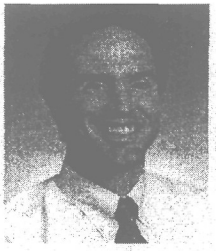
The author is grateful to Sudhakar Pamarti, Eric Siragusa, and Ashok Swaminathan for their helpful discussions and advice regarding this paper.

REFERENCES

1. P. M. Gardner, "Charge-pump phase-lock loops," *IEEE Transactions on Communications*, vol. COM-28, pp. 1849-1858, November 1980.
2. B. Razavi, *Design of Analog CMOS Integrated Circuits*, McGraw Hill, 2001.
3. Bluetooth Wireless LAN Specification, Version 1.0, 2000.
4. U. L. Rohde, *Microvave and Wireless Synthesizers Theory and Design*, John Wiley & Sons, 1997.
5. B. Miller, B. Conley, "A multiple modulator fractional divider," *Annual IEEE Symposium on Frequency Control*, vol. 44, pp. 559-568, March 1990.
6. B. Miller, B. Conley, "A multiple modulator fractional divider," *IEEE Transactions on Instrumentation and Measurement*, vol. 40, no. 3, pp. 578-583, June 1991.
7. T. A. Riley, M. A. Copeland, T. A. Kwasniewski, "Delta-sigma modulation in fractional-N frequency synthesis," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 5, pp. 553-559, May, 1993.
8. S. K. Tewksbury, R. W. Hallock, "Oversampled, linear predictive and noise-shaping coders of order *N* >1," *IEEE Transactions on Circuits and Systems*, vol. CAS-25, pp. 436-447, July 1978.
9. G. Lainey, R. Saintlaurens, P. Senn, "Switched-capacitor second-order noise-shaping coder," *IEE Electronics Letters*, vol. 19, pp. 149-150, February 1983.
10. I. Galton, "Granular quantization noise in a class of delta-sigma modulators," *IEEE Transactions on Information Theory*, vol. 40, no. 3, pp. 848-859, May 1994.
11. N. He, F. Kuhlmann, A. Buzo, "Multiloop sigma-delta quantization," *IEEE Transactions on Information Theory*, vol. 38, no.3, pp.1015-1028, May 1992.
12. I. Galton, "One-bit dithering in delta-sigma modulator-based D/A conversion," Proc. of the IEEE International Symposium on Circuits and Systems, 1993.
13. S. W. Golomb, *Shift Register Sequences*. Laguna Hills, CA: Aegean Park Press, 1982
14. E. J. McCluskey, *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
15. S. K. Tewksbury, R. W. Hallock, "Oversampled, linear predictive and noise-shaping coders of order *N* >1," *IEEE Transactions on Circuits and Systems*, vol. CAS-25, pp. 436-447, July 1978.
16. W. Rhee, B. S. Song, A. Ali, "A 1.1-GHz CMOS fractional-N frequency synthesizer with a 3-b third-order ΔΣ modulator," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 10 , pp. 1453-1460, October 2000.
17. W. L. Lee, C. G. Sodini, "A topology for higher order interpolative coders," *Proceedings of the 1987 IEEE International Symposium on Circuits and Systems*, vol. 2, pp.459-462, May 1987.
18. K. C.-H. Chao, S. Nadeem, W. L. Lee, C. G. Sodini, "A higher order topology for interpolative modulators for oversampling A/D converters," *IEEE Transactions on Circuits and Systems*, vol. 37, no.3, p.309-318, March 1990.
19. Y. Matsuya, K. Uchimura, A. Iwata, T. Kobayashi, M. Ishikawa, T. Yoshitome, "A 16-bit oversampling A-to-D conversion technology using triple integration noise shaping," *IEEE Journal of Solid-State Circuits*, vol. SC-22, pp. 921-929, December 1987.
20. K. Uchimura, T. Hayashi, T. Kimura, A. Iwata, "Oversampling A-to-D and D-to-A converters with multistage noise shaping modulators," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. AASP-36, pp. 1899-1905, December 1988.
21. J. Craninckx, M. S. J. Steyaert, "A fully integrated CMOS DCS-1800 frequency synthesizer," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 2054=2065, December 1998.
22. S. Pamarti, "Techniques for Wideband Fractional-*N* Phase-Locked Loops," PhD Dissertation, University of California, San Diego, 2003.
23. S. R. Norsworthy, R. Schreier, G. C. Temes, Eds. *Delta-Sigma Data Converters, Theory, Design, and Simulation*, New York: IEEE Press, 1997.
24. L. Lin, L. Tee, P. R. Gray, "A 1.4 GHz differential low-noise CMOS frequency synthesizer using a wideband PLL architecture", *IEEE ISSCC Digest of Technical Papers*, pp. 204-205, Feb. 2000.
25. J. G. Proakis, *Digital Communications*, fourth ed., McGraw Hill, 2000.
26. S. Heinen, S. Beyer, J. Fenk, "A 3.0 V 2 GHz transmitter IC for digital radio communication with integrated VCO's," Digest of Technical Papers, *IEEE International Solid-State Circuits Conference*, vol. 38, pp. 150-151,

Feb. 1995.

27. S. Heinen, K. Hadjizada, U. Matter, W. Geppert, V. Thomas, S. Weber, S. Beyer, J. Fenk, E. Matshke, "A 2.7 V 2.5 GHz bipolar chipset for digital wireless communication," Digest of Technical Papers, *IEEE International Solid-State Circuits Conference*, vol. 40, pp. 306-307, Feb. 1997.
28. N. Filiol, et. al., "A 22 mW Bluetooth RF transceiver with direct RF modulation and on-chip IF filtering," Digest of Technical Papers, *IEEE International Solid-State Circuits Conference*, vol. 43, pp. 202-203, Feb. 2001.
29. M. H. Perrott, T. L. Tewksbury III, C. G. Sodini, "A 27-mW CMOS fractional-N synthesizer using digital compensation for 2.5-Mb/s GFSK modulation," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 12, pp. 2048-2059, Dec. 1997.
30. S. Willingham, M. Perrott, B. Setterberg, A. Grzegorek, B. McFarland, "An integrated 2.5GHz ΣΔ frequency synthesizer with 5μs settling and 2Mb/s closed loop modulation," Digest of Technical Papers, *IEEE International Solid-State Circuits Conference*, vol. 43, pp. 200-201, Feb. 2000.



Ian Galton received the Sc.B. degree from Brown University in 1984, and the M.S. and Ph.D. degrees from the California Institute of Technology in 1989 and 1992, respectively, all in electrical engineering.

Since 1996 he has been a professor of electrical engineering at the University of California, San Diego where he teaches and conducts research in the field of mixed-signal integrated circuits and systems for communications. Prior to 1996 he was with UC Irvine, the NASA Jet Propulsion Laboratory, Acuson, and Mead Data Central. His research involves the invention, analysis, and integrated circuit implementation of key communication system blocks such as data converters, frequency synthesizers, and clock recovery systems. The emphasis of his research is on the development of digital signal processing techniques to mitigate the effects of non-ideal analog circuit behavior with the objective of generating enabling technology for highly integrated, low-cost, communication systems. In addition to his academic research, he regularly consults at several communications and semiconductor companies and teaches portions of various industry-oriented short courses on the design of data converters, PLLs, and wireless transceivers. He has served on a corporate Board of Directors and several corporate Technical Advisory Boards, and his is the Editor-in-Chief of the IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing.

Designing Bang-Bang PLLs for Clock and Data Recovery in Serial Data Transmission Systems

Richard C. Walker

Abstract - **Clock recovery using phase-locked loops (PLL) with binary (bang-bang) or ternary-quantized phase detectors has become increasingly common starting with the advent of fully monolithic clock and data recovery (CDR) Circuits in the late 1980's. Bang-bang CDR circuits have the unique advantages of inherent sampling phase alignment, adaptability to multi-phase sampling structures, and operation at the highest speed at which a process can make a working flip-flop. This paper gives insight into the behavior of the nonlinear bang-bang PLL loop dynamics, giving approximate equations for loop jitter, recovered clock spectrum, and jitter tracking performance as a function of various design parameters. A novel analysis shows that the bang-bang loop output jitter grows as the square-root of the input jitter as contrasted with the linear dependence of the linear PLL.**

I. Introduction

Prior to the advent of fully monolithic designs, clock recovery was traditionally performed with some variant of the circuit in Fig. 1. The clock frequency component was typically extracted from

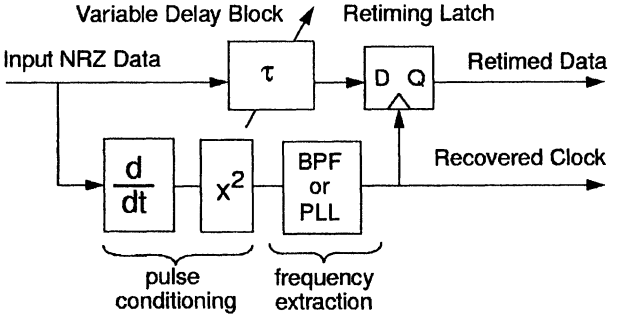


Fig. 1. Traditional non-monolithic clock and data recovery architecture.

the data stream using some combination of differentiation, rectification and filtering. The bandpass frequency filtering was provided by LC tank, surface acoustic wave (SAW) filter, dielectric resonator or PLL. Because the clock recovery path was separate from the data retiming path, it was difficult to maintain optimum sampling phase alignment over process, temperature, data-rate, and voltage variations. Even the PLL techniques had the drawback of using phase detectors with different set-up times than the retiming flip-flop so that the recovered clock was not intrinsically aligned to the optimum sampling point in the data eye. Circuits utilizing SAW resonator filtering typically required hand matching of SAW and circuit temperature coefficients along with custom cut coaxial delay lines for setting the timing of the recovered sampling clock with respect to the data eye [1].

Early monolithic CDR designs imitated these discrete block diagrams. The propagation delay differences between data and clock paths could be ignored as long as the gate delay skew was a negligible fraction of the total bit time, or unit interval. The need for higher link speeds grew faster than Moore's law, and as clock frequencies approached the effective f_T of the active devices, it became increasingly difficult to maintain an optimum sampling phase alignment between the recovered clock and the data over process, temperature, data-rate, and voltage variations.

A second problem was that most linear phase detectors produced narrow pulses with widths proportional to the phase error between the timing of the data and the clock [2], [3]. These narrow pulses required a process speed in excess of that required to simply sample data at a given rate. The timing skew and speed of linear phase detector circuits then became the limiting factor for aggressive designs.

Both these difficulties are eliminated by a family of circuits which simultaneously retime data and measure phase error by using matched flip-flops to sample both the middle of each data bit and the transitions between the data bits. Fig. 2 shows such an

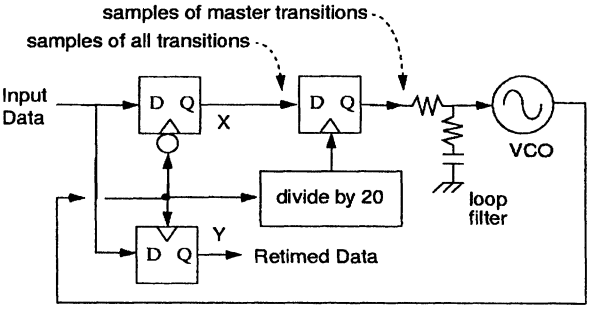


Fig. 2. A simple bang-bang loop using a flip-flop for a phase detector to lock onto a data stream with a guaranteed "0" to "1" transition every 20 bits.

early gigabit-rate monolithic example of such a circuit [4] which samples data with two matched flip-flops. Flip-flop "Y" samples the middle of each data bit on the rising edge of the VCO clock to produce retimed data, while flip-flop "X" samples the transition of each bit using the falling edge of the VCO clock.

The loop is designed to use the 16B/20B line code of Fig. 3 which guarantees a "01" "master transition" every 20 bits. The divide by 20 circuit and associated flip-flop in Fig. 2 discard every

R. Walker is with Agilent Laboratories, 3500 Deer Creek Road, MS 26-U4, Palo Alto CA 94304. (e-mail: rick_walker@labs.agilent.com).

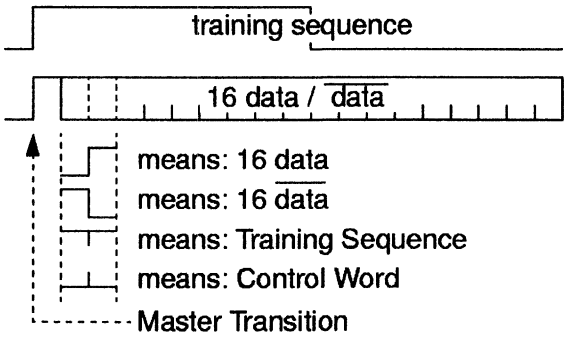


Fig. 3. Format of 16B/20B line code used with bang-bang CDR of.Fig. 2.

transition sample except for this master transition sample. During link start-up a training sequence is sent that has only one rising transition at the location of the master transition. Once the loop is locked, arbitrary data is allowed to be sent at the other 18 bits of the frame, while the transition sampler pays attention only to the data stream in the vicinity of the master transition. If the VCO frequency is too high, the transition flip-flop starts sampling prior to the master transition and outputs a "0" to the loop filter. A slightly lower VCO frequency, on the other hand, will cause the loop to be driven by 1's.

The loop drives the falling edge of the VCO into alignment with the data transitions based on the binary-quantized phase error. Because the clock-to-Q delay of the retiming flip-flop is monolithically matched with the phase detector flip-flop, the PLL aligns the recovered clock precisely in the middle of the data eye with no first-order timing skew over process and temperature variations. Because the narrowest pulse is the output of a flip-flop, such detectors operate at the full speed at which a process is capable of building a functioning flip-flop. This ensures that the phase detector will not be the limiting factor in building the fastest possible retiming circuit.

An additional advantage of flip-flop-based phase detectors is that since they only require simple processing of digital values, they easily generalize to multi-phase sampling structures allowing CDR operation at frequencies in which it would be impossible to build a working full-speed flip-flop. In contrast, most linear phase detectors require at least some analog processing at the full bit rate, limiting process speed and poorly generalizing to multi-phase sampling architectures.

Because of these compelling advantages, the bang-bang loop has become a common design choice for state-of-the CDR designs which are pushing the capability of available IC processes. Fig. 4 surveys CDR designs presented from 1988 to 2001 at the International Solid State Circuits Conference. Designs are plotted by year of presentation against each design's ratio of link speed to effective f_T. The majority of current designs utilize a combination of multiphase sampling structures and bang-bang PLLs. In addition, all CDRs operating at data rates greater than 0.4 f_T are bang-bang designs.

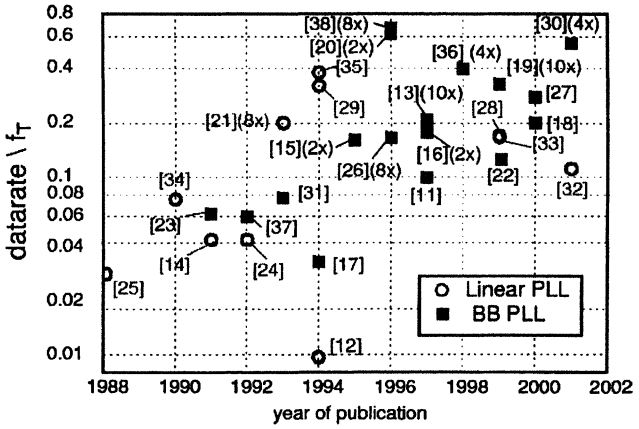


Fig. 4. CDR PLL designs over time. The ratio of link speed to effective process transit frequency is plotted vs year of publication. Multi-phase BB PLLs predominate as data rate approaches the process transit frequency limit. (The number of retiming phases used in each design is given in parentheses.)

II. First-order Loop Dynamics

Unfortunately, transition-sampling flip-flop-based phase detectors can provide only binary (early/late) or ternary (early/late + hold) phase information. This amounts to a hard non-linearity in the loop structure, leading to an oscillatory steady-state and rendering the circuit unanalyzable with standard linear PLL theory. Precise loop behavior can be simulated efficiently with time-step simulators, but this is cumbersome to use for routine design. Fortunately, simple approximate closed-form expressions can be derived for performance parameters of interest, such as loop jitter generation, recovered clock spectrum, and jitter tracking performance as a function of various design parameters.

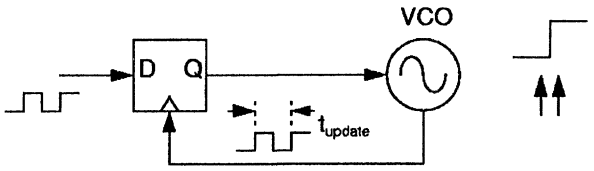


Fig. 5. A simple bang-bang loop using a flip-flop for a phase detector to lock onto square-wave input.

A simple BB PLL is shown in Fig. 5. A flip-flop is used as a phase detector to lock onto a square wave input signal. Depending on whether the VCO phase samples slightly before or after the rising edge of the input square wave, the flip-flop output is either low or high, adjusting the VCO period in such a way as to move the sampling phase error back towards zero. The dynamics of such a binary-quantized loop are equivalent to a data-driven phase detector operating on alternating 0,1 data with 100% transition density, or a master-transition based loop similar to that shown in Fig. 2. For simplicity, we assume that a valid binary phase determination can be made at every timestep. The consequence of random data

and the introduction of a ternary hold mode are considered in a later section.

