Application Note. PLL jitter measurements.

The procedures described in this application note are deliberately broad and generic. Your specific design might require procedures that are slightly different from those described here.

Purpose

This application note illustrates how to use the Spectre and SpectreRF simulators within the Analog Design Environment (ADE) to measure jitter characteristics of the PLL circuits.

Audience

Designers of PLL circuits or their blocks who are interested to use Cadence's tools for a noise and jitter performance verification.

Overview

Phase Lock Loops are widely used in electronics and communication for such timing related functions as clock generation, clock recovery, demodulation, and clock skew reduction. The undesired variation of the PLL operation due to internal and external noise sources are hard to predict and simulate. But it is essential to be able to predict and verify the timing performance of PLL in the presence of internally and externally generated jitter and phase noise.

We introduce the step by step methodology of jitter computation for the PLL using voltage-domain behavioral models for its blocks. The major steps of the process are the following:

i. SpectreRF simulation of the individual blocks to measure the jitter and operating parameters of the models.

ii. Creation or modification of the existing behavioral models of the blocks with the jitter.

iii. Time domain simulation of the original PLL using behavioral models of the blocks.

iv. Post processing of the simulation results to find the jitter and noise characteristics of the entire PLL.

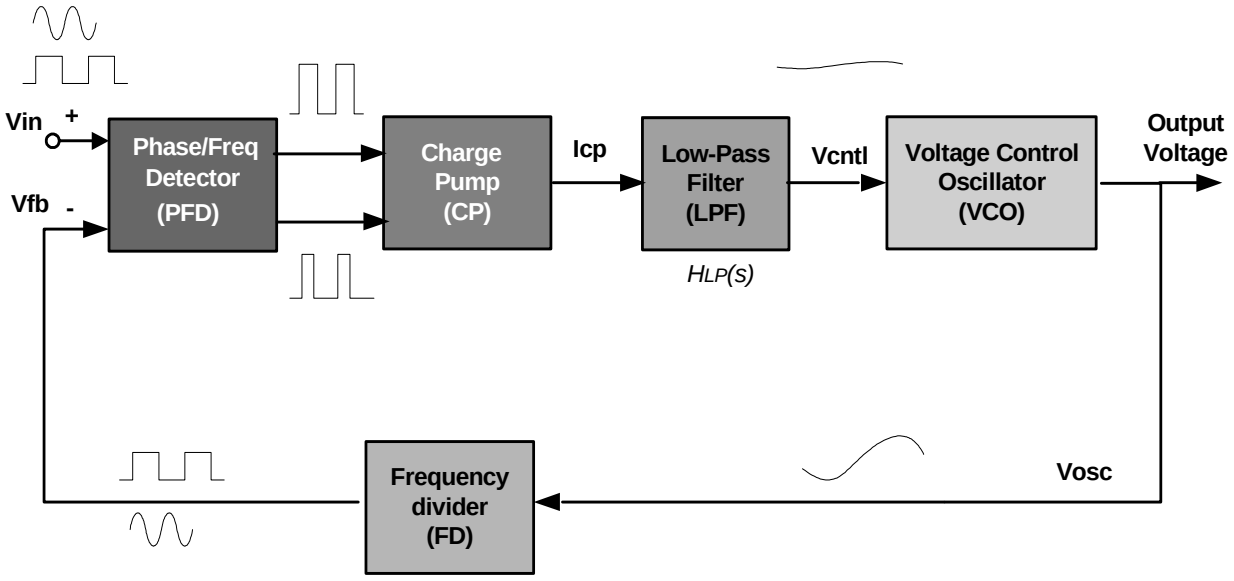
We assume that the user is familiar with the general PLL functionality and his/her particular design under investigation. This will be essential to properly setup the test benches for the simulations. It is also assumed that there are test benches for each of the block to run at least a simple transient simulation under typical operating conditions. User is also advised to familiarized her/himself with the basic ideas of the Jitter measurements under SpectreRF, for which we recommend [1]. The familiarity with the SpectreRF setup, Direct Plot and Calculator is also preferred.

PLL jitter and its contributors

Simple Charge Pump PLL

A typical charge pump PLL is shown at Figure 1. It consists of phase/frequency detector (PFD), charge pump (CP), loop filter (LPF), voltage controlled oscillator (VCO) and frequency dividers (FD), also called counters. The phase and frequency of VCO is forced to follow the input signal periodic signal using the negative feedback. The loop is locked when the frequency of both PFD inputs is the same and the phase skew is constant or zero.

Figure 1 The block diagram of a charge pump PLL.



In case of the frequency multiplier, the input is a signal from a reference oscillator (not shown) which could be passed through a frequency divider/pre-scaler (not shown) before it enters the PLL. There also could be several outputs taken from the VCO using post-scalers (not shown).

Phase noise and timing jitter

The operation of the phase lock loop is mostly affected by the phase noise. Except for the control voltage of the VCO, the main characteristic of the signals in the loop is a phase or a frequency.

The main sources of noise in PLL are input noise, phase/frequency detector and VCO. The transfer function of the loop typically has low-pass characteristic, therefore the input an phase detector noise will dominate the output at the low frequency range. Beyond the loop bandwidth, the VCO phase noise will pass to the output while the rest are attenuated.

There could be other significant noise contributors in the loop, but they often indicate an unusual design or the problems with existing design.

As we discussed in [1], the phase noise or a noise during a transition when a signal is sampled by a threshold will lead to a jitter. In communication system it will negatively impact the transmission quality. Jitter generation by the circuit, jitter transfer by the circuit and jitter tolerance by the locked PLL circuit are the common specification of PLL.

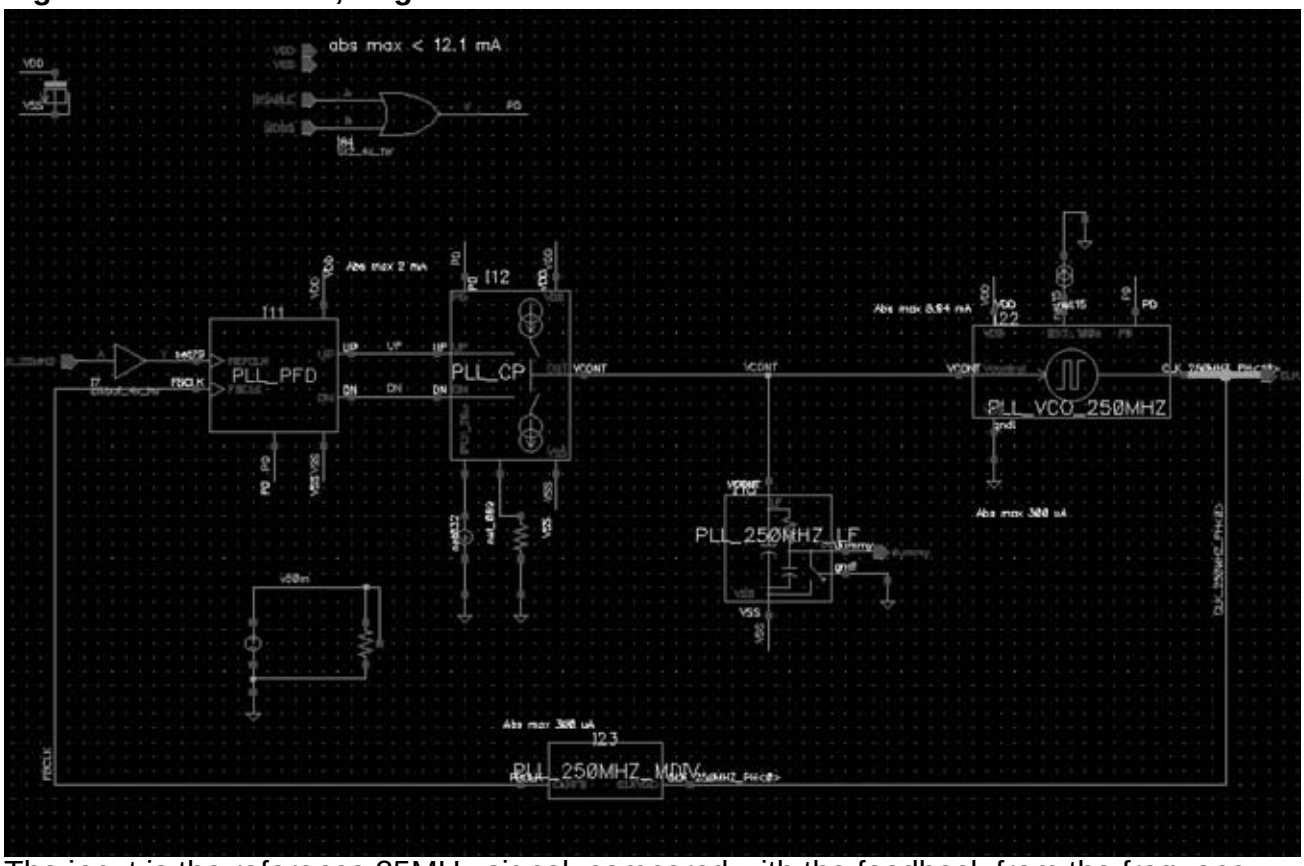
In the following example we will mainly concentrate on the jitter generation by the PLL itself*. We assume the noiseless reference input signal and we measure the jitter at the output of the circuit.

*To be augmented by a jitter transfer verification in the later revisions.

Design under investigation. 250MHz PLL circuit

The PLL was designed using 90nm PDK as a part of AMS methodology flow. The output frequency is 250MHz, with 25MHz reference clock input. The top level schematic of the original PLL is shown at Figure 2. We simplified an original testbench used in the demo of AMS flow. For this Application Note we removed unneeded nose sources and complex network of bond pads on the testbench, the same for the frequency detection instrumental block. Inside PLL, we also reduced the loop filter by removing extra structures to simplify its start and reset.

Figure 2 250MHz PLL, original schematic with reduced LPF.



The input is the reference 25MHz signal, compared with the feedback from the frequency divider, both are entering PFD. The RFD sends two sets of pulses using nodes UP and DN, to speed up and slow down the VCO. Charge pump, receiving those pulses, generates a current pulse, whose sign and duration will depend on the mismatch at the input of the PFD. The current will be transformed into a smooth voltage wave at the VCONT node by the loop filter. VCO will respond to the control and its output, which is expected to stabilize around 250MHz is passed through the frequency divider with the ratio of 10. The outputs from the ring VCO are taken at 10 different phase offsets from the output that is used in the loop for the feedback. The PD signal is used to reset the circuit functional blocks at the beginning of the simulation.

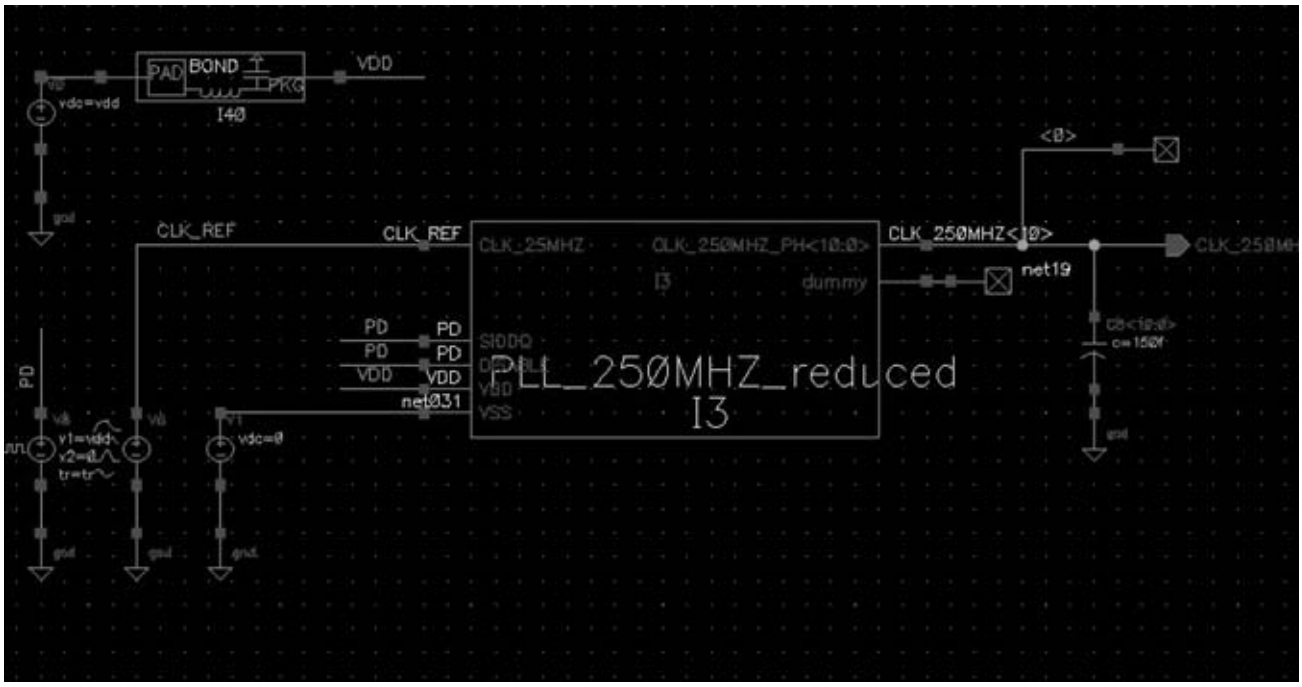
Top level testbench to find “locked” operating conditions

Testbench

The PLL circuits described above is setup for timedomain transient simulation in Spectre. Two DC voltage sources are used for VDD (power supply) and VSS (ground) signals. Current example doesn’t not include an actual reference oscillator with a pre-scaler counter. A voltage

source *V6* produces a pulse with the final rise and fall slopes. It represents the reference input signal with constant frequency, 25MHZ, and amplitude, 2.5V. Another voltage source with a single pulse at the beginning of the simulation will supply with the reset *PD* signal. After a preset delay time, 1ns, it drops from initial VDD voltage to a ground value. To make it constant afterwards, the period of the pulse is setup to be much longer than the expected simulation time in transient analysis. All delays, slopes and periods of the waves are parametrized. The typical value that are used in this circuit are 2.5V for a VDD power supply, 100ps. transition from high to low and back at the reference signal (clock) input. The output of the VCO is loaded with a capacitor at each of its 11 shifted in phase taps, see Figure 3. Only one of this outputs is used for the feedback loop.

