

Fractional-N Frequency Synthesizer Design Using The PLL Design Assistant and CppSim Programs

Michael H. Perrott
<http://www.cppsim.com>

July 2008

Copyright © 2004-2008 by Michael H. Perrott
All rights reserved.

Table of Contents

Setup.....	2
Introduction.....	2
A. GSM Synthesizer Specifications for a Transmitter Application.....	2
B. Preliminary Synthesizer Design Specifications.....	3
Noise Analysis using the PLL Design Assistant.....	4
A. Performing Basic Noise Analysis.....	4
B. Adjusting the PLL Architecture to Meet the Noise Specifications.....	5
C. Adding in the Effect of Detector Noise.....	6
D. Accounting for Parameter Variations.....	7
Dynamic Analysis Using the PLL Design Assistant.....	9
A. Checking Stability.....	9
B. Checking the Settling Time.....	10
C. Examining the Influence of Zero Variations.....	11
Preliminary CppSim Simulation Analysis.....	13
A. Entering the Design into Sue2.....	13
B. Setting Up the CppSim Simulation File.....	16
C. Examining the Simulated Step Response.....	17
D. Examining the Simulated Phase Noise.....	18
Advanced CppSim Simulation Analysis.....	19
A. Observing Cycle Slipping.....	20
B. Examining the Effects of Charge Pump Mismatch.....	21
C. Moving the Nominal Phase Error Away from Zero.....	22
D. Producing Fractional Spurs.....	25
Conclusion.....	26

Setup

Download and install the CppSim Version 3 package (i.e., download and run the self-extracting file named **setup_cppsim3.exe**) located at:

<http://www.cppsim.com>

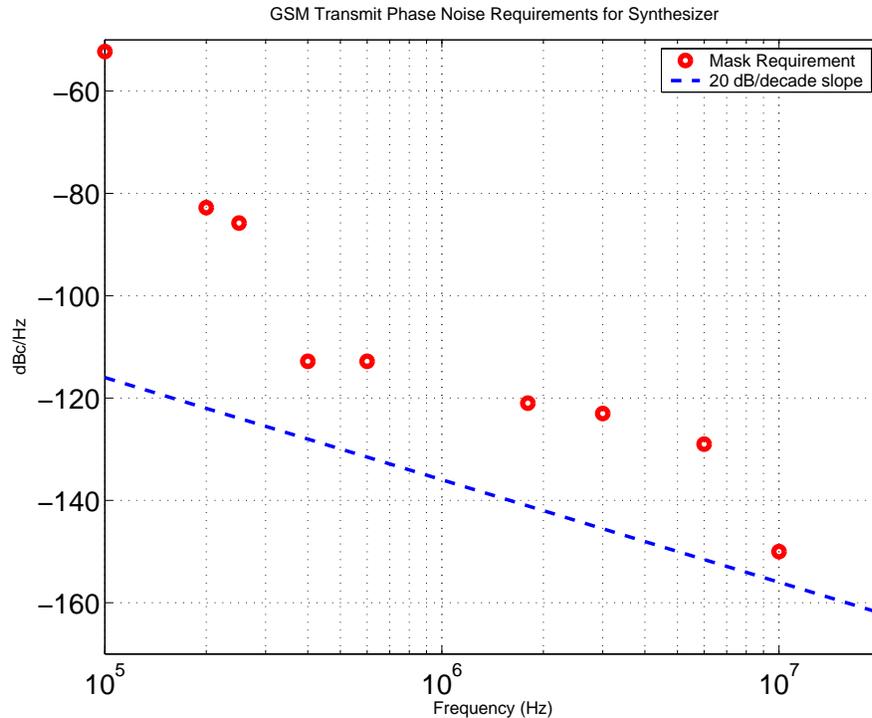
Upon completion of the installation, you will see icons on the Windows desktop corresponding to the PLL Design Assistant, CppSimView, and Sue2. Please read the “**CppSim (Version 3) Primer**” document, which is also at the same web address, to become acquainted with CppSim and its various components. Please read the manual “**PLL Design Using the PLL Design Assistant Program**”, which is located at <http://www.cppsim.com>, to obtain more information about the PLL Design Assistant.

Introduction

In this tutorial we will focus on the design of a Sigma-Delta frequency synthesizer intended for a direct conversion GSM cell phone transmitter. In this section we will review the key performance specifications, and then present the initial PLL specifications for the first-pass design. Note that the specifications are meant to be realistic, but are not necessarily the same as would be encountered for a practical synthesizer for GSM applications (i.e., the design procedure is meant to serve only as an *example* of doing fractional-N frequency synthesizer design).

A. GSM Synthesizer Specifications for a Transmitter Application

The key specifications are to achieve a settling time less than 150 microseconds with less than 10 ppm of frequency error, to achieve better than -80 dBc spurious performance, and to meet the phase noise requirements imposed by the GSM standard, which are shown below in both plot and table form. Note that the 20 dB/decade slope in the plot is shown for reference – the phase noise of the VCO will have this slope for much of its frequency range of interest. Also note that practical GSM synthesizers often strive for comparable settling times with less than 0.1 ppm of frequency error – we chose 10 ppm here since it works well for the example presented (i.e., the goal is to provide a tutorial, not a practical design).



GSM Transmit Phase Noise Specification for Synthesizer

100 kHz	200 kHz	250 kHz	400 kHz	600 kHz	1.8 MHz	3.0 MHz	6.0 MHz	10 MHz	20 MHz
-52.3 dBc/Hz	-82.8 dBc/Hz	-85.8 dBc/Hz	-112.8 dBc/Hz	-112.8 dBc/Hz	-121 dBc/Hz	-123 dBc/Hz	-129.0 dBc/Hz	-150.0 dBc/Hz	-162.0 dBc/Hz

B. Preliminary Synthesizer Design Specifications

The local PLL expert in your company has suggested that you try implementing the synthesizer as a fractional-N topology with first-cut specifications as described below:

- PLL System specifications
 - Bandwidth: 100 kHz (hopefully this will be high enough to meet the settling time requirement)
 - Order: 2 (we want a simple implementation)
 - Shape: Butterworth (there's no reason to think the other shapes are better at this point)
 - Type: 2 with $f_z/f_o = 1/8$ (these are fairly standard values for practical PLL's)
- Implementation details
 - Third order MASH Σ - Δ (avoid second order due to its problem with fractional spurs)
 - Reference Frequency: 26 MHz (this is standard for GSM applications)
 - Output Frequency: 900 MHz (this is set by having a direct conversion transmitter)
- PLL noise parameters

- PFD-referred noise: the PLL expert wasn't sure what you need here
- VCO: -165 dBc/Hz at 20 MHz frequency offset (you'll need this to meet the GSM noise specification with a bit of margin)

Your job is to examine the suitability of using a fractional-N synthesizer architecture with the given specifications, and to investigate the impact of several non-idealities at the system level. Fortunately, you just learned of some tools to make this task easier – the PLL Design Assistant and the CppSim behavioral simulator.

Noise Analysis using the PLL Design Assistant

To check the suitability of the above architecture, we will do noise analysis in four steps. The first is to do a basic check of noise performance with the system parameters given to us. The next is to tweak the system to account for any problems we encounter in meeting the noise specification. We will then investigate what levels of PFD-referred noise will be required to meet the specification. Finally, we will look at the impact of parameter variations on the noise performance to be sure that process and temperature variations do not push us out of spec.

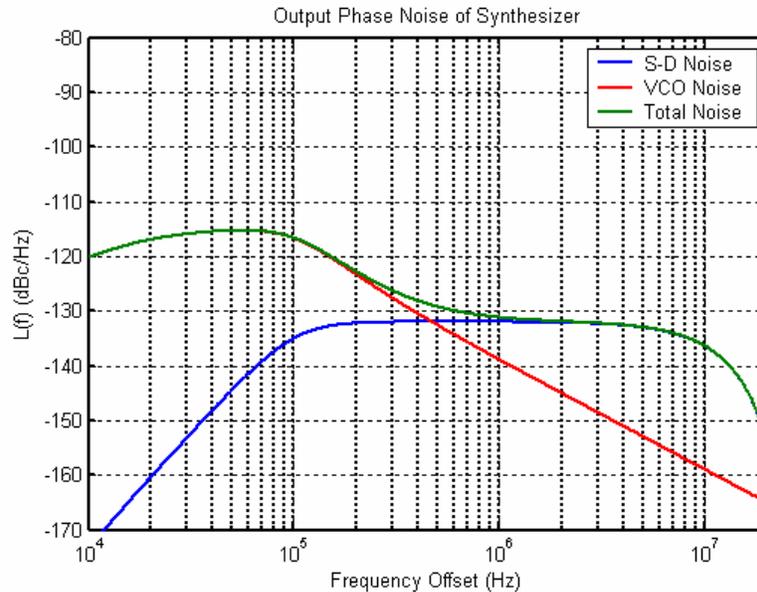
A. Performing Basic Noise Analysis

Put in the above parameters into the PLL Design Assistant, and then click on the **Noise Plot** radio button to see the estimated phase noise of the synthesizer. Now set the axis values of the noise plot to

- Noise axis limits: **10e3 20e6 -170 -80**

As a check to your work, the figures below illustrate what the PLL Design Assistant and resulting noise plot should look like upon completion of this task.