The first-order BB PLL of Fig. 5 can be rendered into a block diagram for analysis as shown in Fig. 6. The loop phase error

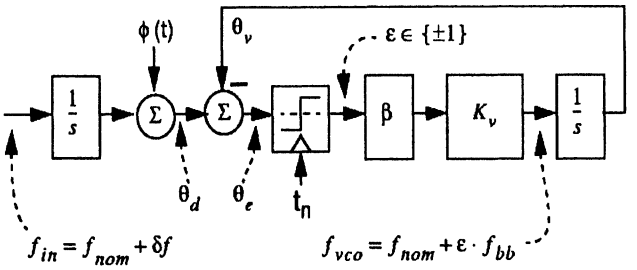


Fig. 6. Block diagram of first order loop showing definition of signal names.

$\theta_e(t_n)$, is defined as the difference between the data phase $\theta_d(t_n)$ and the VCO phase $\theta_v(t_n)$ at the nth sampling time t_n. For convenience, phase is measured with respect to an ideal clock source running at f_{nom}.

The frequency of the incoming data signal differs from the VCO center frequency by δf, and has a zero mean phase jitter of $\phi(t)$. In other words, the data can be considered to have been generated by a pattern generator clocked on the rising edges of the jittered clock signal $\sin[2\pi(f_{nom}+\delta f)t+\phi(t)]$. The data phase $\theta_d(t_n)$ is then $2\pi\delta f t_n + \phi(t_n)$.

The phase detector binary-quantizes the loop phase error at each sampling time to give $\varepsilon_n = \text{sign}[\theta_e(t_n)]$. (Note: In the case of a ternary data-driven phase detector, ε_n may be set to 0 when it is not possible to make a determination of phase error due to consecutive identical bits in the data stream. The consequence of this "hold" state is treated in a later section). The error signal drives the VCO through an attenuator β, to produce a change in frequency of $f_{bb} = \beta K_{vco}$. From time t_n until time t_{n+1}, the VCO operates at one of the two frequencies given by $f_{nom} + \varepsilon_n f_{bb}$.

Because the VCO frequency changes on each cycle, the system has non-uniform sampling times. The time of phase sample $t_{n+1} = t_n + 1/(f_{nom} + \varepsilon_n f_{bb})$. In a typical CDR, f_{bb} is on the order of 0.1% of f_{nom}, so that an analysis assuming uniform time steps of $t_{update} = 1/f_{nom}$ is sufficiently accurate for most purposes. However, for loop analyses requiring exact charge pump balance, such as wide-range loop pull-in without a frequency detector, these non-uniform sampling times must be accounted for.

With the uniform time step approximation, the VCO phase changes up or down (or "walks off") by $\theta_{bb} = 2\pi(f_{bb}/f_{nom})$ radians during each update period.

In summary, the first order loop obeys a simple set of discrete time difference equations:

$$\theta_d(t_n) = \theta_d(0) + 2\pi\delta f t_n + \phi(t_n) \quad (1)$$

$$\theta_v(t_{n+1}) = \theta_v(t_n) + \varepsilon_n\theta_{bb} \quad (2)$$

$$\varepsilon_n = \text{sign}[\theta_d(t_n) - \theta_v(t_n)] \quad (3)$$

As long as the VCO frequency step brackets the input signal frequency error, the loop will remain phase locked. Assuming $\phi(t)$ small, the lock range is: $-f_{bb} < \delta f < f_{bb}$. The loop generates an excess hunting jitter with a peak-to-peak value of two bang-bang phase steps $J_{pp} = 4\pi(f_{bb}/f_{nom})$.

For the loop to be locked, the average VCO frequency must equal the average data frequency. The phase detector duty cycle C, must satisfy the relation

$$\delta f = C(f_{bb}) + (1-C)(-f_{bb}).$$

The value of C is then given by

$$C = \left(\frac{1}{2} + \frac{\delta f}{2f_{bb}}\right).$$

The phase detector duty cycle, and therefore its average output voltage are proportional to the loop frequency error. Fig. 7 shows a simulated loop with a range of input frequencies. The loop is

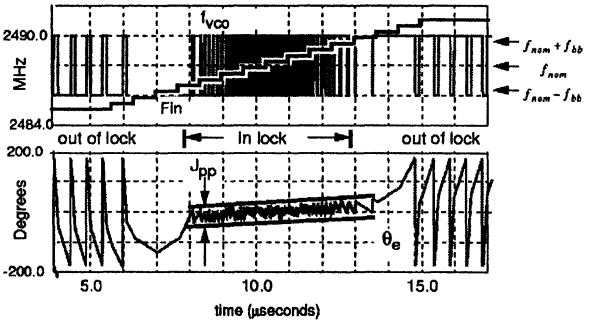


Fig. 7. Simulated response of first-order PLL to a range of input frequencies.

"locked" whenever the input frequency is bracketed by the two VCO frequencies. The rapid alternation between frequencies

slightly too high and slightly too low creates a bounded hunting jitter (J_{pp}).

The derivative of the input data phase deviation, $d[\phi(t)]/dt$, adds to the frequency error that must be tolerated by the loop. Assuming $\delta f = 0$, then for $\phi(t) = A\sin(2\pi f_{mod}t)$, the maximum amplitude A of phase modulation at frequency f_{mod} before onset of slew-rate limiting is $|f_{bb}|/f_{mod}$. Fig. 8. demonstrates the loop at the onset of jitter-induced slew-rate limiting. Although the average input frequency lies within the lock range of the loop, the added sinusoidal jitter causes the instantaneous input frequency deviation to exceed $\pm f_{bb}$. The loop stops toggling and goes into slew rate limiting, leading to a transient phase error.

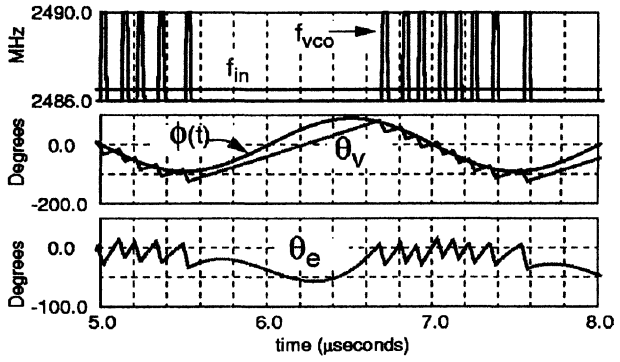


Fig. 8. Simulated response of first-order PLL to sinusoidal input jitter just slightly beyond the tracking capability of the loop.

A. Summary of First-Order Loop

The first-order bang-bang loop has only one degree of freedom. Jitter generation, lock range, and jitter tolerance are all inconveniently controlled by one parameter, f_{bb}. This situation can be improved by using a second control loop to dynamically adjust the nominal VCO frequency f_{nom} to be equal to the incoming data frequency. Because the phase detector duty cycle is proportional to the loop frequency error, this dynamic centering of VCO frequency can be accomplished by adjusting the VCO center frequency in a feedback loop to drive the phase detector duty cycle C to 50%. This decouples the lock range from jitter tolerance and jitter generation, giving more design freedom.

III. Second-Order Loop Dynamics

To extend the loop tracking range independent of the jitter generation, an extra integrator is added between the phase detector and the VCO as in Fig. 9. Since the first-order loop dynamic produces a phase detector duty cycle proportional to the loop frequency error, this added integrator can be viewed as an automatic means for keeping the first-order portion of the loop properly centered on the average incoming data frequency. If certain assumptions are met, as described later, we can consider the system to be composed of two non-interacting loops. These are the loops labeled "bang-bang branch" and "integral branch." If the center frequency control loop is slow enough, the resulting loop behavior will be very similar to a simple first-order loop, but with an extended frequency lock range.

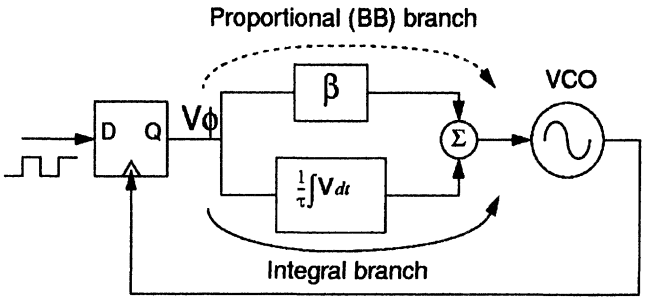


Fig. 9. Second-order bang-bang loop schematic.

A. Stability Factor

To preserve the desirable qualities of the first order loop, it is critical that the phase change due to the proportional branch dominate over the phase change from the integral branch.

The loop phase change in one update time due to the proportional connection is $\Delta\theta_{bb} = \beta V_\phi K_v t_{update}$. The phase change due to the integral branch is $\Delta\theta_{int} = V_\phi K_v t_{update}^2/(2\tau)$. The ratio of these two is the stability factor of the loop

$$\xi \equiv \frac{\Delta\theta_{proportional}}{\Delta\theta_{integral}} = \frac{2\beta\tau}{t_{update}}.$$

The reader should be careful not to confuse the bang-bang loop stability factor ξ with the linear loop damping factor ζ [5].

The discrete time difference equations for the second-order loop can be written as

$$\theta_d(t_n) = \theta_d(0) + 2\pi\delta f t_n + \phi(t_n) \quad (1)$$

$$\theta_v(t_{n+1}) = \theta_v(t_n) + \theta_{bb}\left(\varepsilon_n + \frac{\varepsilon_n}{\xi} + \frac{2}{\xi}\sum_0^n \varepsilon_n\right) \quad (4)$$

$$\varepsilon_n = \text{sign}[\theta_d(t_n) - \theta_v(t_n)] \quad (3)$$

From this, it can be seen that the second-order loop has two degrees of freedom, the loop phase step θ_{bb} (or equivalently, the loop frequency step f_{bb}) and the stability factor ξ. The added loop integrator extends the frequency tracking range, leaving θ_{bb} free to control jitter tolerance and jitter generation.

B. Simulations of Second-Order Loop

Fig. 10 shows two block diagrams for the second-order loop. The upper diagram is a straightforward translation of the schematic in Fig. 9. The lower diagram is a topological re-arrangement

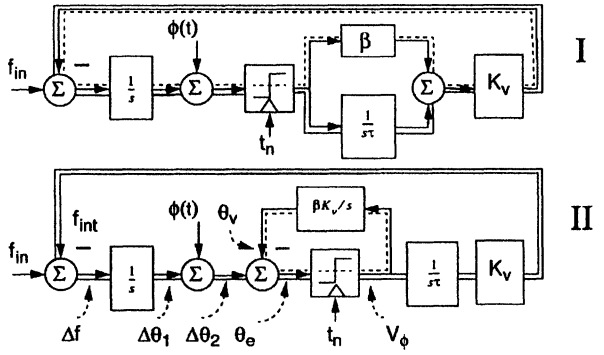


Fig. 10. Two equivalent second-order bang-bang loop block diagrams. The proportional phase-control signal flow is highlighted with a dashed line, and the integral frequency-control loop with a solid line.

which places an inner first-order phase tracking loop inside an outer frequency tracking loop. If one writes the transfer function from the output of the non-linear quantizer block back to the input of the quantizer, it can be shown that both diagrams are exactly equivalent. Some of the signals in the second diagram do not correspond to actual physical variables in the circuit, but they are helpful in understanding the operation of the loop.

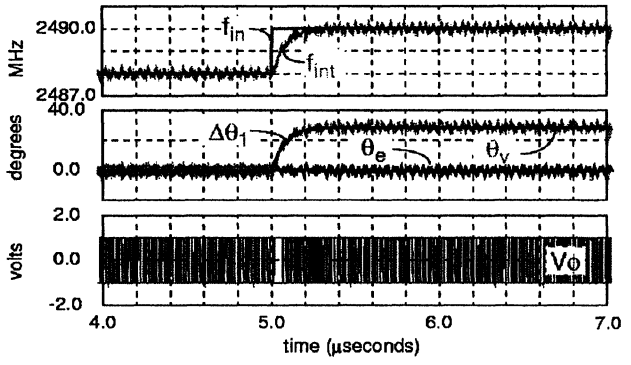


Fig. 11. Second-order loop response to instantaneous frequency step smaller than f_{bb}.

Fig. 11 shows the second-order loop responding to a step change in input frequency f_{in}, producing a slow response f_{int} in the outer integral loop. The resulting phase error $\Delta\theta_1$ is tracked by the inner bang-bang loop θ_v to produce the final sampler phase error θ_e. Notice that, unlike linear PLLs, if the power-supply noise-induced VCO frequency modulation is limited to $\pm f_{bb}$, then there is *no* jitter accumulation or phase transient at the sampling flip-flop.

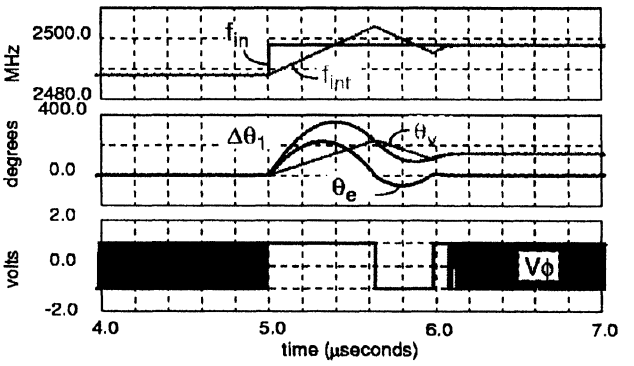


Fig. 12. Second-order loop response to instantaneous frequency step larger than f_{bb}.

Fig. 12 is a simulation in which the input frequency step is bigger than f_{bb}, so the loop goes into slew rate limiting, leading to a transient phase error θ_e at the sampler.

C. Response to Phase Step

For a normalized transient phase step of $\Delta = \theta_{step}/\theta_{bb}$, a first-order loop relocks in Δ update times. The total time for relocking is then $\theta_{step}/(2\pi f_{bb})$.

During the relocking transient of the second-order loop, the loop integrator overshoots the correct steady-state VCO tune voltage. This causes a quadratic overshoot in the phase trajectory.

Fig. 13 shows the second-order phase step response with ξ as a parameter. Up to the first zero crossing, the phase trajectory is given by

$$\frac{\theta(t)}{\theta_{bb}} = \Delta - \left(n + \frac{n^2}{\xi}\right),$$

with $n = t/t_{update}$. The time of the first zero crossing approaches Δ as $\xi \to \infty$, consistent with a first-order loop. In general, the second-order loop is quicker to reach zero phase error than the first-order loop, but pays for this with an oscillatory overshoot. As a conservative rule of thumb, the magnitude of the oscillatory transient of a second-order step response can be considered bounded by the simple linear transient of the first-order loop. The time required to reach steady state, given a step of Δ is always less than or equal to Δ timesteps, independent of ξ.

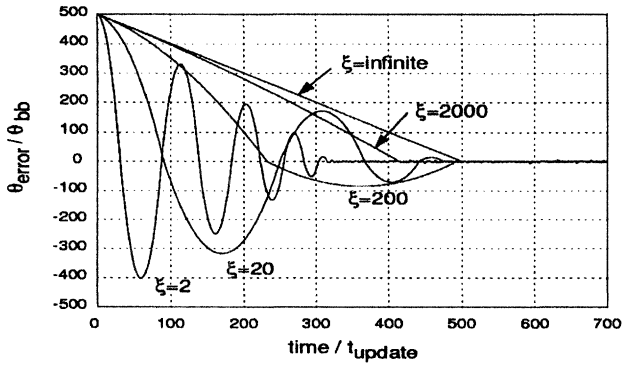


Fig. 13. Noise-free loop response to a phase step with stability factor ξ as a parameter.

IV. Slope Overload

Many systems, such as SONET, specify jitter tolerance in the form of a sinusoidal jitter at various frequencies.

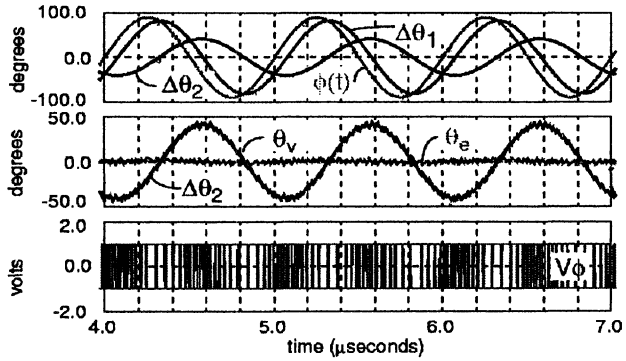


Fig. 14. Second-order loop response to sinusoidal input jitter.

Fig. 14 shows the loop response with a sinusoidal input phase jitter $\phi(t)$. The outer integral loop tracks the input jitter at $\Delta\theta_1$ with a slight phase lag. The resulting phase error $\Delta\theta_2$ is tracked by the inner bang-bang loop θ_v to produce the final sampler phase error θ_e. The duty-cycle of the PD output V_ϕ varies with the slope of $\Delta\theta_2$ which is proportional to the instantaneous frequency error of the outer loop.