Figure 3 PLL testbench for the top level transient simulation.



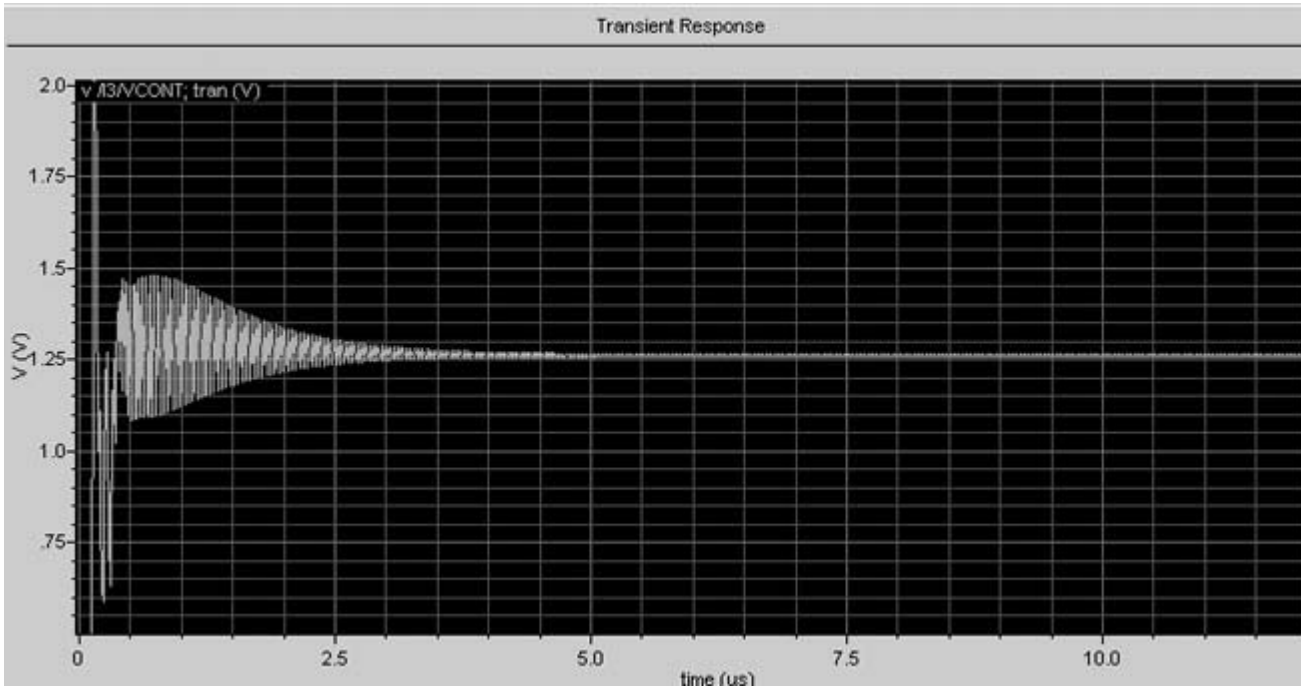
Transient analysis

Setup a *tran* analysis to run for 12us, using *moderate* accuracy settings. Use Virtuoso ADE “Outputs” menu to save the voltages on the nodes interconnecting PLL blocks and also the currents at the terminals connected to the *Vcont* node. Select them directly from schematic by descending inside the PLL circuit.

Transient simulation results

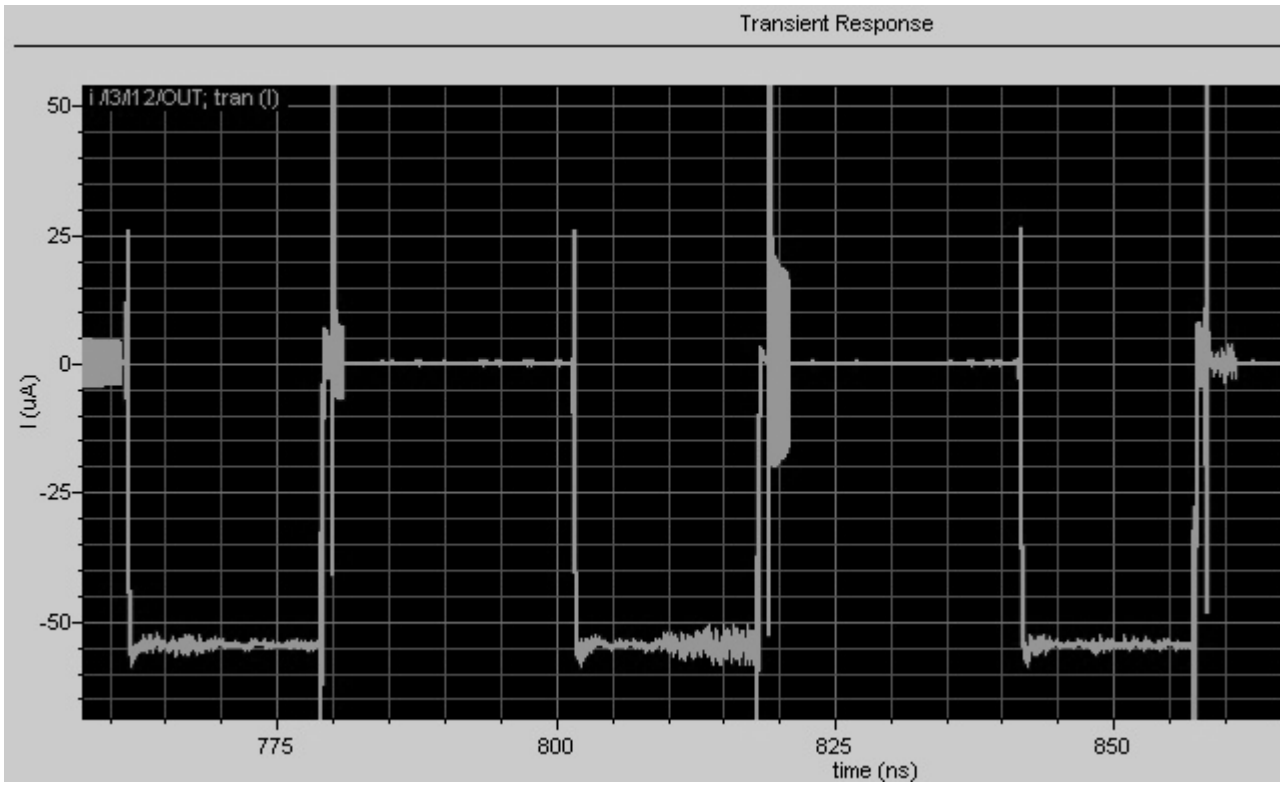
Display results using Direct Plot. Plot control voltage to confirm the locking of the VCO, seeFigure 4. The way to confirm that the frequency is stabilized would be to plot the output of the VCO. Load the wave into calculator. Use “cross” function with “1.25V” (half VDD) threshold, plotting versus “cycles”, “rising” edge, “multiple” occurrences. Then on the resulting curve, use the “deriv” function. The final results will be the period of the VCO output versus cycle. It can be seen that the value is stable at 4ns.

Figure 4 VCO control voltage. TRAN analysis.



Also plot the current at the output terminal of the Charge Pump. At the beginning, before PLL is settled, it is easy to “read” the value of the output current which will be used in the behavioral model. In out case it is -55uA, see Figure 5. It could be also confirmed later, while simulating the block alone.

Figure 5 Charge Pump output current. TRAN analysis.



Then plot the inputs of the PFD and zoom in to see the last couple of transitions. Estimate and record the input slopes of the signals coming into PFD. The simple visual estimate at the rising edges gives us 100ps. The same observation for the input of the FD in the feedback produces 150ps.

PLL blocks. Jitter parameters measurements

Split the circuit into the blocks, preserving all the sources connected to each block. In addition, estimate the output load for each block, or use the first device from the next block to represent the actual loading effects.

VCO

VCO testbench

The testbench, Figure 6, has the same DC sources as the original PLL schematic. In addition, the input control voltage is supplies by a parametrized DC source *V3*, whose value id fixed to "*vcntl*". The load is the first inverting stage from the divider and possibly the PLL output load.

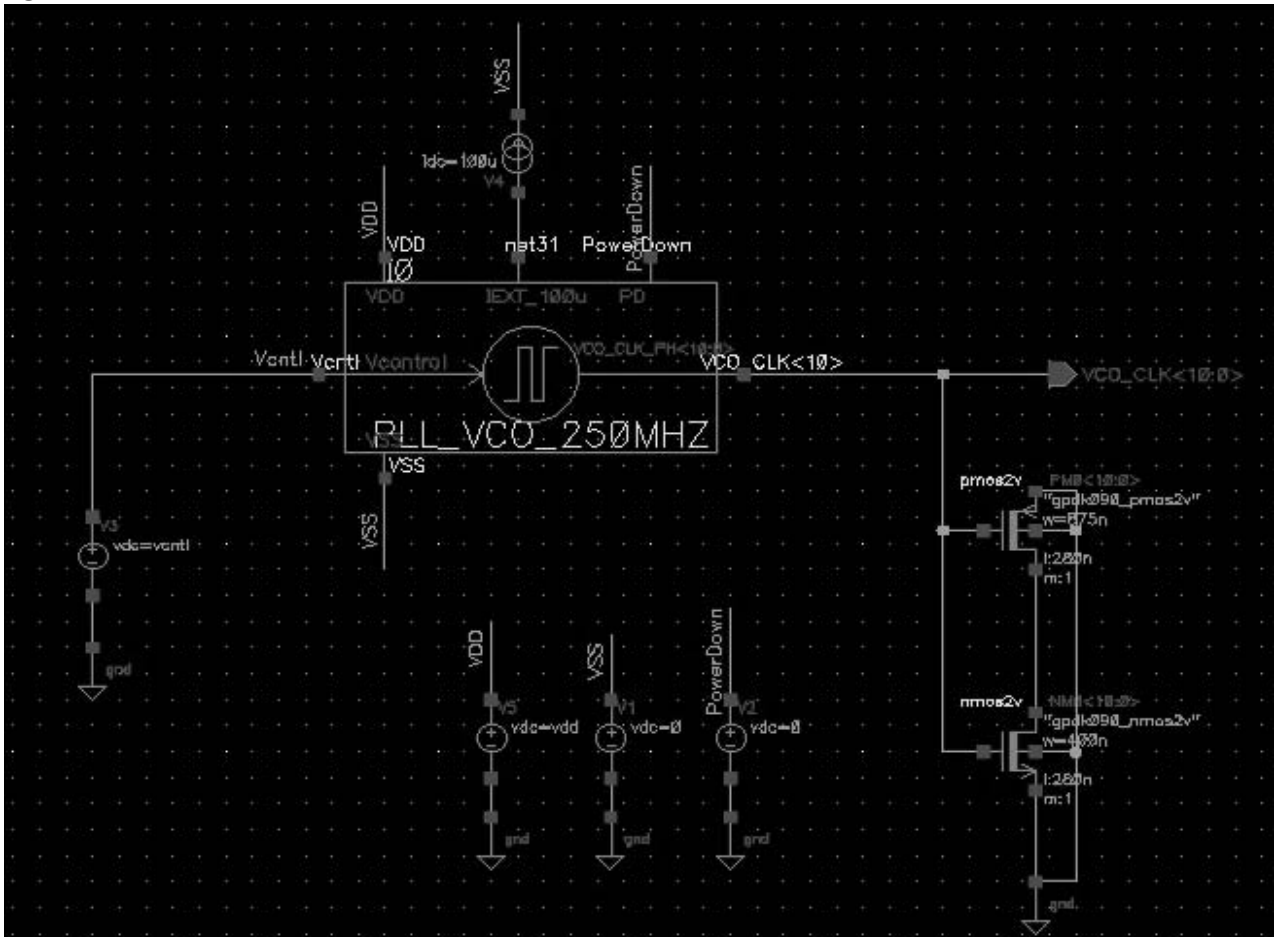
First, we find the operating condition that will provide the desirable output frequency of 250MHz. For more details, see Application Note for VCO [5].

Setup PSS analysis in ADE. Estimate the frequency, lets say 100M, better underestimate. Use *moderate* accuracy settings. For *tstab* we use several estimated periods,

Initial conditions are used to start the oscillations for this ring oscillator. Open "*Simulation*" menu and in "*convergence aids*" set up "*initial conditions"* to be 2.5V (high) on the odd or even nodes between the stages of the ring oscillator. In PSS analysis "*Options"* check "*nodes*" for "*initial condition parameters*".

We sweep the control voltage. Use *Sweep* option on the PSS form. Setup *Variable Name* to "*vcntl"* and set it in a reasonable range of 0.1V to 2.4V. Setup *linear* sweep with 10 or more steps. Run simulation.

Figure 6 VCO testbench schematic.