Dynamic Parameters		Noise Parameters	
fo: 100e3 Hz	paris. pole: _____ Hz <input type="button" value="On"/>	ref. freq: 26e6 Hz	
order: <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3	paris. Q: _____ <input type="button" value="On"/>	out freq: 900e6 Hz	
shape: <input checked="" type="radio"/> Butter <input type="radio"/> Bessel	paris. pole: _____ Hz <input type="button" value="On"/>	Detector: _____ dBc/Hz <input type="button" value="On"/>	
<input type="radio"/> Cheby1 <input type="radio"/> Cheby2 <input type="radio"/> Elliptical	paris. Q: _____ <input type="button" value="On"/>	VCO: -165 dBc/Hz <input type="button" value="On"/>	
ripple: _____ dB	paris. pole: _____ Hz <input type="button" value="On"/>	freq. offset: 20e6 Hz	
type: <input type="radio"/> 1 <input checked="" type="radio"/> 2	paris. pole: _____ Hz <input type="button" value="On"/>	S-D: <input type="radio"/> 1 <input type="radio"/> 2 <input type="button" value="On"/> <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="button" value="On"/>	
fz/fo: 1/8	paris. zero: _____ Hz <input type="button" value="On"/>		
	paris. zero: _____ Hz <input type="button" value="On"/>		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: 3.827e+010	alter: _____ <input type="button" value="On"/>	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: 1.566e+005 Hz	alter: _____ <input type="button" value="On"/>	<input type="radio"/> Step Response	<input checked="" type="radio"/> Noise Plot
fz: 1.250e+004 Hz	alter: _____ <input type="button" value="On"/>	10e3	20e6
Qp: _____	alter: _____ <input type="button" value="On"/>	-170	-80
<input type="button" value="Apply"/>		rms jitter: 244.372 fs	
PLL Design Assistant		Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)	

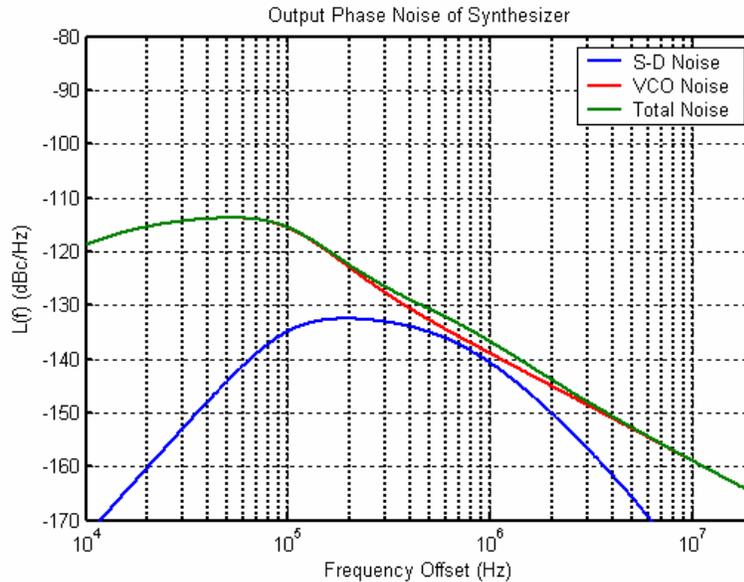


B. Adjusting the PLL Architecture to Meet the Noise Specifications

Notice that the GSM phase noise specifications are not met in the above design due to the Σ - Δ quantization noise. We will now examine two different approaches to fix this issue.

- Strategy 1: try changing the PLL order to be 3 instead of 2
 - The GSM phase noise specs are met, but now the PLL loop filter is much harder to implement
- Strategy 2: try adding some parasitic poles on purpose – one at 500 kHz, and one at 1 MHz
 - The GSM phase noise specs are again met, and this type of loop filter is fairly easy to implement (you just need to cascade the base loop filter with two RC filters). Note that the choice of frequencies here (i.e., 500 kHz and 1 MHz) is fairly arbitrary – you could also make these two frequencies the same value. A chief constraint is that parasitic pole frequencies should generally be a factor of 10 higher than the PLL bandwidth to allow proper pole placement of the dominant poles of the PLL. Putting both poles at 500 kHz might be pushing it, but you can try.

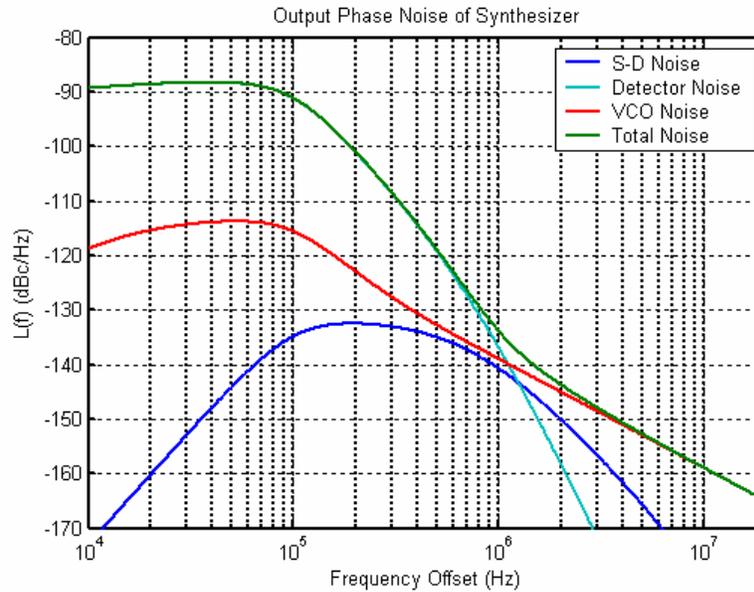
Let's go with Strategy 2, in which case the phase noise of the synthesizer should appear as shown below.



C. Adding in the Effect of Detector Noise

Now let's consider the performance requirements of PFD-referred noise caused by charge pump noise, divider jitter, and reference jitter. We will do so by using the PLL Design Assistant to determine the allowable detector noise for which the GSM transmit phase noise specification is still met. For convenience, the PLL Design Assistant specifies detector noise referenced to the output of the PLL. Once the detector noise value and PLL parameters (such as the charge pump current, etc.) are known, it is fairly straightforward to refer this noise to the charge pump current sources and/or reference/divider input nodes. Please see the manual, "PLL Design Using the PLL Design Assistant Program", for more details on this issue.

- Enter -85 dBc/Hz for **Detector** noise in the PLL Design Assistant.
 - Notice that the GSM phase noise specification is *not* met at the 400 kHz offset.
- Now enter -90 dBc/Hz for **Detector** noise
 - Verify that the GSM phase noise specification is now met at all frequencies. The noise plot from the PLL Design Assistant should look as shown below.



D. Accounting for Parameter Variations

In reality, there will be parameter variations in the implemented synthesizer due to process and temperature variations, and it will be important to check that such variations do not prevent the phase noise specification from being met. Given your experience with the integrated circuit process and PLL blocks you are dealing with, you decide that the following are reasonable values

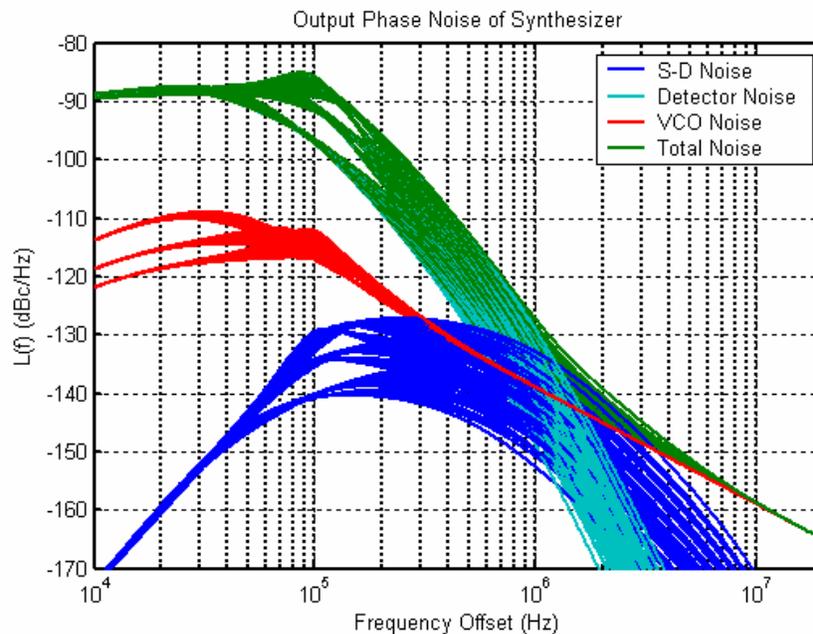
- +/- 40 % variation of the PLL open loop gain, K
 - Note that the open loop gain is a function of the gain of each individual PLL block, including the VCO (K_v), the loop filter (gain), and the charge pump (i_{cp})
- +/-30 % variation of the loop filter pole, f_p .
- +/-30 % variation of the parasitic poles at 500 kHz and 1 MHz.

Note that, in an actual design, the values of variation would be determined from detailed SPICE simulations of the actual transistor-level PLL blocks run across the different process and temperature corners of interest. Since we need to design the circuitry for these blocks *after* the preliminary system design has been completed, we will assume the above with knowledge that we will need to go back later in the design process and update the above parameters with better approximations. We see that the design process is iterative between system and circuit level analysis and simulations!

- Enter in the above variations into the PLL Design Assistant and then hit **Apply** to see the impact of these variations on the noise performance. As an example, to examine the impact of the open loop gain having two different values, 40% *below* nominal value and 40% *above* nominal value, you enter the expression **[-0.4 0.4]** into the alter: entry of the PLL Design Assistant. Since the program must consider all combinations, it may take some time for the results to appear. As a check to your work, the figures below show how the PLL Design Assistant should look upon entry of these values and the resulting phase noise plot.

- o Note that it is important to put the *nominal* value of parasitic poles *first* when they are entered with variation values since only the first value influences the computation of the open loop PLL parameters (i.e., **K**, **fp**, **fz**). In the example figure below, parasitic pole values of 500e3 and 1e6 are assumed when computing the open loop PLL parameters since the parasitic poles are entered as 500e3*[1.0 0.7 1.3] and 1e6*[1.0 0.7 1.3].

Dynamic Parameters		Noise Parameters	
fo: 100e3 Hz	paris. pole: 500e3*[1.0 0.7 1.3] Hz	ref. freq: 26e6 Hz	
order: 2	paris. Q: []	out freq.: 900e6 Hz	
shape: Butter	paris. pole: 1e6*[1.0 0.7 1.3] Hz	Detector: -90 dBc/Hz	<input type="checkbox"/> On
ripple: [] dB	paris. Q: []	VCO: -165 dBc/Hz	<input type="checkbox"/> On
type: 2	paris. pole: [] Hz	freq. offset: 20e6 Hz	
fz/fo: 1/8	paris. pole: [] Hz	S-D: 3	<input type="checkbox"/> On
	paris. pole: [] Hz		
	paris. zero: [] Hz		
	paris. zero: [] Hz		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: 3.272e+010	alter: [-0.4 0.4]	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: 2.173e+005 Hz	alter: [-0.3 0.3]	<input type="radio"/> Step Response	<input checked="" type="radio"/> Noise Plot
fz: 1.250e+004 Hz	alter: []	10e3	20e6
Qp: []	alter: []	-170	-80
<input type="button" value="Apply"/>		rms jitter: 2.287 ps (min), 4.105 ps (max)	
PLL Design Assistant		Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)	



We see from the above results that the GSM transmit noise specification is not met at 400 kHz, so that we must adjust our design. We have several options

- Lower the PLL bandwidth to get more suppression of the Detector noise at 400 kHz
- Lower the Detector noise to a suitable value

Due to our design requirement of having a settling time less than 150 microseconds, we will see later that we really can't afford to lower the PLL bandwidth further. Therefore, we will assume at this point that we can achieve lower Detector noise than -90 dBc/Hz (referred to the PLL output), and examine what value is necessary to meet the desired GSM specifications. In practice, the designer would need to run SPICE simulations on transistor level implementations of the charge pump and other circuits (and perform other design tasks) to determine what level of detector noise is practical. We see again that one must iterate between system and circuit level simulations and analysis in a practical design process.