In Fig. 15, the phase modulation is increased until the instantaneous frequency error exceeds the inner loop's ability to track. Slew-rate limiting produces a tracking error at the sampler θ_e. A CDR would normally be designed such that slewing would never occur for any valid signal allowed by a particular standard. The next two sections develop an analytic expression for slope overload so that a loop can be easily designed to never slew for signals meeting a typical frequency-domain jitter tolerance specification.

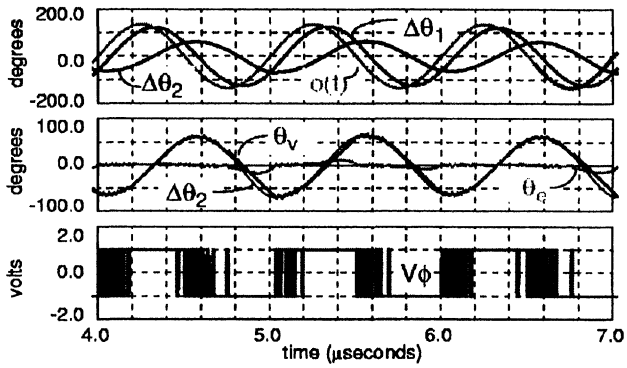


Fig. 15. Second-order loop response to large sinusoidal input jitter.

A. Delta-Sigma Analogy

Before developing an analytic equation for slope overload, it is helpful to introduce a further rearrangement of block diagram II from Fig. 10. Fig. 16 transforms the loop by pulling two integrators through the last summing node prior to the quantizer. The update time interval is set to 1. The definition for bang-bang frequency step $f_{bb} = \beta K_v V_\phi$, and stability factor $\xi = 2\beta\tau / t_{update}$ are also substituted in.

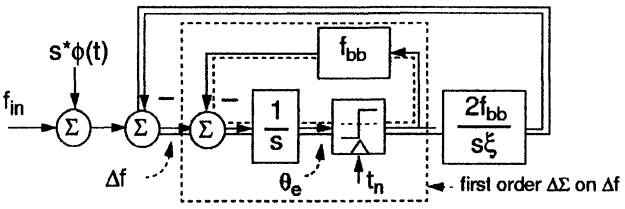


Fig. 16. Redrawing of the loop to show inner $\Delta\Sigma$ inner modulator operating on the loop frequency error.

The shaded area in Fig. 16 shows how the proportional feedback loop can be thought of as an inner $\Delta\Sigma$ modulator producing a phase detector duty cycle proportional to the VCO frequency error [6],[7].

Fig. 17 summarizes an analysis of the first order delta-sigma (after [8]). When the loop is not in slew rate limiting, or in a periodic limit-cycle, the quantizer (e.g., PD) can be replaced with a unity gain element and a noise source $Q(z)$ with the same $A\sin(2\pi tft/t_{update})/(2\pi ft/t_{update})$ noise characteristics as a random binary bitstream. Both these constraints are met in practice as the VCO phase noise is sufficient to eliminate any deterministic limit cycles, and the loop is designed to never slew rate limit on any conforming input signal. This insight is critical as

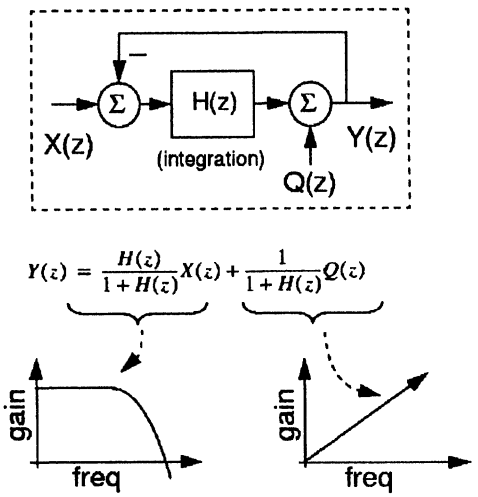


Fig. 17. Simplified analysis of delta-sigma circuit.

it allows linear analysis to be applied whenever the bang-bang loop is not in slew rate limiting.

With the $\Delta\Sigma$ substitution, the inner loop becomes a wide-band unity-gain block as seen from the viewpoint of the outer integral frequency control loop. The noise in the delta-sigma core is first-order frequency shaped towards high frequencies. However, when the frequency noise is converted to phase noise, the shaping is lost and the noise becomes flat.

B. Expression for Slope Overload

A closed-form analysis of slope overload can now be derived. Referring to Fig. 16, the system slews when $|\Delta F| > f_{bb}$. Assuming no slew rate limiting, we can use the results from the $\Delta\Sigma$ analysis to justify replacing the loop quantizer with a unity gain element. The maximum input phase jitter in UI as a function of frequency, $\sigma_j^{max}(s)$, normalized to θ_{bb} can then be calculated using Laplace transforms.

We want to find an input excitation $F(s)$, for which $|\Delta F| = f_{bb}$ at all frequencies. The inner $\Delta\Sigma$ of Fig. 16 has a linearized transfer function of $1/(s+f_{bb})$. Using standard feedback loop theory, the expression for ΔF can then be written as

$$\Delta F = \frac{F(s)}{1+\left(\frac{2f_{bb}}{s\xi}\right)\left(\frac{1}{s+f_{bb}}\right)}.$$

Setting $\Delta F = f_{bb}$, and normalizing the equation by letting f_{bb} and $t_{update} = 1$, we can solve for $F(s)/s$ to get the maximum normalized input phase as a function of normalized frequency

$$\frac{\sigma_j^{max}(s)}{\theta_{bb}} = \left(\left(s^2+s+\frac{2}{\xi}\right)/(s^3+s^2)\right).$$

This is a curious bootstrapped analysis, in that it assumes a lack of slewing to justify the linearization which permits the computation of the onset of slew rate limiting.

Fig. 18 shows a good agreement between this expression and simulated loop performance in which slewing is defined as a contiguous sequence of ten or more identical phase-error indications. This expression can be used to design a loop for a given jitter tolerance. The tolerance plots are single-pole slope for high ξ and high jitter frequency, becoming double-pole at lower frequencies and small ξ. At high frequencies, all of the curves become asymptotic to the single-pole tolerance of a first-order bang-bang PLL. The operating region below each of these curves is where the $\Delta\Sigma$ approximation is valid, and where a linear loop analysis is justified.

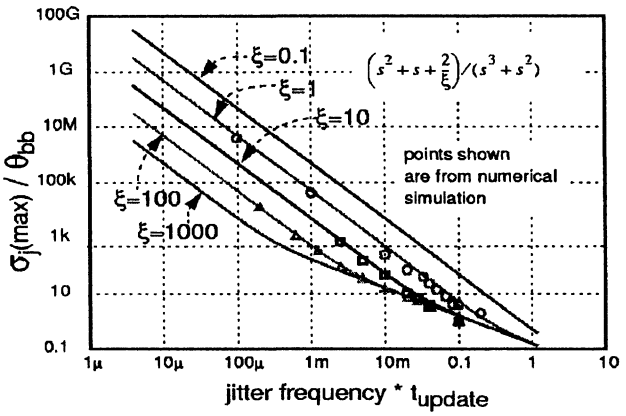


Fig. 18. Normalized amplitude of sinusoidal jitter just sufficient to cause slope overload as a function of normalized jitter frequency and with ξ as a parameter.

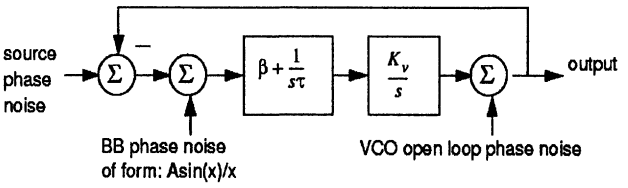


Fig. 19. Loop redrawn replacing phase detector with unity gain element and additive quantization noise.

V. Jitter Generation

With these insights, it is possible to accurately predict the loop jitter generation in the frequency domain. Fig. 19 is a redrawing of the loop replacing the phase detector by a unity gain element, and an additive noise source. The forward loop gain is

$$H(s)= \frac{K_{vco}}{s}\left(\beta + \frac{1}{s\tau}\right).$$

From this can be calculated two transfer functions: the lowpass seen by both the source phase noise and the PD noise to the output, $A(s)= 1/[1+H(s)]$, and the high-pass transfer function from VCO phase noise to the output, $B(s)= H(s)/[1+H(s)]$. As shown in Fig. 20, with a source phase noise P(s), a PD phase noise Q(s), and a VCO phase noise R(s), the total loop jitter generation spectrum becomes the RMS combination of each of the three weighted terms $J(s)= \sqrt{(PA)^2+(QA)^2+(RB)^2}$. The source phase noise is

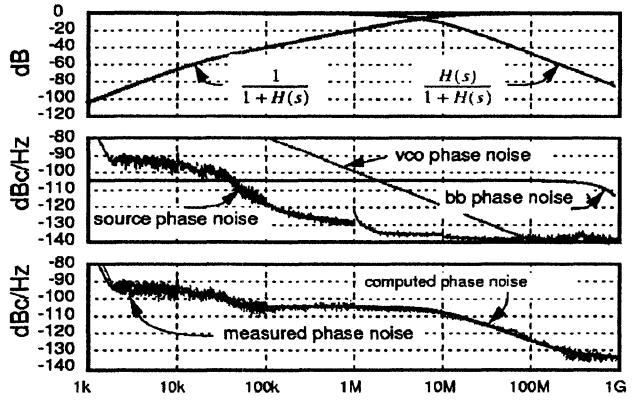


Fig. 20. Example computation of loop jitter generation spectrum with parameters from [11].

generally taken to be the spectrum of the clock driving the data source or BERT, or in the case of a clock multiplying circuit, the spectrum of the reference clock corrected by 20 times the log of the loop frequency multiplication ratio.

The phase noise power is given by

$$S_{RMS} = \int_0^{w_{max}} J^2(s)d\omega.$$

The RMS jitter in unit intervals is then $J_{RMS} = \text{atan}(\sqrt{S_{RMS}})/\pi$.

It should be noted that the linearized loop model is only suitable for computation of the jitter spectrum but not for computing the actual sampling point phase error or other time-domain transient response. The linearized response only covers the dynamics of the outer frequency tracking loop, but does not capture the extra tracking of the internal nonlinear $\Delta\Sigma$ core.

VI. Gaussian Input Noise

Fig. 21 is a plot of output jitter vs input jitter with ξ as a

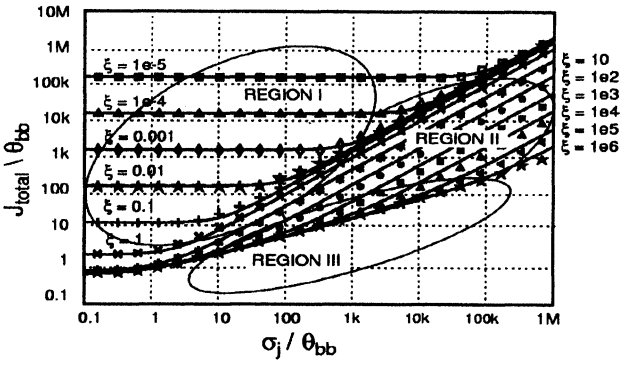


Fig. 21. Normalized output jitter vs input jitter sigma with ξ as a parameter. Simulation is for a non-tristated loop, with square wave data input, 10^8 timesteps per point, and ignoring phase wrapping.

parameter. For convenience, all jitter sigmas are normalized to θ_{bb}, the loop phase step size. The total loop output jitter can be approximated by three regions of operation: $J_{total} \approx J_{idle} + J_{linear} + J_{walk}$. In Region I, the output jitter is independent of input jitter σ_j. This occurs when the self-generated hunting jitter exceeds the input jitter. The RMS jitter in this region is empirically determined to be well approximated by $J_{idle} \approx 0.6 + (1.65/\xi)$. In Region II, the output jitter is proportional to the input jitter. This occurs when the input jitter is so high that, for a given ξ, the bang-bang dynamic is unable to control the second-order portion of the loop. This leads to large quadratic trajectories in the phase domain, causing the loop phase to "hunt" towards the limits of the input jitter distribution. As the loop phase nears the limits of the input jitter distribution, the bang-bang hunting has more effect on stabilizing the second-order loop. In this region, the output jitter is proportional to the input jitter: $J_{lin} \approx 2\sigma_j/(1+\sqrt{\xi})$. In Region III, the output RMS jitter J_{walk} is approximately equal to $0.7 \cdot \sqrt{\sigma_j}$. This surprising result says that loops with large ξ have output jitter which grows as the square root of the input jitter. Contrast this with a linear PLL which simply low-pass filters the input jitter and thus has an output jitter which grows linearly with the input jitter.

An approximate analysis of loop jitter can shed light on this curious square-root dependence of output jitter on input jitter. Assume a zero-mean input jitter distribution with a sigma σ_j. Using a linearized approximation to the standard probability distribution function, the probability of getting an "early" phase error indication for small loop phase deviation $\Delta\theta$, is approximately

$$p_e \approx \frac{1}{2} - \frac{\Delta\theta}{\sigma_j\sqrt{2\pi}}.$$

The expected phase change in the loop after one update time is

$$\theta_{bb}((1-p_e)-p_e) = \frac{2\Delta\theta}{\sigma_j\sqrt{2\pi}}\theta_{bb}$$

The discrete time equation for the average evolution of loop phase under the condition of a small input phase error can then be expressed as

$$\Delta\theta(t_{n+1}) = \left(1 - \frac{2\theta_{bb}}{\sigma_j\sqrt{2\pi}}\right)\Delta\theta(t_n),$$

This equation has the same form as a discrete time approximation to the capacitor voltage in an RC lowpass filter. By analogy, when time is expressed in units of loop update times, any transient phase error in the bang-bang loop can then be said to decay to zero with a time constant of $\tau = \sigma_j\sqrt{2\pi}/(2\theta_{bb})$.

This "lowpass" loop characteristic is being driven by random energy from the early/late phase detector output. A related problem is the computation of the baseline wander voltage generated by passing a random NRZ data stream through a coupling capacitor. It can be shown that the sigma on the capacitor voltage is given by $\sigma_{BLW} = V_{pp}\sqrt{t_{bit}/(8\tau)}$. Extending this analogy to the loop, we can consider the output of the phase detector as a 50% duty-cycle random NRZ data stream. Given that the output from each "bit" must cause a loop phase change of θ_{bb}, we can compute that the effective V_{pp} to satisfy our loop difference equation must be $\sqrt{2\pi}\sigma_j$. We can then compute the loop jitter by using the analogous baseline wander expression with the effective loop V_{pp} and τ. The result is

$$\sigma_{bb} = \sqrt{\frac{\theta_{bb}\sigma_j\sqrt{2\pi}}{8}} \approx 0.79\sqrt{\theta_{bb}\sigma_j},$$

which is consistent with empirical analysis of simulation results.

One further insight into this behavior is offered. The second-order loop drives the phase detector output to a steady-state 50% duty-cycle. In this condition, the loop phase splits the input jitter distribution into equal early and late halves. This means that the bang-bang loop phase is servoed to the *median* of the input jitter distribution rather than to the *mean* as would be the case with a linear loop. Because of this, the bang-bang loop makes a constant modest correction in response to large jitter outliers, rather than the proportionally large overcompensation of a linear loop. This insight supports the idea that the bang-bang loop jitter should only be sub-linearly affected by the magnitude of the input jitter.

VII. Data-Driven Phase Detectors

Unless the data contains a guaranteed periodic transition, the CDR will be required to lock onto random transitions embedded in the data stream. The effects of runlength and transition density on loop performance must then be considered. The effect of these two data attributes is dependent on the type of phase detector used. Most modern codes use some variation of Alexander's phase detector [9] shown in Fig. 22.Two matched flip-flops form the front-end of Alexander's phase detector, with the first flip-flop driven on the rising edge of the 50% duty-cycle clock, and the second flip-flop driven on the falling edge of the same clock. (Using a fully-differential monolithic ring-oscillator, it is possible to achieve a very precise 50% duty-cycle clock source). When the loop is locked, the rising-edge retiming flip-flop samples the center of each data bit and produces a retimed data bit at (A) and the following retimed bit at (B). The falling-edge flip-flop functions as a phase detector by sampling the transition (T) between the data bits (A,B). To improve the circuit's operating speed, the (T) sample is delayed an extra half bit time by a latch so that the logic on (A,T,B) has a full bit time for resolution.

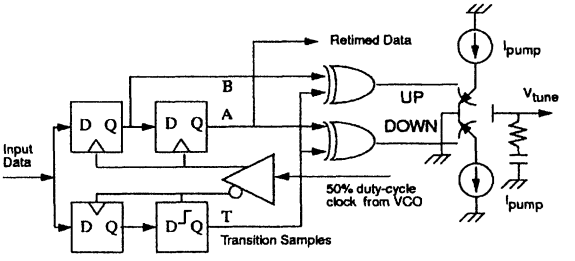


Fig. 22. Modified form of Alexander's ternary-quantized phase detector for NRZ data along with a typical charge pump for driving the VCO tuning input.

