Clock and Data Recovery (CDR) Design Using The PLL Design Assistant and CppSim Programs

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

July 2008

Table of Contents

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 clock and data recovery (CDR) circuit that meets the SONET OC192 Standard (i.e. for 10 Gb/s data rates). 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 CDR for SONET applications (i.e., the design procedure is meant to serve only as an *example* of doing CDR design).

A. CDR Specifications for the SONET OC192 Standard

The OC 192 SONET specification applies to 10 Gbit/s data rates, and specifies three benchmark specifications that must be met by CDR circuits used in this application – jitter generation, jitter tolerance, and jitter transfer. We now examine each of these specifications in turn.

Jitter Generation

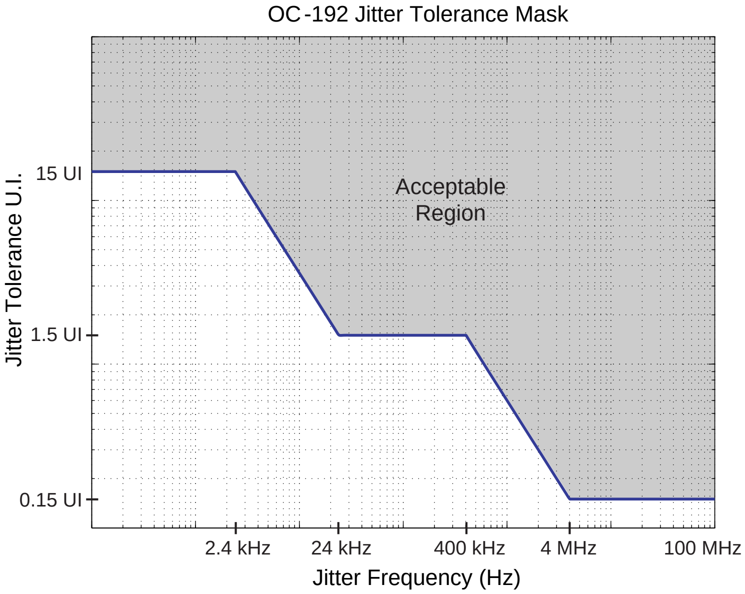
Given an input data stream with no jitter, the output of the CDR must meet the following jitter requirements:

- rms jitter < 10 mUI (i.e., rms jitter < 1 ps) (although SONET doesn't technically insist in meeting this rms spec, it is an appropriate benchmark to apply in order to meet the required peak-to-peak jitter spec below)
- peak-to-peak jitter < 100 mUI (i.e., peak jitter < 10 ps)

To test the CDR for compliance to the jitter generation specification, an input data stream with very low jitter is input into the CDR, and the jitter of the output data and clock stream is measured. Each of those streams (i.e., both clock and data) should have jitter amplitude that is less than the above specification. The jitter generation specification is often tested with several pseudo-random bit stream (PRBS) patterns generated with a linear feedback shift register (LFSR), where the LFSR has a length of either 7, 23, or 31. Low jitter is harder to maintain with longer lengths for the LFSR, so that it is desirable from a performance standpoint to meet the jitter generation specification with an LFSR of length 31. However, it is not uncommon for venders to limit their jitter generation specification to length 7 and 23 LFSR patterns.

Jitter Tolerance

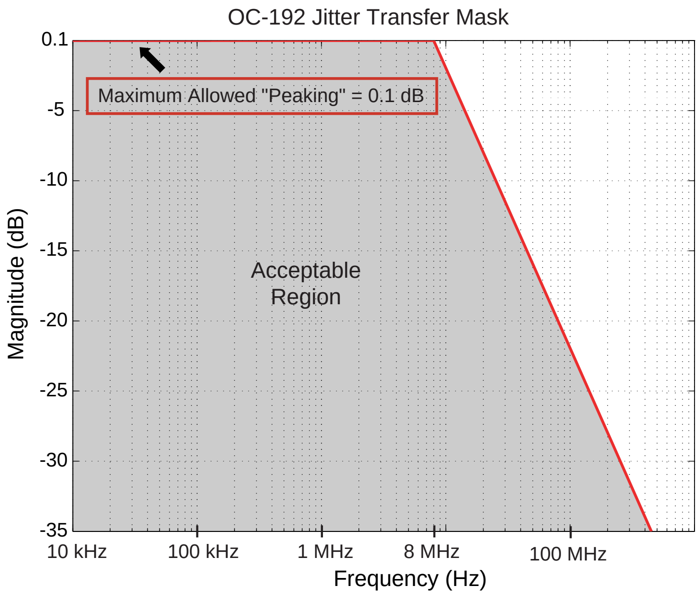
Given an input data stream whose instantaneous phase experiences a given minimum jitter amplitude at a specified frequency, as shown in the figure below, the CDR must maintain bit error rates (BER) below 1e-12. Note that higher jitter amplitude must be tolerated at lower frequencies since such jitter can be tracked by the PLL. The jitter amplitude at higher frequencies cannot be tracked by the PLL, and becomes a constant value at frequencies above the PLL bandwidth, as shown in the figure. Note that the high frequency jitter tolerance is, in practice, limited by the setup and hold times of the re-clocking register for the data stream in combination with the phase error offset under steady-state conditions.



To test the CDR for adherence to the above jitter tolerance specification, the instantaneous amplitude and frequency of the *phase* of the input data stream is varied according to the above figure, and the bit error rate of the CDR is checked at each amplitude and frequency value to determine whether it remains below 1e-12.

Jitter Transfer

Given an input data stream whose instantaneous phase experiences a given jitter amplitude at a specified frequency, the corresponding jitter amplitude at the output of the CDR at the same frequency must be attenuated as specified by the figure below.



To test the CDR for compliance to the jitter transfer specification, the instantaneous *amplitude* and *frequency* of the *phase* of the input data stream is varied according to the jitter *tolerance* specification (such that the BER remains below 1e-12), and a Fourier transform is performed on the instantaneous phase of the CDR output at each frequency. Given the Fourier transform results, the ratio of the output to input phase jitter is computed at each frequency and then checked against the specification.

One should note that the performance requirements of SONET CDR circuits differ from the performance requirements for high speed link CDR circuits used in chip-to-chip communication. In particular, high speed link CDR circuits typically do not need to meet a jitter transfer specification, and are instead focused on simply achieving adequately low bit error rates. In addition, the jitter generation performance of high speed link CDR circuits is often much more relaxed than required of SONET CDR circuits. However, despite the differences, this exercise will hopefully provide insight into general design principles for CDR circuits.