A few iterations of changing the Detector noise entry of the PLL Design Assistant reveals that we need -95 dBc/Hz for the detector noise (referred to the PLL output) to achieve worst case PLL phase noise performance of -113 dBc/Hz at 400 kHz offset. (Note that is helpful to change the axis limits to verify this fact – simply change the noise plot axis limits to **399e3 401e3 -120 -110**).

Dynamic Analysis Using the PLL Design Assistant

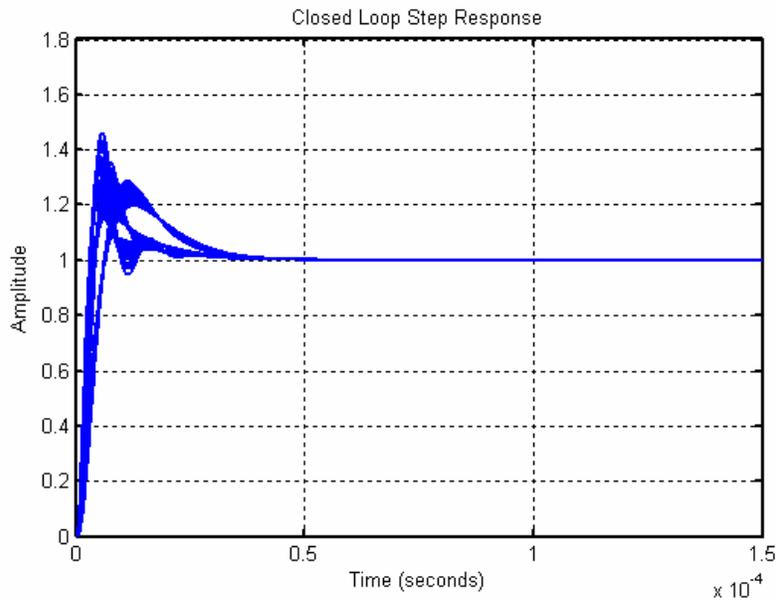
Now that we have met the GSM noise specifications, let us examine the issue of settling time. Here we will assume simplistically that we only need to consider the settling time in the case that the PLL starts and remains in frequency lock. This scenario is not realistic for a synthesizer that must settle from a power-up condition, but is reasonable for a fractional-N PLL that is already in lock (recall that the divide value of a fractional-N synthesizer can be ramped gradually to avoid loss of frequency lock).

A. Checking Stability

- In the PLL Design Assistant, click on the **Step Response** radio button to see the synthesizer time domain response to a unit step function.
 - Key in axis limits of: **0 150e-6 0 1.8**

Your results should match the figures below

Dynamic Parameters		Noise Parameters	
fo: <input type="text" value="100e3"/> Hz	paris. pole: <input type="text" value="500e3*[1.0 0.7 1.3]"/> Hz <input type="button" value="On"/>	ref. freq: <input type="text" value="26e6"/> Hz	<input type="button" value="On"/>
order: <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3	paris. Q: <input type="text"/>	out freq.: <input type="text" value="900e6"/> Hz	<input type="button" value="On"/>
shape: <input checked="" type="radio"/> Butter <input type="radio"/> Bessel	paris. pole: <input type="text" value="1e6*[1.0 0.7 1.3]"/> Hz <input type="button" value="On"/>	Detector: <input type="text" value="-95"/> dBc/Hz <input type="button" value="On"/>	<input type="button" value="On"/>
<input type="radio"/> Cheby1 <input type="radio"/> Cheby2 <input type="radio"/> Elliptical	paris. Q: <input type="text"/>	VCO: <input type="text" value="-165"/> dBc/Hz <input type="button" value="On"/>	<input type="button" value="On"/>
ripple: <input type="text"/> dB	paris. pole: <input type="text"/> Hz <input type="button" value="On"/>	freq. offset: <input type="text" value="20e6"/> Hz	<input type="button" value="On"/>
type: <input type="radio"/> 1 <input checked="" type="radio"/> 2	paris. pole: <input type="text"/> Hz <input type="button" value="On"/>	S-D: <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="button" value="On"/>	<input type="button" value="On"/>
fz/fo: <input type="text" value="1/8"/>	paris. zero: <input type="text"/> Hz <input type="button" value="On"/>		
	paris. zero: <input type="text"/> Hz <input type="button" value="On"/>		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: <input type="text" value="3.272e+010"/>	alter: <input type="text" value="[-0.4 0.4]"/> <input type="button" value="On"/>	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: <input type="text" value="2.173e+005"/> Hz	alter: <input type="text" value="[-0.3 0.3]"/> <input type="button" value="On"/>	<input checked="" type="radio"/> Step Response	<input type="radio"/> Noise Plot
fz: <input type="text" value="1.250e+004"/> Hz	alter: <input type="text"/> <input type="button" value="On"/>	<input type="text" value="0"/> <input type="text" value="150e-6"/> <input type="text" value="0"/> <input type="text" value="1.8"/>	
Qp: <input type="text"/>	alter: <input type="text"/> <input type="button" value="On"/>	rms jitter: <input type="text"/>	
PLL Design Assistant		Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)	



The above step response waveforms reveal that the PLL is stable across all parameter variations, and that it appears to settle quickly in the time window examined.

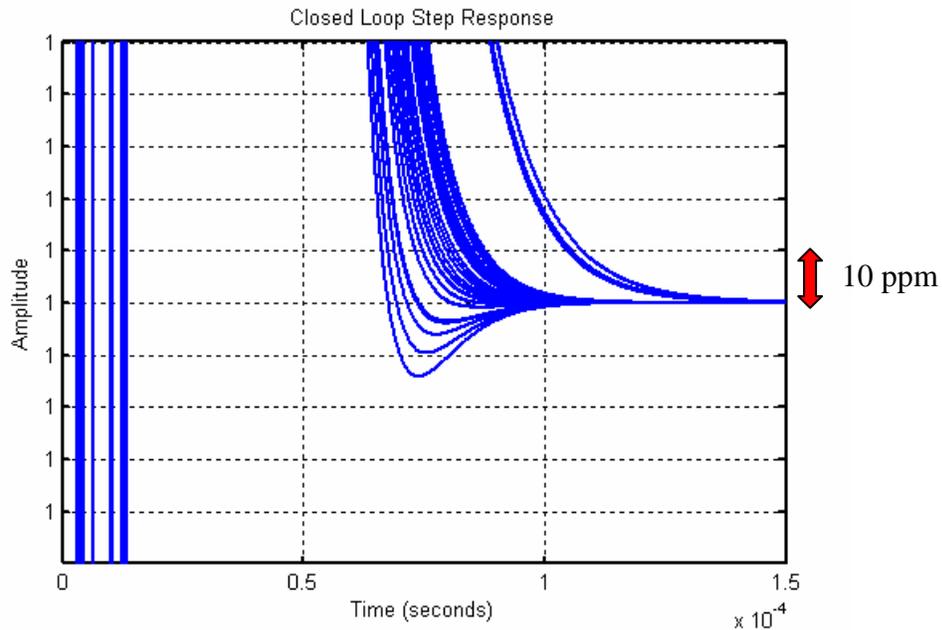
B. Checking the Settling Time

Although we have now verified that the synthesizer is stable across all parameter variations, we have not verified that it settles within 10 ppm of its target frequency in less than 150 microseconds. To do so, we simply need to change the scale parameters in the PLL Design Assistant on the y-axis to +/- 50 ppm about the final value of 1.

- Alter the axis values to: **0 150e-6 1-50e-6 1+50e-6**

Dynamic Parameters		Noise Parameters	
fo: <input type="text" value="100e3"/> Hz	paris. pole: <input type="text" value="500e3*[1.0 0.7 1.3]"/> Hz <input type="button" value="On"/>	ref. freq: <input type="text" value="26e6"/> Hz	
order: <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3	paris. Q: <input type="text"/>	out freq.: <input type="text" value="900e6"/> Hz	
shape: <input checked="" type="radio"/> Butter <input type="radio"/> Bessel	paris. pole: <input type="text" value="1e6*[1.0 0.7 1.3]"/> Hz <input type="button" value="On"/>	Detector: <input type="text" value="-95"/> dBc/Hz <input type="button" value="On"/>	
<input type="radio"/> Cheby1 <input type="radio"/> Cheby2 <input type="radio"/> Elliptical	paris. Q: <input type="text"/>	VCO: <input type="text" value="-165"/> dBc/Hz <input type="button" value="On"/>	
ripple: <input type="text"/> dB	paris. pole: <input type="text"/> Hz <input type="button" value="On"/>	freq. offset: <input type="text" value="20e6"/> Hz	
type: <input type="radio"/> 1 <input checked="" type="radio"/> 2	paris. pole: <input type="text"/> Hz <input type="button" value="On"/>	S-D: <input type="radio"/> 1 <input type="radio"/> 2 <input type="button" value="On"/>	<input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="button" value="On"/>
fz/fo: <input type="text" value="1/8"/>	paris. zero: <input type="text"/> Hz <input type="button" value="On"/>		
	paris. zero: <input type="text"/> Hz <input type="button" value="On"/>		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: <input type="text" value="3.272e+010"/>	alter: <input type="text" value="[-0.4 0.4]"/> <input type="button" value="On"/>	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: <input type="text" value="2.173e+005"/> Hz	alter: <input type="text" value="[-0.3 0.3]"/> <input type="button" value="On"/>	<input checked="" type="radio"/> Step Response	<input type="radio"/> Noise Plot
fz: <input type="text" value="1.250e+004"/> Hz	alter: <input type="text"/> <input type="button" value="On"/>	<input type="text" value="0"/>	<input type="text" value="150e-6"/> <input type="text" value="1-50e-6"/> <input type="text" value="1+50e-6"/>
Qp: <input type="text"/>	alter: <input type="text"/> <input type="button" value="On"/>	rms jitter: <input type="text"/>	
PLL Design Assistant		Written by Michael Perrott (http://www.mtl.mit.edu/~perrott)	

You should obtain a settling time plot as shown below.



We see that the settling response meets the 10 ppm requirement within 150 microseconds.

