

SMASH

Release 5.6 – January 2006

PLL Jitter

Tutorial

DOLPHIN

INTEGRATION
39, avenue du Granier
B.P. 65 ZIRST

38242 Meylan France
tel : +(33) 4 76 41 10 96
fax : +(33) 4 76 90 29 65

web : <http://www.dolphin-integration.com>
e-mail : medal@dolphin-integration.com

This tutorial presupposes that the user knows how to use the basic functions of SMASH, and how a PLL works.

1 Introduction

1.1 What is Jitter?

Continuous advances in communication and measurement systems require higher levels of performances from system clocks. The most important, and certainly the least understood measurement of clock performance, is jitter. For most designers, jitter evaluation is still a mystic problem for which it is difficult to find efficient solutions.

Jitter can be defined as the deviation in a clock's output transition from their ideal positions. The deviation can either be leading or lagging the ideal position. The distribution of these time errors can be completely random (random jitter) or depend on other events (correlated jitter)¹. For computing Jitter, two common methods are known:

- the first one consists in using RF simulator algorithms, such as harmonic balance analysis,
- the second one consists in using Transient noise analysis, as possible with SPICE-like simulators.

Each of these solutions has its pros and cons. With the first method, all forms of jitter cannot be simulated, especially cycle Jitter and cycle-to-cycle Jitter, since only the phase noise can be simulated. With the latter, all forms of jitter can be simulated, but only for PLLs with low complexity as it is too time consuming. As a consequence, neither of these two methods can provide efficient solutions to PLL designers or SoC integrators for predicting accurately Jitter in a reasonable amount of time.

The aim of this tutorial is to provide an efficient methodology for simulating Jitter. Our approach allows a noise/jitter characterization of Virtual Components (ViC), in this case a PLL, in order to accelerate simulation.

1.2 Transient simulation method

A noise analysis is generally based on a frequency analysis, as the circuit works in small signal and has a linear relationship between noise source and output nodes.

However, this type of analysis is not relevant as it computes noise for a defined bias point. This is why RF simulation tools, based on harmonic balance or more recently on nonlinear stochastic differential equation, are preferred but these methods provide only long time jitter evaluation.

The noise effects can be simulated in transient domain, if the noise of each component can be simulated (if a noise model or the noise spectrum is available). To allow the simulation of any kind of jitter, instead of a method based on the stochastically differential equations, SMASH provides a method based on the inverse fast Fourier transform function (IFFT)².

¹ For a complete explanation on jitter refer to Annexe A

² For a complete explanation on transient method refer to Annexe B

1.3 Methodology

A method combining multi-level simulation with calibration provides a breakthrough solution by simulating properly all forms of Jitter with shortened simulation times.

The first step of the multi-level modeling with calibration method consists in characterizing, at the electrical level (SPICE), the noise in the frequency domain. The noise spectrum of each block is simulated using a small-signal noise simulation, or a transient noise simulation post-processed with an FFT. Each bloc is analyzed to choose the most suited type of simulation.

- The low-pass filter cells (LPF) or the voltage to current converter cells (CVC), used in the PLL, have working mode quite constant around the bias point; this allows a small-signal noise simulation.
- On the contrary, the phase-frequency detector cell (PFD) or the current controlled oscillator (CCO) have a working mode with a wide range of voltage changes; hence these cells must be simulated in transient domain.

Then, the jitters are simulated at behavioral level using the noise spectrum from the previous SPICE simulations. Noise sources are included in the behavioral models, enabling a fast simulation of the jitters of the complete PLL. The noise source inputs are the noise spectrum from the previous SPICE simulation results which can be:

- the FFT of a transient noise simulation result,
- the result of a small-signal noise simulation.

The noise source output is a transient waveform issued from an IFFT of the input noise spectrum.

2 Design steps

First, the PLL is sliced into different blocks and described at a behavioral level in VHDL-AMS or Verilog-A. The different blocks of the PLL of this tutorial are:

- Phase-Frequency Detector + charge pump (PFD),
- Low Pass Filter + Voltage to Current Converter (LPFCVI),
- Current Controlled Oscillator (CCO),
- Loop Divider (LD).

To give you an insight into the method, the behavioral modeling is different for each part of the PLL:

- Modeling the current controlled oscillator with piece-wise-linear function of the transfer function as well as for the voltage to current converter,
- Modeling the low pass filter using generic components, resistors, capacitors
- Modeling the Phase-frequency detector with standard gates models in Verilog/VHDL as well as for the loop divider
- Modeling the charge pump with only 2 current sources and 2 switches.