Use Direct Plot to plot the tuning curve, Select *Harmonic frequency* and choose the first

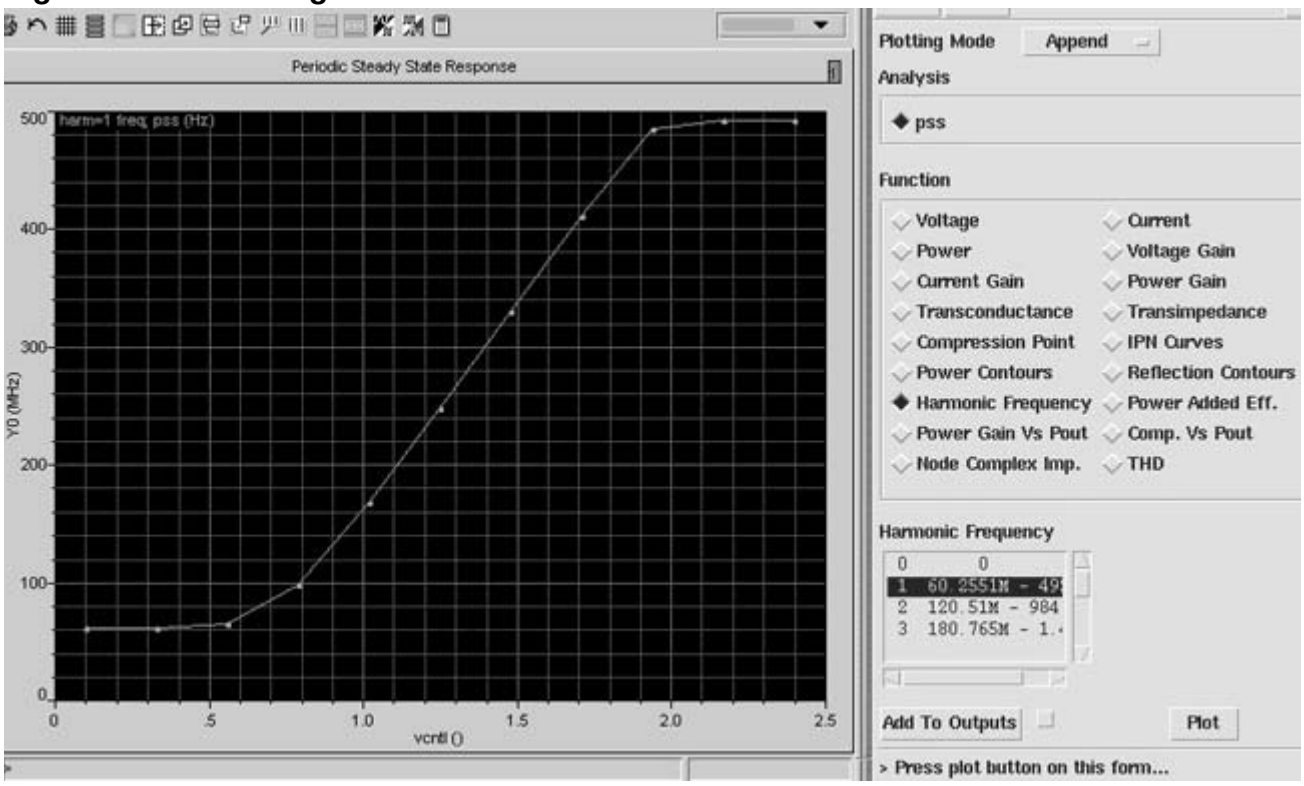
harmonic, see Figure 7.

Load the curve in calculator and find *cross* with 250MHz. The value is 1.255V which is close to the stable value during the transient simulation of the original PLL.

From the same curve, we can estimate the additional parameters needed for the behavioral model. They represent the linear tuning range:

Fmin = 97.65MHz; Fmax = 485MHz; Vmin = 0.79V; Vmax = 1.94V; Kvco = 420MHz/V.

Figure 7 VCO Tuning Curve.



VCO Phase Noise and Jitter.

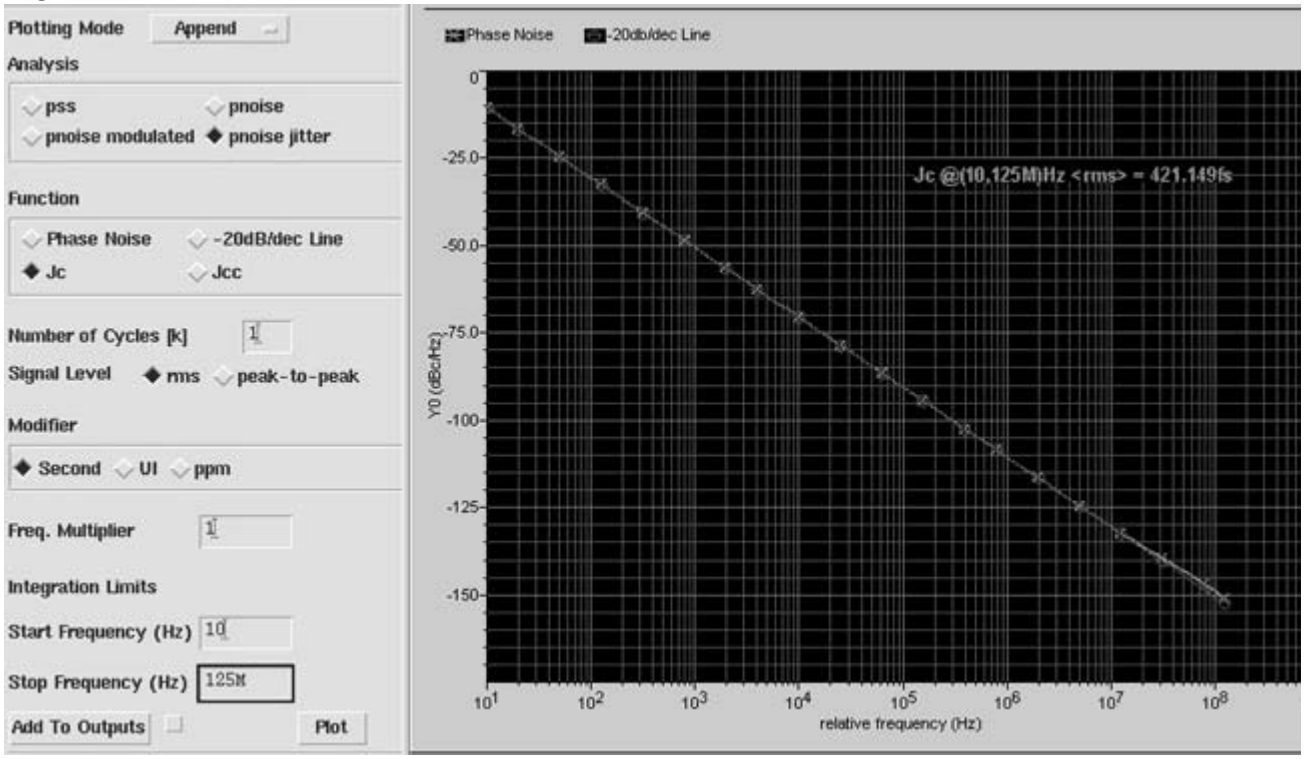
Use PSS setup but disable the sweep option. Fix the value of "*vcntl*" parameter to 1.255V. Also adjust the *Beat Frequency* estimate to 250MHz.

Select PNOISE analysis. We are interested in the noise near first harmonic at the output, which contribute to the jitter through the phase noise component. Set *relative* sweep near 1 harmonic. Specify the sweep limits* from 10 to 125MHz with 10 points per decade to get a

good smooth curve for integrating the jitter later on. Increase number of *Sidebands* to 100.** Specify the voltage output at the node which is used for the feedback. Input source is not important in this case. Enable "*jitter" Noise Type.* Run simulation.

Open Direct Plot and plot the *Phase Noise* from "pnoise jitter" analysis section of it. Then, to see which type of noise is dominant in this circuit, one might add the -20dB/dec line, using the same Direct Plot form. Select a frequency where the graph is seems to be straight. In this circuit, it is pretty much over entire frequency range up to 100MHz. Type 1000 in *Frequency* selector and push *Plot.* The curve will lay exactly on top of the most of the plot. This confirm that there is no significant flicker noise in this VCO in the given range. The graph picks up in the high frequency range, where the white or flicker PM noise is dominant.

Figure 8 VCO Phase Noise and Jitter.



Plot the period (i.e. cycle) jitter by using *Jc* button. Use *rms* type in seconds and select an entire frequency range of 10 to 125MHz. Do not modify a *Frequency Multiplier* - it is only used when the signal frequency is not the same as the PSS analysis' fundamental, which would happen when there is a frequency division or multiplication in the circuit. The final result is close to the estimate that we get using the simple white jitter Equation 8 on page 33(see also [2]):

(1) $f^{sample} = 100KHz; f_c = 250MHz; S_{\phi_{FM}}(f^{sample}) = -90.64(dBc/Hz)$

(2) $J_c(kT_c) = \sqrt{\frac{1}{2} S_{\phi_{FM}}(f^{sample}) \cdot \frac{f^{sample^2}}{f_c^3}} = 528\text{fs}$

The difference is due to the assumption about an infinite bandwidth during the analytical integration in Equation 6.

*> The limits will be affected by several factors: the bandwidth of the circuit or the system requirements for the PLL are the obvious ones. There are several assumptions for the jitter calculation from the phase noise, which assume that one will not include the noise too far from the modulated carrier frequency. We will exclude the noise outside the f_osc/2 region where it begins to change up and down around other harmonics.

** >This number might need an adjustment later on, to see if we included enough noise frequency to cover the bandwidth of the circuit. Typically, one will increase the value until the noise increase is significant to the desired accuracy. Then, one might reduce the number of sidebands if the performance is important while the drop in accuracy could be tolerated. In the current design, and increase to 200 sideband did not produce any significant change in the results. There seems to be that even the much lesser number will work just fine.

LPF, transfer function and jitter.

Before computing the jitter for PFD/CP, we need to determine a noise transfer function of the filter to know how it will affect the noise or a jitter generated by the charge pump. In addition, we can confirm if the noise from the filter could be neglected in the process of modelling. Since we eventually will run the transient analysis and not Noise or PNoise, the noise generated by components of the LPF, if any, will have to be included explicitly into one of the surrounding device models.

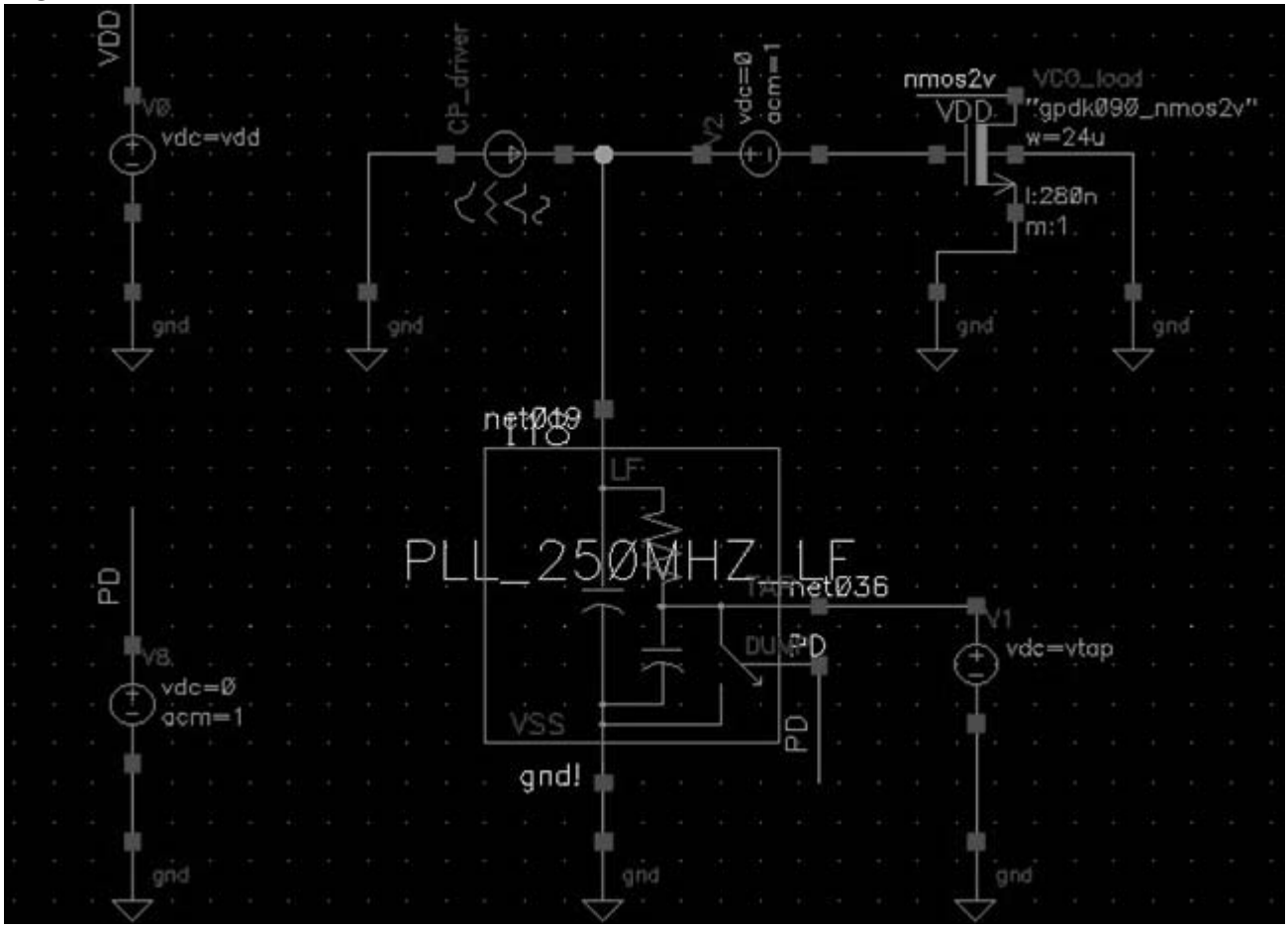
Setup PSS simulation using the simple testbench shown at Figure 9. The load is the first stage of the VCO. Use a zero DC voltage source as a current probe. The bias and power supply are two DC voltage sources. The input signal is a piece wise linear current source with a 100ps rise and fall slope and a short flat regions of 20ps in the maximum (+55uA) and minimum (-55uA). The PSS fundamental frequency is set to the reference frequency. Use *moderate* accuracy settings.