B. Preliminary Synthesizer Design Specifications

The local CDR expert in your company has suggested that you try implementing the design using the following architecture:

- PLL system parameters
  - Bandwidth: 4 MHz (this must be low enough to meet the jitter transfer specification across process and temperature variations and high enough to meet the tolerance specifications across process and temperature variations)
  - Order: 1 (the jitter transfer specification requires only a first order transfer function)
  - Shape: not applicable (only applies when the PLL order is greater than or equal to 2)
  - Type: 2 with fz/fo unknown by the PLL expert (you want Type 2 in order to get a consistent phase error between the data input and output clock such that the output clock is always “centered” within the eye of the data input)
- Implementation details
  - Data Rate: 10 Gbit/s (this is standard for SONET OC 192 systems)

    - Output Frequency: 10 GHz (this is standard for SONET OC 192 systems)
- PLL noise parameters
    - PFD-referred noise: the PLL expert wasn't sure what you need here
    - VCO: the PLL expert wasn't sure about this, either.

Your job is to examine the suitability of using the above architecture with the given specifications, and to investigate the impact of several non-idealities at the system level. Fortunately, you can apply the PLL Design Assistant and the CppSim behavioral simulator to make your job easier.

Jitter Transfer Analysis using the PLL Design Assistant

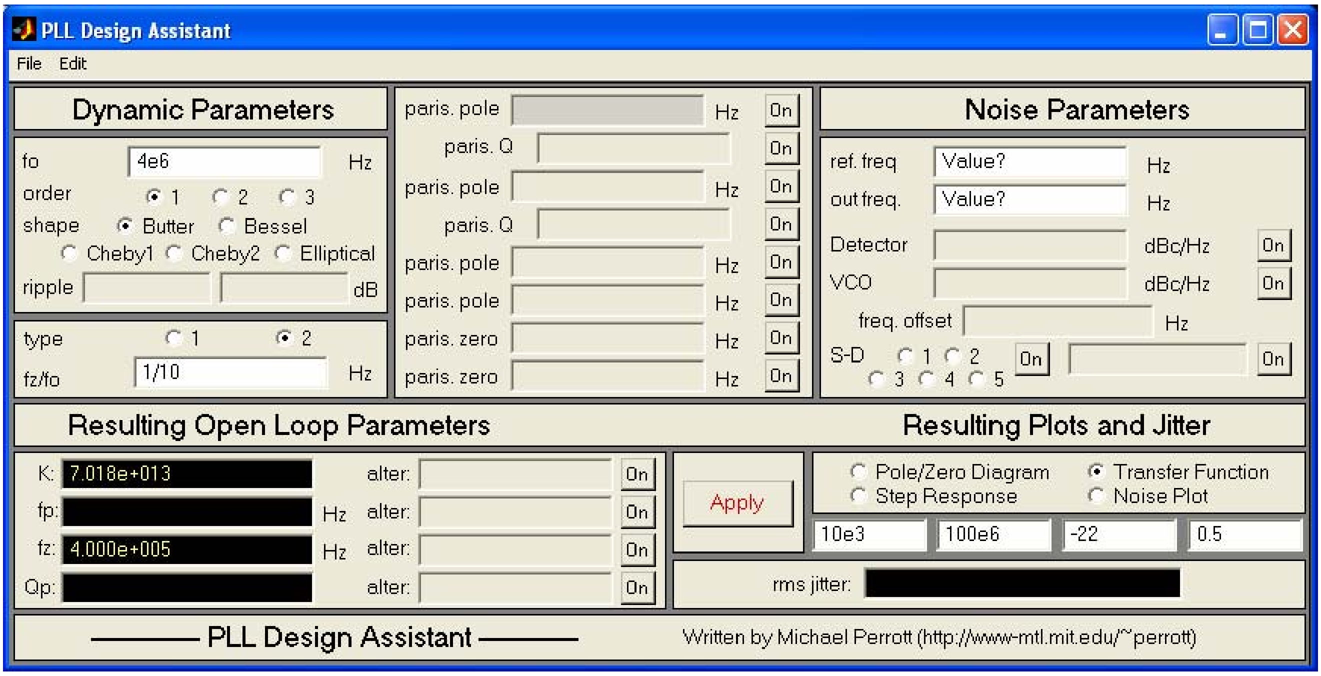
For SONET CDR circuits, it is often best to start the design process by focusing on jitter transfer performance. By doing so, we can first deal with the issue of setting the CDR dynamics independently from noise considerations, and then use the resulting configuration to determine the required noise performance of the various CDR blocks.

Based on the preliminary design given above, we will first determine an appropriate value for the ratio fz/fo by changing its value until the jitter transfer specification is met. We will then include the impact of a parasitic pole and parameter variations, and appropriately adjust the PLL design parameters to insure that the transfer specification is met under such conditions.

A. Calculation of PLL Parameters for the Nominal Design

- Enter the closed loop bandwidth, order, and type parameters into the PLL Design Assistant according to the preliminary settings presented in the introduction.
- Since the ratio fz/fo is unknown at this time, choose an initial value of 1/10 to get an initial evaluation of the PLL transfer characteristics.
- Click on the **Transfer Function** radio button, and set the axis limits to:
    - Transfer function axis limits: **10e3 100e6 -22 0.5**

The above axis limit settings will allow us to focus on the key frequency range of 10 kHz to 100 MHz, and to conveniently look at the jitter transfer spec (which must less than 0.1 dB at low frequencies, and less than 22 dB at 100 MHz frequency offset). The PLL Design Assistant and the resulting jitter transfer curve should look as shown in the figures below upon clicking on **Apply** with the above parameters entered.