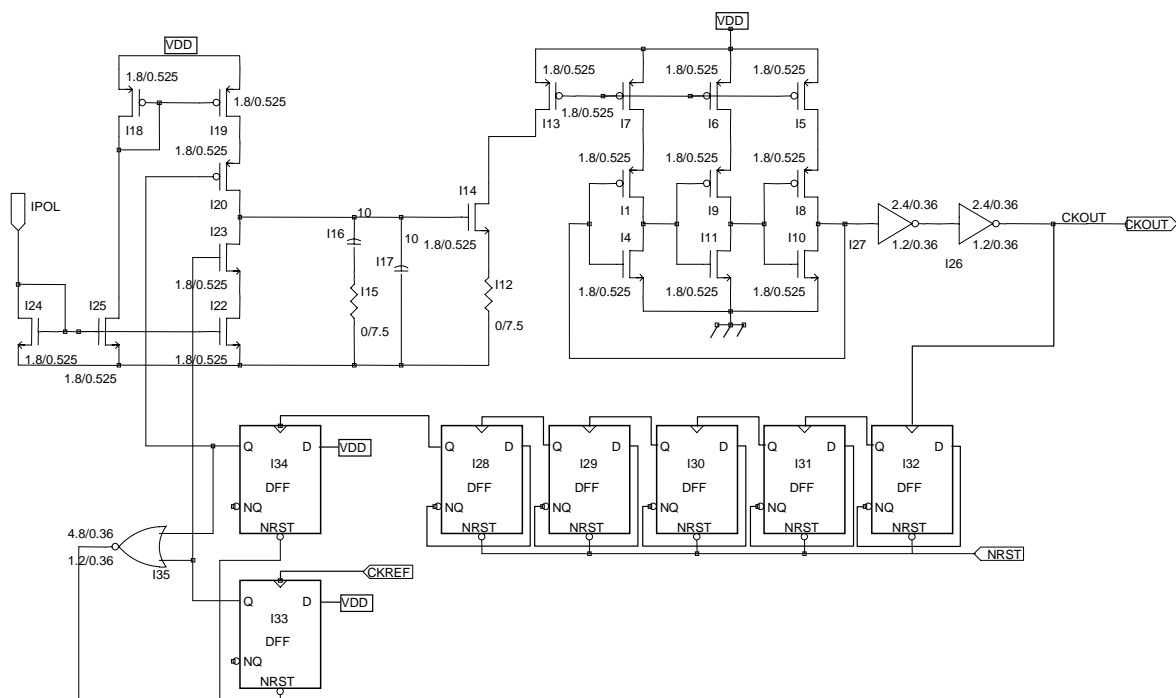


Figure 2-1: PLL Schematic

The multi-level simulation with calibration method starts with the characterization of the noise in the first bloc: *Phase-Frequency Detector + charge pump*.

2.1 PFD calibration

Load the PFD SPICE netlist:

```
<installdir>/examples/Tutorial/Jitter/spice/pfd/pfd_spice.nsx
```

Because the PFD+CP bloc is non-linear with a wide range of voltage changes when working in transient, we have to run a transient noise simulation. The second main characteristic of the PFD+CP is that it does not memorize the noise (contrarily to the CCO that integrates the noise). This second characteristic allows us to compute the noise impact as the output difference between a noisy block and quiet (without noise) block.

In the netlist, this difference is computed by:

```
Enoise noise 0 value='I(Vtest)-I(VtestNN)'
```

2.1.1 SPICE Setup and Simulation

You have to set the working conditions as in the locked mode of the PLL.

- The input signals are in phase so we just have to apply the same signal at both inputs.
- The output voltage is equal to 0.9V.

```
XI1 RST CKIN CKIN...
Vtest IOUT 0 0.9
```

Noise contribution of the cell is evaluated thanks to the current difference 'I(Vtest)-I(VtestNN)' computed by the voltage source Enoise.

To force the noise contribution of each component to be higher than the noise due to numerical computation errors, the noise of each component is multiplied by ten using the GNOISE=10 parameter (it will be the same for each SPICE block):

```
.param Gnoise=10
```

Run the transient analysis by selecting the menu *Simulate -> Transient -> Run*. In the *Transient analysis parameters* dialog box, be sure to check checkbox *Transient noise step* (see Figure 2-1):

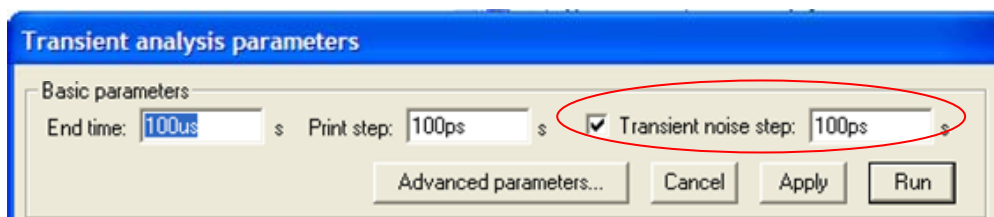


Figure 2-2: Setting the PFD SPICE simulation

Or, directly in the spice netlist:

```
.TRAN 0.1ns 100us 0s NOISE=yes NOISESTEP=0.1ns
```

The main parameters are:

- *End Time* because it defines the minimum frequency of the noise spectrum
 $F_{min} = 1/Endtime$

- *Transient noise step* because it defines the maximum frequency of the noise spectrum
 $F_{max} = 0.5/\text{Noise Step}$

Simulate for 100µs and choose a transient noise step of 100ps. Thus, our noise spectrum bandwidth will be 10kHz-5GHz.

2.1.2 Extraction of SPICE results

Current through the voltage source characterizes the noise of the PFD+CP. To achieve the noise characterization, an FFT is applied on this current to obtain its noise spectrum in the frequency domain. Select the I(ENOISE) signal in the transient window and select the FFT item on the contextual menu as shown in Figure 2-2.

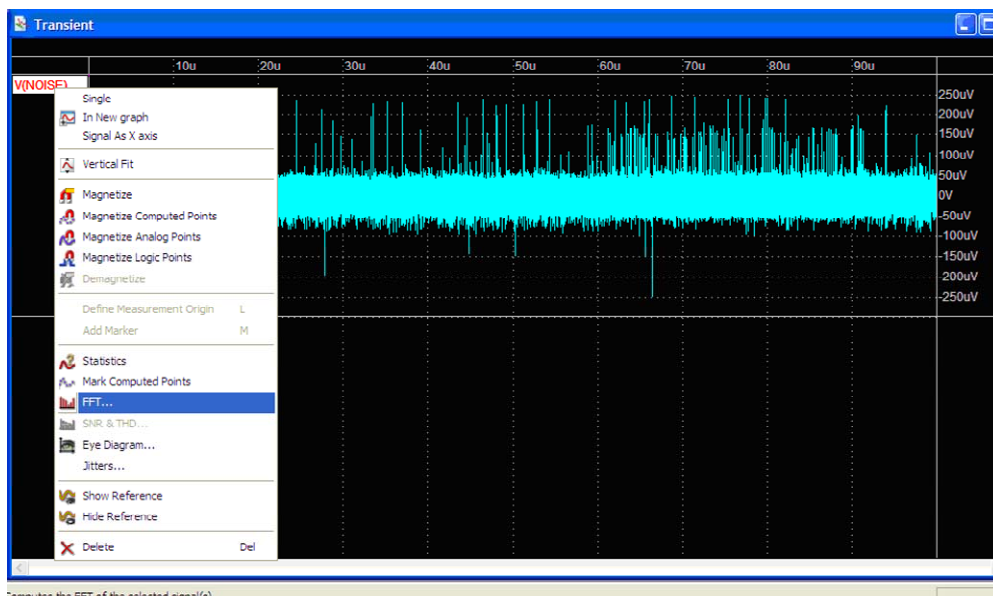


Figure 2-3: Calling FFT tool for SPICE simulation output signal

The *Fast Fourier Transform* window appears:

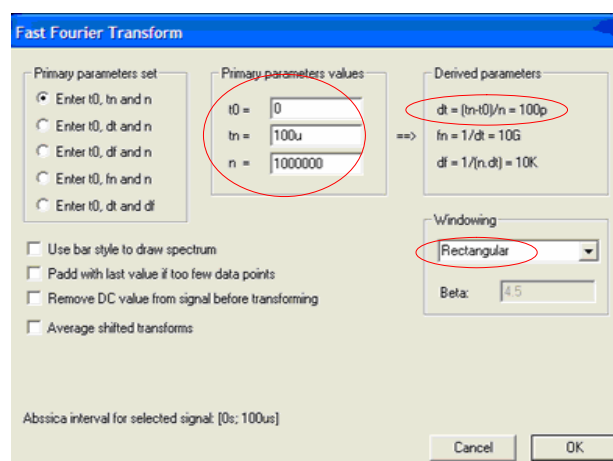


Figure 2-4: setting the FFT for extracting the noise spectrum of the PFD

The Main parameters are:

- *t0* and *tn* the start time and end time that define the transient window of the FFT
- *n* the number of points, to define the FFT resolution

As you can see in Figure 2-4, you may specify a number of points high enough to take into account all calculated points. For a 100µs simulation time with a noise step of 100ps, you have to compute 1 million points.

Note:

We use a rectangular windowing to preserve the spectrum noise integrity. Another windowing (for example Hamming) would multiply by zero the beginning and the end of the noise spectrum deforming it and changing the nature of the signal after the IFFT used to compute the transient noise in the behavioral model.

Once the FFT is computed, you have to store the result in a '.dat' file in order to re-use it as the input of the noise source included in the behavioral model. To do so, select the menu *Tools -> Dump in Text Format* (see Figure 2-5). SMASH offers the possibility to save just a part of the noise spectrum. As the noise spectrum is symmetric, you can store only half of it. Store your FFT results from 10kHz to 500Meghz.

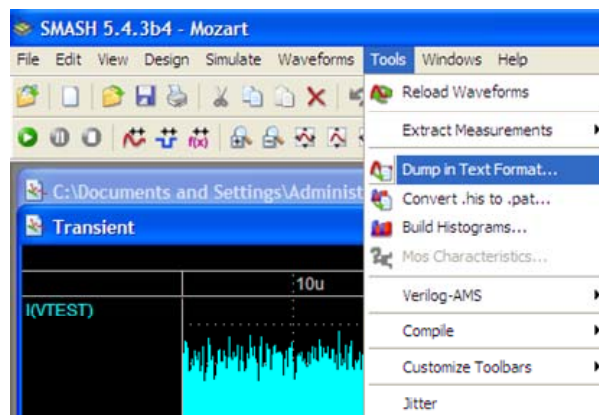


Figure 2-5: Saving FFT in a '.dat' file

2.1.3 AMS Setup and Simulation using the SPICE results

Load the PFD VHDL netlist:

```
<installdir>/examples/Tutorial/Jitter/vhdl/pfd/pfd_vhdl.nsx
```

The unit of the FFT result is in dB by bin, this is power in a bandwidth corresponding to the frequency sampling of the FFT. As the behavioral noise source expects a noise spectrum in $V/\sqrt{\text{Hz}}$, we have to convert the FFT result. This conversion can be done through the GNOISE parameter of the noise source, for this we will use an intermediary parameter named AMS_GNOISE.

```
.param AMS_GNOISE=1.41e-2
```

The noise current source is defined as follows:

```
.model INOISE_PFD I level=NOISE GNOISE='AMS_GNOISE'  
+YSCALE="LOG" XSCALE="LIN" FILE="pfd_spice.dat"  
+SIGNAL="REC(V(noise))"
```

The main parameters are:

- FILE: name of the '.dat' file where the FFT result is stored.

- SIGNAL: noise spectrum
- AMS_GNOISE= $\sqrt{(2/\Delta f)}$: where Δf is the FFT frequency sampling. It is a conversion factor to give the result in $V/\sqrt{\text{Hz}}$.

The PFD was simulated in SPICE during $100\mu\text{s}$, thus our FFT frequency sampling is 10kHz. Therefore, the AMS_GNOISE value is given by $\sqrt{(2/10,000)} = 1.41\text{e-}2$.

Launch the same transient noise analysis as in the first part. Finally, compare the FFT of the noise signal obtained from SPICE simulation and VHDL-AMS one (see Figure 2-6).

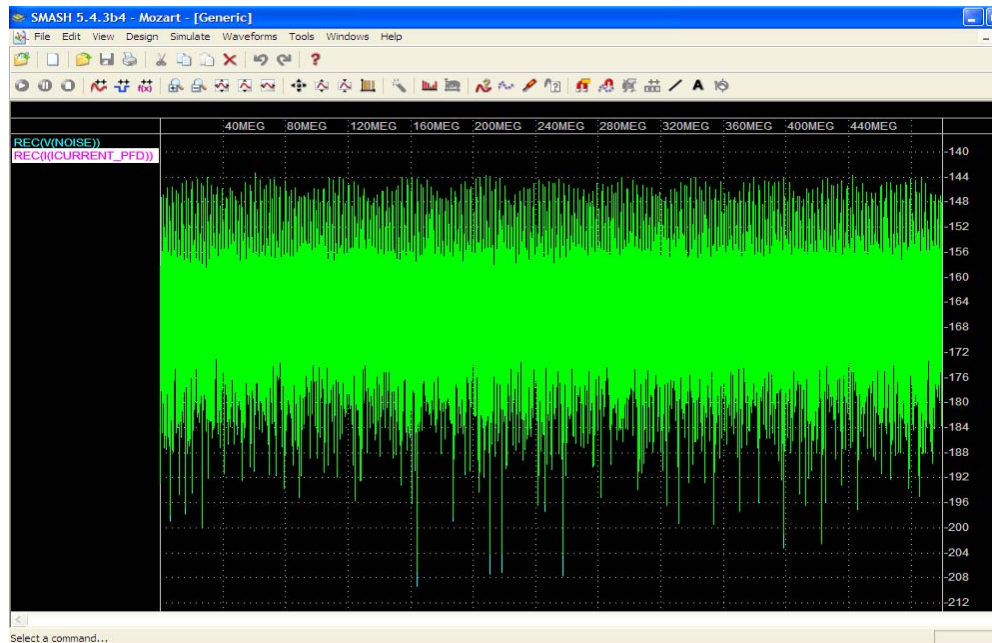


Figure 2-6: Comparing PFD noise spectrum results

Note:

The GNOISE parameter used to multiply the noise of the SPICE components in the SPICE netlist is implicitly conserved in the behavioral model. So there is no need to introduce it in the comparison of the results. But when we will evaluate jitter, we will have to divide the final result by a factor equal to $\text{GNOISE}=10$.