The transition sample is then compared to the surrounding data bits to determine whether the clock sampling phase is early or late to derive a binary-quantized (bang-bang) or ternary phase error indication. A truth-table for the logic in Fig. 22 is given in Table 1.

TABLE 1. Truth table for logic in Fig. 22.

State	A	T	B	UP	DOWN	Meaning
0	0	0	0	0	0	hold
1	0	0	1	0	1	early
2	0	1	0	1	1	hold
3	0	1	1	1	0	late
4	1	0	0	1	0	late
5	1	0	1	1	1	hold
6	1	1	0	0	1	early
7	1	1	1	0	0	hold

The states 2 and 5 in Table 1 correspond to the normally impossible condition of sampling a "1" midway between two "0" bits. A custom truth table can use these states to detect either a high bit-error-rate condition [10], a VCO running grossly too slowly (eg: lump these states into the "late" condition), or taken as an indication that a link has locked onto its own VCO crosstalk, perhaps by amplification of power supply noise by pick up from a high-gain optical transimpedance amplifier [11].

Since the mid-bit samples (A,B) straddle the (T) transition sample, it is also possible to detect the lack of a transition. This condition corresponds to states 0 and 7 in Table 1. This information can be used to create an extra ternary hold-state in the PD output, causing the charge pump to hold its value during long runlengths. Both binary and ternary PDs will be discussed in turn, along with their implications on loop performance.

A. Run-length and Latency

Binary phase detectors have no hold state, so the PD continues to put out the last valid phase error indication during long data runlengths. In this situation, the loop idling jitter will be multiplied from the expected value by the maximum runlength of the data. For example, an 8B/10B code has a maximum code runlength of 5 and will have a peak jitter walk-off five times the value of that computed for a "10" repetitive data pattern. The average RMS jitter will be a function of the runlength distributions of each particular code. There is also a trade-off in effective stability factor as a repetitive pattern such as "11110000" will be equivalent to a loop with an effective update time 4 times larger than the expected $t_{update} = 1/f_{nom}$. Since the stability factor is inversely dependent on update time, it is possible for binary PDs to become unstable with data patterns containing very long runs due to the delay in timely phase-error feedback.

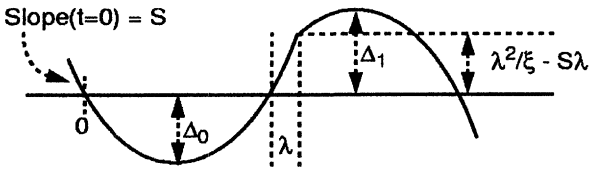


Fig. 23. Setup for computing onset of loop instability with latency λ.

Fig. 23 shows the loop phase trajectory during an acquisition transient. At t=0, the loop crosses zero phase error with $d\phi/dt = S$. From this we can compute an overshoot Δ_o. When the loop phase again crosses zero phase error, the phase detector is late in responding by a time λ. This time is a combination of runlength, latency in the phase detector logic, and high-order poles in the VCO tuning characteristic.

Due to the loop latency λ, the loop overshoots zero phase by $\lambda^2/\xi - S\lambda$ before the "braking" effect of the proportional branch starts to act. The onset of catastrophic instability occurs when $\Delta_1 > \Delta_0$, for this implies exponential growth of the acquisition transient. The convergence is guaranteed whenever $\xi > 2\lambda$.

Although usable for tightly constrained block codes such as 8b/10B, binary phase detectors are essentially unusable for codes such as 10Gb Ethernet 64b/66b or SONET which can have very long runlengths of up to 66 or 80 bits, respectively.

B. Ternary Phase-Detector

The 3-state, or ternary phase detector provides superior jitter performance for data with long runs [12]. Ternary PDs neither charge nor discharge the loop filter during long runs causing the loop to hold the current estimate of the data frequency. Such loops effectively "stop time" during long runs.

If the charge pump does not have a hold-mode, it is possible to emulate a ternary loop, with some loss of performance, by continuously toggling the phase-detector output to approximately maintain the current charge pump voltage during long runs.

The peak idling jitter for ternary loops is unchanged from the simple 100% transition density analysis. The RMS jitter will be reduced by the average transition density. Because the loop phase cannot change during hold mode, the jitter tolerance will be derated by the average transition density. This can easily be taken into account by increasing θ_{bb} appropriately for the characteristics of the code to be used.

C. VCO Tuning Bandwidth

The previous analyses all assumed an infinite VCO tuning bandwidth for the proportional tuning input. A VCO time-constant τ_{vco}, can slightly reduce hunting jitter if it is small compared to the loop update time.

Timeconstants larger than the loop update time prevent the loop from reversing phase slope within an update period and lengthen the loop limit cycle. If the extra pole is thought of as an extra latency $2\tau_{vco}$, then the result of the previous section can be used to give an approximate bound on loop stability. To avoid divergence: $\tau_{vco} < \xi t_{update}/4$. Comparison with simulation verifies this equation as a conservative limit on τ_{vco}.

However, it cannot be recommended to flirt with this boundary. Unless one meticulously checks performance by numerical simulation, it is safest to design the VCO to essentially respond fully in one update time. This is usually very easy to achieve in ring-oscillators and possible with some care using low-Q LC VCOs.

VIII. Conclusion

Bang-bang CDR circuits have the unique advantages of inherent sampling phase alignment, adaptability to multi-phase sam-

pling structures, and operation at the highest speed at which a process can make a working flip-flop. Approximate equations for loop jitter, recovered clock spectrum, and jitter tracking performance as a function of various design parameters have been derived. The median-tracking property of the bang-bang loop resulting in an output jitter equal to the square root of the input jitter has been presented.

ACKNOWLEDGMENT

The author is grateful to the contributions of Birdy Amrutur, Bill Brown, John Corcoran, Craig Corsetto, Dave DiPietro, Brian Donoghue, Jeff Galloway, Andrew Grzegorek, Tom Hornak, Jim Horner, Tom Knotts, Benny Lai, Adolf Leiter, Bill McFarland, Charles Moore, Rasmus Nordby, Cheryl Owen, Pat Petruno, Kent Springer, Guenter Steinbach, Hugh Wallace, Bin Wu, J.T. Wu, and Chu Yen for technical discussions and helpful insights into bang-bang loop behavior.

REFERENCES

[1] C. B. Armitage, "SAW Filter Retiming in the AT&T 432 Mb/s Lightwave Regenerator," in *Conference Proceedings: AT&T Bell Labs*, pp. 102-103, Sept. 3-6, 1984.

[2] C. R. Hogge, Jr., "A Self Correcting Clock Recovery Circuit," *IEEE Transactions on Electron Devices,* vol. ED-32, no. 12, pp. 2704-2706, Dec. 1985.

[3] J. Tani, Crandall, D., Corcoran, J. Hornak, T., "Parallel Interface ICs for 120Mb/s Fiber Optic Links," in *ISSCC Digest of Technical Papers*, pp. 190-191,390, Feb. 1987.

[4] R. C. Walker, T. Hornak, C. Yen and K. H. Springer, "A Chipset for Gigabit Rate Data Communication," in *Proceedings of the 1989 Bipolar Circuits and Technology Meeting*, pp. 288-290 September 18-19 1989.

[5] F. Gardner, *Phaselock Techniques,* New York: John Wiley & Sons, 1979, pp. 8-14.

[6] I. Galton, "Higher-order Delta-Sigma Frequency-to-Digital Conversion," in *Proceedings of IEEE International Symposium on Circuits and Systems, pp.* 441-444, May 30 - June 2, 1994.

[7] I. Galton, "Analog-Input Digital Phase-Locked Loops for Precise Frequency and Phase Demodulation," *Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, vol.* 42, no. 10, pp. 621-630, Oct. 1995.

[8] M. W. Hauser, "Principles of Oversampling A/D Conversion," *J. Audio Eng. So. vol 39*, no. 1/2, pp 3-26, Jan./ Feb. 1991.

[9] J. D. H. Alexander, "Clock Recovery from Random Binary Signals," *Electronics Letters,* vol. 11, no. 22, pp. 541-542, Oct. 1975.

[10] J. Hauenschild, D. Friedrich, J. Herrle, J. Krug, "A Two-Chip Receiver for Short Haul Links up to 3.5Gb/s with PIN-Preamp Module and CDR-DMUX," in *ISSCC Digest of Technical Papers*, pp. 308-309,452, Feb. 1996.

[11] R. C. Walker, C. Stout and C. Yen, "A 2.488Gb/s Si-Bipolar Clock and Data Recovery IC with Robust Loss of Signal Detection," in *ISSCC Digest of Technical Papers* pp. 246-247,466, Feb. 1997.

[12] N. Ishihara and Y. Akazawa, "A Monolithic 156 Mb/s Clock and Data Recovery PLL Circuit Using the Sample-and-Hold Technique," *IEEE Journal of Solid-State Circuits,* vol. 29, no. 12, pp. 1566-1571, Dec. 1994.

[13] D. Chen, and M. O. Baker, "A 1.25 Gb/s, 460mW CMOS Transceiver for Serial Data Communication," in *ISSCC Digest of Technical Papers*, pp. 242- 243,465 Feb. 1997.

[14] L. DeVito, J. Newton, R. Goughwell, J. Bulzacchelli and F.Benkley, "A 52MHz and 155 MHz Clock-Recovery PLL," in *ISSCC Digest of Technical Papers*, pp. 142-143,306, Feb. 1991.

[15] J. F. Ewen, A. X. Widmer, M. Soyuer, K. R. Wrenner, B. Parker and H. A. Ainspan, "Single-Chip 1062Mbaud CMOS Transceiver for Serial Data Communication," in *ISSCC Digest of Technical Papers*, pp. 32-33,336, Feb. 1995.

[16] A. Fiedler, R. Mactaggart, J. Welch and S. Krishnan, "A 1.0625Gbps Transceiver with 2x-Oversampling and Transmit Signal Pre-Emphasis," in *ISSCC Digest of Technical Papers, pp. 238-239,464,* Feb. 1997.

[17] B. Guo, A. Hsu, Y. Wang and J. Kubinec, "125Mb/s CMOS All-Digital Data Transceiver Using Synchronous Uniform Sampling," in I*SSCC Digest of Technical Papers*, pp. 112-113, Feb. 1994.

[18] Y. M. Greshishchev, P. Schvan, J. L. Showell, M. Xu, J. J. Ojha and J. E. Rogers, "A Fully Integrated SiGe Receiver IC for 10-Gb/s Data Rate," *IEEE Journal of Solid State Circuits*, vol. 35, no. 12, pp. 1949-1957, Dec. 2000.

[19] R. Gu, J. M. Tran, H. Lin, A. Yee and M. Izzard, "A 0.5-3.5Gb/s Low-Power Low-Jitter Serial Data CMOS Transceiver," in *ISSCC Digest of Technical Papers,* pp. 352-353,478, Feb. 1999.

[20] J. Hauenschild, C. Dorshcky, T. W. Mohrenfels and R. Seitz, "A 10Gb/s BiCMOS Clock and Data Recovery 1:4-Demultiplexer in a Standard Plastic Package with External VCO," in *ISSCC Digest of Technical Papers*, pp. 202-203,445, Feb. 1996.

[21] T. He, and P. Gray, "A Monolithic 480 Mb/s AGC/Decision/Clock Recovery Circuit in 1.2 um CMOS," *IEEE Journal of Solid State Circuits,* vol. 28, no. 12, pp. 1314-1320, Dec. 1993.

[22] P. Larsson, "A 2-1600MHz 1.2-2.5V CMOS Clock-Recovery PLL with Feedback Phase-Selection and Averaging Phase-Interpolation for Jitter Reduction," in *ISSCC Digest of Technical Papers*, pp. 356-357, Feb. 1999.

[23] B. Lai, and R. C. Walker, "A Monolithic 622Mb/s Clock Extraction Data Retiming Circuit," in *ISSCC Digest of Technical Papers,* pp. 144,145, Feb. 1991.

[24] T. H. Lee, and J. F. Bulzacchelli, "A 155MHz Clock Recovery Delay- and Phase-Locked Loop," *IEEE Journal of Solid State Circuits* vol. 27, no. 12, pp. 1736-1746, Dec. 1992.

[25] R. H. Leonowich, and J. M. Steininger, "A 45-MHz CMOS phase/frequency-locked loop timing recovery circuit," in *ISSCC Digest of Technical Papers*, pp. 14-15,278-279, Feb. 1988.

[26] I. Lee, C. Yoo, W. Kim, S. Chai and W. Song, "A 622Mb/s CMOS Clock Recovery PLL with Time- Interleaved Phase Detector Array," in *ISSCC Digest of Technical Papers*, pp. 198-199,444, Feb. 1996.

[27] M. Meghelli, B. Parker, H. Ainspan and M. Soyuer, "A SiGe BiCMOS 3.3V Clock and Data Recovery Circuit for 10Gb/s Serial Transmission Systems," in *ISSCC Digest of Technical Papers*, pp. 56-57, Feb. 2000.

[28] T. Morikawa, M. Soda, S. Shiori, T. Hashimoto, F. Sato and K. Emura, "A SiGe Single-Chip 3.3V Receiver IC for 10Gb/s Optical Communication System," in I*SSCC Digest of Technical Papers*, pp. 380-381,481, Feb. 1999.

[29] A. Pottbacker, and U. Langmann, "An 8GHz Silicon Bipolar Clock-Recovery and Data-Regenerator IC," *IEEE Journal of Solid State Circuits* vol. 29, no. 12, pp. 1572-1576, Dec. 1994.

[30] M. Reinhold, C. Dorschky, F. Pullela, E. Rose, P. Mayer, P. Paschke, Y. Baeyens, J. Mattia and F. Kunz, "A Fully-Integrated 40Gb/s Clock and Data Recovery / 1:4 DEMUX IC in SiGe Technology," in *ISSCC Digest of Technical Papers,* pp. 84-85,435, Feb. 2001.

[31] M. Soyuer, and H. A. Ainspan, "A Monolithic 2.3 Gb/s 100mW Clock and Data Recovery Circuit," in *ISSCC Digest of Technical Papers*, pp. 158-159,282, Feb. 1993.

[32] S. Ueno, K. Watanabe, T. Kato, T. Shinohara, K. Mikami, T. Hashimoto, A. Takai, K. Washio, R. Takeyar and T. Harada, "A Single-Chip 10Gb/s Transceiver LSI using SiGe SOI/BiCMOS," in *ISSCC Digest of Technical Papers,* pp. 82-83,435, Feb. 2001.

[33] H. Wang, and R. Nottenburg, "A 1Gb/s CMOS Clock and Data Recovery Circuit," in *ISSCC Digest of Technical Papers*, pp. 354-355,477, Feb. 1999.

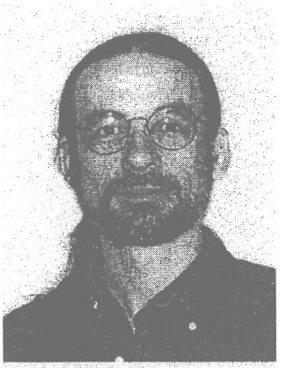
[34] P. Wallace, R. Bayruns, J. Smith, T. Laverick and R. Shuster, "A GaAs 1.5Gb/s Clock Recovery and Data Retiming Circuit," in *ISSCC Digest of Technical Papers*, pp. 192-193, Feb. 1990.

[35] Z. Wang, M. Berroth, J. Seibel, P. Hofmann, A. Hulsmann, Kohler, B. Raynor and J. Schneider, "19GHz Monolithic Integrated Clock Recovery Using PLL and 0.3um Gate-Length Quantum-Well HEMTs," in *ISSCC Digest of Technical Papers*, pp. 118-119, Feb. 1994,

[36] R. C. Walker, K. Hsieh, T. A. Knotts and C. Yen, "A 10Gb/s Si-Bipolar TX/RX Chipset for Computer Data Transmission," in *ISSCC Digest of Technical Papers,* pp. 302-303,450, Feb. 1998.

[37] R. C. Walker, J. Wu, C. Stout, B. Lai, C. Yen, T. Hornak and P. Petruno, "A 2-Chip 1.5Gb/s Bus-Oriented Serial Link Interface," in *ISSCC Digest of Technical Papers,* pp. 226-227,291, Feb. 1992.

[38] C. K. Yang, and M. A. Horowitz, "0.8um CMOS 2.5Gb/s Oversampled Receiver for Serial Links," *IEEE Journal of Solid State Circuits* vol. 31, no. 12, pp. 20150-2023, Dec. 1996.



Richard Walker was born in San Rafael CA, in 1960. He received the B.S. degree in Engineering and Applied Science from the California Institute of Technology in 1982, and an M.S. degree in Computer Science from California State University, Chico, CA in 1992. Rick joined Agilent Laboratories (formerly Hewlett-Packard Laboratories) in 1981, where he is currently a Principal Project Engineer. Since that time, he has worked in the areas of broadband-cable modem design, solid-state laser characterization, phase-locked-loop theory, linecode design, and gigabit-rate serial data transmission. He holds 15 U.S. patents.