PLL Design Assistant

File Edit

Dynamic Parameters

fo 4e6 Hz

order ● 1 ○ 2 ○ 3

shape ● Butter ○ Bessel ○ Cheby1 ○ Cheby2 ○ Elliptical

ripple dB

type ○ 1 ● 2

fz/fo 1/10 Hz

paris. pole Hz On

paris. Q On

paris. pole Hz On

paris. Q On

paris. pole Hz On

paris. pole Hz On

paris. zero Hz On

paris. zero Hz On

Noise Parameters

ref. freq Value? Hz

out freq. Value? Hz

Detector dBc/Hz On

VCO dBc/Hz On

freq. offset Hz

S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 7.018e+013 alter: On

fp: Hz alter: On

fz: 4.000e+005 Hz alter: On

Qp: alter: On

Apply

Resulting Plots and Jitter

○ Pole/Zero Diagram ● Transfer Function

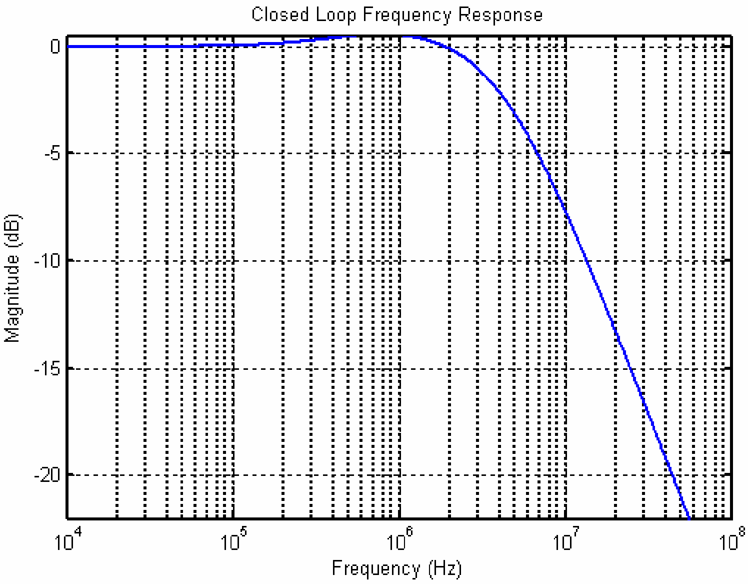
○ Step Response ○ Noise Plot

10e3 100e6 -22 0.5

rms jitter:

PLL Design Assistant

Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)

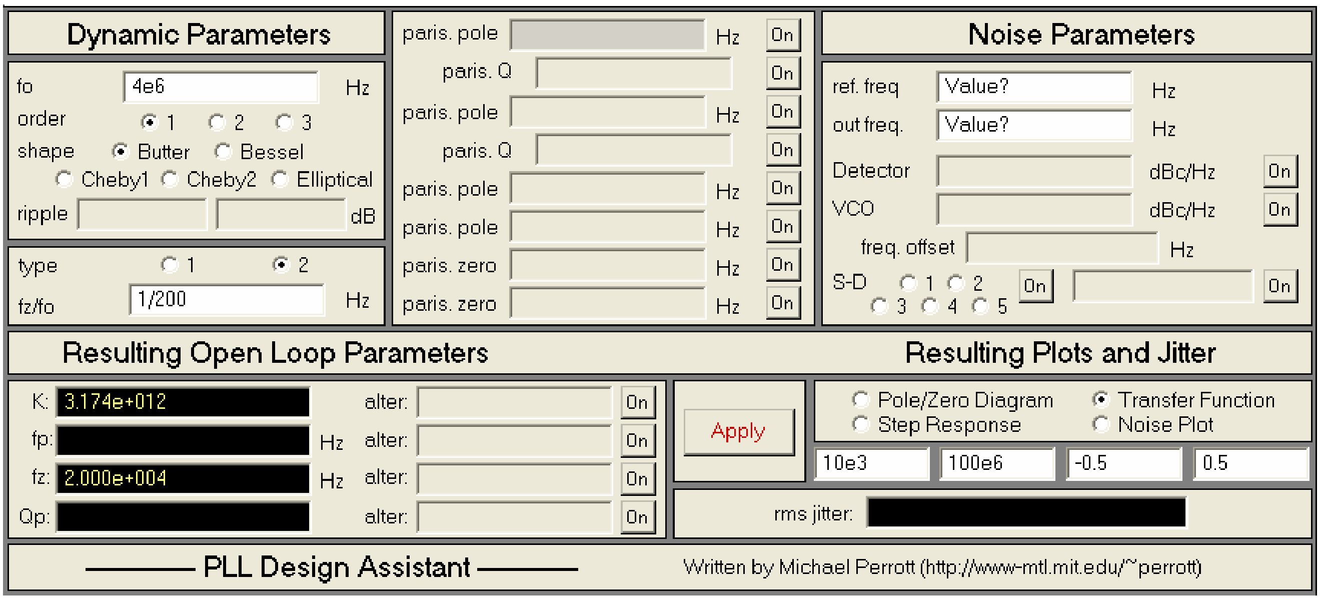


The above transfer function curve reveals that the peak jitter transfer specification is not being met at low frequencies since the peak of the transfer curve is above 0.1 dB. The problem is that the ratio fz/fo is too high – we need to reduce it in order to lower the peak value of the jitter transfer curve.

In response to the above issue, change the PLL Design Assistant parameters as follows:

- Enter a value for fz/fo of 1/200 in order to reduce the peak of the transfer curve
- Change the axis limits in order to better see the peak transfer curve value
    - New transfer function axis limits: **10e3 100e6 -0.5 0.5**

After completion of the above, the PLL Design Assistant and corresponding transfer function plot should look as shown below. We see that the peak of the transfer curve is now below 0.1 dB.