2.2 LPFCVI calibration

Load the LPFCVI SPICE netlist:

```
<installdir>/examples/Tutorial/Jitter/spice/lpfcvi/lpfcvi_spice.nsx
```

First, we have to make the difference between this cell and the previous one. Indeed, contrarily to the PFD+CP, during transient simulation, the voltage swing is small around the working point, so we can use the small-signal noise simulation. This type of simulation is attractive as it is faster and provides a spectral density in the right unit ($V/\sqrt{\text{Hz}}$).

2.2.1 SPICE Setup and Simulation

You have to set the working conditions as in the locked mode of the PLL.

- The input voltage is equal to the output of the PFD (0.9V).
- The output voltage (input of the CCO) is equal to 0.9V.

```
.IC V(VIN)=0.9
Vtest IOUT 0 0.9
```

Run the small signal noise analysis by selecting the menu *Simulate -> Noise -> Run*.

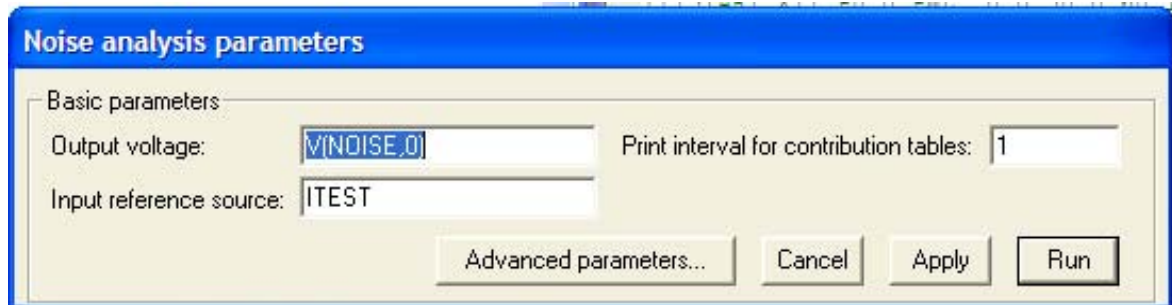


Figure 2-7: Launching noise simulation

This directive specifies the parameters for a small signal noise analysis that is performed in conjunction with the small signal analysis parameters:

```
.AC dec 5 10 1G
.NOISE V(NOISE,0) ITEST 1
```

For the V_{out} field, you must specify a differential voltage $V(Noise, 0)$. The input reference is the name of an independent current source. The noise analysis computes the RMS sum of all noise contributions at the output and the equivalent input noise at the specified input source, using the small signal transfer function between input and output. Analysis is performed from 10hz to 10Ghz, as specified by the .AC directive.

2.2.2 Extraction of the SPICE results

The small signal noise simulation gives directly the noise spectrum of the output signal. There is no need to compute a FFT. The simulation result is saved in an '.nmf' file that can be directly read by the noise source included in the behavioral model.

2.2.3 AMS Setup and Simulation using the SPICE results

Load the LPFCVI VHDL netlist:

```
<installdir>/examples/Tutorial/Jitter/vhdl/lpfcvi/lpfcvi_vhdl.nsx
```

The noise current source is defined as follow:

```
.param AMS_ GNOISE=1
.model INOISE_CVI I level=NOISE GNOISE='AMS_Gnoise'
+YSCALE="LOG" XSCALE="LIN" FILE="LPFCVI_SPICE.nmf"
+SIGNAL="ONoise"
```

As the result of the SPICE simulation is directly given in the right unity (V/\sqrt{Hz}), the AMS_GNOISE parameter of the behavioral current source is set to one.

We will now run a transient noise simulation post processed with an FFT to be able to compare the SPICE and the AMS simulation. Important parameters have already been discussed.

Simulate during 100 μ s and with a noise step of 100ps. Validation of the behavioral description is obtained by comparing the FFT obtained on the signal `IVtest` and the signal `DB(Onoise)` available in the ‘.nmf’ file obtained after the SPICE simulation.

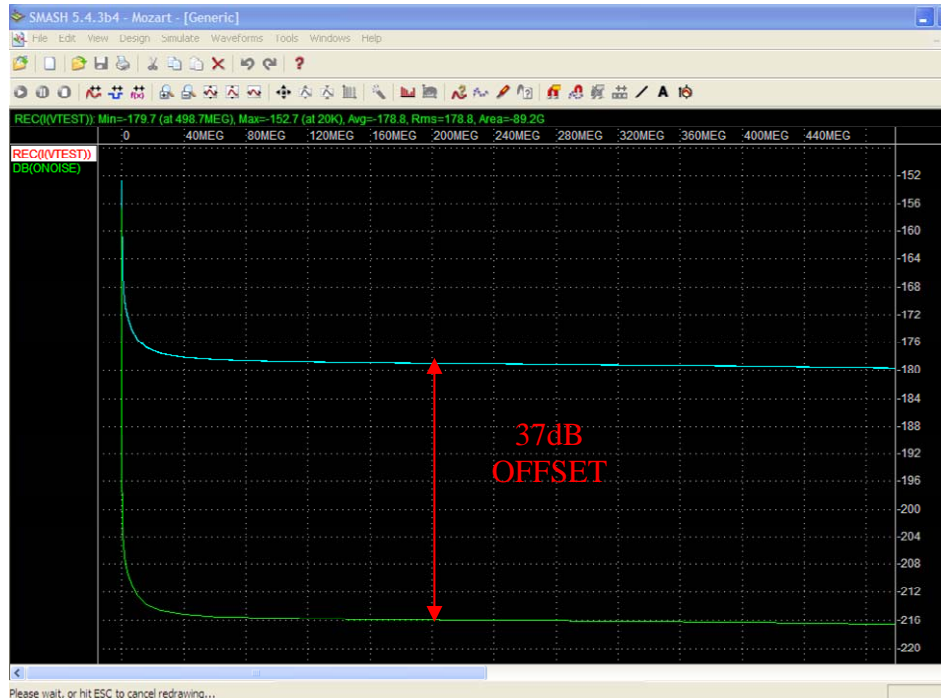


Figure 2-8: Comparing LPFCVI noise spectrum results

In Figure 2-8, we can observe that both signals are equal but with an offset of 37dB. This offset depends on the simulation duration as we compare values in different units. This offset is due to the FFP parameters and can be computed as:

$$10 \times \log_{10}(\Delta f / 2) = 10 \times \log_{10}(2 / 10,000) = 36.99 \text{ dB}$$

2.3 CCO calibration

Load the CCO SPICE netlist:

```
<installdir>/examples/Tutorial/Jitter/spice/cco/cco_spice.nsx
```

As evoked in the PFD discussion, the CCO transfer function is an integrator for the noise and the cell behavior is not linear. As a consequence, none of the methods used before can apply here. We will run a transient noise simulation and use the DSP function *Jitters* to compute all types of jitters from the output signal of the CCO.