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

Kenneth S. Kundert

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

I. Introduction

Phase-locked loops (PLLs) are used to implement a variety of timing related functions, such as frequency synthesis, clock and data recovery, and clock de-skewing. Any jitter or phase noise in the output of the PLL used in these applications generally degrades the performance margins of the system in which it resides and so is of great concern to the designers of such systems. Jitter and phase noise are different ways of referring to an undesired variation in the timing of events at the output of the PLL. They are difficult to predict with traditional circuit simulators because the PLL generates repetitive switching events as an essential part of its operation, and the noise performance must be evaluated in the presence of this large-signal behavior. SPICE is useless in this situation as it can only predict the noise in circuits that have a quiescent (time-invariant) operating point. In PLLs the operating point is at best periodic, and is sometimes chaotic. Recently a new class of circuit simulators has been introduced that are capable of predicting the noise behavior about a periodic operating point [1]. SpectreRF is the most popular of this class of simulators and, because of the algorithms used in its implementation, is likely to be the best suited for this application [2]. These simulators can be used to predict the noise performance of PLLs. The ideas presented in this paper allow those simulators to be applied even to those PLLs that have chaotic operating points.

A. Frequency Synthesis

The focus of this paper is frequency synthesis. The block diagram of a PLL operating as a frequency synthesizer is shown in Figure 1 [3]. It consists of a reference oscillator (OSC), a phase/frequency detector (PFD), a charge pump (CP), a loop filter (LF), a voltage-controlled oscillator (VCO), and two frequency dividers (FDs). The PLL is a feedback loop that, when in lock, forces f_{fb} to be equal to f_{ref}. Given an input frequency f_{in}, the frequency at the output of the PLL is

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

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

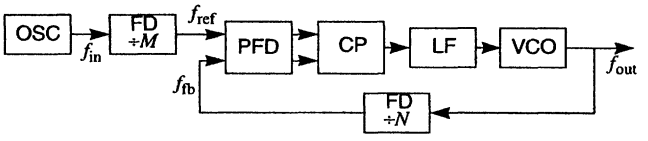


Fig. 1. The block diagram of a frequency synthesizer.

B. Direct Simulation

In many circumstances, SpectreRF† can be directly applied to predict the noise performance of a PLL. To make this possible, the PLL must at a minimum have a periodic steady state solution. This rules out systems such as bang-bang clock and data recovery circuits and fractional-N synthesizers because they behave in a chaotic way by design. It also rules out any PLL that is implemented with a phase detector that has a dead zone. A dead zone has the effect of opening the loop and letting the phase drift seemingly at random when the phase of the reference and the output of the voltage-controlled oscillator (VCO) are close. This gives these PLLs a chaotic nature.

To perform a noise analysis, SpectreRF must first compute the steady-state solution of the circuit with its periodic steady state (PSS) analysis. If the PLL does not have a periodic solution, as the cases described above do not, then it will not converge. There is an easy test that can be run to determine if a circuit has a periodic steady-state solution. Simply perform a transient analysis until the PLL approaches steady state and

Ken Kundert is with Cadence Design Systems, San Jose, California, *kundert@cadence.com*.

† Spectre is a registered trademark of Cadence Design Systems.

then observe the VCO control voltage. If this signal consists of frequency components at integer multiples of the reference frequency, then the PLL has a periodic solution. If there are other components, it does not. Sometimes it can be difficult to identify the undesirable components if the components associated with the reference frequency are large. In this case, use the strobing feature of Spectre's transient analysis to eliminate all components at frequencies that are multiples of the reference frequency. Do so by strobing at the reference frequency. In this case, if the VCO control voltage varies in any significant way the PLL does not have a periodic solution.

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

For PLLs that are candidates for direct simulation using SpectreRF, simply configure the simulator to perform a PSS analysis followed by a periodic noise (PNoise) analysis. The period of the PSS analysis should be set to be the same as the reference frequency as defined in Figure 1. The PSS stabilization time (tstab) should be set long enough to allow the PLL to reach lock. This process was successfully followed on a frequency synthesizer with a divide ratio of 40 that contained 2500 transistors, though it required several hours for the complete simulation [4].

C. *When Direct Simulation Fails*

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

D. *Monte Carlo-Based Methods*

Demir proposed an approach for simulating PLLs whereby a PLL is described using behavioral models simulated at a high level [5, 6]. The models are written such that they include jitter in an efficient way. He also devised a simulation algorithm based on solving a set of nonlinear stochastic differential equations that is capable of characterizing the circuit-level noise behavior of blocks that make up a PLL [6, 7]. 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 [8]. Once everything is ready, simulation of the PLL occurs with the blocks of the PLL being described with behavioral models that exhibit jitter. The actual jitter or phase noise statistics are observed during this simulation. Generally tens to hundreds of thousands of cycles are simulated, but the models are efficient so the time required for the simulation is reasonable. This approach allows prediction of PLL jitter behavior once the noise behavior of the blocks has been characterized. However, it requires the use of an experimental simulator that is not readily available to characterize the jitter of the blocks.

In an earlier series of papers [9, 10], the relevant ideas of Demir were adapted to allow use of a commercial simulator, Spectre [11], and an industry standard modeling language, Verilog-A† [12]. These ideas are further refined in the later half of this paper.

E. *Predicting Noise in PLLs*

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

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

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

The other approach formulates the models in terms of voltage, which are referred to as voltage-domain models. The advantage of voltage-domain models is that they can be refined to implementation. In other words, as the design process transitions to being more of a verification process, the abstract behavioral models initially used can be replaced with detailed gate- or transistor-level models in order to verify the PLL as implemented.

A voltage-domain model is strongly nonlinear and never has a quiescent operating point, making it incompatible with a SPICE-like noise analysis. Often such models have a periodic operating point and so can be analyzed with small-signal RF noise analysis (SpectreRF), but it is also common for that not to be the case. For example, a fractional-*N* synthesizer does

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

not have a periodic operating point. Occasionally, the circuit is sensitive enough that the noise affects the large-signal behavior of the PLL, such as with bang-bang clock-and-data recovery PLLs, which invalidates any use of small-signal noise analysis.

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

The process of predicting the jitter of a PLL with voltage-domain models involves:

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

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

II. Phase-Domain Model

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

This is all true when simulating the PLL in terms of voltages and currents. When doing so, one is said to be using *voltage-domain models*. However, that is not the only option available. It is also possible to formulate models based on the phase of the signals. In this case, one would be using *phase-domain models*. The high frequency variations associated with the voltage-domain models are not present in phase-domain models, and so simulations are considerably faster. In addition, when in lock the phase-domain-based models generally have constant-valued operating points, which simplifies small-signal analysis, making it easier to study the closed-loop dynamics and noise performance of the PLL using either AC or noise analysis.

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

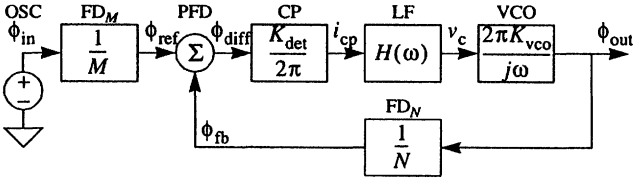


Fig. 2. Linear time-invariant phase-domain model of the synthesizer shown in Figure 1.

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

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

that the signal is driving is edge triggered. For simplicity, we assume that such inputs are sensitive to the rising edges of the signal, that t_0 specifies the time of a rising edge, and that the signal is transitioning between 0 and 1.

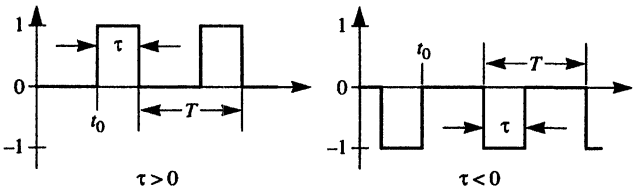


Fig. 3. The pulse train waveform represented by $\Pi(t_0, \tau, T)$.

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

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

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

There are many different types of phase detectors that can be used, each requiring a somewhat different model. Consider a simple phase-frequency detector combined with a charge pump [13]. In this case, the detector takes two inputs, $v_1 = \Pi(t_1, T)$ and $v_0 = \Pi(t_0, T)$ and produces an output $i_{cp} = I_{max}\Pi(t_0, t_1 - t_0, T)$ where I_{max} is the maximum output current of the charge pump. The output of the charge pump immediately passes through a low pass filter that is designed to suppress signals at frequencies of $1/T$ and above, so in most cases the pulse nature of this signal can be ignored in favor of its average value, $\langle i_{cp} \rangle$. Thus, the transfer characteristic of the combined PFD/CP is

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

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

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

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

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

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

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

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

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

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

or in the frequency domain,

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

A. Small-Signal Stability

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

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

to be the forward gain,

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

to be the feedback factor, and

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

to be the loop gain. The loop gain is used to explore the small-signal stability of the loop. In particular, the phase margin is an important stability metric. It is the negative of the difference between the phase shift of the loop at unity gain and 180°, the phase shift that makes the loop unstable. It should be no less than 45° [14]. When concerned about phase noise or jitter, the phase margin is typically 60° or more to reduce peaking in the closed-loop gain, which results in excess phase noise.

B. Noise Transfer Functions

In Figure 4 various sources of noise have been added. These noise sources can represent either the noise created by the blocks due to intrinsic noise sources (thermal, shot, and flicker noise sources), or the noise coupled into the blocks from external sources, such as from the power supplies, the substrate, etc. Most are sources of phase noise, and denoted

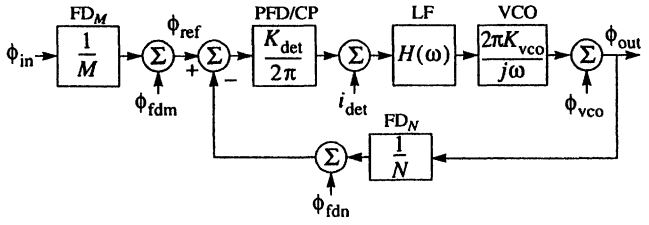


Fig. 4. Linear time-invariant phase-domain model of the synthesizer shown in Figure 2 with representative noise sources added. The ϕ's represent various sources of noise.

ϕ_{in}, ϕ_{fdm}, ϕ_{fdn}, and ϕ_{vco}, because the circuit is only sensitive to phase at the point where the noise is injected. The one exception is the noise produced by the PFD/CP, which in this case is considered to be a current, and denoted i_{det}.

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

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

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

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

and by inspection,

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

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

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

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

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

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

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

C. Noise Model

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

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

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

Listing 1 — Phase-domain model for a PLL configured as a frequency synthesizer.

```
`include "discipline.h"

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

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

endmodule
```

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

Listing 2 — Signal flow discipline definition for phase signals (the nature *Angle* is defined in *discipline.h*).

```
`include "discipline.h"

discipline phase
    potential Angle;
enddiscipline
```

III. Oscillators

Oscillators are responsible for most of the noise at the output of the majority of well-designed frequency synthesizers. This is because oscillators inherently tend to amplify noise found near their oscillation frequency and any of its harmonics. The reason for this behavior is covered next, followed by a description of how to characterize and model the noise in an oscillator. The origins of oscillator phase noise are described in a conceptual way here. For a detailed description, see the papers by Käertner or Demir et al [15, 16, 17].

A. Oscillator Phase Noise

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

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

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

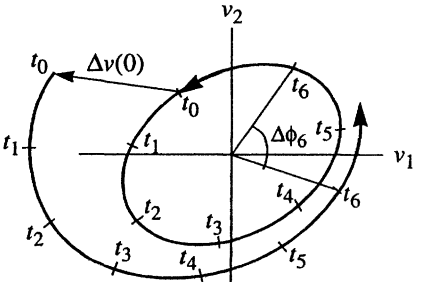


Fig. 5. The trajectory of an oscillator shown in state space with and without a perturbation Δv. By observing the time stamps (t_0, ..., t_6) one can see that the deviation in amplitude dissipates while the deviation in phase does not.

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

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

a unit step $s(t)$. The phase shift over time for an arbitrary input disturbance u is

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

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

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

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

Notice that there is only one degree of freedom — the phase of the oscillator as a whole. There is no restoring force when the phase of all signals associated with the oscillator shift together, however there would be a restoring force if the phase of signals shifted relative to each other. This observation is significant in oscillators with multiple outputs, such as quadrature or ring oscillators. The dominant phase variations appear identically in all outputs, whereas relative phase variations between the outputs are naturally suppressed by the oscillator or added by subsequent circuitry and so tend to be much smaller [8].

B. Characterizing Oscillator Phase Noise

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

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

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

Then from (19) the response will take the form

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

where the factor of $(2\pi)^2$ in the denominator of (19) has been absorbed into n, the constant of proportionality. Thus, the response of the oscillator to white and flicker noise sources is characterized using just two parameters, n and f_c, where n is the portion of S_ϕ attributable to the white noise sources alone at $\Delta f = 1$ Hz and f_c is the flicker noise corner frequency. As shown in Figure 6, n is extracted by simply extrapolating to 1 Hz from a frequency where the noise from the white sources dominates.

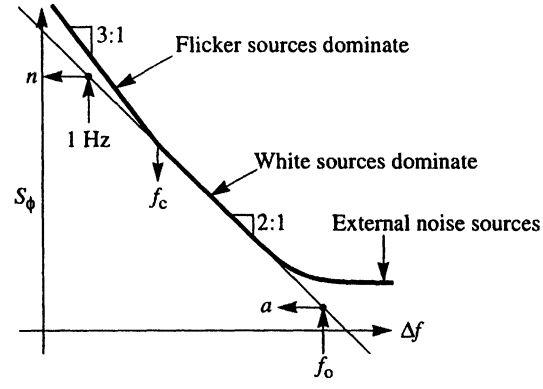


Fig. 6. Extracting the noise parameters, n, a, and f_c, for an oscillator. The parameter a is an alternative to n where $n = af_o^2$. It is used later. The graph is plotted on a log-log scale.

S_ϕ is not directly observable and often difficult to find, so now S_ϕ is related to $\mathcal{L}$, the power spectral density of the output voltage noise S_v normalized by the power in the fundamental tone. S_v is directly available from either measurement with a spectrum analyzer or from RF simulators, and $\mathcal{L}$ is defined as

$$\mathcal{L}(\Delta f) = \frac{S_v(f_o + \Delta f)}{2|V_1|^2}, \tag{22}$$

where V_1 is the fundamental Fourier coefficient of v, the output signal. It satisfies

$$v(t) = \sum_{k=-\infty}^{\infty} V_k e^{j2\pi k f_o t}. \tag{23}$$

In (41) of [15], Demir et al shows that for a free-running oscillator perturbed only by white noise sources†

$$\mathcal{L}(\Delta f) = \frac{1}{2}\frac{n}{n^2\pi^2 + \Delta f^2}, \tag{24}$$

which is a Lorentzian process with corner frequency of

$$f_{corner} = n\pi \ll f_o. \tag{25}$$

At frequencies above the corner,

$$\mathcal{L}(\Delta f) = \frac{n}{2\Delta f^2} = \frac{1}{2}S_\phi(\Delta f), \tag{26}$$

which agrees with Vendelin [18].

Use (21) to extract f_c. Then use both (21) and (26) to determine n by choosing Δf well above the flicker noise corner frequency, f_c, and the corner frequency of (25), f_{corner}, to avoid ambiguity and well below f_o to avoid the noise from other sources that occur at these frequencies.

† Demir uses c rather than n, where $n = cf_o^2$.

C. Phase-Domain Models for the Oscillators

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

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

Listing 3 — Phase-domain oscillator noise model.

```
`include "discipline.h"

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

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

Listing 4 — Phase-domain VCO noise model.

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

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

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

IV. Loop Filter

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

Listing 5 — Loop filter model.