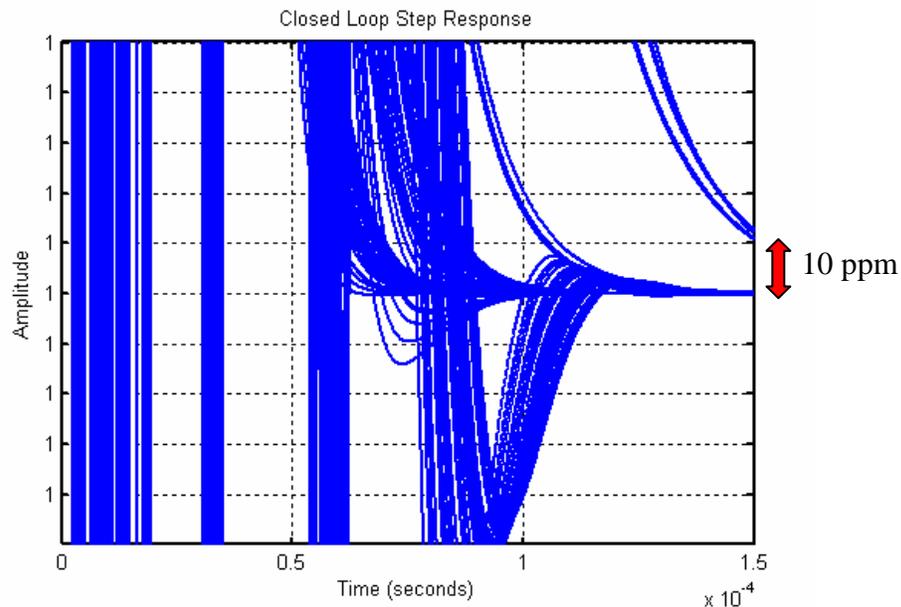
C. Examining the Influence of Zero Variations

After giving more thought to the design, you realize that you forgot to include the impact of the variations of the PLL zero due to process and temperature variations. It seems reasonable to assume similar variation as for the f_p parameter, which is $\pm 30\%$.

- Enter $\pm 30\%$ variation into the **alter:** form for f_z , as shown below.

Dynamic Parameters		Noise Parameters	
fo: <input type="text" value="100e3"/> Hz	paris. pole: <input type="text" value="500e3*[1.0 0.7 1.3]"/> Hz <input type="button" value="On"/>	ref. freq: <input type="text" value="26e6"/> Hz	
order: <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3	paris. Q: <input type="text"/>	out freq.: <input type="text" value="900e6"/> Hz	
shape: <input checked="" type="radio"/> Butter <input type="radio"/> Bessel	paris. pole: <input type="text" value="1e6*[1.0 0.7 1.3]"/> Hz <input type="button" value="On"/>	Detector: <input type="text" value="-95"/> dBc/Hz <input type="button" value="On"/>	
<input type="radio"/> Cheby1 <input type="radio"/> Cheby2 <input type="radio"/> Elliptical	paris. Q: <input type="text"/>	VCO: <input type="text" value="-165"/> dBc/Hz <input type="button" value="On"/>	
ripple: <input type="text"/> dB	paris. pole: <input type="text"/> Hz <input type="button" value="On"/>	freq. offset: <input type="text" value="20e6"/> Hz	
type: <input type="radio"/> 1 <input checked="" type="radio"/> 2	paris. pole: <input type="text"/> Hz <input type="button" value="On"/>	S-D: <input type="radio"/> 1 <input type="radio"/> 2 <input type="button" value="On"/>	<input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="button" value="On"/>
fz/fo: <input type="text" value="1/8"/>	paris. zero: <input type="text"/> Hz <input type="button" value="On"/>		
	paris. zero: <input type="text"/> Hz <input type="button" value="On"/>		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: <input type="text" value="3.272e+010"/>	alter: <input type="text" value="[-0.4 0.4]"/> <input type="button" value="On"/>	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: <input type="text" value="2.173e+005"/> Hz	alter: <input type="text" value="[-0.3 0.3]"/> <input type="button" value="On"/>	<input checked="" type="radio"/> Step Response	<input type="radio"/> Noise Plot
fz: <input type="text" value="1.250e+004"/> Hz	alter: <input type="text" value="[-0.3 0.3]"/> <input type="button" value="On"/>	<input type="text" value="0"/>	<input type="text" value="150e-6"/> <input type="text" value="1-50e-6"/> <input type="text" value="1+50e-6"/>
Qp: <input type="text"/>	alter: <input type="text"/> <input type="button" value="On"/>	<input type="text" value="rms jitter:"/>	
PLL Design Assistant		Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)	

You should see a plot similar to that shown below, which unfortunately reveals that the settling time specification is no longer met! Even though it is close, we don't want to live on the edge in an actual production situation.



At this point, you have several options to correct this problem:

- Increase the PLL bandwidth (and keep $G(f)$ second order)
 - Unfortunately, this leads to an increase in the impact of both Detector and $\Sigma-\Delta$ quantization noise on the output PLL phase noise.
- Increase the PLL bandwidth, but also make $G(f)$ third order
 - Unfortunately, making $G(f)$ third order will greatly increase the implementation complexity of the system. Also, going to third order adds yet another parameter, Q_p , which will also be sensitive to process and temperature variations.

- Try to create circuit designs for the loop filter implementation that have less variation than the assumed value of $\pm 30\%$, and/or strive for less variation in the open loop gain.
 - This is a nice approach, but may not be easy!
- Apply architectural innovation to solve this problem
 - This is usually the best way to go. One common approach is to dynamically change the PLL bandwidth such that it is increased during settling, and lowered when in lock (to achieve the required noise performance). One recent example of such an approach is found in:

M. Keaveney, P. Walsh, M. Tuthill, C. Lyden, B. Hunt, “A 10 μs Fast Switching PLL Synthesizer for GSM/EDGE Base-Stations”, ISSCC, Feb. 2004

Given the above options, we will assume that further investigation must be done to solve the settling time issue. Note that practical GSM synthesizers will often have even more stringent settling time requirements than specified in this tutorial, such as demanding < 0.1 ppm of frequency error in less than 150 microseconds. Therefore, we see that the settling time performance metric is not a trivial one to meet.

We want to be aware of any other issues that must be kept in mind before circuit design begins, so we now embark on further system level investigation of the system through the use of behavioral simulation.

Preliminary CppSim Simulation Analysis

CppSim has several synthesizer examples as part of its installation which can be leveraged in our system-level investigation of the synthesizer. In this section, we first modify one of those examples in Sue2 to properly reflect the parameters of our system, and then create a CppSim simulation file to produce signals for both dynamic and noise analysis. We will then run CppSim, plot the simulated step response and phase noise of the synthesizer, and compare our results to that obtained from the PLL Design Assistant.

A. Entering the Design into Sue2

- Start up Sue2, and select the **sd_synth_tristate_fast** cell within the **Synthesizer_Examples** library.
- Select **File->Save as** and then save the file as **project_synth** within the **Synthesizer_Examples** directory.

The PLL Design Assistant has provided the *overall* system parameters for the synthesizer, but there are still many detailed PLL parameters to consider. You go back to the expert PLL designer, and she provides you with first-cut values for the PLL parameters as listed below. In practice, these values are adjusted as circuit level design of the individual components progresses, so we again catch a glimpse of the iterative cycle between system and circuit level design that must occur for a practical design.

- Reference frequency source:

- Center frequency = 26 MHz (this value is fixed for most GSM applications)
- $K_v = 1$ (this is arbitrary, and doesn't matter for the simulator since the phase input is always 0 for this block)
- VCO
 - Center frequency = 900 MHz (fixed by GSM standard)
 - $K_v = 50 \text{ MHz/V}$ (this would be determined by VCO implementation)
 - Noise: to meet the GSM specification with some margin, let us assume that we can achieve -165 dBc/Hz at 20 MHz offset.
- Divider – this is buried within the VCO implementation in the CppSim schematic. Its nominal value will be set by the input to the **sd_modulator** block, which is, in turn, fed from the **step** block.
 - Nominal divide value = $900 \text{ MHz} / 26 \text{ MHz} = 34.6154$
 - Therefore, we will set **in_gl** of the step block to 34.6154 within the **test.par** file
- Phase-Frequency Detector – Tristate design
 - $\alpha = 1$ for given topology (see PLL Design Assistant manual for explanation of the meaning of this parameter). We'll use this value in the calculation of the loop filter gain below.
 - $\text{reset_delay} = 2.5 \text{ ns}$ (we would need to decrease the time step, T_s , of the simulator to decrease this further. For now, this is OK, but would need to adjust this value once SPICE simulations of the PFD reveals its true value).
- Charge pump
 - $i_val = 100 \text{ microamps}$ for both the up and down current sources (this is a function of desired noise performance you need and the amount of loop filter capacitance available to you – you need to do noise analysis of the charge pump transistors in SPICE, and other circuit level design tasks, to get a good estimate of this. A value of 100 microamps is a reasonable assumption to begin with).
 - $i_variance = 0$ (let's ignore detector noise for now)
- Sigma-Delta Modulator – assume MASH structure (as currently implemented in the schematic)
 - $\text{order} = 3$
- Loop Filter – consists of a lead/lag filter followed by two RC filters
 - RC filters (rcfilter block in CppSimModules)
 - One with $f_o = 500 \text{ kHz}$, and one with $f_o = 1 \text{ MHz}$ (as previously assumed)
 - An important point here: cascading two rfilter blocks in CppSim is *not* the same as cascading two RC networks – CppSim does not account for loading between blocks. CppSim is simply implementing two first order poles that are placed at the values specified – 500 MHz and 1 MHz. If you cascade two RC networks whose isolated pole frequencies were located at these values, then the frequencies would shift when you