2.3.1 SPICE Setup and Simulation

You have to set the working conditions as in the locked mode of the PLL.

- Input source current equal to 125 μ A:

```
I_POLP POLP 0 125u
```

As the output of the CCO is the output of the PLL, the noise of this cell will be directly calculated from output signal. There is no need to add a voltage or current source. Apply a transient noise simulation of 100 μ s with a 100ps noise step.

2.3.2 Extraction of the SPICE results

Select the output signal in the transient window and select the *Jitters* item in the contextual menu as shown in Figure 2-9.

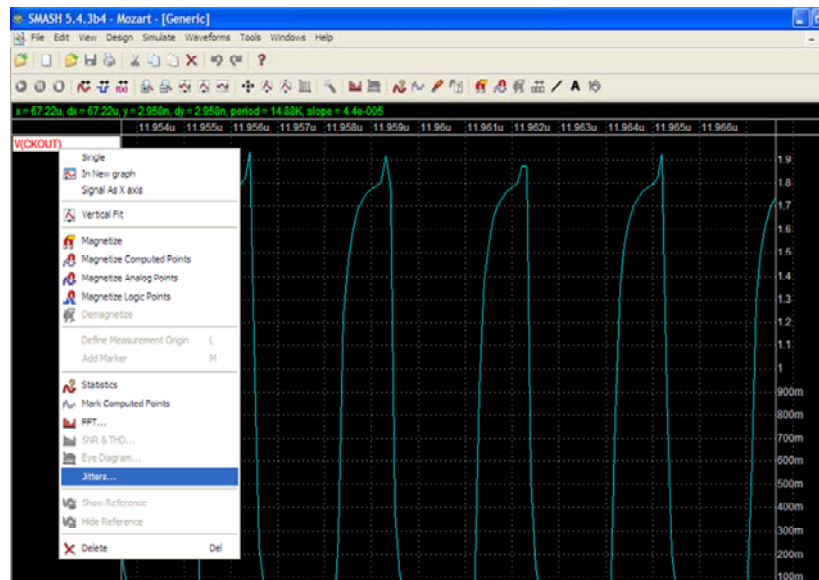


Figure 2-9: Calling Jitter DSP tool

All kinds of jitter (cycle, cycle to cycle, long term) are measured directly on the output signal of the CCO as shown in Figure 2-10.

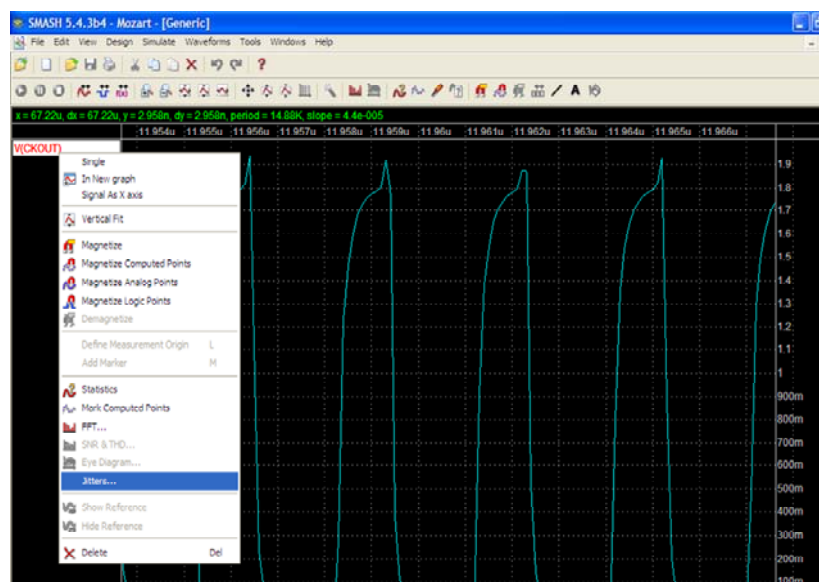


Figure 2-10: Jitters on the CCO output

The noise of the components causes the variation of the CCO output frequency $freq(V(CKOUT))$. For the input of the behavioral noise sources, we have to compute the

spectrum of this variation with an FFT. Perform an FFT post-processing on the frequency signal $freq(V(CKOUT))$ and dump it in text format as seen in previous sections. Do not forget to remove the DC value in the FFT settings as we want to extract the spectrum of the frequency variation and not the mean frequency value.

2.3.3 AMS Setup and Simulation using the SPICE results

Load the CCO VHDL netlist:

```
<installdir>/examples/Tutorial/Jitter/vhdl/cco/cco_vhdl.nsx
```

```
.param AMS_GNOISE=1.41e-2
.param CCOGain=0.8e12
.model INOISE_CCO I level=NOISE GNOISE=1.41e-2/GAINCCO
+YSCALE="LOG" XSCALE="LIN" FILE="PLL_SPICE_1m.dat"
SIGNAL="REC(V(noise))"
```

CCOGain is a parameter defining the gain of the CCO in Hz/A. It allows simulating the impact of the noise with a current source connected to the CCO input. Launch the same transient noise analysis as for the SPICE simulation and compute the jitter at the CCO output. Finally, compare the FFT obtained (Figure 2-11) for the noise signal with the SPICE simulation and for the VHDL-AMS one.