Setup PXF analysis: wide frequency sweep from 10Hz to 10GHz, *absolute* type. The *output* is the *voltage* across the load. We are interested in the baseband behavior, so the 0 *maximum sideband* will be enough here. (If noise is not important, one can use XF analyses for such a measurements.)

Add PNoise analysis. The same sweep as in PXF but 40 *maximum sidebands*, which is not expensive here. The output is a *voltage* across the load. The Noise Type is set to *sources*.

Specify the *Input Source* on PNoise setup form. Then select the input current source that we use as a signal on the testbench. Set the *Reference sideband* to “0”. *Run simulation.

Figure 9 LPF. Testbench schematic.



Use Direct Plot for PXF analysis to plot the frequency response (transfer function). Plot the spectrum of the sideband “*0”.* We can measure the 40 dB drop at about 680 MHz, see Figure 10. Store this number as “*f*max” to use for PFD/CP. Plot the output noise using PNoise Direct Plot form.

We can get both transfer function and noise in a single PNoise analysis. From PNoise form, select and plot the Transfer *Function*. Change the Y-axis to the log format. That will have the same result as PXF spectrum for “*0*” sideband. In addition, to add LPF noise into the PDF/CP model, we use “*Input Noise”* option for PNoise and integrate it up to “*f*max”. See Figure 11.

Figure 10 LPF. Output Noise and Transfer function.

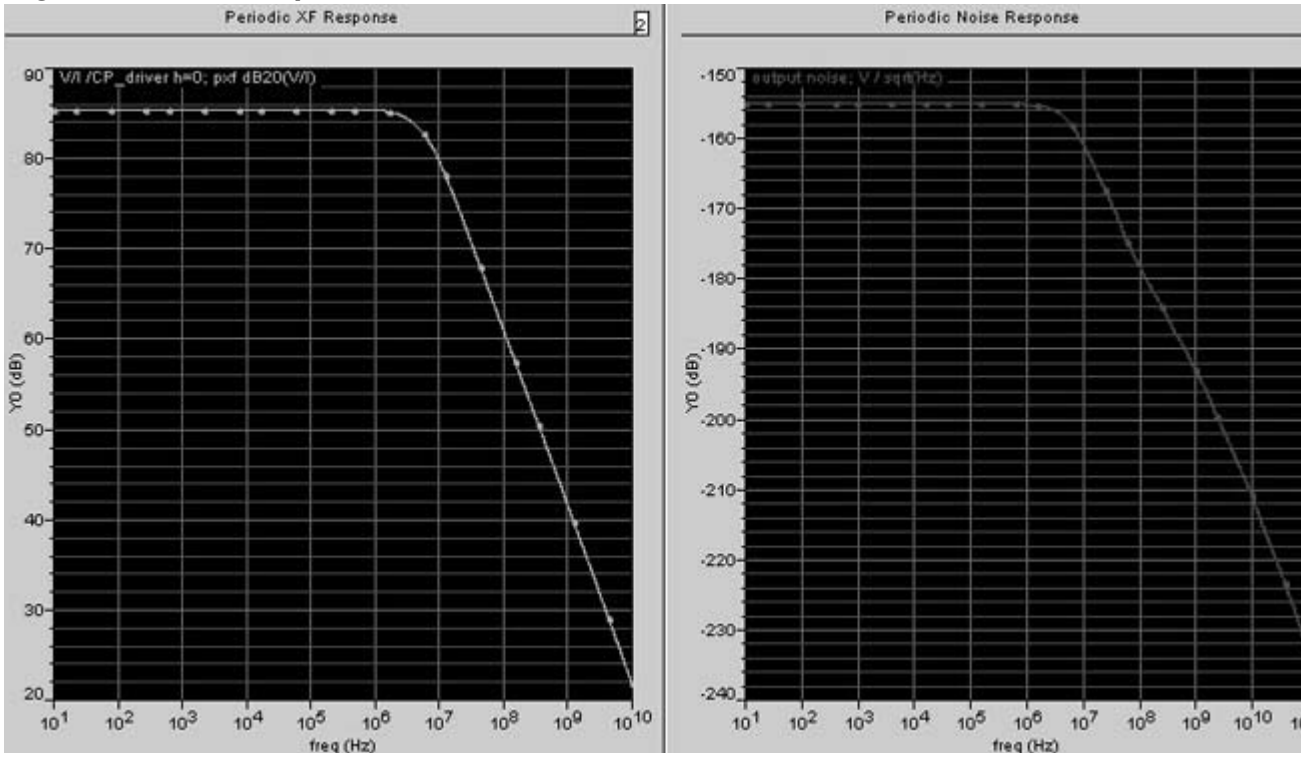
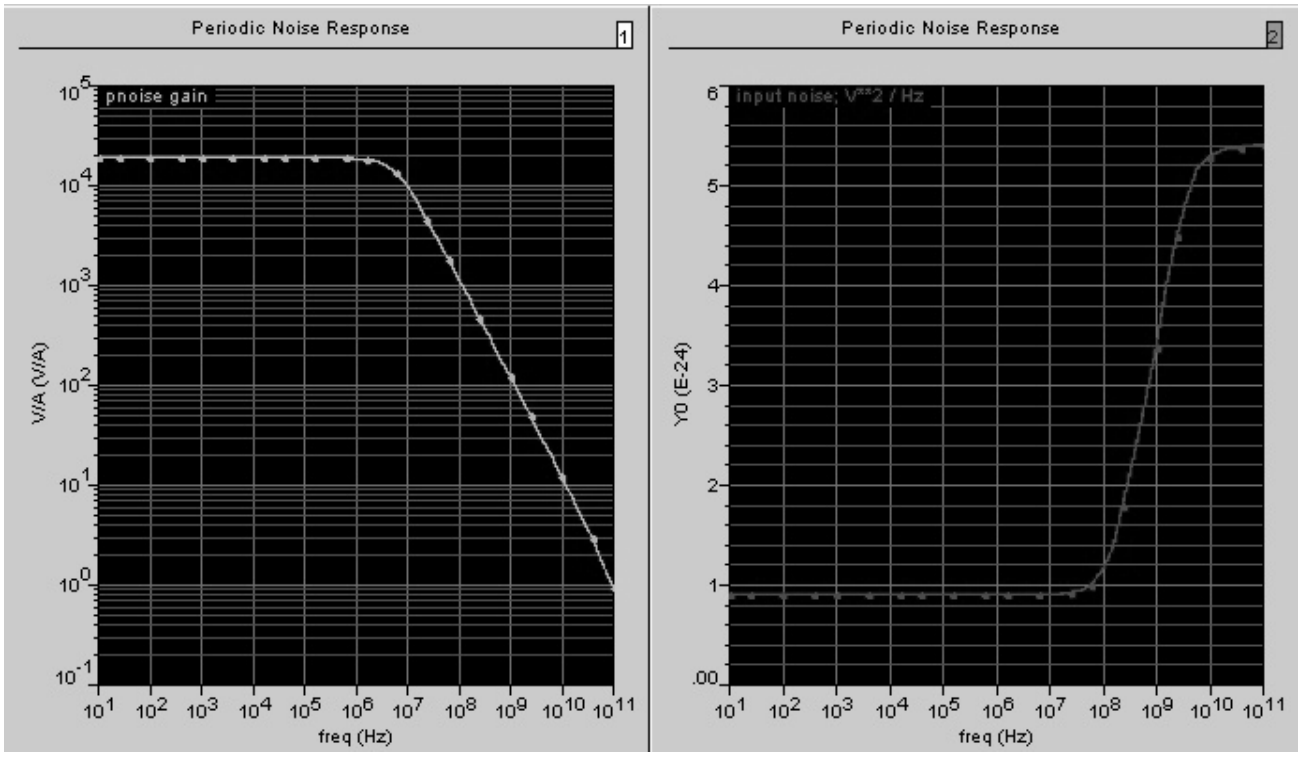


Figure 11 LPF. Noise Gain and Input Referred Noise.



PFD

PFD/CP testbench

The testbench, Figure 12, has two input voltage sources that represent the reference and feedback signals coming into the PFD. Set them to be *pulse* type. Parametrize the periods, rise and fall times, the delay time and pulse width. Use 100ps for transition, and reference signal (25MHz) period for both inputs. The delay is set the same - 5ns - in one simulation. In the second simulation, we introduce the mismatch of 2ns. in the delay. Power supply and bias are DC sources - VDD, VCC, PD(set as *dc=0*), same as in the original PLL design. The load is a voltage source that is biasing the output consistent with the operating conditions of 250MHz VCO - control voltage of 1.245V.

Setup PSS analysis. 25MHz "*Beat Frequency*". *Moderate* accuracy settings, small *tstab=0.4u*. Setup PNoise analysis with wide frequency range from 10Hz to 1GHz. A few points per decade is enough, more could be added if needed. Specify output load as a "*probe"* for the *Output*. Use "*sources"* type for the simulation. Run simulation.

Figure 12 PFD/CP testbench schematic.

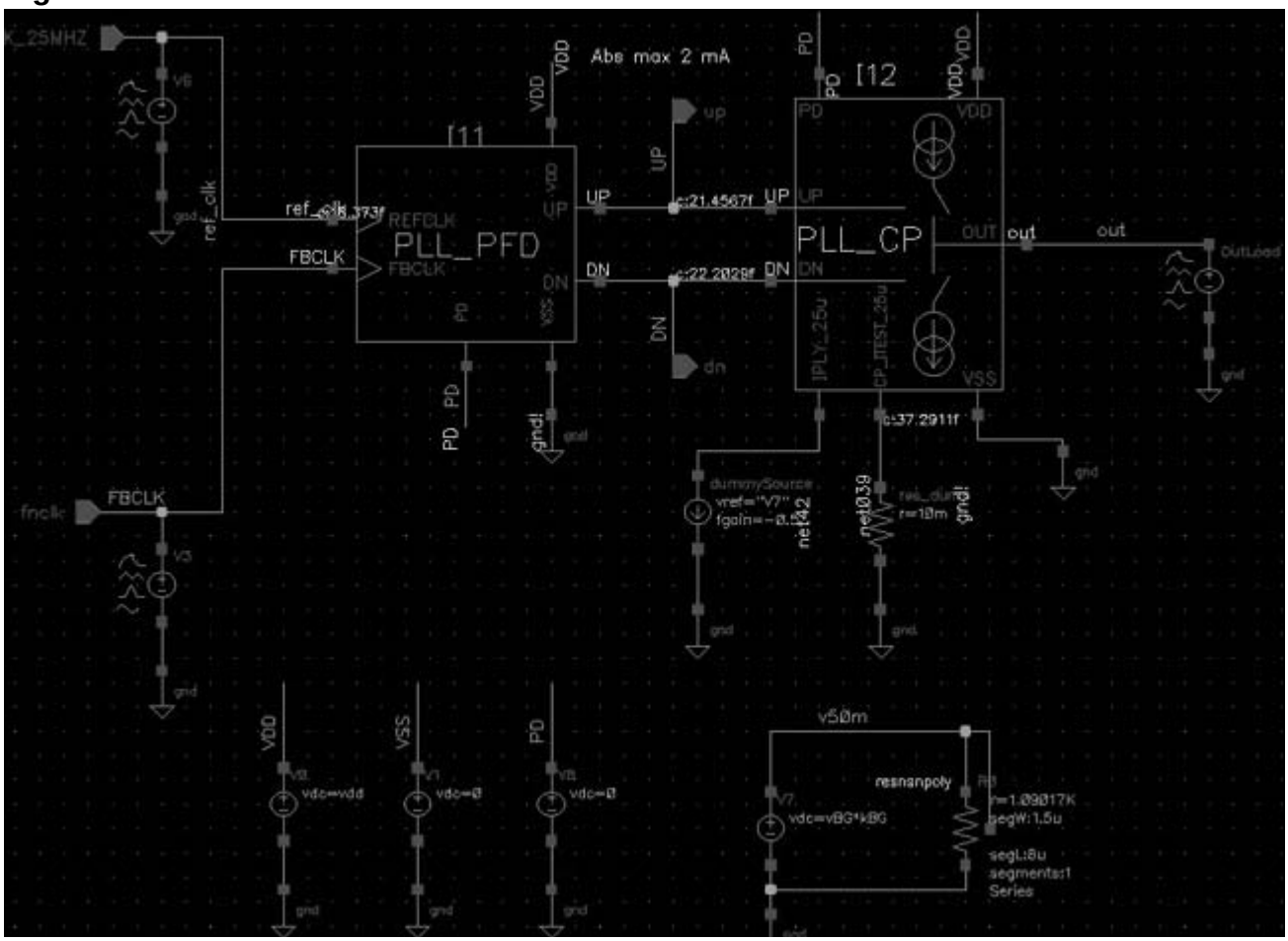
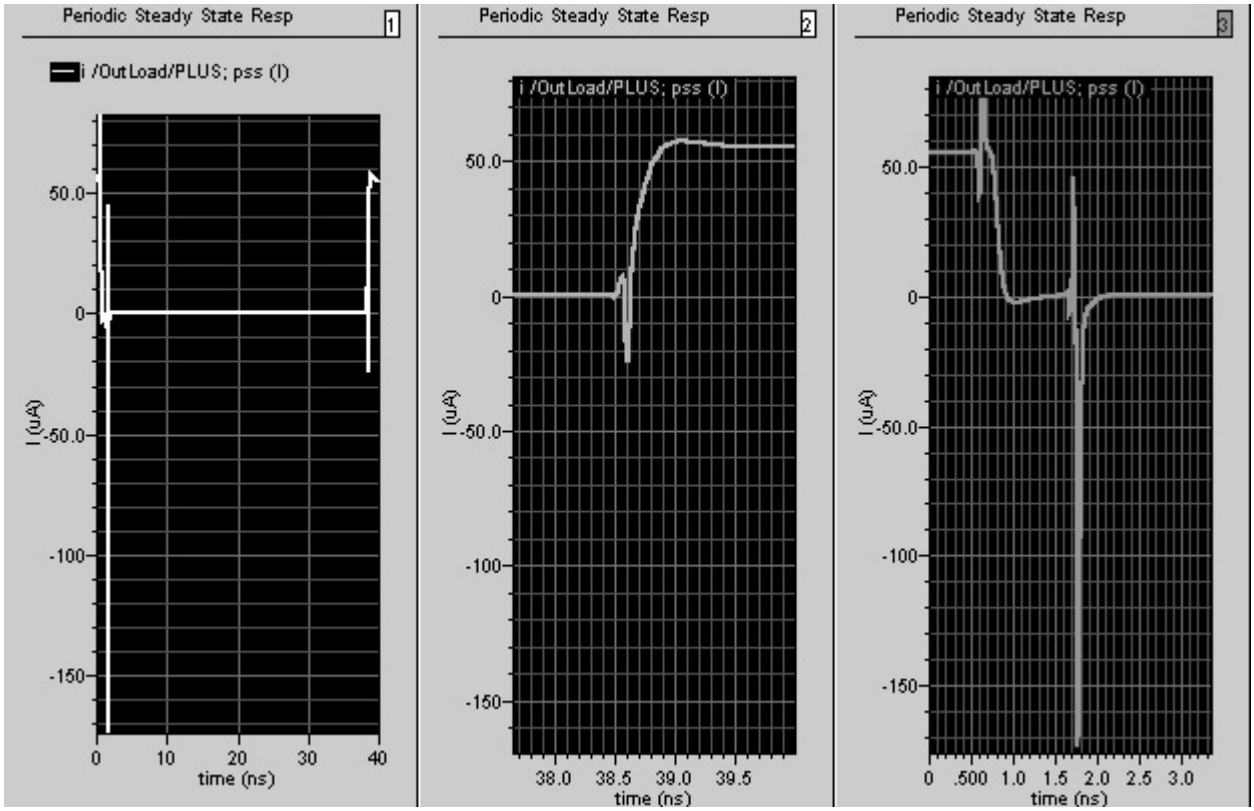