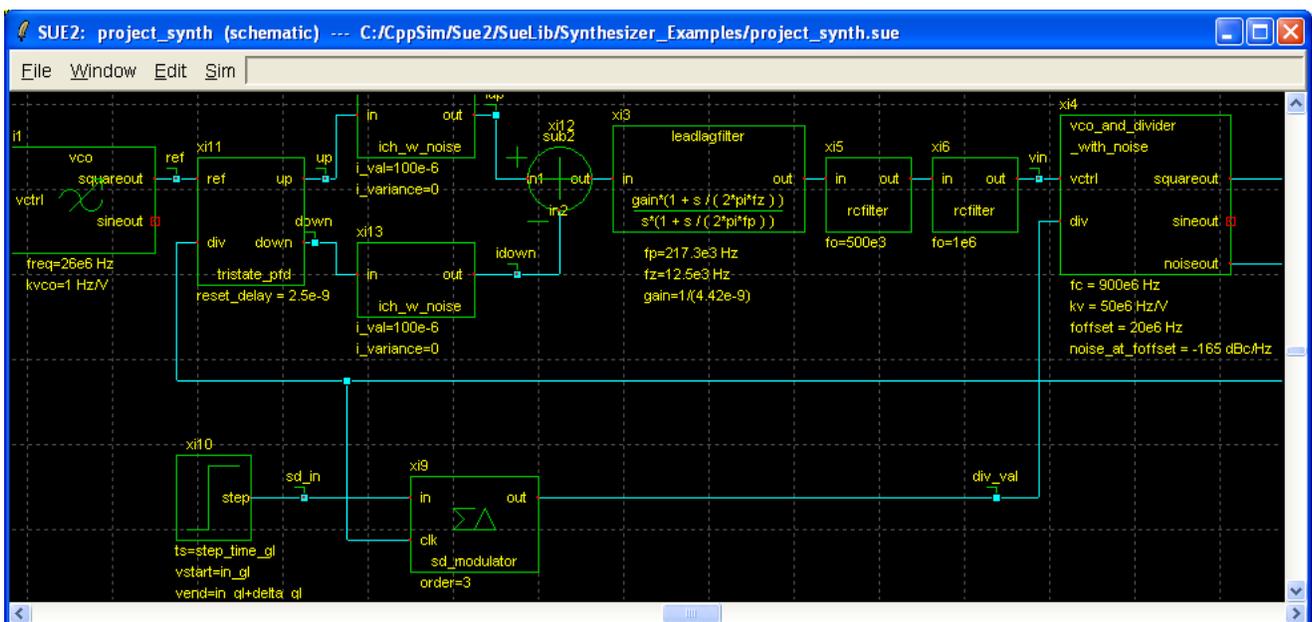
connected them due to loading effects. One can figure out how to set the RC values of a cascaded RC filter network to match the desired pole values by simply solving for the transfer function of the overall network – this strategy should be followed in the circuit implementation of the loop filter.

- Lead/lag filter – use values from PLL Design Assistant calculated previously
 - $f_p = 217.3 \text{ kHz}$, $f_z = 12.5 \text{ kHz}$ (from the PLL Design Assistant)
 - gain – must be calculated from PLL parameters

$$\text{gain} = K \frac{N_{\text{nom}}}{\alpha K_v I_{\text{cp}}}$$

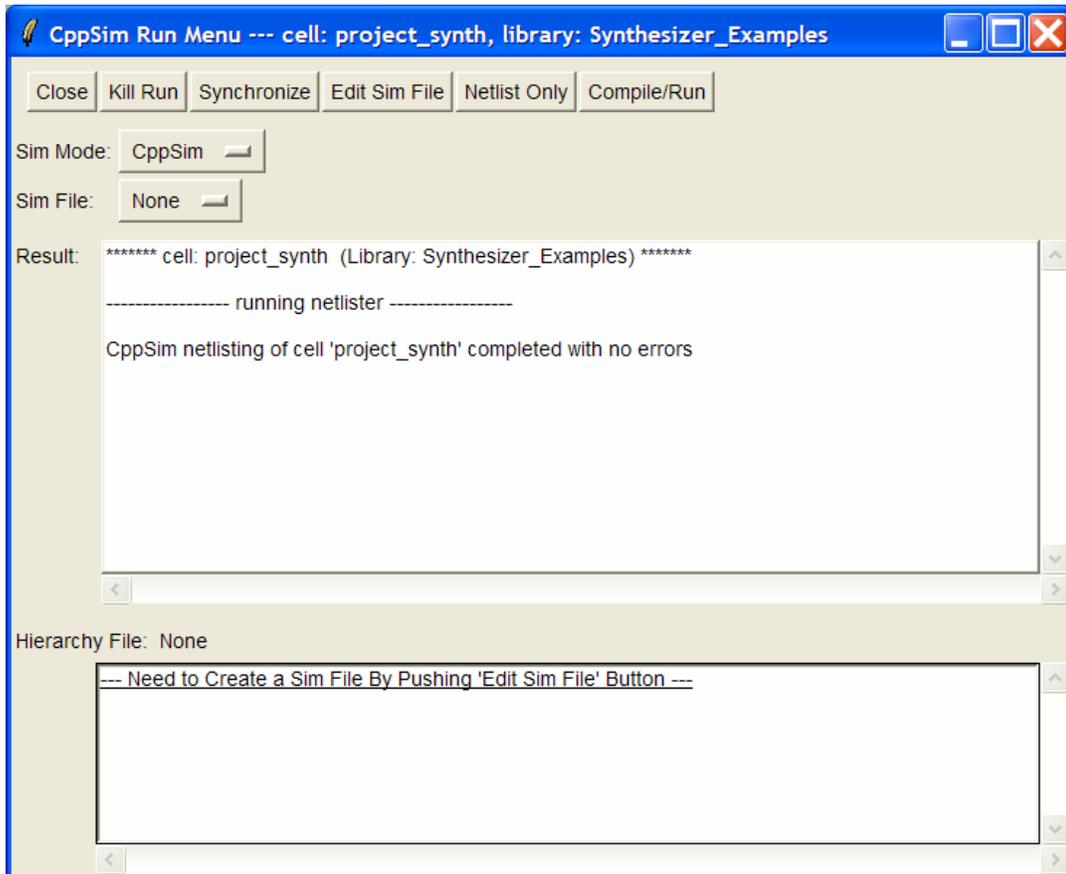
- $K = 3.272 \times 10^{10}$ (from the PLL Design Assistant)
- $N_{\text{nom}} = 34.6$, $\alpha = 1$, $K_v = 50 \times 10^6$, $I_{\text{cp}} = 100 \times 10^{-6}$ (from above)
- **Result:** $\text{gain} = 1/(4.42 \times 10^{-9})$
 - This corresponds to 4.4 nF of capacitance in the loop filter! Therefore, the loop filter capacitance would likely need to be implemented off-chip in this design. Note that the above formula allows us to see that reduction of the charge pump current, I_{cp} , would also allow reduction of the capacitance value. However, reduction of I_{cp} would, in turn, lead to an increased impact of charge pump noise on the PLL output. The best value for I_{cp} , and loop filter capacitance, would need to be decided later after SPICE simulations of the charge pump and other PLL blocks are performed.

Implementation of the above parameters and PLL configuration into Sue2 should yield a circuit whose primary circuit blocks are similar to those shown below.



B. Setting Up the CppSim Simulation File

- Within the Sue2 window, select **Tools**→ **CppSim Simulation**. You should see the Cppsim Run Menu that pops up. Left click on the **Edit Sim File** button, and an Emacs window should pop up. Now open CppSimView and make sure it synchronizes to your new schematic. If not, left-click the **Synch** button on the CppSimView window to do it.



- Within the Emacs windows that pops up, enter the following values

```
// Number of time steps needs to be large enough to achieve noise analysis
num_sim_steps: 3e6

// Choose a time step such that its inverse is over a factor ten higher than the reference
frequency
Ts: 1/400e6

// For the transient response, look at a 150 microsecond span starting at 50 microseconds
output: test start_time=50e-6 end_time=200e-6

probe: vin sd_in

// For the noise performance, start recording after transients have died away
```

```
output: test_noise start_time=200e-6
```

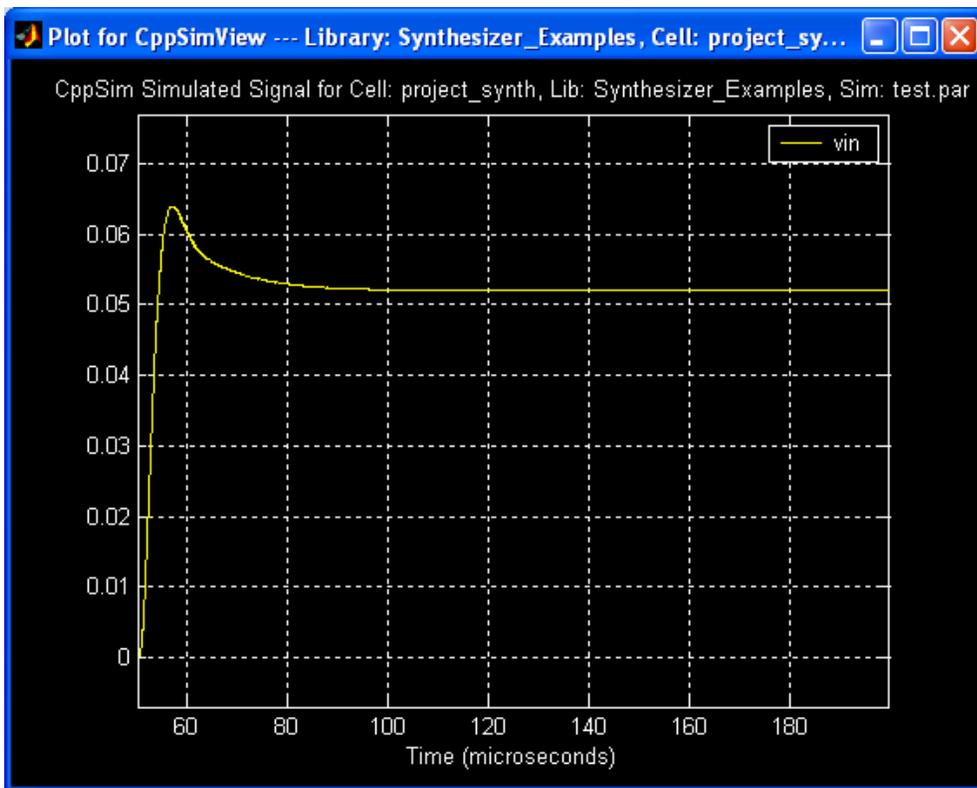
```
probe: noiseout
```

```
// Set the nominal divide value to 34.6154, and apply a step of 0.1 at 50 microseconds
```

```
global_param: in_gl=34.6154 delta_gl=0.1 step_time_gl=50e-6
```

C. Examining the Simulated Step Response

- Now run the CppSim simulation, and then plot signal **vin** to observe the transient response. You should obtain a plot as shown below.



Compare the above step response to that produced by the PLL Design Assistant.

- To do so, go back to the PLL Design Assistant, turn off the **alter:** statements, and remove the variations from the parasitic poles.

The PLL Design Assistant should then look as shown below, and the corresponding step response should match that of CppSim quite well.