fo 4e6 Hz

order ● 1 ○ 2 ○ 3

shape ● Butter

type ○ 1 ● 2

fz/fo 1/200 Hz

ref. freq Value? Hz

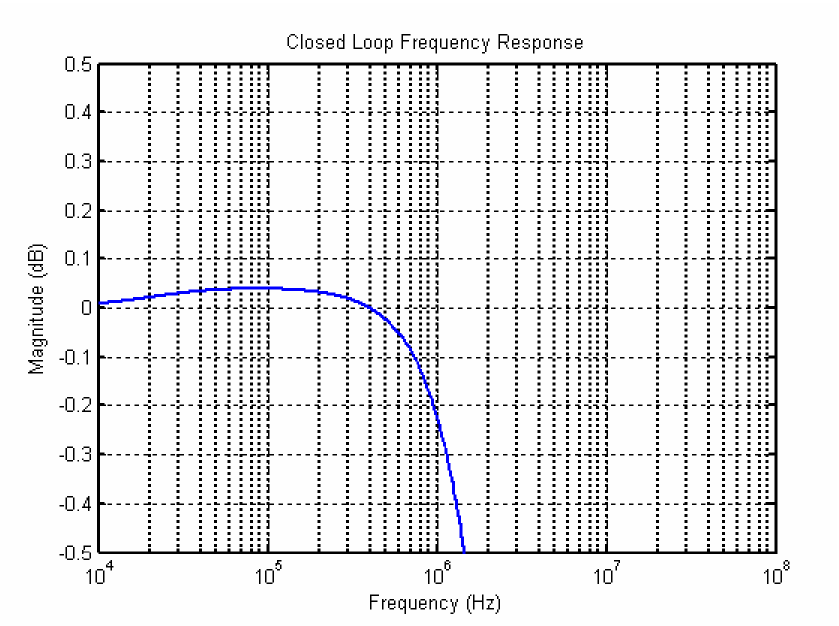
out freq. Value? Hz

K: 3.174e+012

fz: 2.000e+004 Hz

● Transfer Function

10e3 100e6 -0.5 0.5



B. Adding in Parasitic Poles and Parameter Variations

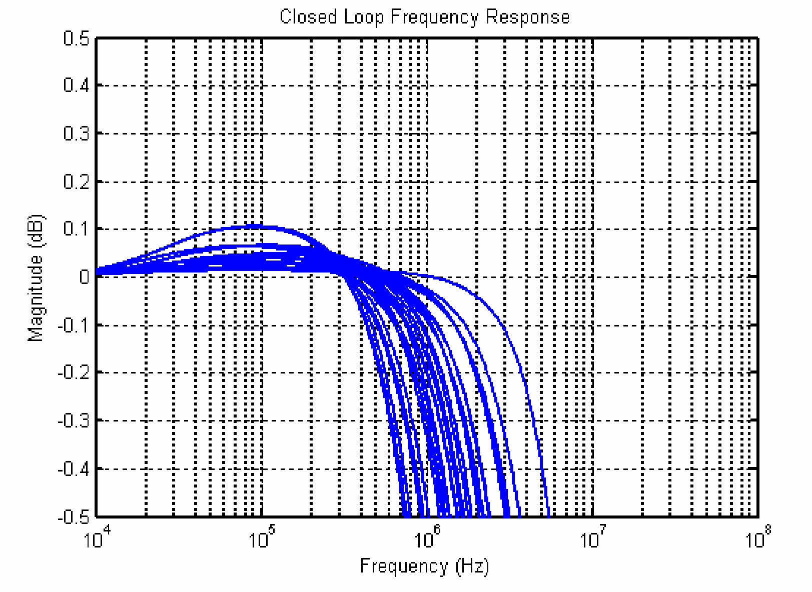
In practice, there will be parasitic poles present in the PLL open loop transfer function whose influence should be examined at an early stage. The value of such parasitic poles are calculated by running SPICE simulations across process and temperature variations once the transistor level schematics are created for the charge pump, loop filter, and VCO. At this early stage, let us simply assume the existence of one parasitic pole at ten times the PLL bandwidth (i.e., 40 MHz). Note that this value would need to be altered once SPICE simulations revealed a more realistic value, and other parasitic poles and/or zeros might also need to be added.

We also need to consider process and temperature variations on the open loop parameters of the CDR to insure that the jitter transfer specification will be met across such variations. Let us assume that the open loop gain will vary +/- 40% due to variations in the charge pump current, loop filter components, and the Kv of the VCO, and that the open loop zero, f_z, will vary +/- 30% due to such variations. In addition, let us assume that the parasitic pole varies across a wide range of 20 MHz to 80 MHz. Of course, the actual value of such variations would need to be evaluated with SPICE once the transistor level circuit design has been completed – the values assumed here are only preliminary guesses.

Make the following changes to the PLL Design Assistant configuration

- Enter the variations for the open loop gain and zero values within their respective **alter:** entry forms (see the **alter**: entries in the PLL Design Assistant figure below to see how this is done)
- Enter the nominal value of the parasitic pole along with the limits of its variation within one of the **paris. pole** entry forms (again, see the PLL Design Assistant figure below to see how this is done). Note that it is important to specify the *nominal* parasitic pole value first (i.e., [**40e6** 20e6 80e6]) since the open loop PLL parameters (i.e., **K** and $\mathbf{f_z}$) are computed based *only* on the first value (which is 40e6 in this example).

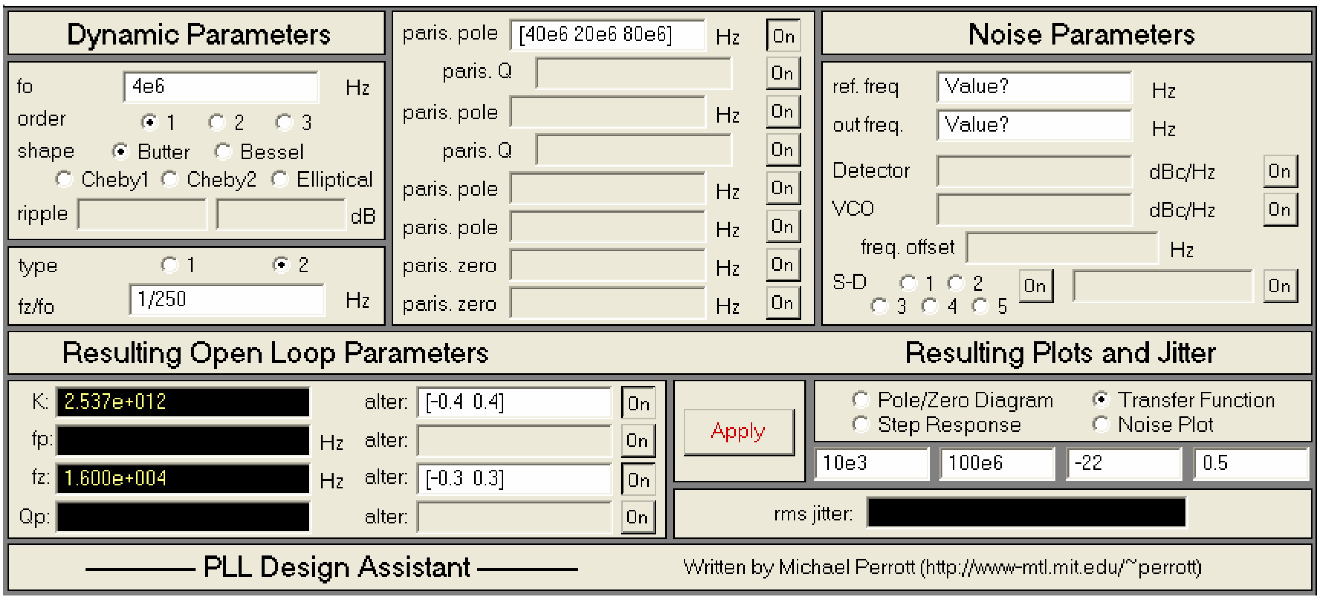
The resulting frequency response plot, which is shown below, reveals that the peaking specification is not met across all parameter variations.



To meet the PLL peaking specification, we should further lower the ratio of fz/fo:

- Change the ratio **fz/fo** from 1/200 to 1/250, and hit **Apply** to verify that the peaking specification is now met,
- Given that the peaking specification is met, check the overall transfer characteristic by changing the axis limits to: **10e3 100e6 -22 0.5**

The PLL Design Assistant, and the corresponding transfer function plot, should now appear as shown below. Note that the OC192 Transfer Spec. line and label were added to the plot within this document – the PLL Design Assistant will not produce this line and label on its own. We see that the transfer specification is now met. Of course, once more information is known about the actual circuits from SPICE simulations, this exercise should be repeated with updated variation parameters in order to obtain the best design configuration.