Figure 13 PFD/CP Output Current, 2ns inputs mismatch.



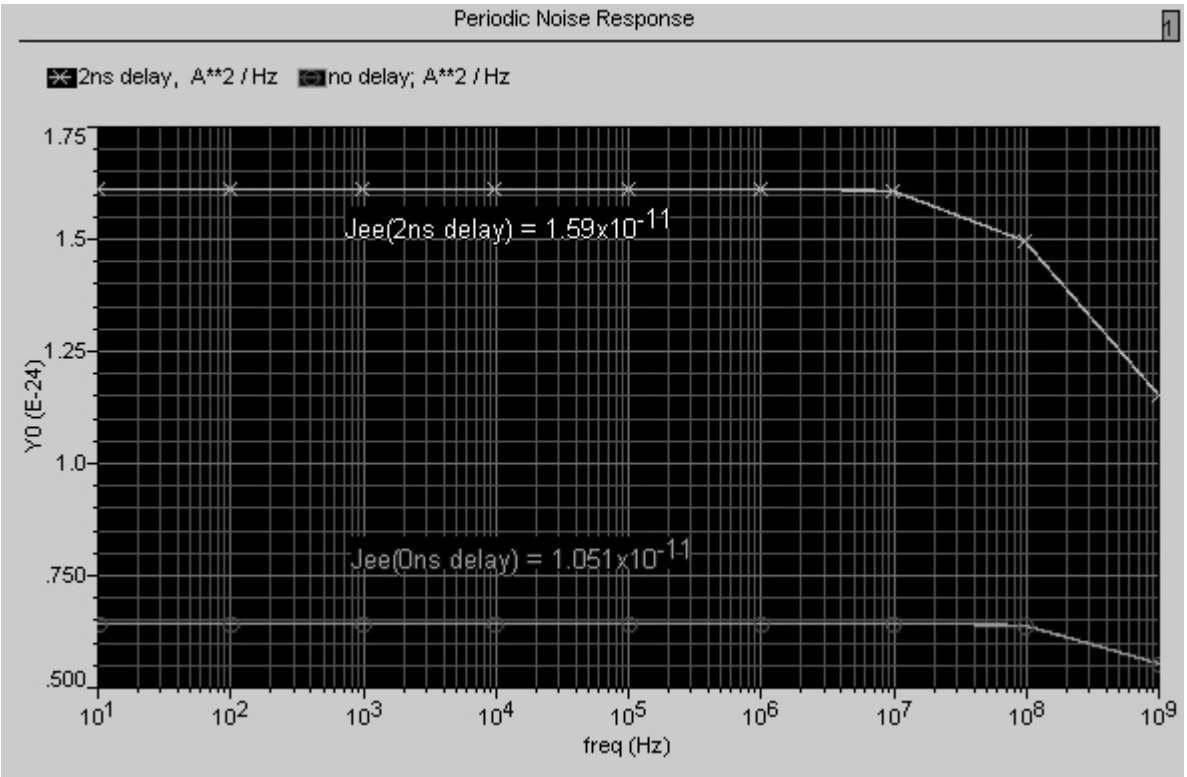
Open Direct Plot. On PSS form, select "*Current*". Select the current at the terminal of the load. We confirm that the *Imax* is 55uA just as in the original PLL transient simulation, see Figure 13.

PFD/CP Noise Current and input-referred Jitter.

The VCO will be sensitive to the voltage level on the control node, not the timing of its variation. As a result we are interested in the voltage noise as the node.

Use Direct Plot, PNoise form to plot the *Output Noise*. On Figure 14, you see the compared noise curves for no mismatch and for 2ns delay between PFD inputs. The units are current squared per Hz. Integrated over the total frequency range, the current noise is translated to the input noise in seconds, using *Kdet* which is defined as transfer function from relative time delay on the input of PFD/CP to the output current of the block.

Figure 14 PFD/CP Noise Current and input referred Jitter.



Open the wave in Calculator tool and implement the following operations to determine the input referred jitter:

(3) $$J_{ee_{PFD/CP}} = \frac{T}{K_{det}}\sqrt{\frac{1}{2}\int_{f_{min}}^{f_{max}} S(f)\,df},$$

where the *Kdet* equal the charge pump current (55uA) and *T* is the reference signal period (40ns). The integration is up to the 40dB drop limit that we found in the LPF simulation - 680MHz.

The jitter value depends on the operating condition - the noise is larger as the phase mismatch in inputs is increasing. For no mismatch, the estimate is 10ps while for 2ns advanced feedback signal, it grows to 16ps. Depending on the situation, we can select the worst case or the locked (no mismatch) number.

Frequency divider.

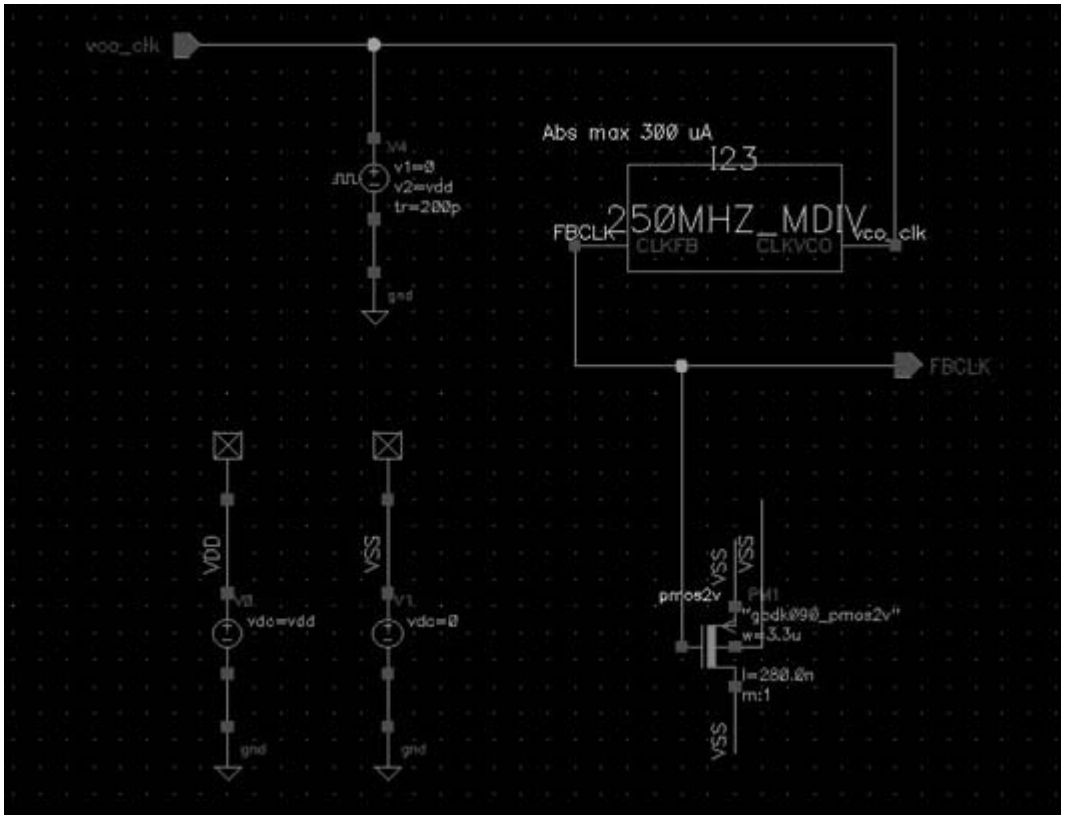
The jitter extraction for dividers is based on the notion that the output of the divider will

typically enter the logical or digital circuits with the threshold sensitive behavior. The time of the threshold crossing will be affected by the additive noise. The time of the crossing will be affected by the synchronous jitter [2]. We need to find the noise at the location of the crossing by the noiseless signal and use the slew rate to convert it into the jitter. In the current SpectreRF implementation, it is done automatically as the jitter measurements for driven circuits.

The testbench for the divider is shown at Figure 15. The input is the voltage source with the typical transition times (100ps). The *Period* of the pulse corresponds to the VCO output of 250MHz. The load represent the input stage of the PFD.

Setup PSS analyses the fundamental frequency equal to the output frequency of FD, 25MHz, or period of 40ns. Short *tstab* of 100ns is enough. Add PNoise analysis with the frequency sweep over the wide range, up to half of the output frequency, which is 12.5MHz. Set the sweep to be *relative* to the first harmonic. Output *voltage* is from output node *FBCLK* to the ground. Activate *Jitter* option for PNoise and specify the threshold value (VDD/2 = 1.25V) and the *rise* direction for the signals transition. Set "*Maximum sidebands"* to 400. Run simulation.

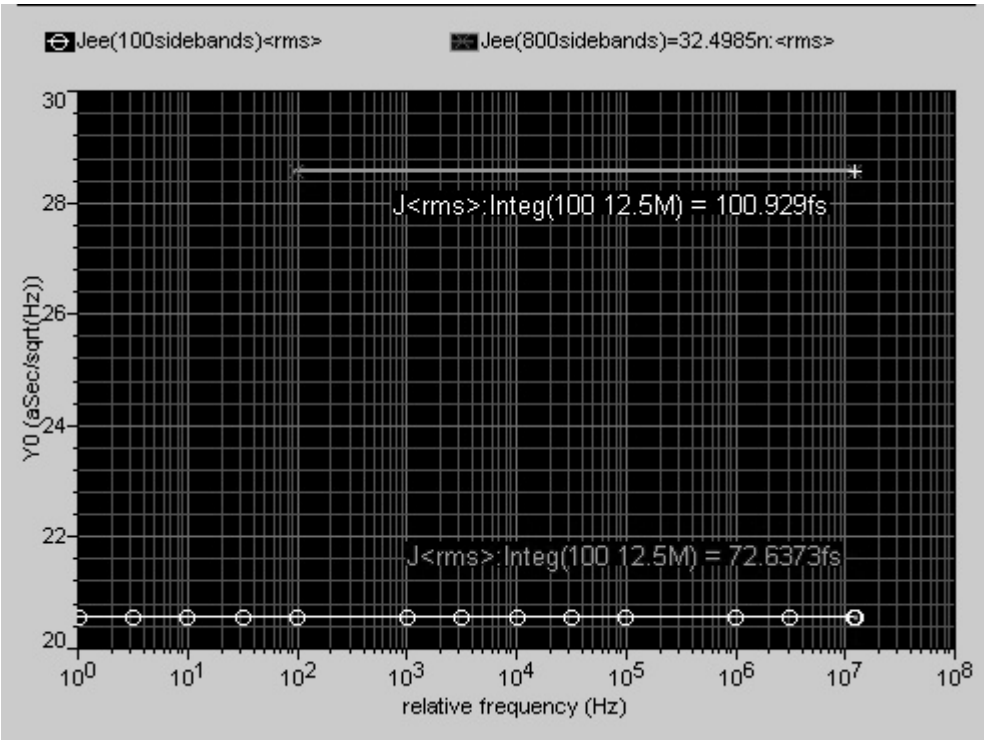
Figure 15 Frequency Divider. Testbench schematic.