Dynamic Parameters		Noise Parameters	
fo: <input type="text" value="100e3"/> Hz	paris.pole: <input type="text" value="500e3"/> Hz <input type="button" value="On"/>	ref.freq: <input type="text" value="26e6"/> Hz	
order: <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3	paris.Q: <input type="text"/> <input type="button" value="On"/>	out.freq: <input type="text" value="900e6"/> Hz	
shape: <input checked="" type="radio"/> Butter <input type="radio"/> Bessel	paris.pole: <input type="text" value="1e6"/> Hz <input type="button" value="On"/>	Detector: <input type="text" value="-95"/> dBc/Hz <input type="button" value="On"/>	
<input type="radio"/> Cheby1 <input type="radio"/> Cheby2 <input type="radio"/> Elliptical	paris.Q: <input type="text"/> <input type="button" value="On"/>	VCO: <input type="text" value="-165"/> dBc/Hz <input type="button" value="On"/>	
ripple: <input type="text"/> dB	paris.pole: <input type="text"/> Hz <input type="button" value="On"/>	freq.offset: <input type="text" value="20e6"/> Hz	
type: <input type="radio"/> 1 <input checked="" type="radio"/> 2	paris.pole: <input type="text"/> Hz <input type="button" value="On"/>	S-D: <input type="radio"/> 1 <input type="radio"/> 2 <input type="button" value="On"/>	<input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="button" value="On"/>
fz/fo: <input type="text" value="1/8"/>	paris.pole: <input type="text"/> Hz <input type="button" value="On"/>		
	paris.pole: <input type="text"/> Hz <input type="button" value="On"/>		
	paris.zero: <input type="text"/> Hz <input type="button" value="On"/>		
	paris.zero: <input type="text"/> Hz <input type="button" value="On"/>		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: <input type="text" value="3.272e+010"/>	alter: <input type="text"/> <input type="button" value="On"/>	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: <input type="text" value="2.173e+005"/> Hz	alter: <input type="text"/> <input type="button" value="On"/>	<input checked="" type="radio"/> Step Response	<input type="radio"/> Noise Plot
fz: <input type="text" value="1.250e+004"/> Hz	alter: <input type="text"/> <input type="button" value="On"/>	<input type="text" value="0"/> <input type="text" value="150e-6"/> <input type="text" value="0"/> <input type="text" value="1.8"/>	
Qp: <input type="text"/>	alter: <input type="text"/> <input type="button" value="On"/>	rms jitter: <input type="text"/>	
PLL Design Assistant		Written by Michael Perrott (http://www.mtl.mit.edu/~perrott)	

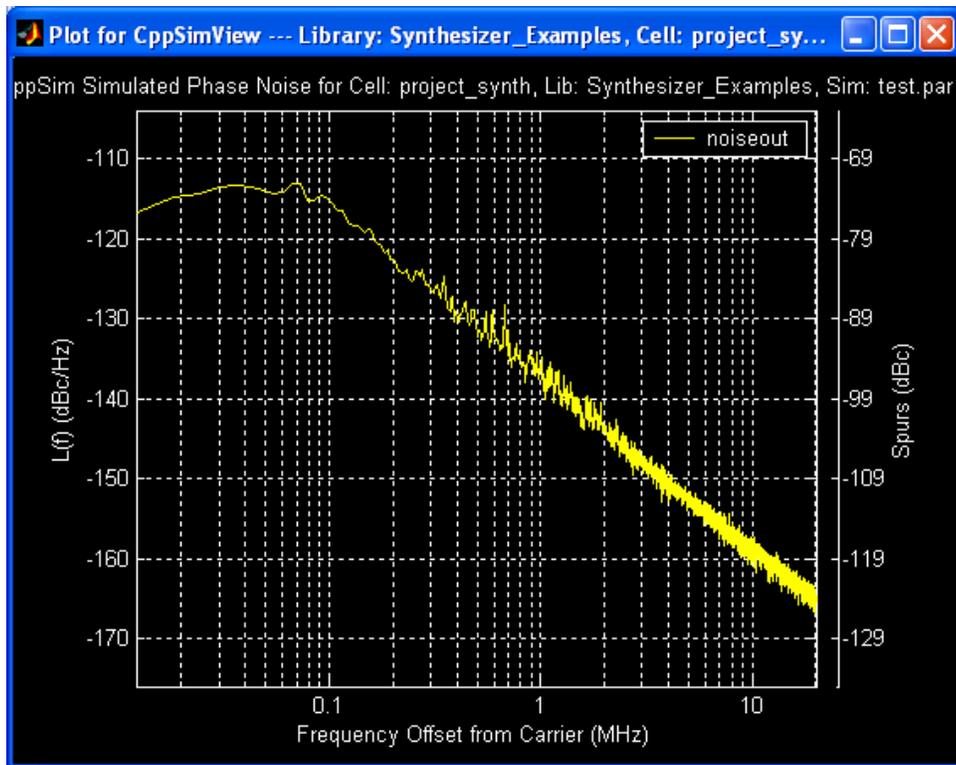
D. Examining the Simulated Phase Noise

In CppSimView, switch to the **test_noise.tr0** output file, and then run the plot function

- `plot_pll_phasenoise(x,10e3,20e6,'noiseout')`

The above function specifies that we look at the output phase noise of the PLL at offset frequencies that span from 10 kHz to 20 MHz. You should see a plot similar to that shown below. Note that the frequency does not span all the way down to 10 kHz due to the limited number of points at frequencies in that range. To see this, rerun the plot function and have it span from *1 kHz* to 20 MHz, and then left-click on the **Zoom** button and then on **(L)ineStyle** – you should see that there are very few points at low frequencies, and that the lowest frequency point is quite inaccurate due to the spectral density calculation method employed. The issue of limited low frequency points is improved if we substantially increase the number of simulation sample points (and decimate the resulting signals using the **trigger** command in the **test.par** file to keep a reasonable output file size), but doing so carries the negative effect of increasing the time for the simulation.

Note that the very low frequency sample points in the simulated phase noise plot have a large amount of variance compared to their high frequency counterparts, and are therefore unreliable in their accuracy (i.e., don't make conclusions that you are seeing $1/f$ noise or other phenomenon based on the lowest frequency portion of the simulated phase noise plots). To get better accuracy, you need to increase the number of simulation sample points (at the expense of longer simulation times).



- Now go back to the PLL Design Assistant, and plot the phase noise of the synthesizer with the detector noise turned off.

The PLL Design Assistant should look as shown below, and the results between CppSim and PLL Design Assistant should match quite well.

Dynamic Parameters		Noise Parameters	
fo: 100e3 Hz	order: <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3	ref. freq: 26e6 Hz	out freq.: 900e6 Hz
shape: <input checked="" type="radio"/> Butter <input type="radio"/> Bessel	paris. pole: 500e3 Hz	Detector: <input type="text"/> dBc/Hz	VCO: -165 dBc/Hz
ripple: <input type="text"/> dB	paris. Q: <input type="text"/>	freq. offset: 20e6 Hz	S-D: <input type="radio"/> 1 <input type="radio"/> 2 <input checked="" type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
type: <input type="radio"/> 1 <input checked="" type="radio"/> 2	paris. pole: 1e6 Hz		
fz/fo: 1/8	paris. Q: <input type="text"/>		
	paris. pole: <input type="text"/> Hz		
	paris. pole: <input type="text"/> Hz		
	paris. zero: <input type="text"/> Hz		
	paris. zero: <input type="text"/> Hz		
Resulting Open Loop Parameters		Resulting Plots and Jitter	
K: 3.272e+010	alter: <input type="text"/> On	<input type="radio"/> Pole/Zero Diagram	<input type="radio"/> Transfer Function
fp: 2.173e+005 Hz	alter: <input type="text"/> On	<input type="radio"/> Step Response	<input checked="" type="radio"/> Noise Plot
fz: 1.250e+004 Hz	alter: <input type="text"/> On	10e3	20e6
Qp: <input type="text"/>	alter: <input type="text"/> On	-170	-80
		rms jitter: 186.311 fs	
PLL Design Assistant		Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)	

Advanced CppSim Simulation Analysis

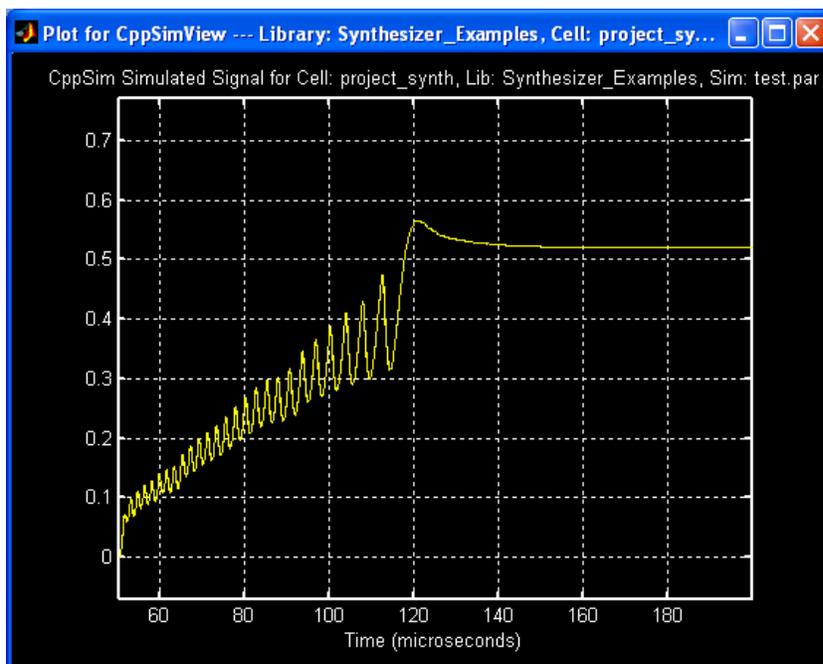
Let us now examine some non-ideal effects that are not predicted by the PLL Design Assistant, but can be captured by the time-domain simulation approach that CppSim offers. We will first observe the impact of increasing the divider step value so that the PLL loses frequency lock and creates a condition of cycle slipping. Mismatch between the up and down current sources will then be introduced in the charge pump, which will reveal increased phase noise due to noise folding of the Σ - Δ quantization noise. To deal with such mismatch, it is common to purposefully introduce a phase offset into the synthesizer through the addition of current at the charge pump output – we will show the positive and negative impact of this approach. Finally, we will change the input value to the Σ - Δ modulator in order to observe fractional spurs that are produced.

A. Observing Cycle Slipping

Let us look at the effect of introducing a larger step value in the divide value on the transient performance of the synthesizer.

- Click on the **Edit Sim File** button in CppSim Run Menu, and change the **delta_gl** parameter in **test.par** from 0.1 to 1.
- Rerun the CppSim simulation, load the **test.tr0** (rather than **test_noise.tr0**) output file, and then run the **plotsig(...)** function on signal **vin**.