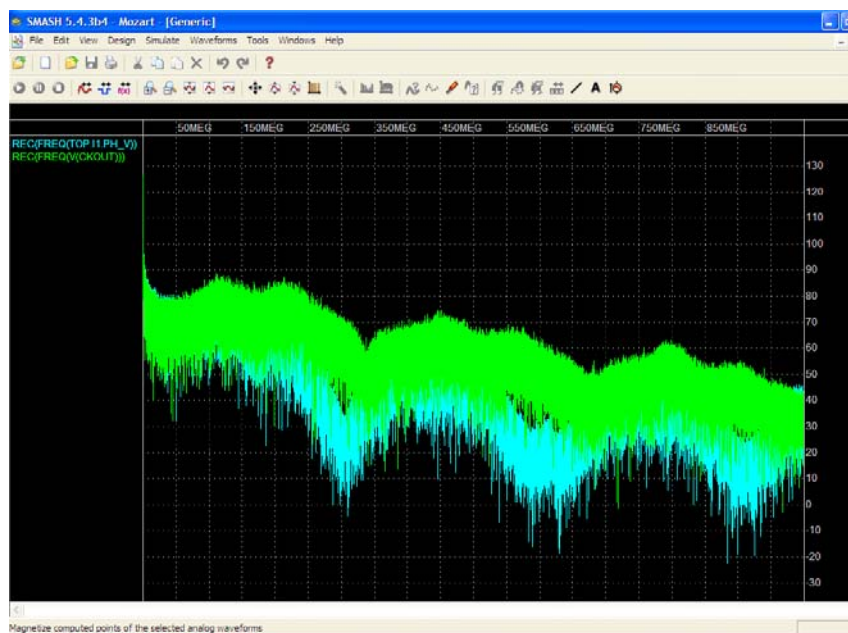


Figure 2-11: Comparing CCO noise spectrum results

3 Behavioral PLL

Once the noise sources have been calibrated with SPICE simulations (see chapter 2), we use the VHDL-AMS (or VERILOG-A) model of the whole PLL.

Load the PLL VHDL netlist:

```
<installdir>/examples/Tutorial/Jitter/vhdl/pll/pll_vhdl.nsx
```

And run a transient noise simulation.

All types of jitter are obtained with a significant simulation time reduction (some minutes compared to several ten of hours).

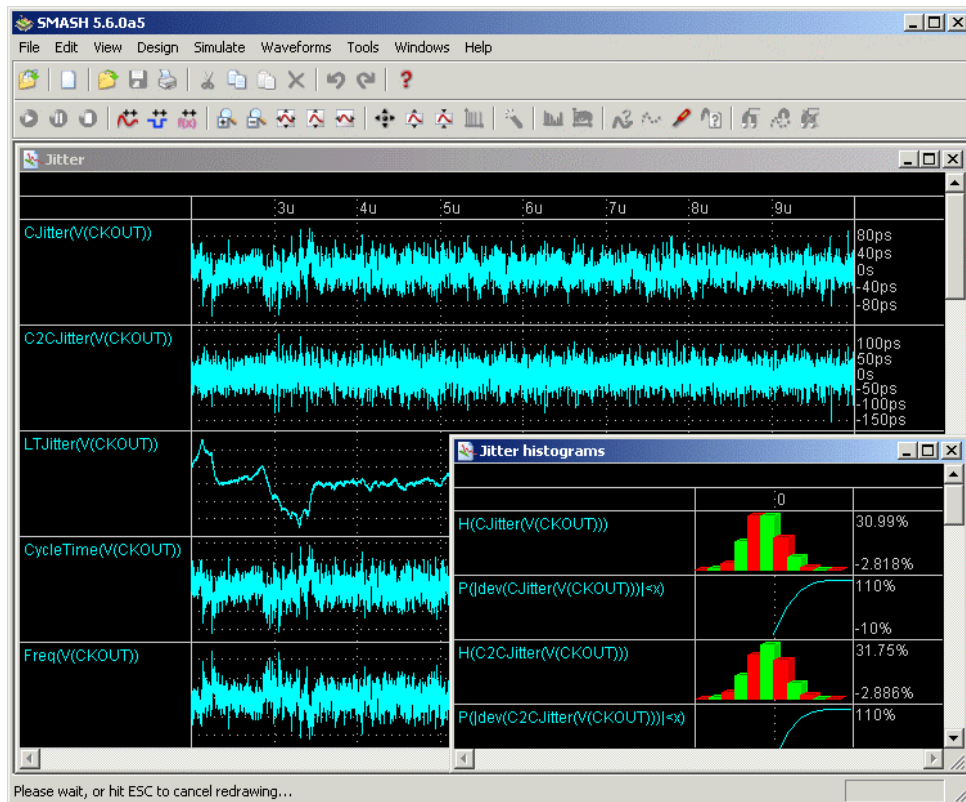


Figure 3-1: Jitters of the PLL

The table below summarizes the jitter contribution of each block of the PLL.

	PFD+CP	LPFCVI	CCO	PLL
Long term Jitter RMS	27.6ps	212.5ns	207ns	144.4ns
Cycle to Cycle Jitter RMS	33.1ps	6.4ns	27.8ps	2.9ns
Cycle Jitter RMS	20.7ps	5.2ns	35.35ps	3.3ns

Tableau 1: Jitter contribution of each bloc

Note:

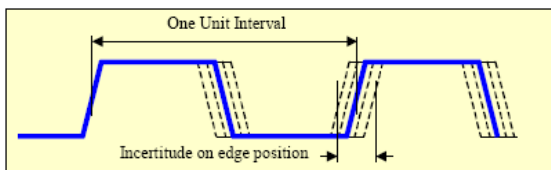
As mentioned at the beginning, we have to divide the result (peak to peak or rms) by the GNOISE factor (10). Remember that it was chosen to stay clear of numerical computation errors...

4 ANNEXES

4.1 ANNEXE A

Understanding Jitter

Continuous advances in communication and measurement systems require higher levels of performances from system clocks. The most important and certainly the least understood measurement of clock performance is jitter. Jitter can be defined as the deviation in a clock's output transition from their ideal positions. The deviation can either be leading or lagging the ideal position. The distribution of these time errors can be completely random (random jitter) or depends on other events (correlated jitter).