Use Direct Plot. Open "*pnoise jitter*" form to plot PSD of the time jitter. *Jee* represents the

edge to edge jitter from the internal noise sources of the divider. The complication arises from the fact that the aliasing due to sampling of the divider output requires very high number of the sidebands. The Figure 16 depicts the increase in the value from 400 to 800 sidebands. The simulation time could be very long for 800 sidebands.

Figure 16 Frequency Divider. PSD of the time Jitter.

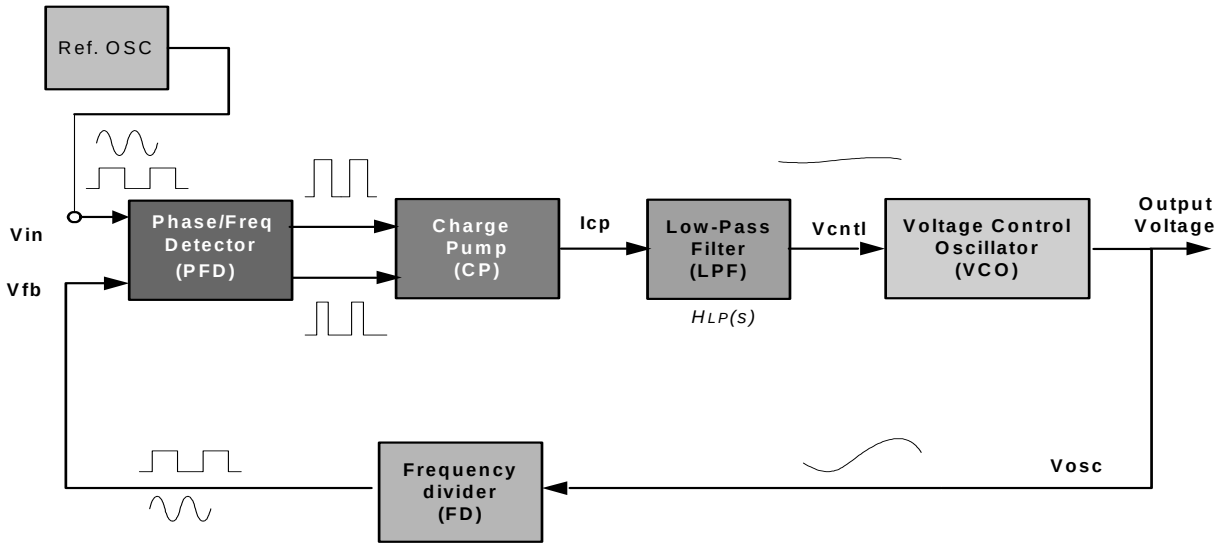


Input Jitter from the reference signal

In addition to the original example, where reference signal is an ideal voltage source, we present additional results for the circuit with a reference oscillator present. Since no transistor level design was available, we used some typical values for the reference jitter and other parameters of the model.

The diagram for the modified PLL is shown at Figure 17.

Figure 17 The block diagram of a PLL with reference oscillator.



Power supply and substrate contribution to the jitter

For a simple stationary noise on the power supply or a substrate in the VCO, one can use a modulated PXF analyses to extract the phase noise due to such noise source.

Such noise could be added to the internal noise of the VCO computed above.

The usage of PXF for jitter contribution of the power supply and substrates will be demonstrated on the future examples.

Behavioral models

A small library of the VerilogA models with corresponding symbols was created for this PLL circuit. The blocks are generic and could be always improved or their functionality augmented for more accurate representation of the PLL components and to include more noise/jitter effects.

VCO

The simple VCO model is presented below. The parameters from the VCO simulation are Fmin, Fmax, Vmin, Vmax, transition rate *tt* and the period jitter. The period of the VCO output signal will be stored in the file in the Matlab format. We set the time to start the output using a parameter "*outStart"*.

Inside the model, the period jitter is converted into the random frequency variations that lead to the variations in the phase of the output and its transition timing.

Listing 1 VCO model

```
// Verilog-A model for VCO with accumulating jitter.
// The user accepts full responsibility for the use of this model.
`include "disciplines.h"
`include "constants.h"

module PLL_VCO(vco_clk_ph_0, Vcontrol);

input Vcontrol;
output vco_clk_ph_0;
electrical Vcontrol, vco_clk_ph_0;
parameter real Vmin=0;
parameter real Vmax=Vmin+1 from (Vmin:inf);
parameter real Fmin=1M from (0:inf);                    // minimum output frequency
parameter real Fmax=2*Fmin from (Fmin:inf);             // maximum output frequency
parameter real Vhi=1;                                   // high output voltage
parameter real Vlo=0;                                   // low output voltage
parameter real tt=0.01/Fmax from (0:inf);               // output transition time
parameter real jitter=0 from [0:0.25/Fmax);             // period jitter (produces white accumulating jitter)
parameter outStart=10m from (1/Fmin:inf);               // start saving periods of
VCO to a file
parameter real ttol=1u/Fmax from (0:1/Fmax);            // time tolerance
real freq, phase, dT, prev, curPeriod, diffPeriod, prevPeriod;
integer n, seed, fp, fjcc;

analog begin
   @(initial_step) begin
                seed = -561;
                fp = $fopen("periods.m");
                fjcc = $fopen("jcc.m");
   end

   // compute the freq from the input voltage
   freq = (V(Vcontrol) - 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 2p
   phase = 2*`M_PI*idtmod(freq, 0.0, 1.0, -0.5);

   // update jitter twice per period
   // `M_SQRT2=sqrt(K), K=2 jitter updates/period
   @(cross(phase + `M_PI/2, +1, ttol) or cross(phase - `M_PI/2, +1, ttol)) begin
                dT = `M_SQRT2*jitter*$dist_normal(seed,0, 1);
                n = (phase >= -`M_PI/2) && (phase < `M_PI/2);
                if(n == 1) begin
                   if($abstime >= outStart) begin
                                curPeriod = $abstime - prev;
                                diffPeriod = curPeriod - prevPeriod;
```

```
                        $fstrobe(fp, "%0.10e", curPeriod);
                        $fstrobe(fjcc, "%0.10e", diffPeriod);
                    end
                    prev = $abstime;
                    prevPeriod = curPeriod;
                end
    end

    // generate the output
    V(vco_clk_ph_0) <+ transition(n ? Vhi : Vlo, 0, tt);
end
endmodule
```

Frequency divider.

The frequency divider model has typical timing parameters as transition times and time delay from input to the output. The maximum, minimum and threshold voltages are the other set of parameters. The synchronous edge to edge jitter from the above simulation of FD is used to generate the random dither at the output.

The model is a counter of the crossing transition in one preset direction (another parameter), generates random jitter and uses the *transition* function to generate the continues transitions at the output when the counter is matching the divider *ratio.*

Listing 2 Frequency Divider model

```
// Verilog-A model of frequency divider with synchronous jitter.
// The user accepts full responsibility for the use of this model.
`include "disciplines.h"

module PLL_250MHZ_MDIV(clkfb, clkvco);

output     clkfb; // output
input      clkvco; // input (edge triggered)
electrical clkfb, clkvco;

parameter integer ratio=2 from [2:inf); // divide ratio
parameter real Vhi=+1;          // output voltage in high state
parameter real Vlo=0;           // output voltage in low state
parameter real Vth=(Vhi+Vlo)/2;   // threshold voltage at input
parameter integer dir=1 from [-1:1] exclude 0;
                            // dir=1 for positive edge trigger
                            // dir=-1 for negative edge trigger
parameter real tt=100p from (0:inf);      // transition time of output signal
parameter real td=0 from (0:inf);       // average delay from input to output
parameter real jitter=0 from [0:td/5);  // edge-to-edge jitter
parameter real ttol=1p from (0:td/5);   // time tolerance, recommend ttol << jitter
integer count, n, seed;
real dt;

analog begin
    @(initial_step) seed = -311;
    @(cross(V(clkvco) - Vth, dir, ttol)) begin
        count = count + 1; // count input transitions
        if (count >= ratio)
            count = 0;
        n = (2*count >= ratio);
        dt = jitter*$dist_normal(seed,0,1); // add jitter
    end
```

```
   V(clkfb) <+ transition(n ? Vhi : Vlo, td+dt, tt);
end
endmodule
```

PFD/CP.

The model below represents an example of a simple three state phase-frequency detector. The amplitude of the output current *Iout* represent the Imax that was measured in the simulation for the block. the synchronous gaussian jitter is added to the output and it changes the shape of the transition during the rise and fall.

Listing 3 PFD/CP model.

```
// Verilog-A model for PFD/CP with synchronous jitter.
// The user accepts full responsibility for the use of this model.
`include "disciplines.h"

module PLL_PFD_CP(out, refclk, fbclk);

input refclk, fbclk;
output out;
electrical refclk, fbclk, out;

parameter real Iout=50u;
parameter real Vtrans= 0 from [-10:10]; // voltage above which the transition is
parameter integer dir=1 from [-1:1] exclude 0;                    // dir=1 for positive edge trigger
                                                                  // dir= 1 for negative edge trigger

parameter real tt=1n from (0:inf);
parameter real td=0 from (0:inf);
parameter real jitter=0 from [0:td/5); // edge-to-edge jitter
parameter real ttol=1p from (0:td/5); // recommend ttol << jitter

integer state, seed, fdbg;
real dt;

analog begin
        @(initial_step) begin
         seed = 716;
//       fdbg = $fopen("PFDCPdebug.txt");
        end

        @(cross(V(refclk) - Vtrans, dir, ttol)) begin
           if (state > -1) state = state  -1;
           dt = jitter*$dist_normal(seed,0,1);
        end

        @(cross(V(fbclk) - Vtrans, dir, ttol)) begin
           if (state < 1) state = state + 1;
           dt = jitter*$dist_normal(seed,0,1);
        end
//      $fstrobe(fdbg, "%d\n", state);
//      $fstrobe(fdbg, "%0.10e\n\n", dt);

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

Reference oscillator.

The simple OSC model is presented below. The parameters from the VCO simulation are Fmin, Fmax, Vmin, Vmax, transition rate *tt* and the period jitter. The period of the VCO output signal will be stored in the file in the Matlab format. We set the time to start the output using a parameter "*outStart"*.

Inside the model, the period jitter is converted into the random frequency variations that lead to the variations in the phase of the output and its transition timing..

Listing 4 OSC model.

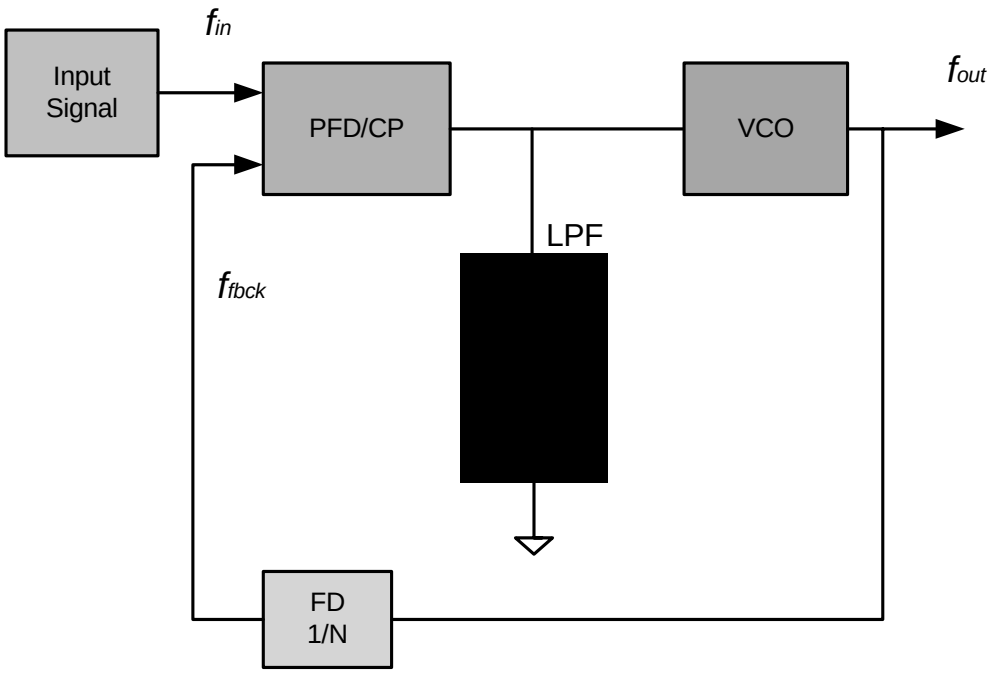
```
//
// VerilogA model for OSC with accumulating jitter.
// Original author: Ken Kundert
//
`include "discipline.h"
`include "constants.h"