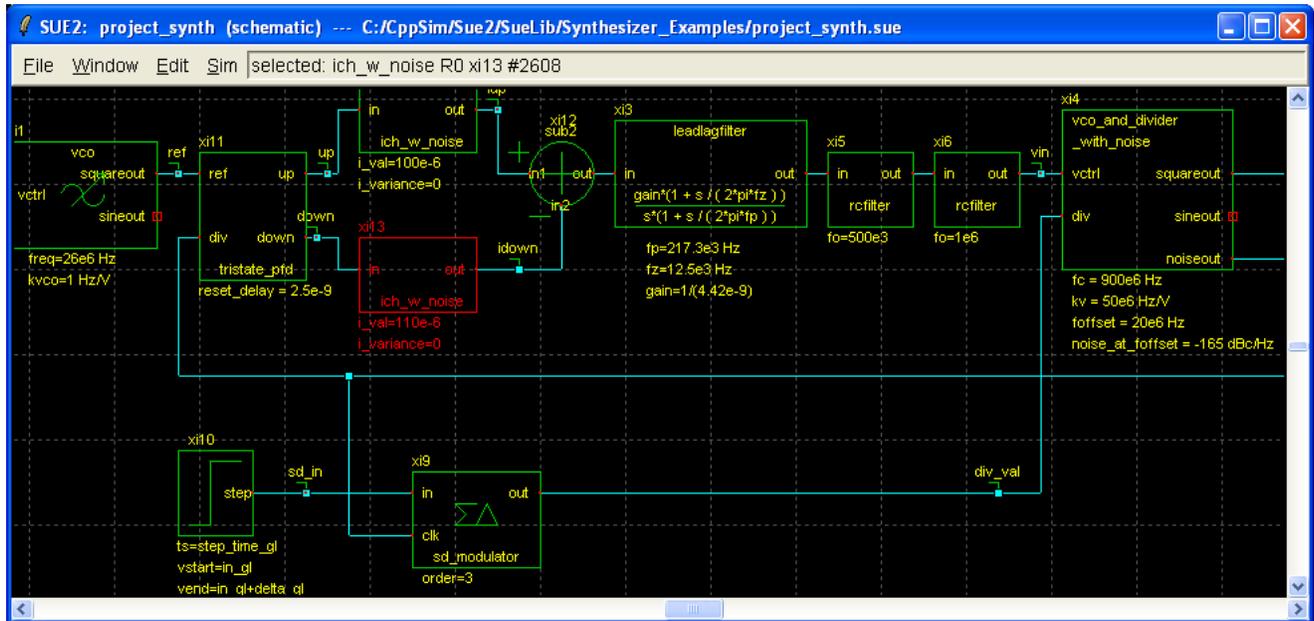
The resulting step response should be similar to that shown below – it no longer looks as predicted by the PLL Design Assistant! The reason for the discrepancy is that the PLL Design Assistant assumes a linearized PLL model that is valid only when the PLL is in lock. In this case, we have put a large enough step to take the synthesizer out of lock, and are seeing the effects of cycle slipping that occurs before it eventually returns to a locked state.



B. Examining the Effects of Charge Pump Mismatch

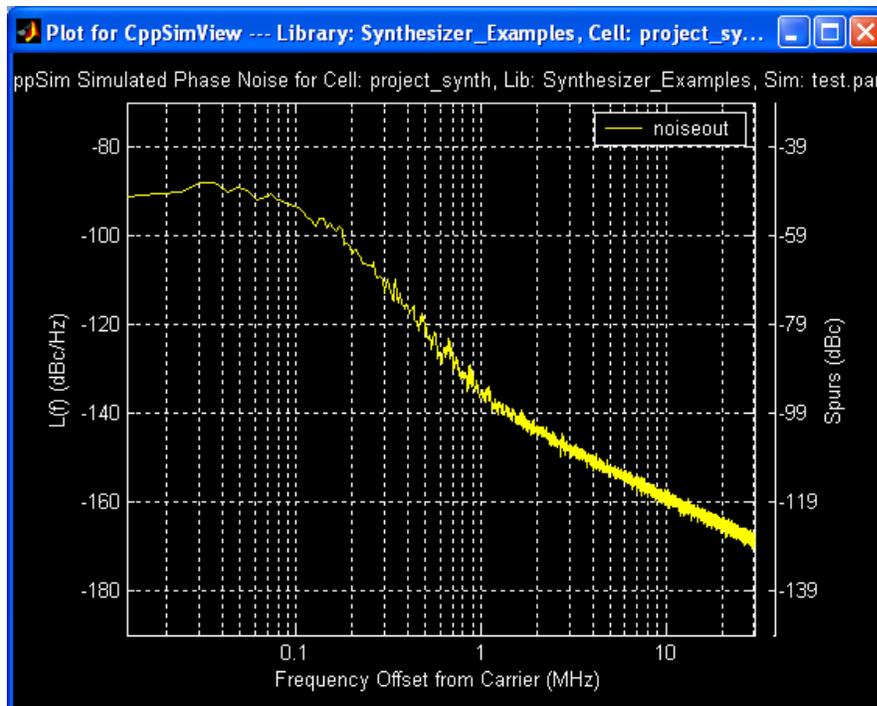
Now let us examine the impact of charge pump current mismatch on the phase noise of the synthesizer.

- Within Sue2, change the bottom charge current from 100 microamps to 110 microamps. This change causes there to be a mismatch between the up and down charge pump currents. Note that the updated Sue2 schematic should match that shown below.



- Rerun the CppSim simulation, load in the **test_noise.tr0** output file, and run the **plot_pll_phasenoise(...)** function on signal **noiseout**. This time, increase the frequency scale in **plot_pll_phasenoise(...)** to span from 10 kHz to 30 MHz.

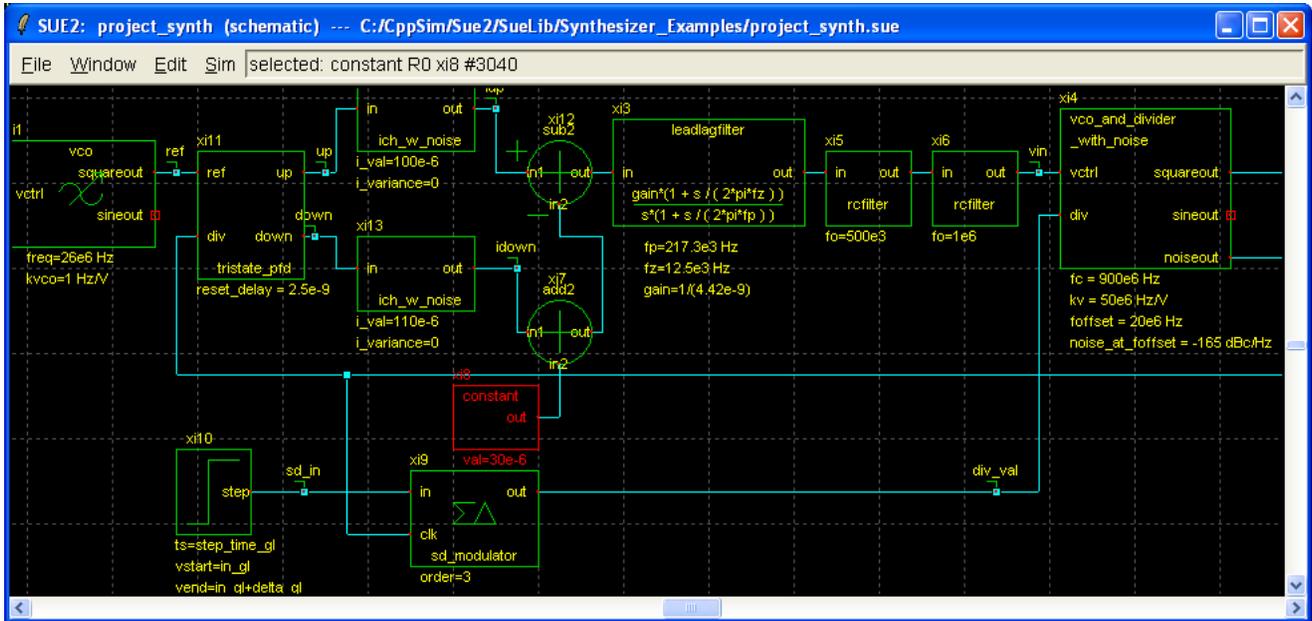
The resulting phase noise plot should be similar to that shown below. Comparing this plot to the earlier one (with matched up and down current sources) reveals significantly more noise at low frequencies. This effect is caused by the effective nonlinearity introduced by the mismatched current, which causes the highpass shaped Sigma-Delta quantization noise to be folded down into lower frequencies.



C. Moving the Nominal Phase Error Away from Zero

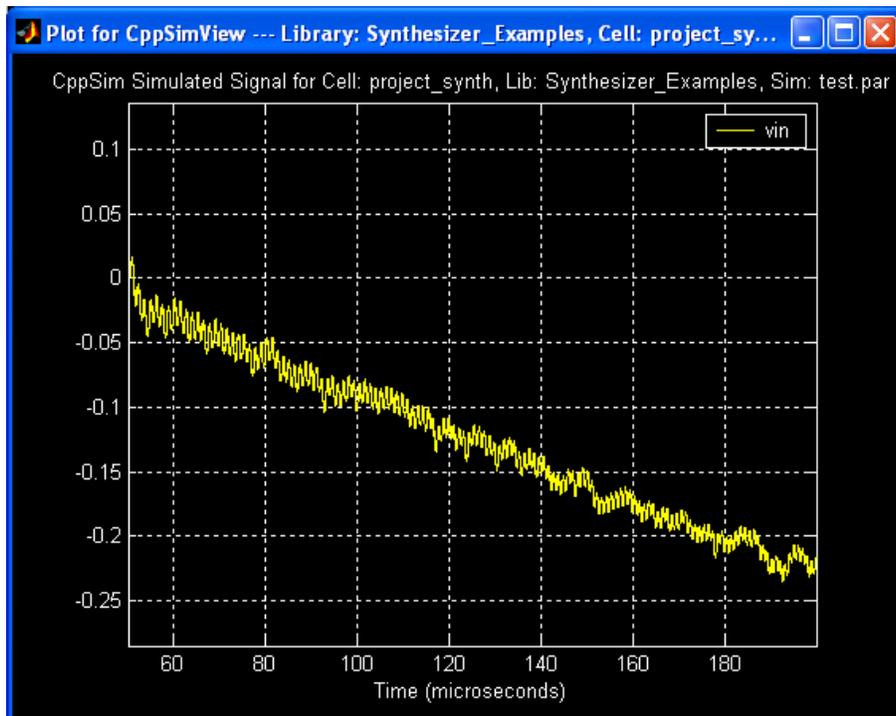
- Now go back to the Sue2 schematic, and add in a constant offset current into the charge pump output of 30 microamps (as shown in the schematic figure below).