```
`include "discipline.h"

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

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

endmodule
```

† The *ground* statement is not currently supported in Cadence's Verilog-A implementation, so instead ground is explicitly passed into the module.

V. Phase Detector and Charge Pump

As with the VCO, the noise of the PFD/CP as needed by the phase-domain model is found directly with simulation. Simply drive the block with a representative periodic signal, perform a PNoise analysis, and measure the output noise current. In this case, a representative signal would be one that produced periodic switching at the output. This is necessary to capture the noise present during the switching process. Generally the noise appears as in Figure 7, in which case the noise is parameterized with n and f_c. n is the noise power density at frequencies above the flicker noise corner frequency, f_c, and below the noise bandwidth of the circuit.

The phase-domain model for the PFD/CP is given in Listing 6. It is based on (3). Alternatively, as before one could

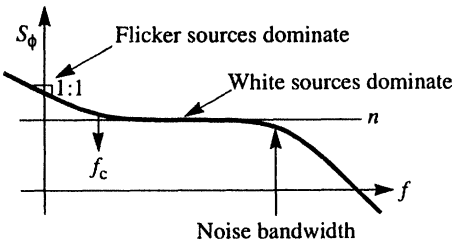


Fig. 7. Extracting the noise parameters, n and f_c, for the PFD/CP. The graph is plotted on a log-log scale.

use the *noise_table* function in lieu of the *white_noise* and *flicker_noise* functions to use the measured noise results directly.

Listing 6 — Phase-domain phase detector noise model.

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

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

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

VI. Frequency Dividers

There are several reasons why the process of extracting the noise produced by the frequency dividers is more complicated than that needed for other blocks. First, the phase noise is needed and, as of the time when this document was written, SpectreRF reports on the total noise and does not yet make the phase noise available separately. Secondly, the frequency dividers are always followed by some form of edge-sensitive thresholding circuit, in this case the PFD, which implies that the overall noise behavior of the PLL is only influenced by the noise produced by the divider at the time when the threshold is being crossed in the proper direction. The noise produced by the frequency divider is cyclostationary, meaning that the noise power varies over time. Thus, it is important to analyze the noise behavior of the divider carefully. The second issue is discussed first.

A. Cyclostationary Noise.

Formally, the term cyclostationary implies that the autocorrelation function of a stochastic process varies with t in a periodic fashion [19, 20], which in practice is associated with a periodic variation in the noise power of a signal. In general, the noise produced by all of the nonlinear blocks in a PLL is strongly cyclostationary. To understand why, consider the noise produced by a logic circuit, such as the inverter shown in Figure 8. The noise at the output of the inverter, n_{out}, comes from different sources depending on the phase of the output signal, v_{out}. When the output is high, the output is insensitive to small changes on the input. The transistor M_P is on and the noise at the output is predominantly due to the thermal noise from its channel. This is region A in the figure. When the output is low, the situation is reversed and most of the output noise is due to the thermal noise from the channel of M_N. This is region B. When the output is transitioning, thermal noise from both M_P and M_N contribute to the output. In addition, the output is sensitive to small changes in the input. In fact, any noise at the input is amplified before reaching the output. Thus, noise from the input tends to dominate over the thermal noise from the channels of M_P and M_N in this region. Noise at the input includes noise from the previous stage and noise from both devices in the form of flicker noise and thermal noise from gate resistance. This is region C in the figure.

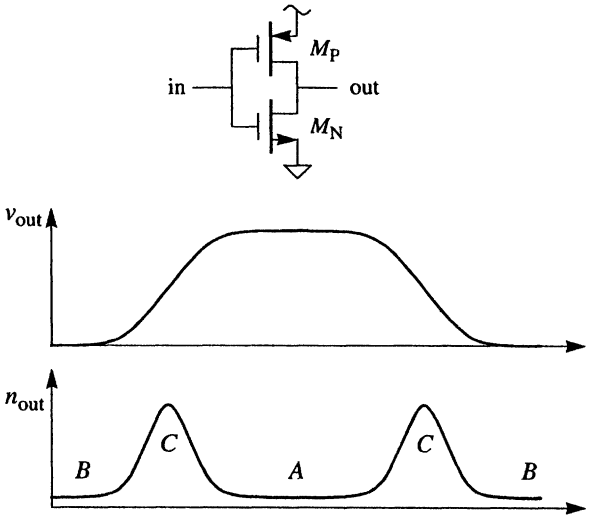


Fig. 8. Noise produced by an inverter (n_{out}) as a function of the output signal (v_{out}). In region A the noise is dominated by the thermal noise of M_P, in region B its dominated by the thermal noise of M_N, and in region C the output noise includes the thermal noise from both devices as well as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

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

circuit is periodically sampled to create a discrete-time random sequence, as shown in Figure 9. SpectreRF then computes the power-spectral density of the sequence. The sample time would be adjusted to coincide with the desired threshold crossings. Since the T-periodic cyclostationary noise process is sampled every T seconds, the resulting noise process is stationary. Furthermore, the noise present at times other than at the sample points is completely ignored.

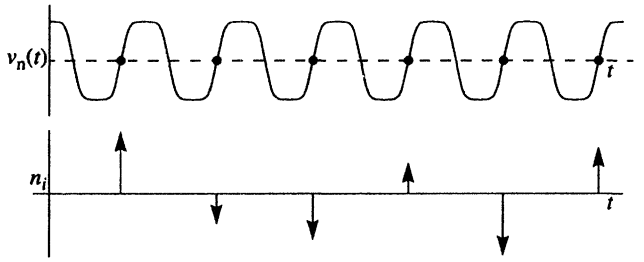


Fig. 9. Strobed noise. The lower waveform is a highly magnified view of the noise present at the strobe points in v_n, which are chosen to coincide with the threshold crossings in v.

B. Converting to Phase Noise

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

$$n_i = v_n(iT) - v(iT). \tag{27}$$

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

$$v_n(t) = v\left(t + \frac{\phi(t)}{2\pi f_0}\right). \tag{28}$$

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

$$v_n(t) \cong v(t) + \frac{dv(t)}{dt}\frac{\phi(t)}{2\pi f_0} \tag{29}$$

and

$$n_i \cong v(iT) + \frac{dv(iT)}{dt}\frac{\phi(iT)}{2\pi f_0} - v(iT) = \frac{dv(iT)}{dt}\frac{\phi(iT)}{2\pi f_0}. \tag{30}$$

Finally, ϕ_i can be found from n_i using

$$\phi_i = 2\pi f_0 n_i / \frac{dv(iT)}{dt}. \tag{31}$$

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

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

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

$$S_\phi(f) = \left[2\pi f_0 / \frac{dv(iT)}{dt}\right]^2 S_n(f). \tag{32}$$

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

C. Phase-Domain Model for Dividers

To extract the phase noise of a divider, drive the divider with a representative periodic input signal and perform a PSS analysis to determine the threshold crossing times and the slew rate (dv/dt) at these times. Then use SpectreRF's strobed PNoise analysis to compute $S_n(f)$. When running PNoise analysis, assure that the *maxsidebands* parameter is set sufficiently large to capture all significant noise folding. A large value will slow the simulation. To reduce the number of sidebands needed, use T as small as possible. $S_\phi(f)$ is then computed from (32). Generally the noise appears as in Figure 10. Notice that the noise is periodic in f with period $1/T$ because n is a discrete-time sequence with period T. The parameters n and f_c for the divider are extracted as illustrated. The high frequency roll-off is generally ignored because it occurs above the frequency range of interest.

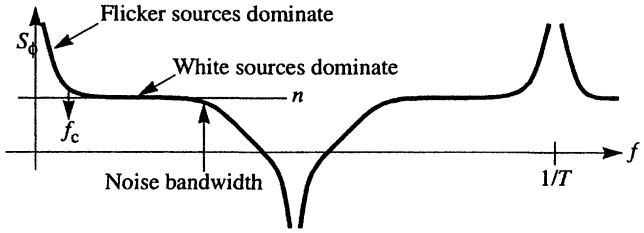


Fig. 10. Extracting the noise parameters, n and f_c, for the divider.

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

$$S_{\phi\,\text{out}} = T_K^2 \sum_{k=0}^{K} \frac{S_{\phi k}}{T_k^2} \tag{33}$$

where $S_{\phi 0}$ and T_0 are the phase noise and signal period at the input to the first stage of the ripple counter.

With undesired variations in the supplies or in the substrate the resulting phase noise in each stage would be correlated, so one would need to compute the transfer function from the sig-

nal source to the phase noise of each stage and combine in a vector sum.

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

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

Listing 7 — Phase-domain divider noise model.

```
`include "discipline.h"

module divider(in, out);
input in; output out;
phase in, out;
parameter real ratio = 1 from (0:inf);// divide ratio
parameter real n = 0 from [0:inf);
    // white output phase noise (rads²/Hz)
parameter real fc = 0 from [0:inf);
    // flicker noise corner frequency (Hz)

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

VII. Fractional-*N* Synthesis

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

The constraint on the loop bandwidth imposed by the required frequency resolution is eliminated if the divide ratio *N* is not limited to be an integer. This is the idea behind fractional-*N* synthesis. In practice, one cannot directly implement a frequency divider that implements non-integer divide ratio except in a few very restrictive cases, so instead a divider that is capable of switching between two integer divide ratios is used, and one rapidly alternates between the two values in such a way that the time-average is equal to the desired non-integer divide ratio [13]. A block diagram for a fractional-*N* synthesizer is shown in Figure 11. Divide ratios of *N* and $N+1$ are used, where *N* is the first integer below the desired divide ratio, and $N+1$ is the first integer above. For example, if the desired divide ratio is 16.25, then one would alternate between the ratios of 16 and 17, with the ratio of 16 being used 75% of the time. Early attempts at fractional-*N* synthesis alternated between integer divide ratios in a repetitive manner, which resulted in noticeable spurs in the VCO output spectrum. More recently, $\Delta\Sigma$ modulators have been used to generate a random sequence with the desired duty cycle to control the multi-modulus dividers [21]. This has the effect of trading off the spurs for an increased noise floor, however the $\Delta\Sigma$ modulator can be designed so that most of the power in its output sequence is at frequencies that are above the loop bandwidth, and so are largely rejected by the loop.

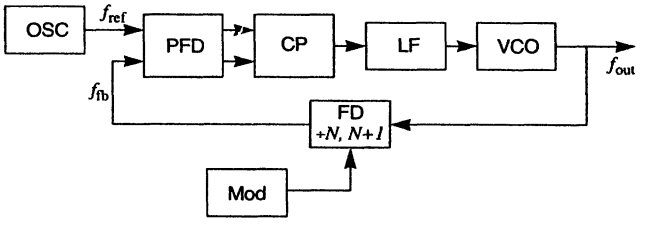


Fig. 11. The block diagram of a fractional-*N* frequency synthesizer.

The phase-domain small-signal model for the combination of a fractional-*N* divider and a $\Delta\Sigma$ modulator is given in Listing 8. It uses the *noise_table* function to construct a simple piece-wise linear approximation of the noise produced in an n^{th} order $\Delta\Sigma$ modulator that is parameterized with the low frequency noise generated by the modulator, along with the corner frequency and the order.