module osc( out);
  output out;
  electrical  out;
                parameter real freq=1M from (0:inf);
                parameter real Vlo=0, 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 //osc
```

PLL jitter measurement using behavioral models.

Testbench.

Final jitter measurement behavioral model for PLL is presented at Figure 18. The only analog block is the original loop filter. The input is the reference frequency voltage source, as in original PLL.

Figure 18 PLL block diagram for the behavioral modeling.



The schematic of the PLL is shown on Figure 19. The PFD and CP are joined in a single model which symbol view is shown on Figure 20.

Figure 19 PLL schematic with VerilogA models for its blocks.

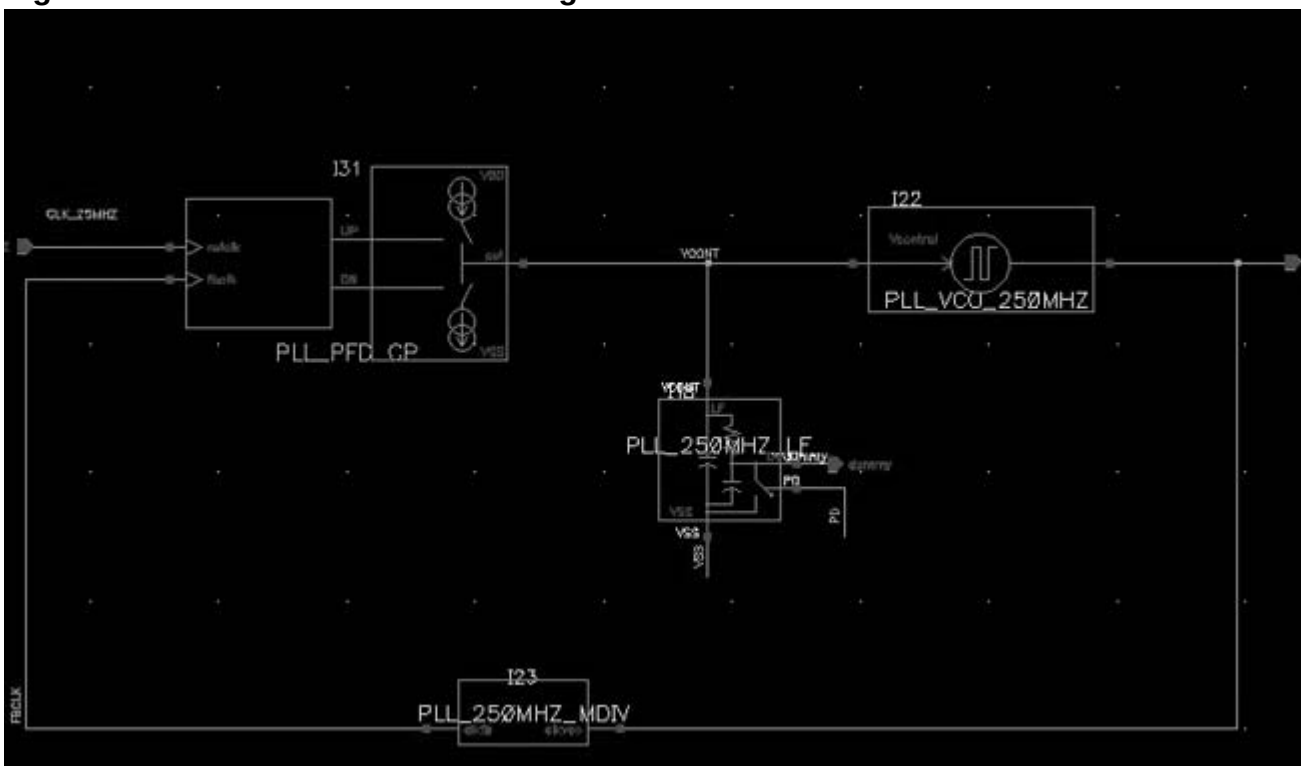
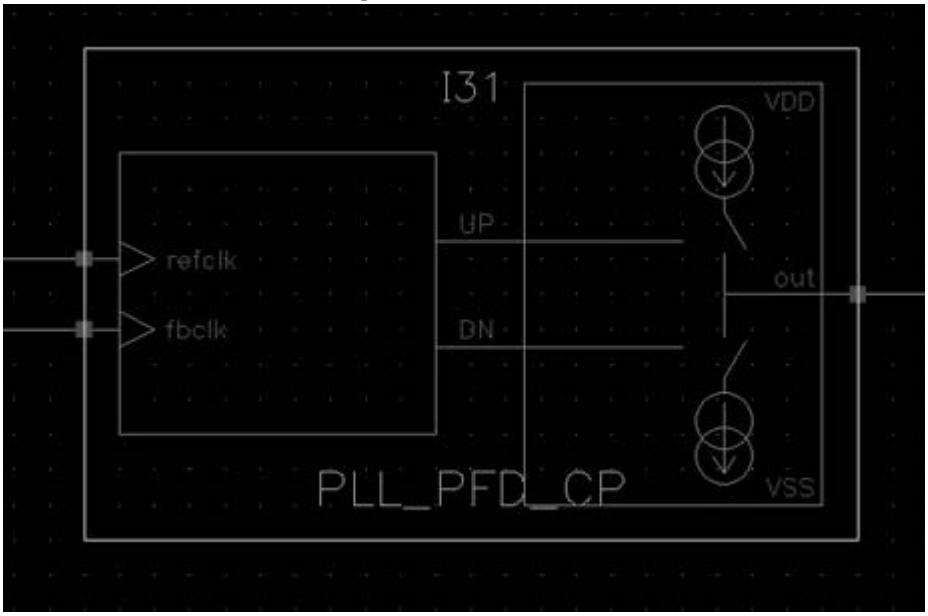
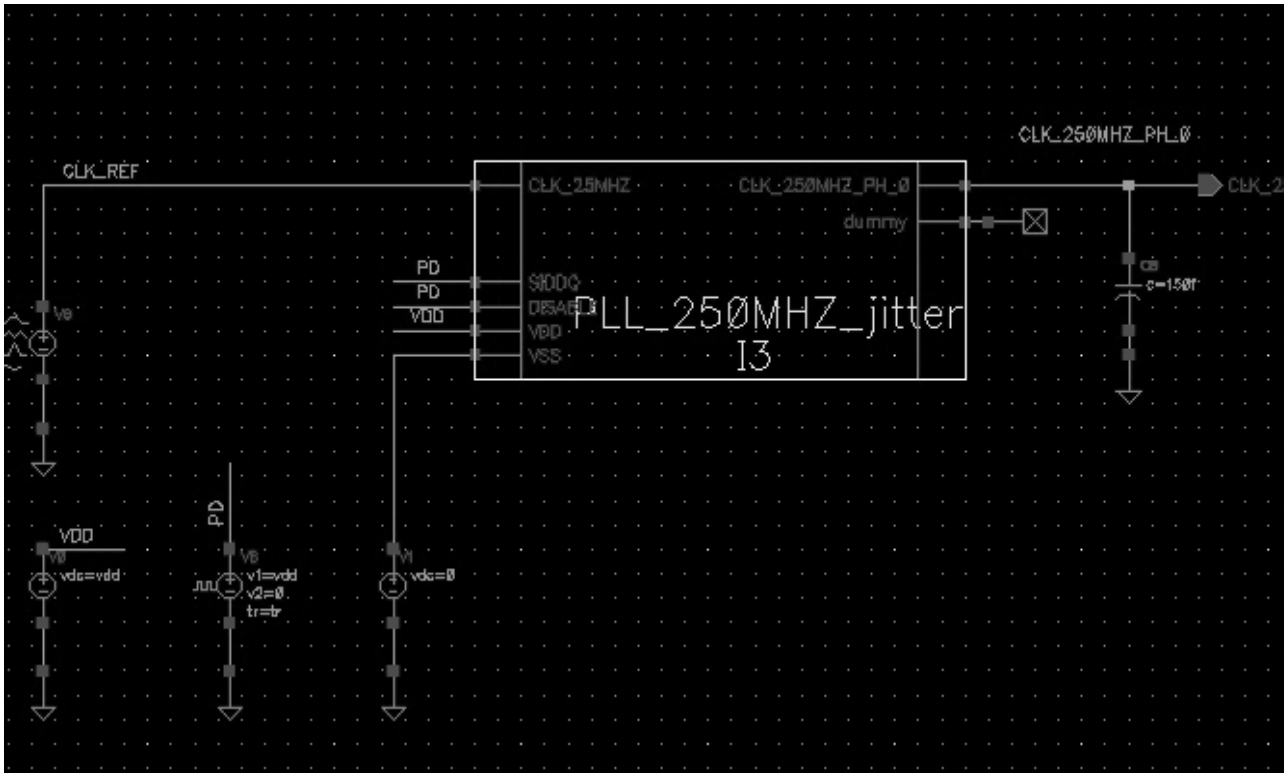


Figure 20 Combined PFD/CP VerilogA block.



The final testbench is similar to the original PLL testbench, Figure 21. The sources are the same. The load represents only a single phase output of PLL.

Figure 21 Testbench for PLL simulation using VerilogA models.

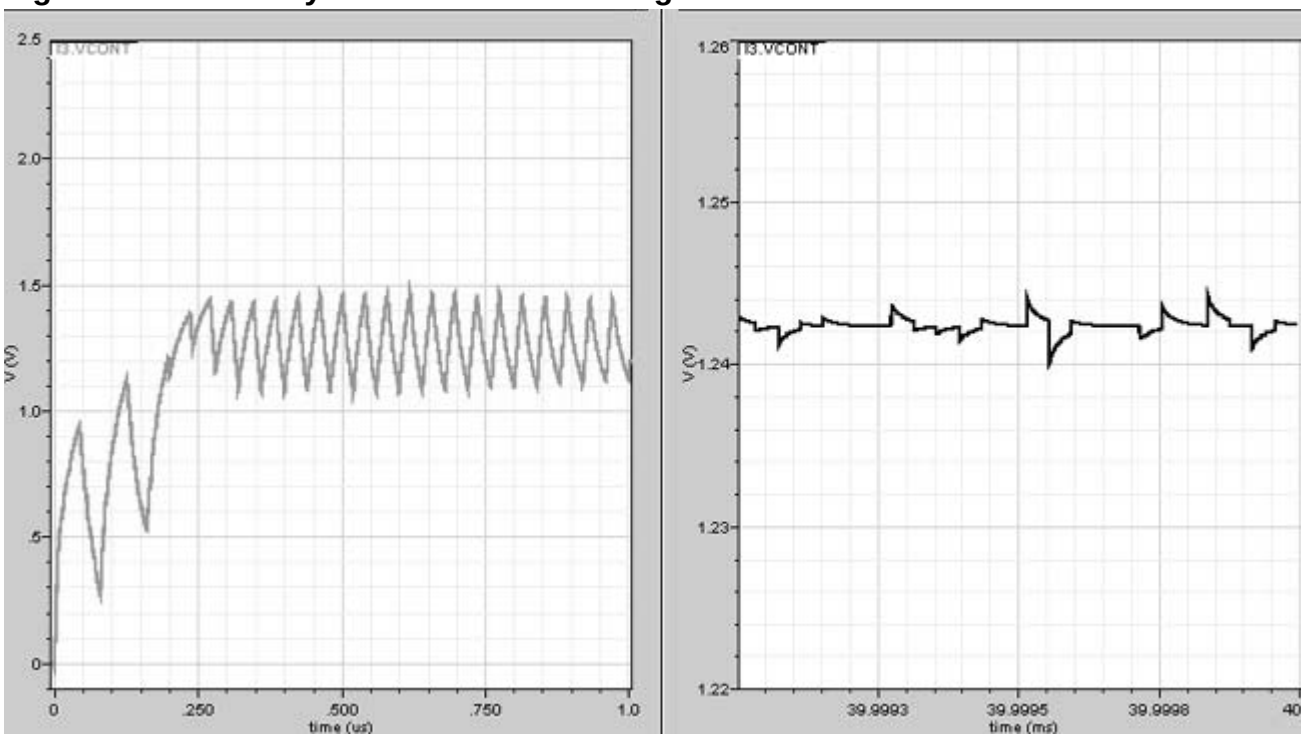


Transient jitter simulation.

Setup a long transient simulation, in our example we use 40ms. The output of the periods by VCO model will begin at the preset time *outStart*, set to 10ms. We select in ADE to output a minimum number of nodes to save time and disk space. Use *moderate* accuracy settings. Run simulation.

Load the results in Direct Plot or Results Browser. We plot several parts of the control voltage output to see the locking happening, see Figure 22. The frequency is also settled before 10ms to a reasonable value.

Figure 22 Tran analysis. VCO control voltage.



Post processing.

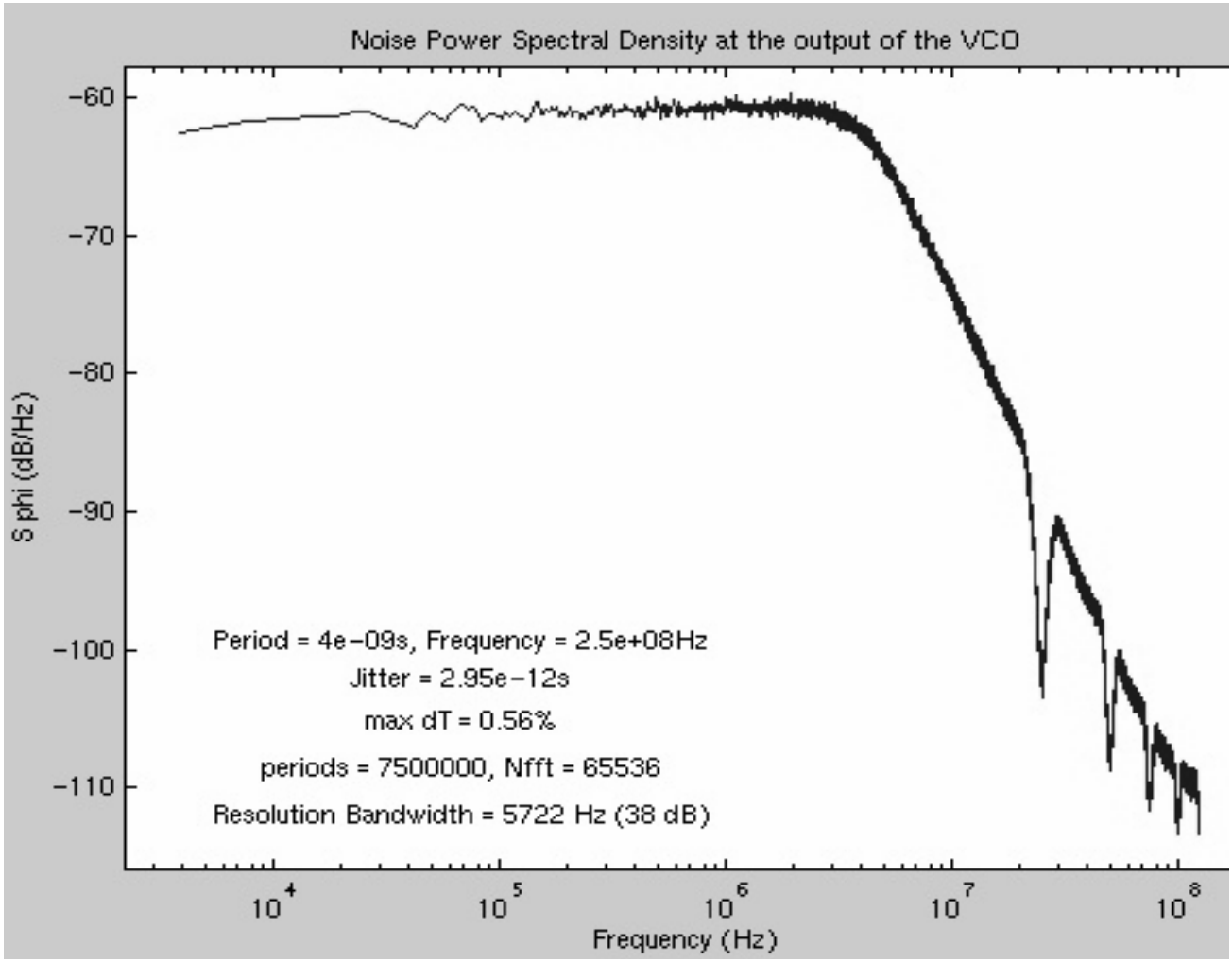
The output file that contain the periods of the output signal is processed using MATLAB tool by MathWorks. The original processing script is written by Ken Kundert in his work [2]. The period jitter is computed according to the definition[1]. The power spectral density of the output phase is computed using standard *psd* function, Listing 5. It uses Welch's method and it requires to select the number of samples to use for computations. After several experiments we use a large number of samples to get the stable result, Figure 23.

Listing 5 processing.m