The effect of this offset current is to shift the relative phase of the reference and divider outputs, so that we are no longer in the portion of the tristate phase detector characteristic that experiences variable up *and* down pulses. In the case below, the down pulse will become fixed in width according to the reset delay, and the up pulse be expanded such that its *average* width is roughly 30% of the reference period (i.e., not considering the effect of the small-width down pulse, the up pulse must be on 30% of the time in order to offset the 30 microamp offset and make the input to the leadlagfilter have a zero average). This schematic change will have some positive and negative effects, as we will now explore.



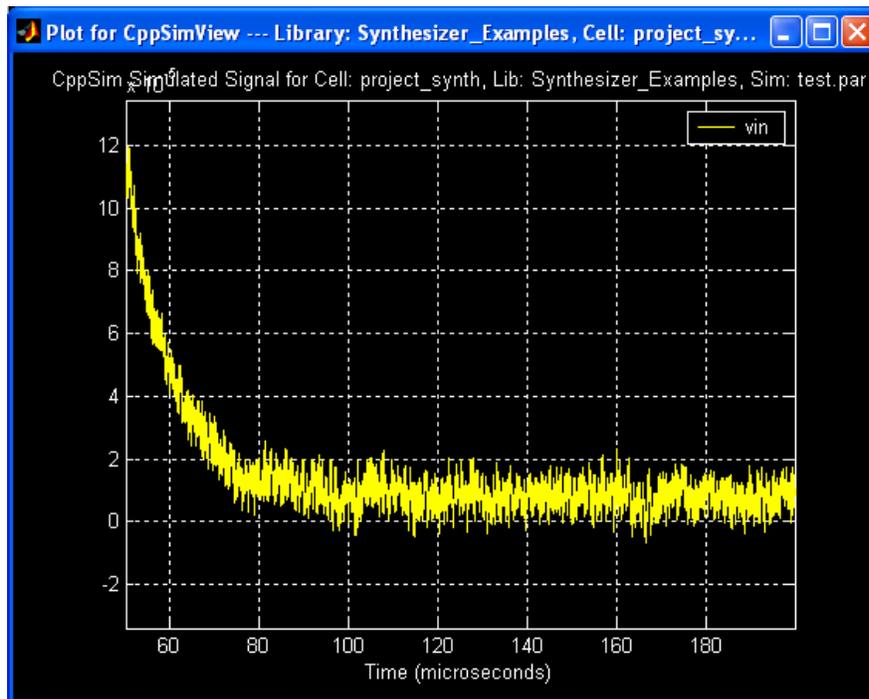
- Run CppSim, and then use the `plotsig(...)` function to plot signal `vin`.

Your plot should match the one shown below, and reveals that the PLL is no longer locking properly. The reason for this behavior is that the offset added to the charge pump output counteracts the offset added by the frequency detection circuitry of the PFD when the PLL is out of lock. In this case, the charge pump offset value is so large (at 30 microamps) that it overrides the frequency detector action to the point that frequency lock is not achieved. In practice, one would not put such a large offset into the charge pump output, but this exercise hopefully illustrates one of the impacts of doing so.



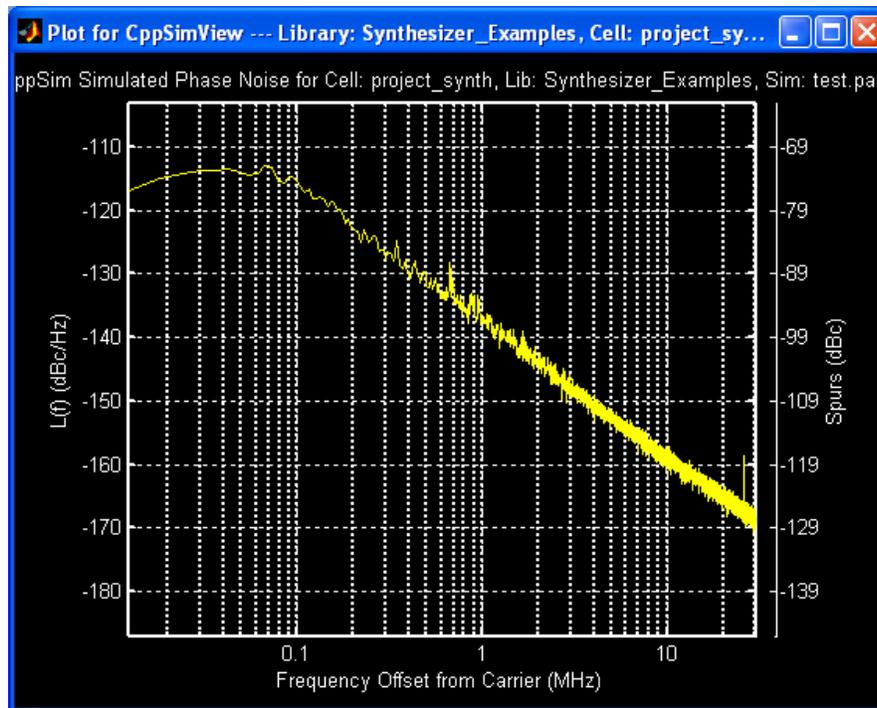
- Now click on **Edit Sim File** in CppSimView, and change the **delta_gl** parameter in the **test.par** file to be zero.
- Rerun the CppSim simulation and again plot **vin** as above.

You should see a plot as shown below, which reveals that the PLL becomes locked within 200 microseconds. In this case, we started close enough to lock at the beginning of the simulation that the offset current does not pose a problem. Again, in a practical design, you could not accommodate such a large current offset at the charge pump output as you would need to be able to regain lock with a large divide value change (especially at power-up). We are simply following through on this exercise for pedagogical reasons.



- Now re-examine the phase noise of the synthesizer by selecting the **test_noise.tr0** output file and then running the **plot_pll_phasenoise(...)** function (from 10 kHz to 30 MHz) on signal **noiseout**.

The resulting plot should appear as shown below. Notice that the mismatch between the up and down current pulses no longer causes significant folding of the Sigma-Delta quantization noise, but that a spur now appears at the reference frequency of 26 MHz. Fortunately, there is a great deal of filtering in the loop filter by the time you hit the reference frequency, so that is still below -100 dBc and not of consequence.

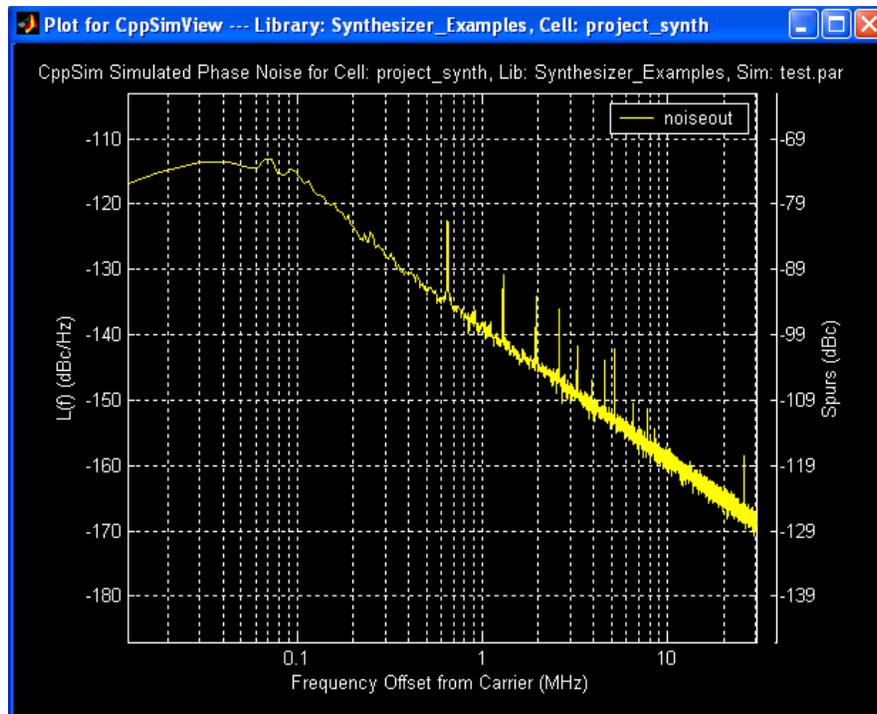


D. Producing Fractional Spurs

- Now click on **Edit Sim File** in CppSimView, and change the **in_gl** parameter to 34.65.
- Rerun CppSim, and again plot the phase noise of the synthesizer as before.

The plot should appear as shown below, and reveals the presence of fractional spurs due to the updated value of 34.65 going into the Sigma-Delta modulator. In practice, if possible, a proper job of frequency planning in the system can avoid having to hit such undesirable input values to the $\Sigma-\Delta$ modulator. Otherwise, dithering methods are a possibility to prevent the creation of such spurs. Fortunately, in this case, we see that the spurs have magnitude less than -80 dBc, and shouldn't pose a problem since they meet the goal of achieving spur performance better than -80 dBc. However, this may not be the case for other Sigma-Delta input values, and thorough investigation of this issue is warranted in an actual design.

Also, please note that the spur peaks shown in the phase noise plot are sampled values of waveforms of very narrow width – the peak that you see may not be the actual peak of the waveform. Therefore, the actual value of the spur may be higher than actually observed here. Phase noise measurement systems perform an algorithm to estimate the spur values, which we haven't bothered to do (i.e., we just provide the right-hand scale to give you a reasonable sense of the spur level values).



Conclusion

In this tutorial, we have used the PLL Design Assistant and CppSim programs to design a frequency synthesizer at the system level that is intended for a GSM transmitter application. Using the PLL Design Assistant, we were able to examine the noise performance of the synthesizer under different configurations and thereby select a candidate approach that met the GSM noise specifications and also met the settling time requirements under nominal conditions. The impact of parameter variations was then assessed, and its effects on phase noise and settling time examined. Using the CppSim behavioral simulator, we were then able to verify the noise and dynamic analysis of the PLL Design Assistant under nominal conditions. CppSim also allowed for examination of non-ideal behavior such as cycle slipping, the impact of charge pump mismatch, and the presence of fractional spurs for a given $\Sigma\text{-}\Delta$ modulator input setting. Hopefully, this exercise has given you helpful insights not only for fractional-N synthesizer design, but also for PLL design in general.