VIII. Jitter

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

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

$$v_n(t) = v(t + j(t)) \tag{34}$$

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

Listing 8 — Phase-domain fractional-N divider model.

```
`include "discipline.h"

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

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

$$\phi(t) = 2\pi f_o j(t), \tag{35}$$

where $f_o = 1/T$ and

$$v_n(t) = v\left(t + \frac{\phi(t)}{2\pi f_o}\right). \tag{36}$$

A. Jitter Metrics

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

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

$$J_{ee}(i) = \sqrt{\text{var}(t_i)}. \tag{37}$$

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

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

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

$$J_k(i) = \sqrt{\text{var}(t_{i+k} - t_i)}. \tag{38}$$

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

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

$$J_{cc}(i) = \sqrt{\text{var}(T_{i+1} - T_i)}. \tag{39}$$

Cycle-to-cycle jitter is like edge-to-edge jitter in that it is a scalar jitter metric that does not contain information about the correlation in the jitter between distant transitions. However, it differs in that it is a measure of short-term jitter that is relatively insensitive to long-term jitter [22]. As such, cycle-to-cycle jitter is the only jitter metric that is suitable for use when flicker noise is present. All other metrics are unbounded in the presence of flicker noise.

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

These jitter metrics are illustrated in Figure 12.

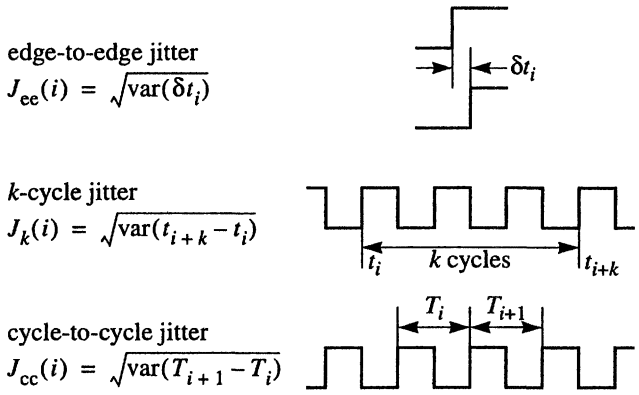


Fig. 12. The various jitter metrics.

B. Types of Jitter

The type of jitter produced in PLLs can be classified as being from one of two canonical forms. Blocks such as the PFD, CP, and FD are driven, meaning that a transition at their output is a direct result of a transition at their input. The jitter

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

exhibited by these blocks is referred to as *synchronous jitter*, it is a variation in the delay between when the input is received and the output is produced. Blocks such as the OSC and VCO are autonomous. They generate output transitions not as a result of transitions at their inputs, but rather as a result of the previous output transition. The jitter produced by these blocks is referred to as *accumulating jitter*, it is a variation in the delay between an output transition and the subsequent output transition. Table I previews the basic characteristics of these two types of jitter. The formulas for jitter given in this table are derived in the next two sections.

TABLE I: THE TWO CANONICAL FORMS OF JITTER.

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

IX. SYNCHRONOUS JITTER

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

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

$$j_{\text{sync}}(t) = \eta(t) \tag{40}$$

$$v_{\text{n}}(t) = v(t + j_{\text{sync}}(t)) \tag{41}$$

exhibits synchronous jitter. If η is further restricted to be a white Gaussian stationary or T-cyclostationary process, then $v_{\text{n}}(t)$ exhibits *simple synchronous jitter*. The essential characteristic of simple synchronous jitter is that the jitter in each event is independent or uncorrelated from the others, and (35) shows that it corresponds to white phase noise. Driven circuits exhibit simple synchronous jitter if they are broadband and if the noise sources are white, Gaussian and small. The sources are considered small if the circuit responds linearly to the noise, even though at the same time the circuit may be responding nonlinearly to the periodic drive signal.

For systems that exhibit simple synchronous jitter, from (37),

$$J_{\text{ee}}(i) = \sqrt{\text{var}(j_{\text{sync}}(t_i))}\,. \tag{42}$$

Similarly, from (38),

$$J_k(i) = \sqrt{\text{var}(t_{i+k} - t_i)}\,, \tag{43}$$

$$J_k(i) = \sqrt{\text{var}([(i+k)T + j_{\text{sync}}(t_{i+k})] - [iT + j_{\text{sync}}(t_i)])}\,, \tag{44}$$

$$J_k(i) = \sqrt{2\text{var}(j_{\text{sync}}(t_i))}\,. \tag{45}$$

$$J_k(i) = \sqrt{2} J_{\text{ee}}(i)\,. \tag{46}$$

Since $j_{\text{sync}}(t)$ is T-cyclostationary $j_{\text{sync}} = j_{\text{sync}}(t_i)$ is independent of i, and so is J_{ee} and J_k. The factor of $\sqrt{2}$ in (46) stems from the length of an interval including the independent variation from two transitions. From (46), J_k is independent of k, and so

$$J_k = J \text{ for } k = 1, 2, \ldots m\,. \tag{47}$$

Using similar arguments, one can show that with simple synchronous jitter,

$$J_{\text{cc}} = J\,, \tag{48}$$

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

A. Extracting Synchronous Jitter

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

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

The amount of displacement in time is determined by the amplitude of the noise signal, $n_v(t)$ and the slew rate of the periodic signal, $dv(t_c)/dt$, as the threshold is crossed, as shown in Figure 13 [23]. If the noise n_v is stationary, then

$$\text{var}(j_{\text{sync}}(t_c)) \cong \frac{\text{var}(n_v)}{[dv(t_c)/dt]^2} \tag{49}$$

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

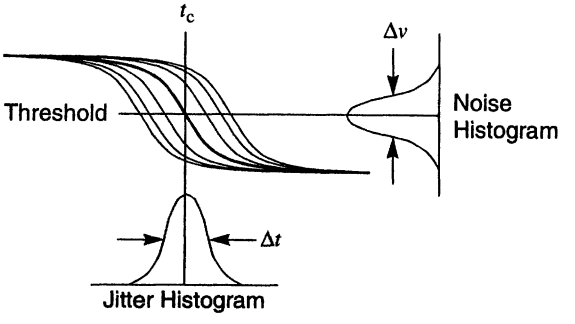


Fig. 13. How a threshold converts noise into jitter.

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

$$\text{var}(j_{\text{sync}}(t_c)) = \frac{\text{var}(n_v(t_c))}{[dv(t_c)/dt]^2}. \tag{50}$$

The jitter is computed from (42) using (49) or (50),

$$J_{ee} = \frac{\sqrt{\text{var}(n_v(t_c))}}{dv(t_c)/dt}. \tag{51}$$

To compute $\text{var}(n_v(t_c))$, one starts by driving the circuit with a representative periodic signal, and then sampling $v(t)$ at intervals of T to form the ergodic sequence $\{v(t_i)\}$ where $t_i = t_c$ for some i. Then the variance is computed by computing the power spectral density for the sequence by integrating from $f = -f_0/2$ to $f_0/2$. Recall that the noise is periodic in f with period $f_0 = 1/T$ because n is a discrete-time sequence with rate T.

In practice, this is done by using the strobed noise capability of SpectreRF† to compute the power spectral density of the sequence. When the strobed noise feature is active, the noise produced by the circuit is periodically sampled to create a discrete-time random sequence, as shown in Figure 9. SpectreRF then computes the power-spectral density of the sequence. The sample time should be adjusted to coincide with the desired threshold crossings. Since the T-periodic cyclostationary noise process is sampled every T seconds, the resulting noise process is stationary. Furthermore, the noise present at times other than at the sample points is completely ignored.

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

$$\text{var}(n_v(t_c)) = \int_{-f_0/2}^{f_0/2} S_{n_v}(f, t_c)df. \tag{52}$$

Then J_{ee} is computed from (51).

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

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

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

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

To extract the input-referred jitter of a PFD/CP, drive both inputs with periodic signals with offset phase so that the PFD/CP produces a representative output. Use SpectreRF's PNoise analysis to compute the output noise over the total bandwidth of the PFD/CP (in this case, use the conventional noise analysis rather than the strobed noise analysis). Choose the frequency range of the analysis so that the total noise at frequencies outside the range is negligible. Thus, the noise should be at least 40 dB down and dropping at the highest frequency simulated. Integrate the noise over frequency and apply Wiener-Khinchin Theorem [24] to determine

$$\text{var}(n) = \int_{-\infty}^{\infty} S_n(f)df, \tag{53}$$

the total output noise current squared [19]. Then either calculate or measure the effective gain of the PFD/CP, K_{det}. Scale the gain so that it has the units of amperes per second. Then divide the total output noise current by the gain and account for there being two transitions per cycle to distribute the noise over to determine the input-referred jitter for the PFD/CP,

$$J_{ee_{\text{PFD/CP}}} = \frac{T}{2\pi K_{\text{det}}}\sqrt{\frac{\text{var}(n)}{2}}. \tag{54}$$

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

X. Accumulating Jitter

Accumulating jitter is exhibited by autonomous systems, such as oscillators, that generate a stream of spontaneous output

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

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

$$j_{\mathrm{acc}}(t) = \int_0^t \eta(\tau)d\tau \tag{55}$$

$$v_{\mathrm{n}}(t) = v(t + j_{\mathrm{acc}}(t)) \tag{56}$$

exhibits accumulating jitter. While η is cyclostationary and so has bounded variance, (55) shows that the variance of j_{acc}, and hence the phase difference between $v(t)$ and $v_{\mathrm{n}}(t)$, is unbounded.

If η is further restricted to be a white Gaussian stationary or T-cyclostationary random process, then $v_{\mathrm{n}}(t)$ exhibits *simple accumulating jitter*. In this case, the process $\{j_{\mathrm{acc}}(iT)\}$ that results from sampling j_{acc} every T seconds is a discrete Wiener process and the phase difference between $v(iT)$ and $v_{\mathrm{n}}(iT)$ is a random walk [19]. As shown next, simple accumulating jitter corresponds to oscillator phase noise that results from white noise sources.

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

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

$$J_k = \sqrt{k}J \text{ for } k = 0, 1, 2, \ldots, \tag{57}$$

where

$$J = \sqrt{\mathrm{var}(j_{\mathrm{acc}}(t_i + T)) - \mathrm{var}(j_{\mathrm{acc}}(t_i))}\,. \tag{58}$$

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

Similarly,

$$J_{\mathrm{cc}} = \sqrt{2}J\,. \tag{59}$$

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

A. Extracting Accumulating Jitter

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

In order to determine the period jitter J of $v_{\mathrm{n}}(t)$ for a noisy oscillator, assume that it exhibits simple accumulating jitter so that η in (55) is a white Gaussian T-cyclostationary noise process (this excludes flicker noise) with a power spectral density of

$$S_\eta(f) = a\,, \tag{60}$$

and an autocorrelation function of

$$R_\eta(t_1, t_2) = a\delta(t_1 - t_2)\,, \tag{61}$$

where δ is a Kronecker delta function. Then

$$j_{\mathrm{acc}}(t) = \int_0^t \eta_T(\tau)d\tau \tag{62}$$

is a Wiener process [19], which has an autocorrelation function of

$$R_{j_{\mathrm{acc}}}(t_1, t_2) = a\min(t_1, t_2)\,. \tag{63}$$

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

$$J^2 = \mathrm{var}(j_{\mathrm{acc}}(t + T) - j_{\mathrm{acc}}(t))\,. \tag{64}$$

$$J^2 = \mathrm{E}[(j_{\mathrm{acc}}(t + T) - j_{\mathrm{acc}}(t))^2] \tag{65}$$

$$J^2 = \mathrm{E}[j_{\mathrm{acc}}(t + T)^2 - 2j_{\mathrm{acc}}(t + T)j_{\mathrm{acc}}(t) + j_{\mathrm{acc}}(t)^2] \tag{66}$$

$$J^2 = \mathrm{E}[j_{\mathrm{acc}}(t + T)^2] - 2\mathrm{E}[j_{\mathrm{acc}}(t + T)j_{\mathrm{acc}}(t)] + \mathrm{E}[j_{\mathrm{acc}}(t)^2] \tag{67}$$

$$J^2 = R_{j_{\mathrm{acc}}}(t + T, t + T) - 2R_{j_{\mathrm{acc}}}(t + T, t) + R_{j_{\mathrm{acc}}}(t, t) \tag{68}$$

$$J^2 = a(t + T) - 2at + at \tag{69}$$

$$J = \sqrt{aT} \tag{70}$$

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

$$\phi_{\mathrm{acc}}(t) = 2\pi f_{\mathrm{o}} j_{\mathrm{acc}}(t) = 2\pi f_{\mathrm{o}} \int_0^t \eta(\tau)d\tau\,, \tag{71}$$

where $f_{\mathrm{o}} = 1/T$. From (60) and (71) the PSD of ϕ_{acc} is

$$S_{\phi_{\mathrm{acc}}}(\Delta f) = a\frac{(2\pi f_{\mathrm{o}})^2}{(2\pi\Delta f)^2} = \frac{af_{\mathrm{o}}^2}{\Delta f^2}\,. \tag{72}$$

From (26)

$$\mathcal{L}(\Delta f) = \frac{1}{2} S_{\phi_{acc}}(\Delta f) = \frac{a f_o^2}{2\Delta f^2}, \tag{73}$$

$$a = 2\mathcal{L}(\Delta f)\frac{\Delta f^2}{f_o^2}. \tag{74}$$

Determine a by choosing Δf well above the corner frequency, f_{corner} to avoid ambiguity and well below f_o to avoid the noise from other sources that occur at these frequencies.

1) Example: To compute the jitter of an oscillator, an RF simulator such as SpectreRF is used to find $\mathcal{L}$ and f_o of the oscillator. Given these, a is found with (74), J is found with (70) and J_k is found with (57). This procedure is demonstrated for the oscillator shown in Figure 14. This is a very low noise oscillator designed in 0.35μ CMOS by of Rael and Abidi [25]. The frequency of oscillation is 1.1 GHz and the resonator has a loaded Q of 6.

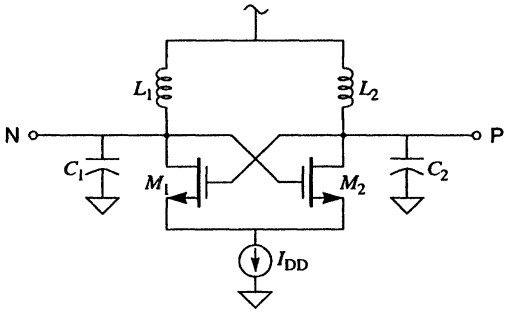


Fig. 14. Differential LC oscillator.

The procedure starts by using an RF simulator such as SpectreRF to compute the normalized phase noise $\mathcal{L}$. Its PNoise analysis is used, with the *maxsidebands* parameter set to at least 10 to adequately account for noise folding within the oscillator.† In this case, $\mathcal{L} = -110$ dBc at 100 kHz offset from the carrier. Apply (74) to compute a from $\mathcal{L}$, where $\mathcal{L}(\Delta f) = 10^{-11}$, $\Delta f = 100$ kHz, and $f_o = 1.1$ GHz,

$$a = 2 \cdot 10^{-11}\left(\frac{10^5}{1.1\times10^9}\right)^2 = 165.3\times10^{-21}. \tag{75}$$

The period jitter J is then computed from (70),

$$J = \sqrt{aT} = \sqrt{\frac{a}{f_o}} = \sqrt{\frac{165.3\times10^{-21}}{1.1\text{ GHz}}} = 12.3\text{ fs}. \tag{76}$$

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

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

XI. Jitter of a PLL

If a PLL synthesizer is constructed from blocks that exhibit simple synchronous and accumulating jitter, then the jitter behavior of the PLL is relatively easy to estimate [26]. Assume that the PLL has a closed-loop bandwidth of f_L, and that $\tau_L = 1/2\pi f_L$, then for k such that $kT \ll \tau_L$, jitter from the VCO dominates and the PLL exhibits simple accumulating jitter equal to that produced by the VCO. Similarly, at large k (low frequencies), the PLL exhibits simple accumulating jitter equal to that produced by the OSC. Between these two extremes, the PLL exhibits simple synchronous jitter. The amount of which depends on the characteristics of the loop and the level of synchronous jitter exhibited by the FDs and the PFD/CP. The behavior of such a PLL is shown in Figure 15.

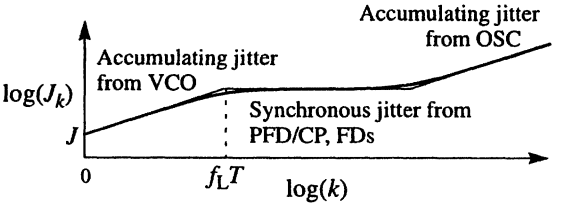


Fig. 15. Long-term jitter (J_k) for an idealized PLL as a function of the number of cycles.

XII. Modeling a PLL with Jitter

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 [27, 28]. Instead, only the techniques for adding jitter to the models are discussed.

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

A. Modeling Driven Blocks

A feature of Verilog-A allows especially simple modeling of synchronous jitter. The *transition*() function, which is used to model signal transitions between discrete levels, provides a delay argument that can be dithered on every transition. The delay argument must not be negative, so a fixed delay that is greater than the maximum expected deviation of the jitter must be included. This approach is suitable for any model that exhibits synchronous jitter and generates discrete-valued outputs. It is used in the Verilog-A divider module shown in Listing 9, which models synchronous jitter with (41) where j_{sync} is a stationary white discrete-time Gaussian random process. It is also used in Listing 10, which models a simple PFD/CP.

1) Frequency Divider Model: The model, given in Listing 9, operates by counting input transitions. This is done in the

Listing 9 — Frequency divider that models synchronous jitter.

```
`include "discipline.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);// edge-to-edge jitter
parameter real ttol=1p from (0:td/5);// ttol << jitter

integer count, n, seed;
real dt;

analog begin
    @(initial_step) seed = -311;

    @(cross(V(in) - (Vhi + Vlo)/2, dir, ttol)) begin
        // count input transitions
        count = count + 1;
        if (count >= ratio)
            count = 0;
        n = (2*count >= ratio);
        // add jitter
        dt = jitter*$dist_normal(seed,0,1);
    end

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

Listing 10 — PFD/CP model with synchronous jitter.

```
`include "discipline.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);// edge-to-edge jitter
parameter real ttol=1p from (0:td/5);// ttol << jitter

integer state, seed;
real dt;

analog begin
    @(initial_step) seed = 716;

    @(cross(V(ref), dir, ttol)) begin
        if (state > -1) state = state - 1;
        dt = jitter*$dist_normal(seed,0,1);
    end

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

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

@cross block. The cross function triggers the @ block at the precise moment when its first argument crosses zero in the direction specified by the second argument. Thus, the @ block is triggered when the input crosses the threshold in the user specified direction. The body of the @ block increments the count, resets it to zero when it reaches ratio, then determines if count is above or below its midpoint (n is zero if the count is below the midpoint). It also generates a new random dither dT that is used later. Outside the @ block is code that executes continuously. It processes n to create the output. The value of the ?: operator is Vhi if n is 1 and Vlo if n is 0. Finally, the *transition* function adds a finite transition time of tt and a delay of td + dt. The finite transition time removes the discontinuities from the signal that could cause problems for the simulator. The jitter is embodied in dt, which varies randomly from transition to transition. To avoid negative delays, td must always be larger than dt. This model expects jitter to be specified as J_{ee}, as computed with (51).

2) PFD/CP Model: The model for a phase/frequency detector combined with a charge pump is given in Listing 10. It implements a finite-state machine with a three-level output, -1, 0 and +1. On every transition of the VCO input in direction dir, the output is incremented. On every transition of the reference input in the direction dir, the output is decremented. If both the VCO and reference inputs are at the same frequency, then the average value of the output is proportional to the phase difference between the two, with the average being negative if the reference transition leads the VCO transition and positive otherwise [3]. As before, the time of the output transitions is randomly dithered by dt to model jitter. The output is modeled as an ideal current source and a finite transition time provides a simple model of the dead band in the CP.

B. Modeling Accumulating Jitter

1) OSC Model: The delay argument of the *transition*() function cannot be used to model accumulating jitter because of the accumulating nature of this type of jitter. When modeling a fixed frequency oscillator, the *timer*() function is used as shown in Listing 11. At every output transition, the next transition is scheduled using the *timer*() function to be $T/K + J\delta/\sqrt{K}$ in the future, where δ is a unit-variance zero-mean random process and K is the number of output transitions per period. Typically, $K = 2$.

C. VCO Model

A VCO generates a sine or square wave whose frequency is proportional to the input signal level. VCO models, given in

Listing 11 — Fixed frequency oscillator with accumulating jitter.

```
`include "discipline.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 jitter=0 from [0:0.1/freq);// period jitter

integer n, seed;
real next, dT;

analog begin
    @(initial_step) begin
        seed = 286;
        next = 0.5/freq + $abstime;
    end

    @(timer(next)) begin
        n = !n;
        dT = jitter*$dist_normal(seed,0,1);
        next = next + 0.5/freq + 0.707*dT;
    end

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

Listings 12 and 13, are constructed using three serial operations, as shown in Figure 16. First, the input signal is scaled to compute the desired output frequency. Then, the frequency is integrated to compute the output phase. Finally, the phase is used to generate the desired output signal. The phase is computed with *idtmod*, a function that provides integration followed by a modulus operation. This serves to keep the phase bounded, which prevents a loss of numerical precision that would otherwise occur when the phase became large after a long period of time. Output transitions are generated when the phase passes $-\pi/2$ and $\pi/2$.

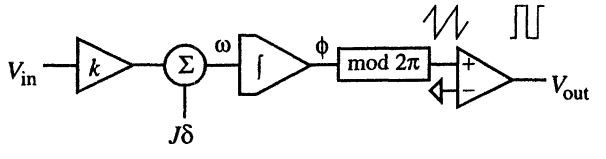


Fig. 16. Block diagram of VCO behavioral model that includes jitter.

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

$$\tau_i = \tau + \Delta\tau_i . \tag{77}$$

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

$$\mathrm{var}(\Delta\tau) = \frac{\mathrm{var}(T)}{K} = \frac{J^2}{K} . \tag{78}$$

Therefore,

$$\Delta\tau_i = \frac{J\delta_i}{\sqrt{K}} \tag{79}$$

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

$$f_i = \frac{1}{K}\left(\frac{1}{\tau + \Delta\tau_i}\right) = \frac{\frac{1}{K\tau}}{1 + \frac{\Delta\tau_i}{\tau}} = \frac{f_c}{1 + K\Delta\tau_i f_c} \tag{80}$$

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

$$f_i = \frac{f_c}{1 + \Delta T_i f_c} . \tag{81}$$

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

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

The final model given in Listing 12. This model can be easily modified to fit other needs. Converting it to a model that generates sine waves rather than square waves simply requires replacing the last two lines with one that computes and outputs the sine of the phase. When doing so, consider reducing the number of jitter updates to one per period, in which case the factor of 1.414 should be changed to 1.

Listing 13 is a Verilog-A model for a quadrature VCO that exhibits accumulating jitter. It is an example of how to model an oscillator with multiple outputs so that the jitter on the outputs is properly correlated.

D. Efficiency of the Models

Conceptually, a model that includes jitter should be just as efficient as one that does not because jitter does not increase the activity of the models, it only affects the timing of particular events. However, if jitter causes two events that would normally occur at the same time to be displaced so that they are no longer coincident, then a circuit simulator will have to use more time points to resolve the distinct events and so will run more slowly. For this reason, it is desirable to combine jitter sources to the degree possible.

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

1) Including Synchronous Jitter into OSC: One can combine the output-referred noise of FD_M and FD_N and the input-

Listing 12 — VCO model that includes accumulating 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);// period jitter
parameter real ttol=1u/Fmax from (0:1/Fmax);

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

analog begin
    @(initial_step) seed = -561;

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

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

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

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

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

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

referred noise of the PFD/CP with the output noise of OSC. A modified fixed-frequency oscillator model that supports two jitter parameters and the divide ratio M is given in Listing 14 (more on the effect of the divide ratio on jitter in the next section). The accJitter parameter is used to model the accumulating jitter of the reference oscillator, and the syncJitter parameter is used to model the synchronous jitter of FD_M, FD_N and PFD/CP. Synchronous jitter is modeled in the oscillator without using a nonzero delay in the transition function. This is a more efficient approach because it avoids generating two unnecessary events per period. To get full benefit from this optimization, a modified PFD/CP given in Listing 15 is used. This model runs more efficiently by removing support for jitter and the td parameter.