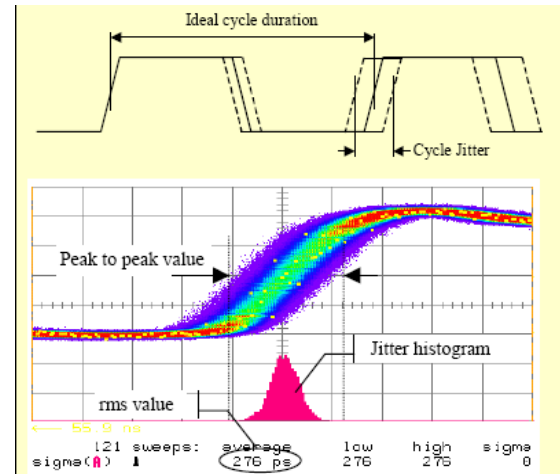
The jitter can be measured as the maximum variation (peak to peak value), or as the standard deviation (rms value). The jitter can be expressed in several units, like:

- Unit of time: Jitter expressed in units of time, describes the magnitude of the jitter in the appropriate order of magnitude, usually picoseconds.
- Degrees: Jitter expressed in degrees, describes the magnitude of the jitter in units of degrees, where one cycle equals 360° .
- Unit Intervals: A single unit interval is one theoretical cycle of the clock. This is the normalized clock period. Jitter expressed in unit intervals, describes the magnitude of the jitter as the fractional of one unit interval, normally expressed in percentage.
- Jitter Power: Jitter expressed in jitter power, described in unit interval

squared, usually expressed in dBui or dB relative to one cycle squared.

Cycle Jitter

Cycle jitter or period jitter measure the change in the cycle duration from the ideal cycle duration.

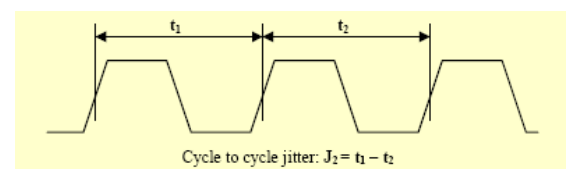


Cycle jitter is used to calculate timing margins in systems. Consider a memory with 1 ns of data setup time. If the maximum cycle jitter of the clock is higher than 1ns, the clock rising edge can occur before the data is valid on the data bus. Hence the system will not work properly. The main applications affected by cycle jitter are:

- Microcontroller-based systems
- Memory
- Digital communication

Cycle-to-Cycle jitter

Cycle to cycle jitter is the variation of the cycle duration from one cycle clock compared to the previous one.



Cycle to cycle jitter is mainly used when the clock drives another PLL. When the cycle-to-cycle jitter is higher than the maximum allowable, the

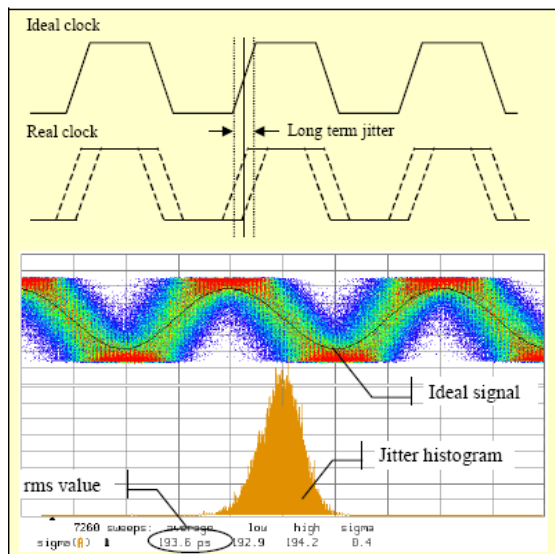
second PLL cannot lock to the reference frequency.

The main applications affected by cycle-to-cycle jitter are:

- Microprocessor including a PLL
- ADC or DAC including a PLL
- Digital communication

Long term Jitter

Long term jitter (also named short or medium term jitter) is the difference between the clock transitions and their ideal positions, over many cycles. The term "many" depends on the application and the period of variation of the output clock. For audio applications, the term "many" usually refers to some milliseconds. For PC motherboards and graphic applications, the term "many" generally refers to some tens of microseconds.



The main applications affect by long term jitter are:

- Audio systems
- Video systems
- Asynchronous communication without clock recovery

Correlated Jitter

The correlated jitter is deterministic in nature and depends on known phenomena generally periodical.

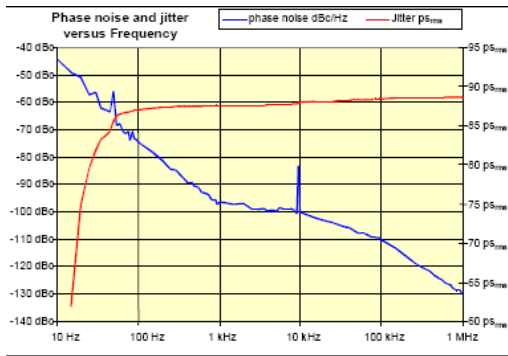
This jitter is usually caused by a coupling with another signal or due to a cyclic working mode of the clock synthesizer, for instance, if the dividing ratio changes periodically. Generally the correlated jitter causes spikes in the spectrum of the output clock, this spikes can be particularly annoying for audio applications or RF transmissions. Potentially any application can be affected by the correlated jitter depending on the period of modulation. High frequency modulations may have the same effect than cycle or cycle-to-cycle jitters; low frequency modulations may have the same effect than long-term jitter. In most cases, the correlated jitter can be detected thanks to the histograms, which is no longer Gaussian.

Jitter and Spectral Content

The dithered edge of clock is a result of noise (random jitter) and frequency modulation (correlated jitter). Noise and modulation have spectral content and power. Spectral content of clock jitter differs greatly depending on the technique used to generate the clock, especially if the dividing ratio is modulated. The spectrum density of jitter is well known as "phase noise". Equivalent jitter over various bandwidths may be calculated from the phase noise of a clock. The relationship between jitter and phase noise inside a known bandwidth is given by:

$$Jitter = \frac{1}{2\pi f_{clock}} \sqrt{\int_{f_1}^{f_2} S(f) df} \approx \frac{1}{2\pi f_{clock}} \sqrt{2 \int_{f_1}^{f_2} \ell(f) df}$$

where $s(f)$ is the phase noise density and $n(f)$ the spectrum density. The figure below shows the phase noise of a clock (curve blue) and its computing jitter (curve red). The phase noise is express in dBc versus the frequency offset from the carrier. The jitter is express in psrms (sigma value), and is computed in the bandwidth comprised between 10 Hz up to the considered point on the x axes.