Dynamic Parameters

fo 4e6 Hz

order ● 1 ○ 2 ○ 3

shape ● Butter ○ Bessel ○ Cheby1 ○ Cheby2 ○ Elliptical

ripple dB

type ○ 1 ● 2

fz/fo 1/250 Hz

paris. pole [40e6 20e6 80e6] Hz On

paris. Q On

paris. pole Hz On

paris. Q On

paris. pole Hz On

paris. pole Hz On

paris. zero Hz On

paris. zero Hz On

Noise Parameters

ref. freq Value? Hz

out freq. Value? Hz

Detector dBc/Hz On

VCO dBc/Hz On

freq. offset Hz

S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 2.537e+012 alter: [-0.4 0.4] On

fp: Hz alter: On

fz: 1.600e+004 Hz alter: [-0.3 0.3] On

Qp: alter: On

Apply

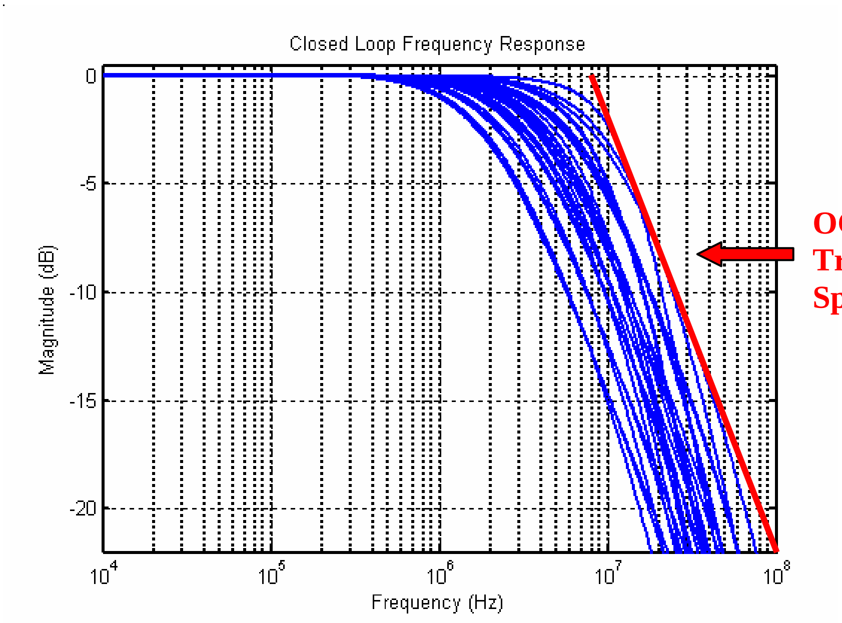
Resulting Plots and Jitter

○ Pole/Zero Diagram ● Transfer Function ○ Step Response ○ Noise Plot

10e3 100e6 -22 0.5

rms jitter:

PLL Design Assistant — Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)



OC192 Transfer Spec.

Although we have focused only on jitter transfer in this section, we have actually achieved the best design, given our current assumptions, for meeting both the jitter transfer *and* jitter tolerance specifications. In particular, since the jitter tolerance performance improves with wider PLL bandwidth (so that the PLL can track out input jitter), it is best to choose the widest PLL bandwidth that still meets the jitter transfer specification in the presence of parameter variations. The figure above reveals that we have achieved this goal given the current assumptions of the PLL parameter variation values.

Jitter Generation Analysis Using the PLL Design Assistant

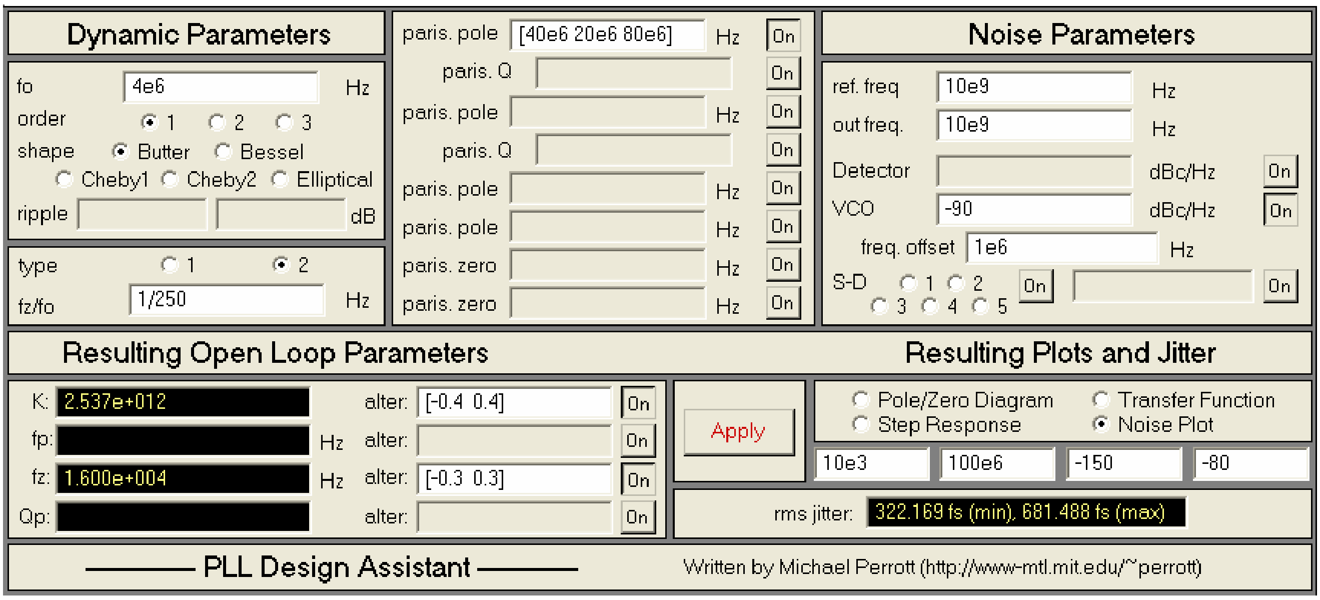
Given that the dynamics of the CDR have now been set according to the jitter transfer and jitter tolerance specifications, we now examine the noise performance required of the VCO and PFD blocks to meet the jitter generation specification.

A. Adding in VCO Noise

Let us now focus on the impact of VCO noise on the jitter of the CDR output clock. To do so, enter in values to the **Noise Parameters** section of the PLL Design Assistant as follows:

- Enter in a value of 10e9 within the **ref. freq** and **out freq.** entry forms,
- Enter in a value of -90 dBc/Hz at 1 MHz offset within the **VCO** entry forms,
- Click on the **Noise Plot** radio button to see the resulting noise plot and jitter estimate,
- Set the axis limits to **[10e3 100e6 -150 -80]** to focus on the frequency offset range of 10 kHz to 100 MHz – the SONET standard specifies this frequency offset range for the calculation of jitter for OC192 CDR circuits. Note that the PLL design assistant calculates jitter by integrating the estimated phase noise over the frequency offset range specified by the axis limits.