Listing 13 — Quadrature Differential VCO model that includes accumulating 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 Fmin=1 from (0:inf);
parameter real Fmax=2*Fmin from (Fmin:inf);
parameter real Vlo=-1, Vhi=1;
parameter real jitter=0 from [0:0.25/Fmax);// period jitter
parameter real ttol=1u/Fmax from (0:1/Fmax);
parameter real tt=0.01/Fmax;

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

analog begin
    @(initial_step) seed = 133;

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

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

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

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

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

    // generate the I and Q outputs
    V(PIout) <+ transition(i ? Vhi : Vlo, 0, tt);
    V(NIout) <+ transition(i ? Vlo : Vhi, 0, tt);
    V(PQout) <+ transition(q ? Vhi : Vlo, 0, tt);
    V(NQout) <+ transition(q ? Vlo : Vhi, 0, tt);
end
endmodule
```

2) Merging the VCO and FD_N: If the output of the VCO is not used to drive circuitry external to the synthesizer, if the divider exhibits simple synchronous jitter, and if the VCO exhibits simple accumulating jitter, then it is possible to include the frequency division aspect of the FD_N as part of the

Listing 14 — Fixed-frequency oscillator with accumulating and synchronous jitter.

```
`include "discipline.h"

module osc (out);

output out; electrical out;

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

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

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

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

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

VCO by simply adjusting the VCO gain and jitter. If the divide ratio of FD_N is large, the simulation runs much faster because the high VCO output frequency is never generated. The Verilog-A model for the merged VCO and FD_N is given in Listing 16. It also includes code for generating a logfile containing the length of each period. The logfile is used in Section XIII when determining S_{VCO}, the power spectral density of the phase of the VCO output.

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

$$J_{FD} = \sqrt{N} J_{VCO}. \tag{82}$$

Listing 15 — PFD/CP without jitter.

```
`include "discipline.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 ttol=1p from (0:inf);

integer state;

analog begin
    @(cross(V(ref), dir, ttol)) begin
        if (state > -1) state = state - 1;
    end
    @(cross(V(vco), dir, ttol)) begin
        if (state < 1) state = state + 1;
    end

    I(out) <+ transition(Iout * state, 0, tt);
end
endmodule
```

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

After simulation, it is necessary to refer the computed results, which are from the output of the divider, to the output of VCO, which is the true output of the PLL. The period jitter at the output of the VCO, J_{VCO}, can be computed with (82).

To determine the effect of the divider on $S_\phi(\omega)$, square both sides of (82) and apply (70)

$$a_{VCO} T_{VCO} = \frac{a_{FD} T_{FD}}{N}. \tag{83}$$

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

$$a_{VCO} = a_{FD} \tag{84}$$

From (72),

$$S_{VCO} \frac{f^2}{f_{VCO}^2} = S_{FD} \frac{f^2}{f_{FD}^2} \tag{85}$$

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

$$S_{VCO} = N^2 S_{FD}. \tag{86}$$

Once FD_N is incorporated into the VCO, the VCO output signal is no longer observable, however the characteristics of the VCO output are easily derived from (82) and (86), which are summarized in Table II.

It is interesting to note that while the frequency at the output of FD_N is N times smaller than at the output of the VCO, except for scaling in the amplitude, the spectrum of the noise close to the fundamental is to a first degree unaffected by the presence of FD_N. In particular, the width of the noise spec-

Listing 16 — VCO with FD_N.

```
`include "discipline.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 ratio=1 from (0:inf);
parameter real Vlo=-1, Vhi=1;
parameter real tt=0.01*ratio/Fmax from (0:inf);
parameter real jitter=0 from [0:0.25*ratio/Fmax);
                    // VCO period jitter
parameter real ttol=1u*ratio/Fmax from (0:ratio/Fmax);
parameter real outStart=inf from (1/Fmin:inf);

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

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

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

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

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

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

    // update jitter twice per period
    @(cross(phase - 0.25, +1, ttol)) begin
        dT = delta * $dist_normal(seed, 0, 1);
        Vout = Vhi;
    end
    @(cross(phase + 0.25, +1, ttol)) begin
        dT = delta * $dist_normal(seed, 0, 1);
        Vout = Vlo;
        if ($abstime >= outStart) $fstrobe( fp, "%0.10e",
                                        $abstime - prev);
        prev = $abstime;
    end
    V(out) <+ transition(Vout, 0, tt);
end
endmodule
```

trum is unaffected by FD_N. This is extremely fortuitous, because it means that the number of cycles we need to simulate is independent of the divide ratio N. Thus, large divide ratios do not affect the total simulation time.

TABLE II: CHARACTERISTICS OF VCO OUTPUT RELATIVE TO THE OUTPUT OF FD_N ASSUMING THE VCO EXHIBITS SIMPLE ACCUMULATING JITTER AND THE FD_N IS NOISE FREE.

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

To understand why FD_N does not affect the width of the noise spectrum, recall that while we started with a jitter that varied continuously with time, $j(t)$ in (34), for either efficiency or modeling reasons we eventually sampled it to end up with a discrete-time version. The act of sampling the jitter causes the spectrum of the jitter to be replicated at the multiples of the sampling frequency, which adds aliasing. This aliasing is visible, but not obvious, at high frequencies in Figure 18. However, especially with accumulating jitter, the phase noise amplitude at low frequencies is much larger than the aliased noise, and so the close-in noise spectrum is largely unaffected by the sampling. The effect of FD_N is to decimate the sampled jitter by a factor of N, which is equivalent to sampling the jitter signal, $j(t)$, at the original sample frequency divided by N. Thus, the replication is at a lower frequency, the amplitude is lower, and the aliasing is greater, but the spectrum is otherwise unaffected.

XIII. SIMULATION AND ANALYSIS

The synthesizer is simulated using the netlist from Listing 18 and the Verilog-A descriptions in Listings 14-16, modifying them as necessary to fit the actual circuit. The simulation should cover an interval long enough to allow accurate Fourier analysis at the lowest frequency of interest (F_{min}). With deterministic signals, it is sufficient to simulate for K cycles after the PLL settles if $F_{min} = 1/(TK)$. However, for these signals, which are stochastic, it is best to simulate for $10K$ to $100K$ cycles to allow for enough averaging to reduce the uncertainty in the result.

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

Unlike $\mathcal{L}(\Delta f)$, $S_\phi(\Delta f)$ can be computed efficiently. The Verilog-A code for the VCO/FD_N given in Listing 16 writes the length of each period to an output file named *periods.m*. Writing the periods to the file begins after an initial delay, specified using outStart, to allow the PLL to reach steady state. This file is then processed by Matlab from MathWorks using the script shown in Listing 17. This script computes $S_\phi(\Delta f)$,

the power spectral density of ϕ, using Welch's method [28]. The frequency range is from $f_{out}/2$ to $f_{out}/nfft$. The script computes $S_\phi(\Delta f)$ with a resolution bandwidth of rbw.† Normally, $S_\phi(\Delta f)$ is given with a unity resolution bandwidth. To compensate for a non-unity resolution bandwidth, broadband signals such as the noise should be divided by rbw. Signals with bandwidth less than rbw, such as the spurs generated by leakage in the CP, should not be scaled. The script processes the output of VCO/FD_N. The results of the script must be further processed using the equations in Table II to remove the effect of FD_N.

Listing 17 — Matlab script used for computing $S_\phi(\Delta f)$. These results must be further processed using Table II to map them to the output of the VCO.

```
% Process period data to compute Sφ(Δf)
echo off;
nfft=512;      % should be power of two
winLength=nfft;
overlap=nfft/2;
winNBW=1.5;    % Noise bandwidth given in bins

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

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

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

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

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

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

† The Hanning window used in the psd() function has a resolution bandwidth of 1.5 bins [29]. Assuming broadband signals, Matlab divides by 1.5 inside psd() to compensate. In order to resolve narrowband signals, the factor of 1.5 is removed by the script, and instead included in the reported resolution bandwidth.

XIV. Example

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

The noise of OSC is –95 dBc/Hz at 100 kHz. Applying (74) to compute a, where $\mathcal{L}(\Delta f) = 316 \times 10^{-12}$, Δf = 100 kHz, and f_o = 25 MHz, gives $a = 10^{-14}$. The period jitter J is then computed from (70), giving J = 20 ps.

The noise of VCO is –48 dBc/Hz at 100 kHz. Applying (74) and (70) with $\mathcal{L}(\Delta f) = 1.59 \times 10^{-5}$, Δf = 100 kHz, and f_o = 2 GHz, gives $a = 7.9 \times 10^{-14}$ and an period jitter of J = 6.3 ps.

The period jitter of the PFD/CP and FDs was found to be 2 ns. The FDs were included into the oscillators, which suppresses the high frequency signals at the input and output of the synthesizer. The netlist is shown in Listing 18. The results (compensated for non-unity resolution bandwidth (–28 dB) and for the suppression of the dividers (80 dB)) are shown in Figures 17-20. The simulation took 7.5 minutes for 450k time-points on a HP 9000/735. The use of a large number of time points was motivated by the desire to reduce the level of uncertainty in the results. The period jitter in the PLL was found to be 9.8 ps at the output of the VCO.

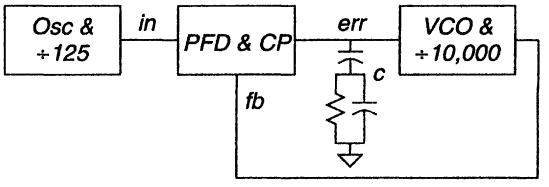
Listing 18 — Spectre netlist for PLL synthesizer.

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

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

Osc  (in)          osc       freq=25MHz ratio=125 \
                             accJitter=20ps syncJitter=2ns
PFD  (err in fb)   pfd_cp    Iout=500ua
C1   (err c)       capacitor c=3.125nF
R    (c  0)        resistor  r=10k
C2   (c  0)        capacitor c=625pF
VCO  (fb err)      vco       Fmin=1GHz Fmax=3GHz \
                             Vmin=-4 Vmax=4 ratio=10000 \
                             jitter=6ps outStart=10ms

JitterSim          tran      stop=60ms
```



The low-pass filter LF blocks all high frequency signals from reaching the VCO, so the noise of the phase lock loop at high frequencies is the same as the noise generated by the open-loop VCO alone. At low frequencies, the loop gain acts to stabilize the phase of the VCO, and the noise of the PLL is dom-

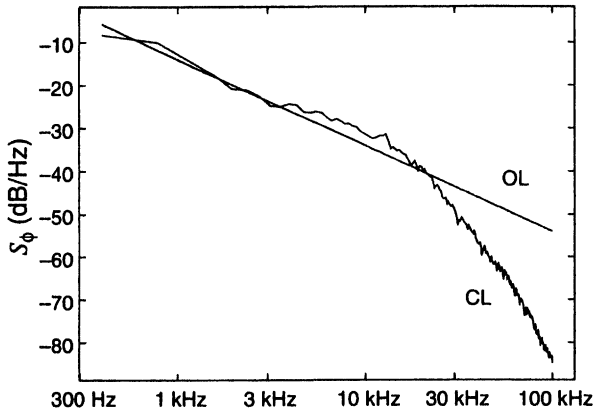


Fig. 17. Noise of the closed-loop PLL at the output of the VCO when only the reference oscillator exhibits jitter (CL) versus the noise of the reference oscillator mapped up to the VCO frequency when operated open loop (OL).

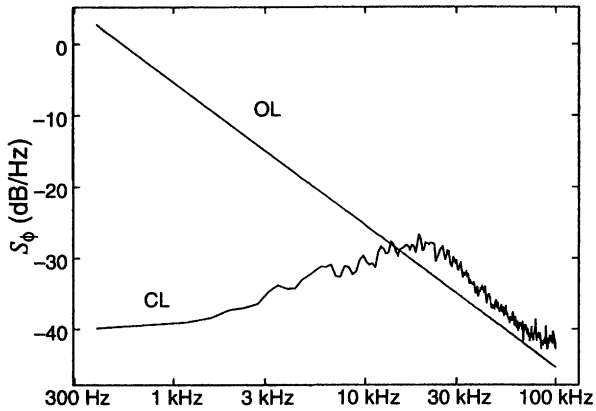


Fig. 18. Noise of the closed-loop PLL at the output of the VCO when only the VCO exhibits jitter (CL) versus the noise of the VCO when operated open loop (OL).

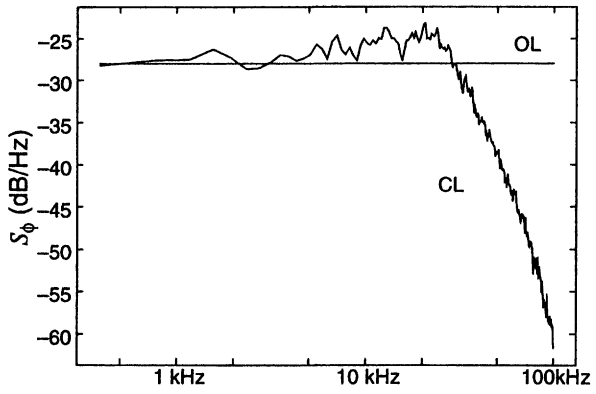


Fig. 19. Noise of the closed-loop PLL at the output of the VCO when only the PFD/CP, FD_M, and FD_N exhibit jitter (CL) versus the noise of these components mapped up to the VCO frequency when operated open loop (OL).

inated by the phase noise of the OSC. There is some contribution from the VCO, but it is diminished by the gain of

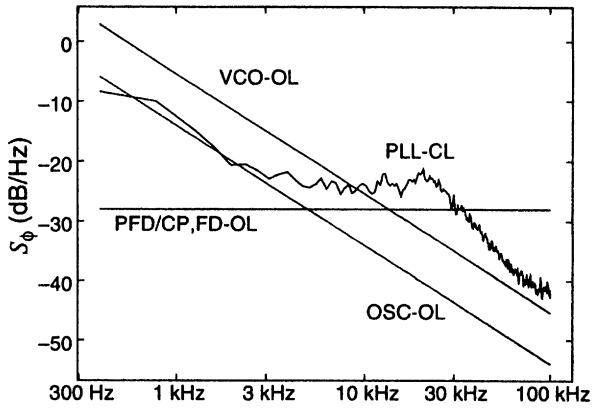


Fig. 20. Closed-loop PLL noise performance compared to the open-loop noise performance of the individual components that make up the PLL. The achieved noise is slightly larger than what is expected from the components due to peaking in the response of the PLL.

the loop. In this example, noise at the middle frequencies is dominated by the synchronous jitter generated by the PFD/CO and FDs. The measured results agree qualitatively with the expected results. The predicted noise is higher than one would expect solely from the open-loop behavior of each block because of peaking in the response of the PLL from 5 kHz to 50 kHz. For this reason, PLLs used in synthesizers where jitter is important are usually overdamped.

XV. Conclusion

A methodology for modeling and simulating the phase noise and jitter performance of phase-locked loops was presented. The simulation is done at the behavioral level, and so is efficient enough to be applied in a wide variety of applications. The behavioral models are calibrated from circuit-level noise simulations, and so the high-level simulations are accurate. Behavioral models were presented in the Verilog-A language, however these same ideas can be used to develop behavioral models in purely event-driven languages such as Verilog-HDL and Verilog-AMS. This methodology is flexible enough to be used in a broad range of applications where phase noise and jitter is important.

References

[1] Ken Kundert. "Introduction to RF simulation and its application." *Journal of Solid-State Circuits,* vol. 34, no. 9, September 1999.

[2] Cadence Design Systems. "SpectreRF simulation option." *www.cadence.com/datasheets/spectrerf.html.*

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

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

[5] 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.

[6] 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.

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

[8] A. Demir, A. Sangiovanni-Vincentelli. *Analysis and Simulation of Noise in Nonlinear Electronic Circuits and Systems*. Kluwer Academic Publishers, 1997.

[9] Ken Kundert. "Modeling and simulation of jitter in phase-locked loops." In *Analog Circuit Design: RF Analog-to-Digital Converters; Sensor and Actuator Interfaces; Low-Noise Oscillators, PLLs and Synthesizers*, Rudy J. van de Plassche, Johan H. Huijsing, Willy M.C. Sansen, Kluwer Academic Publishers, November 1997.

[10] Ken Kundert. "Modeling and simulation of jitter in PLL frequency synthesizers." Available from *www.designers-guide.com.*

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

[12] *Verilog-A Language Reference Manual: Analog Extensions to Verilog-HDL*, version 1.0. Open Verilog International, 1996. Available from *www.eda.org/verilog-ams.*

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

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

[15] A. Demir, A. Mehrotra, and J. Roychowdhury. "Phase noise in oscillators: a unifying theory and numerical methods for characterization." *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 5, May 2000, pp. 655 -674.

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

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

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

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

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

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

[22] Frank Herzel and Behzad Razavi. "A study of oscillator jitter due to supply and substrate noise." *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 46. no. 1, Jan. 1999, pp. 56-62.

[23] T. C. Weigandt, B. Kim, and P. R. Gray. "Jitter in ring oscillators." *1994 IEEE International Symposium on Circuits and Systems* (ISCAS-94), vol. 4, 1994, pp. 27-30.

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

[25] J. J. Rael and A. A. Abidi. "Physical processes of phase noise in differential LC oscillators." *Proceedings of the IEEE Custom Integrated Circuits Conference*, CICC 2000.

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

[27] 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.

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

[29] F. Harris. "On the use of windows for harmonic analysis with the discrete Fourier transform." *Proceedings of the IEEE*, vol. 66, no. 1, January 1978.