Practically, it is not reliable to compute jitter from phase noise, since the accuracy is limited by the measurement exactness, especially by the noise floor of the spectrum analyzer.

Causes of Jitter

In System on Chip (SoC) applications, there are four principal causes of jitter given below in decreasing order of importance.

a) Coupling noise:

- Inside the circuit: the noise due to other cells in the circuit can inject some charges in the substrate, perturbs PLL's oscillator, and thus, the output frequency. This noise is particularly important in case of digital function and usually produces cycle jitter, cycle-to-cycle jitter and correlated jitter.
- Outside the circuit: in case of external low pass filter, the PLL is particularly sensitive to coupling between the external low pass filter and any external signal. This coupling will directly affect the output frequency of the PLL. Depending on the nature of the coupled signal, any kind of jitter can result, but commonly, long term jitter and correlated jitter results.

b) Power supply noise:

- PLL's supply inputs: usually, PLL are very sensitive to supply noise, which appears on output clock as jitter. This noise usually produces cycle jitter, cycle to cycle jitter, and correlated jitter.
- Digital's supply inputs: this noise changes the threshold voltages of the

input of the cells connected to the PLL output. Hence causes the same effect than a jittered clock. For example, a noise equal to 200 mV superimposed on a power voltage equal to 3.3 V causes 60 mV of uncertainty on the gate threshold. This uncertainty will produce an effect equivalent to a jitter equal to 120ps with a 2 ns clock rising edge. This noise on the digital power supply usually produces cycle jitter, or cycle-to-cycle jitter.

c) Jitter on the reference frequency, several phenomena can cause this jitter depending on the nature of this reference. Special attention must be done with mechanical noise from vibrations of the crystal reference.

d) In case of external divider, if the dividing ratio is modulated, it can add some correlated jitter.

4.2 ANNEXE B

The noise effects can be simulated in transient domain, if the noise of each component can be simulated (a noise model or the noise spectrum is available). To avoid any change in the simulator kernel, we do not use a method based on the stochastically differential equations, but the following method based on the inverse fast Fourier transform (IFFT). For a given biasing condition, the power spectrum density $S_i(f)$ of each component is known. During transient simulation, the biasing condition is able to change and to modulate the power spectrum density shape. Using the Priestley form, the power spectrum becomes:

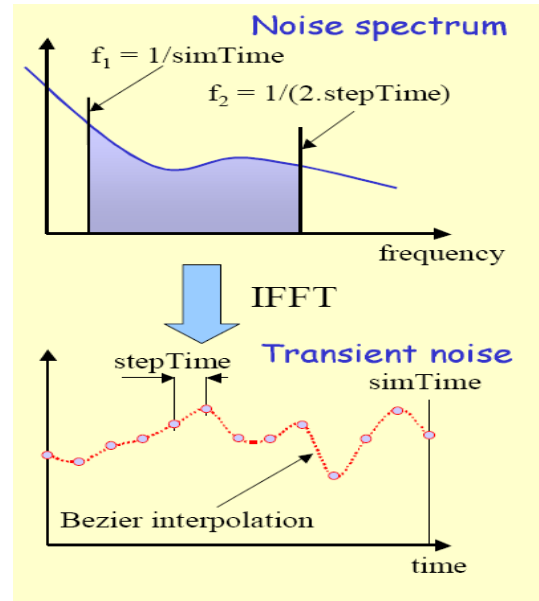
$$\Phi_i(f, t) = P(t) \times S_i(f)$$

Where $P(t)$ is the power spectrum modulated function, so it is a transient function. In stationary case (fixed bias), $\Phi_i(f, t)$ can be simplified to the usual $S_i(f)$. We assume that the noise is locally stationary, that is to say, the transient is equivalent to a succession of balanced states. The modulated function $P(t)$ exclusively depends on the biasing condition at the time t , we presume that no memory effect or no hysteresis effect occurs in the noise expression. The transient representation of $\Phi_i(f, t)$ is a second order stationary signal $o_i(t)$ given by:

$$o_i(t) = P(t) \times RFFT[S_i(f)]$$

Where the phase of $S_i(f)$ is a random function with an homogenous probability on $[-\pi; +\pi]$.

To help the convergence of the simulator, the signal given by the IFFT is interpolated using a Bezier function.



With that method, the following objectives are reached:

- Generating a signal in transient domain with spectrum characteristics given at each time by the power spectrum density defined by the model.
- The total power of the generated signal is equal to that given by power spectrum density defined by the model.
- The generated signals from the same power spectrum density are different and not correlated.

For example, for a MOS transistor, the current spectrum density of the Flicker noise J_f , can be written as:

$$J_f = \frac{KF \times gm^2(t)}{COX \times f^{AF} \times L_{eff} \times W_{eff}}$$

So the transient current of the Flicker noise $i(t)$ begins:

$$i(t) = gm^2(t) \times RFFT \left[\frac{KF}{COX \times f^{AF} \times L_{eff} \times W_{eff}} \right]$$

We have implemented the foregoing algorithm in our SMASHTM simulator to compute the noise contribution of each component (MOS, bipolar, resistor, jfet, etc...) during a transient simulation.

5 Table of contents

1	INTRODUCTION	3
1.1	WHAT IS JITTER?	3
1.2	TRANSIENT SIMULATION METHOD	3
1.3	METHODOLOGY	4
2	DESIGN STEPS	5
2.1	PFD CALIBRATION	6
2.1.1	SPICE SETUP AND SIMULATION	6
2.1.2	EXTRACTION OF SPICE RESULTS	7
2.1.3	AMS SETUP AND SIMULATION USING THE SPICE RESULTS	8
2.2	LPFCVI CALIBRATION	9
2.2.1	SPICE SETUP AND SIMULATION	9
2.2.2	EXTRACTION OF THE SPICE RESULTS	10
2.2.3	AMS SETUP AND SIMULATION USING THE SPICE RESULTS	10
2.3	CCO CALIBRATION	11
2.3.1	SPICE SETUP AND SIMULATION	11
2.3.2	EXTRACTION OF THE SPICE RESULTS	12
2.3.3	AMS SETUP AND SIMULATION USING THE SPICE RESULTS	13
3	BEHAVIORAL PLL	14
4	ANNEXES	17
4.1	ANNEXE A	17
4.2	ANNEXE B	20
5	TABLE OF CONTENTS	21