The PLL Design Assistant and corresponding output plot should now appear as shown below. We see that the rms jitter of the CDR output clock remains below 1 ps across all variations, so that the jitter generation specification is met. You may want to try changing the VCO noise to a few different values to see its impact on the rms jitter of the CDR output clock. However, VCO noise is only one contributor to the output jitter – we must also consider PFD noise sources from such elements as the charge pump within the CDR.



Dynamic Parameters

fo 4e6 Hz

order ● 1 ○ 2 ○ 3

shape ● Butter ○ Bessel ○ Cheby1 ○ Cheby2 ○ Elliptical

ripple dB

type ○ 1 ● 2

fz/fo 1/250 Hz

paris. pole [40e6 20e6 80e6] Hz On

paris. Q On

paris. pole Hz On

paris. Q On

paris. pole Hz On

paris. pole Hz On

paris. zero Hz On

paris. zero Hz On

Noise Parameters

ref. freq 10e9 Hz

out freq. 10e9 Hz

Detector dBc/Hz On

VCO -90 dBc/Hz On

freq. offset 1e6 Hz

S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 2.537e+012 alter: [-0.4 0.4] On

fp: Hz alter: On

fz: 1.600e+004 Hz alter: [-0.3 0.3] On

Qp: alter: On

Resulting Plots and Jitter

Apply

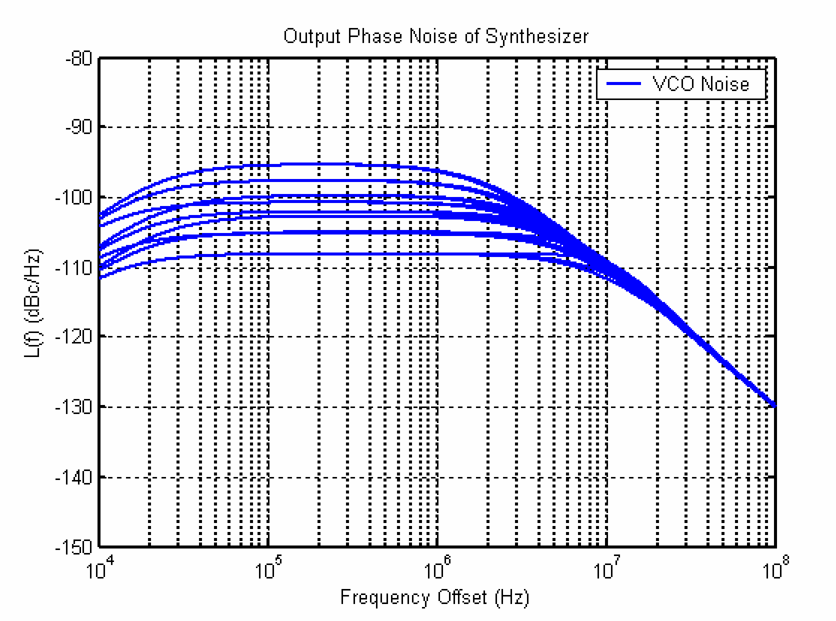
○ Pole/Zero Diagram ○ Transfer Function
○ Step Response ● Noise Plot

10e3 100e6 -150 -80

rms jitter: 322.169 fs (min), 681.488 fs (max)

PLL Design Assistant

Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)

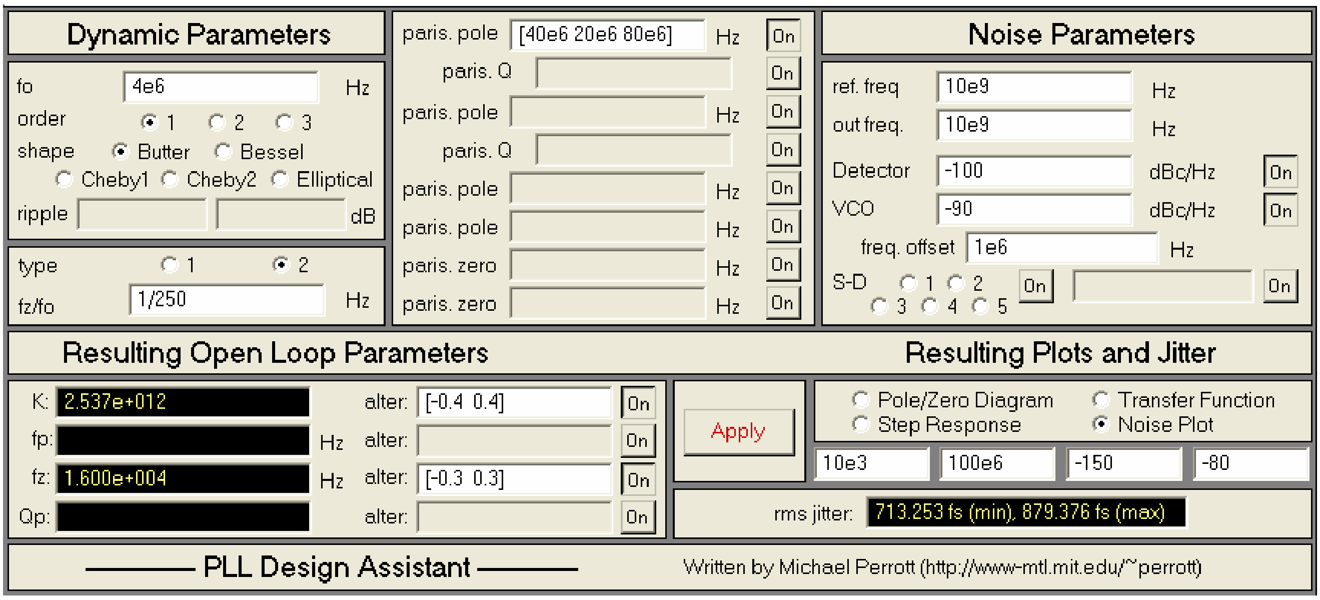


B. Adding in PFD Noise

Let us now add in output-referred PFD noise to the PLL Design Assistant in order to examine its impact on the jitter of the CDR output clock. In practice, such PFD noise is typically dominated by charge pump noise, though other blocks should also be considered (such as jitter added by the various circuit blocks within the phase detector, etc.). Note that one needs to know additional circuit details in order to map the output-referred PFD noise considered here to the acceptable limits of charge pump noise (and other PFD noise sources).