```
echo off;
nfft=32768;
% 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('Noise Power Spectral Density at the output of the PLL');
xlabel('Frequency (Hz)');
ylabel('S phi (dB/Hz)');
rbw = winNBW/(T*nfft);
RBW=sprintf('Resolution Bandwidth = %.0f Hz (%.0f dB)', rbw, 10*log10(rbw));
line1=sprintf('Period = %.3gs, Frequency = %.3gHz\n',T, 1/T);
line2=sprintf('Jitter = %.3gs',J);
line3=sprintf('max dT = %.2g%%\n', 100*maxdT);
line4=sprintf('periods = %d, Nfft = %d\n', length(periods), nfft);

imtext(0.3,0.11, line4);
imtext(0.3,0.17, line3);
imtext(0.3,0.24, line2);
imtext(0.3,0.27, line1);
imtext(0.3,0.07, RBW);
```

Figure 23 Phase jitter Power Spectral Density. Matlab postprocessing.



Results discussion.

The VCO noise is low and the models of the example did not include the flicker noise. Most of the output results are dominated by the PFD/CP noise. It is shaped by the loop phase noise transfer function. Most important noise sources which will need to be considered in the future are the input jitter from reference oscillator, power and substrate noise sources.

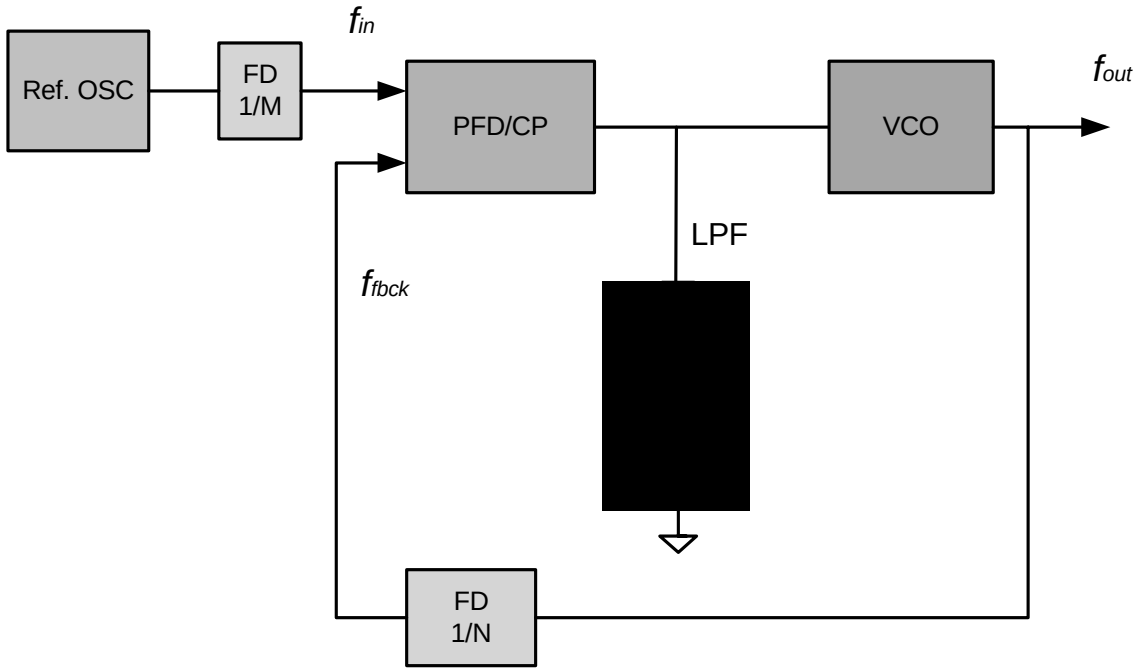
The behavioral simulation is slow since the LPF has components that are causing the small time step. Optimally, they will be replaced by the equivalent simplified models.

Improved behavioral models.

VCO and frequency divider. Reference OSC with frequency divider.

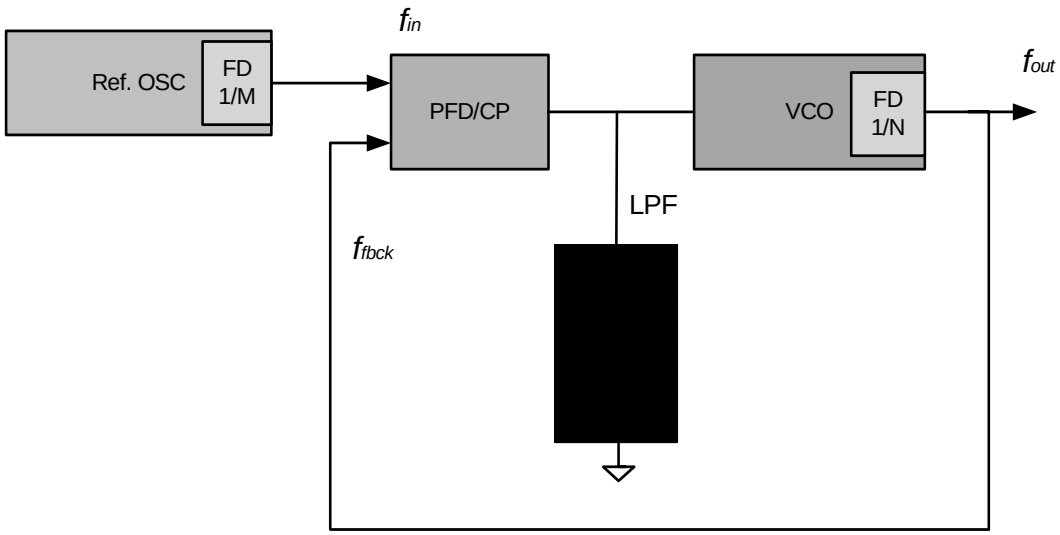
In the more complete examples, we will present the effect of the input jitter from the reference oscillator, using an additional model for a free running oscillator as the input of the PLL, Figure 24.

Figure 24 Complete frequency multiplier (synthesizer).



Additional runtime improvement will be done by including the frequency divider model into the oscillators models, Figure 25. This will save time for simulation and simplify the models. Certain restrictions apply, and additional postprocessing of results will be also needed.

Figure 25 Optimized PLL, dividers included into oscillators.



PFD/CP with jitter referred to the reference oscillator.

Another speed up could be achieved by reducing the number of events due to the jitter on the transitions. It can be done if we combine the jitter from several sources into a fewer number of blocks.

Conclusions.

We demonstrated the flow that can be used for a time domain simulation of the PLL jitter. The flow is as generic as the behavioral models could be. It uses latest updated jitter measurements for autonomous and driven circuits in SpectreRF. Additional upgrades to the flow will be introduced by using the new sampled small signal analyses. It will also benefit from the development of the models in the future.

Future development.

External noise sources.

The external uncorrelated stationary noise sources could be currently added to the flow by using the "*noisefile*" feature for external sources or ports. They will be properly modelled by

PNoise analysis and its variations.

Correlated external noise sources.

Correlated external noise source could be added by using PXF, modulated or sampled. The transfer functions from particular source will have to be later multiplied with the noise PSD, summed up by correlated group and added to the PNoise results.

When automated in MMSIM 6.2 release, it will allow to use more advanced version of PNoise/ Jitter analysis setup and post processed by enhanced Direct Plot.

Appendix.

Jitter estimate.

a) PM Jitter for driven circuits.

GUI supports the ability to specify the timing event by selecting the value and a direction of the crossing event for the output signal. PSS will produce results which will contain the breakpoint for this event, which guarantees the more accurate slew rate and noise computations for the time of the event (t_x). Using the relation between the simple PM jitter (*rms*) in time and time-domain noise at the time of crossing:

(4) $$J_c = \sqrt{2\,var(j_{PM})(t_x)} = \frac{\sqrt{2\,var(n_v, t_x)}}{\frac{d}{dt}v(t_x)}$$

where $var(n_v, t_x)$ is the total instant noise, the result of the sampled PNoise analysis for this particular crossing event. The jitter will be made available in the different metrics and using different units.

User sets up a regular PSS and PNoise analysis. After “Jitter measurement” part of the PNoise form is activated, the user will specify all the details for the events of the interest.

(*If jitter histogram is needed, there could be some extra information needed to limit the amount of the calculations.* - Not yet implemented.)

b) Simple FM Jitter for autonomous circuits.

This calculations will be based on results of the phase noise measurements. User will be

able to choose the frequency range to do the phase noise calculation. Later in the post-processing, a user will be able to select the frequency integration range to estimate the FM jitter. The regular PNoise analysis could be also used instead of the more expansive modulated phase noise computations. This will be accurate if the final measurements is done for the frequency range where the phase noise is by far greater than its AM counterpart.

The phase noise PSD, which is used later on, is assumed to be SSB and is twice the value of the Leeson Number which is computed by the regular PNoise or twice the PM component computed by the modulated PNoise.

Assuming the simple FM Jitter is present, the standard deviation of the variation in the period is:

(5) $$J_c(kT_c) = \sqrt{akT_c} = \sqrt{ak/f_c}$$

The period jitter is derived from the noise process PSD as:

(6) $$J_c^2(kT_c) = \frac{1}{(\pi f_c)^2}\int_0^\infty S_\phi(f)\sin^2(\pi k f T_c)\,df$$

where *f* is an offset frequency from the carrier frequency. The phase noise (or one could also use a ratio of the total voltage noise to the power in the carrier signal, with less accuracy) is used to calculate the a via the following:

(7) $$S_{\phi_{FM}}(f) = \frac{2af_c^2}{f^2}$$

User selects the frequency point f^{sample} to measure the phase noise PSD. To assist in selection the frequency where the slope corresponds to the white FM noise, we provide the function that plots the -20dB/dec slope. The final expression could be written as:

(8) $$J_c(kT_c) = \sqrt{\frac{1}{2}S_{\phi_{FM}}(f^{sample})\cdot\frac{f^{sample^2}}{f_c^3}}$$

References

[1] Cadence Application Note, "*Jitter measurements using SpectreRF",* 2005.

[2] Ken Kundert,"Predicting the Phase Noise and Jitter of PLL-Based Frequency Synthesizers", *The Designer's Guide,* www.designers-guide.org, 2005

[3] W.A. Gardner, *Introduction to Random Processes with Applications to Signals and Systems*, 2nd ed. New York: McGraw Hill, 1989

[4] Cadence Application Note, "*Oscillator Noise Analyses in SpectreRF*".

[5] Cadence Application Note, "*VCO Design Using SpectreRF*", 2004.