- Enter in -100 dBc/Hz into the **Detector** entry form within the **Noise Parameters** section,
- Click on **Apply**

The PLL Design Assistant and associated plot should appear as shown below. We see that the jitter generation specification is still met since the rms jitter has remained below 1 ps.



Dynamic Parameters

fo 4e6 Hz
order ● 1 ○ 2 ○ 3
shape ● Butter ○ Bessel
○ Cheby1 ○ Cheby2 ○ Elliptical
ripple dB
type ○ 1 ● 2
fz/fo 1/250 Hz

paris. pole [40e6 20e6 80e6] Hz On
paris. Q On
paris. pole Hz On
paris. Q On
paris. pole Hz On
paris. pole Hz On
paris. zero Hz On
paris. zero Hz On

Noise Parameters

ref. freq 10e9 Hz
out freq. 10e9 Hz
Detector -100 dBc/Hz On
VCO -90 dBc/Hz On
freq. offset 1e6 Hz
S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

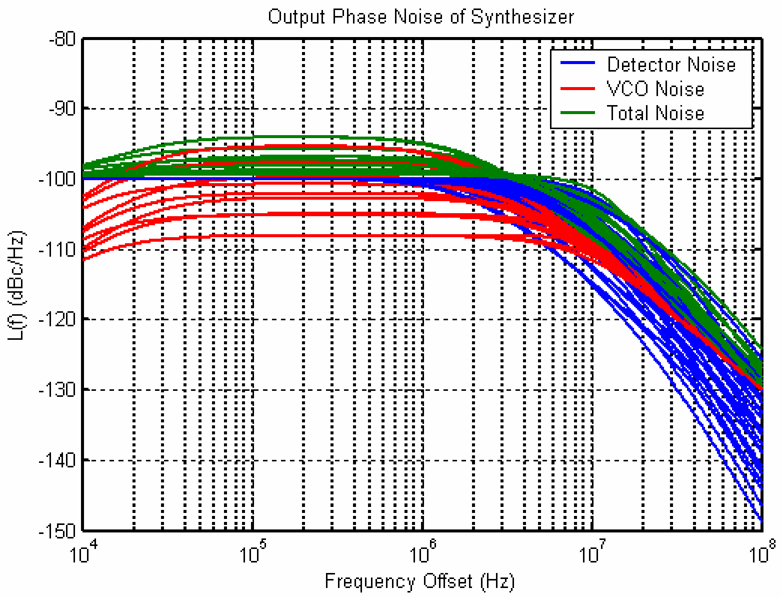
K: 2.537e+012 alter: [-0.4 0.4] On
fp: Hz alter: On
fz: 1.600e+004 Hz alter: [-0.3 0.3] On
Qp: alter: On

Apply

Resulting Plots and Jitter

○ Pole/Zero Diagram ○ Transfer Function
○ Step Response ● Noise Plot
10e3 100e6 -150 -80
rms jitter: 713.253 fs (min), 879.376 fs (max)

PLL Design Assistant
Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)



Given the above exercise, we now have a starting point for circuit design - the preliminary VCO design should aim to achieve output noise that is less than -90 dBc/Hz at 1 MHz offset (assuming a carrier frequency of 10 GHz), and the charge pump current should be set large enough to achieve less than -100 dBc/Hz of output-referred PFD noise. Of course, as the circuit design progresses, the values assumed here should be updated with those obtained through SPICE level simulation. Also, in practice, commercial CDR circuits achieve well below the maximum required jitter generation specification (perhaps by a factor of 2 or 3) in order to be competitive with respect to their performance. Therefore, the VCO and PFD noise requirements calculated here should be significantly exceeded in practice in order to have a marketable product. Finally, one should note that we have not included 1/f noise or other non-ideal effects in our analysis – the PLL Design Assistant is meant only to provide a starting point for design, and should be augmented with Matlab analysis and behavioral/circuit simulation to get a more accurate estimate of noise performance. Note that 1/f noises of the detector and VCO can be included in the PLL Design Assistant if necessary. Please read the manual "**PLL Design Using the PLL Design Assistant Program**" to see how this is done.

We will return to the PLL Design Assistant configuration we have just set up in a later section of this document, so that you may want to save the current configuration using the **Save (As)** command before proceeding to the next section.

Behavioral Simulation of Transient Behavior and Jitter using CppSim

CppSim has several CDR examples as part of its installation which can be leveraged in performing behavioral simulation of the linear CDR we designed in the previous sections. To do so, 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 instantaneous jitter of the CDR, and compare our results to that obtained from the PLL Design Assistant.

A. Entering the Design into Sue2

- Start up Sue2, and select the **linear_cdr** cell within the **CDR_Examples** library.

- Select **File->Save as** and then save the file as **project_linear_cdr** within the **CDR_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.

- PRBS data signal source (i.e. **signal_source** in Sue2):
    - stype = 3 (this selects a PRBS (Pseudo-Random Bit Sequence) data pattern that has a NRZ (Non-Return to Zero) signal characteristic.
    - freq = 10e9 (we nominally want a 10 Gbit/s data sequence for OC192 applications)
- VCO
    - Center frequency = 10 GHz (set by the input PRBS data pattern frequency and the fact that we have chosen a full-rate CDR architecture.)
    - K_v = 200 MHz/V (this would be determined by VCO implementation)
    - Noise: to meet the jitter generation specification, let us assume -90 dBc/Hz at 1 MHz frequency offset as discussed in the previous section.
- Phase Detector – Hogge design
    - α = 1/2 for a PRBS input data pattern (see the **Clock and Data Recovery Circuits** section of the PLL Design Assistant manual for explanation of the meaning of this parameter for CDR circuits). We'll use this value in the calculation of the loop filter gain below.
- Charge pump
    - ival = 500 microamps (this is a function of speed requirements of the charge pump, the desired PFD noise performance you need, and the amount of loop filter capacitance available to you – you need to do transient and noise analysis of the charge pump transistor-level circuit in SPICE to get a good estimate of this. A value of 500 microamps is a reasonable assumption to begin with for modern CMOS processes (i.e., 0.18 micron gate length or less)).
    - Note that the charge pump block in this schematic does not include noise, so that we are ignoring PFD-referred noise in the simulation. Such noise could be added in the future once more information is available from SPICE level simulations.
- Loop Filter – consists of a lead/lag filter
    - Use the parameter values previously calculated from the PLL Design Assistant
        - fp = 40 MHz (the parasitic pole location), fz = 16 kHz (from the PLL Design Assistant)
        - gain – must be calculated from PLL parameters as described within the **Clock and Data Recovery Circuits** section of the PLL Design Assistant manual

$$gain = K\frac{1}{2aKvIcp}$$

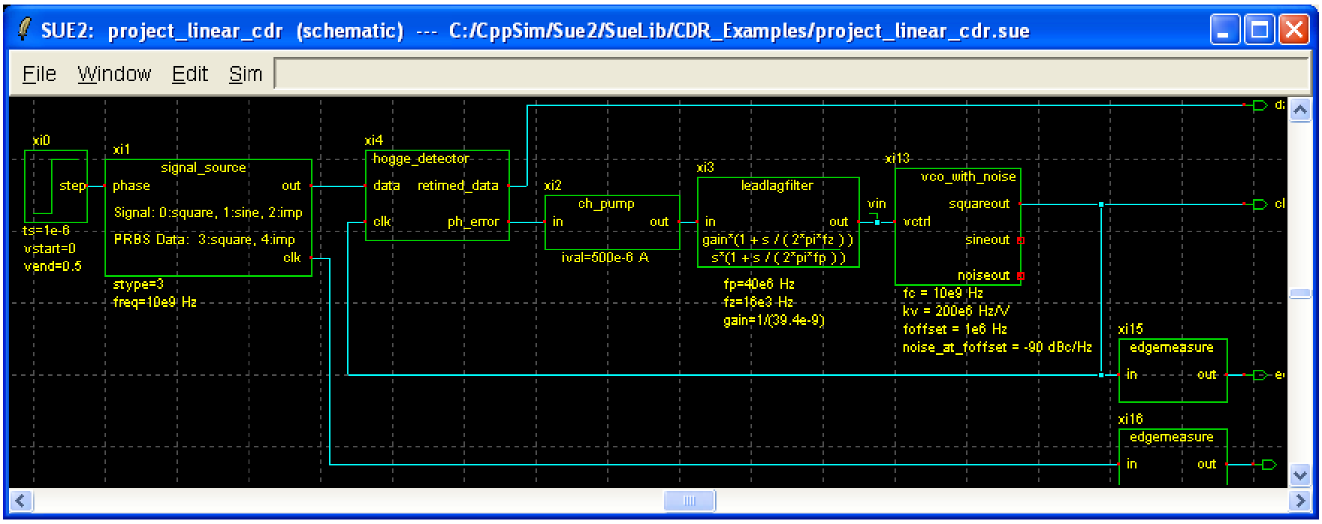
- K = 2.537e12 (from the PLL Design Assistant)
- α = 1/2, Kv = 200e6, Icp = 500e-6 (from above)

- **Result:** gain = 1/(39.4e-9)
  - This corresponds to 39.4 nF of capacitance in the loop filter! Therefore, the loop filter capacitance would 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, Icp, would allow a reduction in the reduction of the capacitance value. However, reduction of Icp would also lead to potentially slower dynamics for the charge pump and an increased impact of charge pump noise on the PLL output. The best value for Icp, and loop filter capacitance, will need to be decided later after SPICE simulations of the charge pump and other PLL blocks are performed.

In addition to the parameter changes made above, we would also like to add a step input to the phase control of the **signal_source** in order to observe the transient performance of the system.

- Delete the **constant** cell that feeds the input **phase** control of the **signal_source** cell.
- Click on the **step_in** primitive within the **icons1** listbox of Sue2, and then move the mouse within the main Sue2 schematic window. Place the **step_in** cell in the place that the **constant** cell once occupied so that its output feeds into the input **phase** control of the **signal_source** cell.
- Double-click on the **step_in** cell, and enter in the parameters **ts=1e-6**, **vstart=0**, and **vend=0.5** in order to create a phase step of 0.5 UI (i.e., half of a period of the data signal) at 1 microsecond.

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

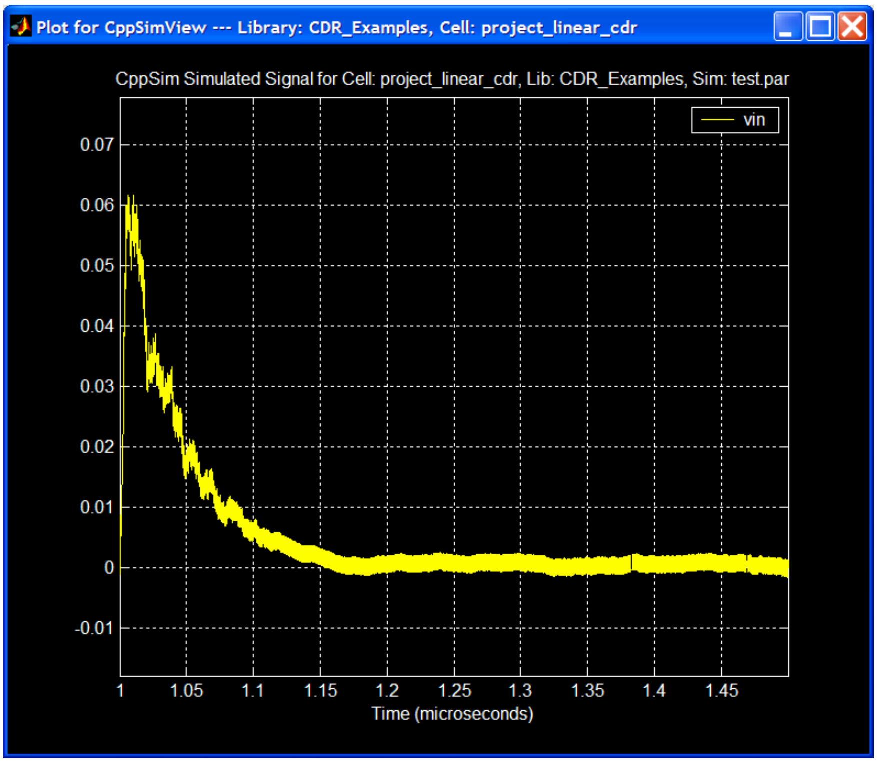
- Within the Sue2 window, select **Tools→ CppSim Simulation**. You should see the Cppsim Run Menu that pops up. Left click on the **Edit Sim File** button, and an Emacs window should pop up**.** Now open CppSimView and make sure it synchronizes to your new schematic. If not, left-click the **Synch**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

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

```
//  Number of time steps needs to be large enough to observe settling and jitter
performance
num_sim_steps: 1e6

// Choose a time step such that its inverse is over a factor of 5 higher than the output
// frequency – you need at least 5 points per VCO period to properly represent its square
// wave output.  I often use 6 points per VCO period, as chosen here
Ts: 1/60e9

// Probe the CDR signals of interest
output: test
probe: vin data_out clk_out edge_clk edge_ref

// Zoom in on the signals over a 0.5 microsecond interval starting at 1 microsecond to
easily
// examine the step response
output: test_closeup start_time=1e-6 end_time=1.5e-6
probe: vin data_out clk_out edge_clk edge_ref
```

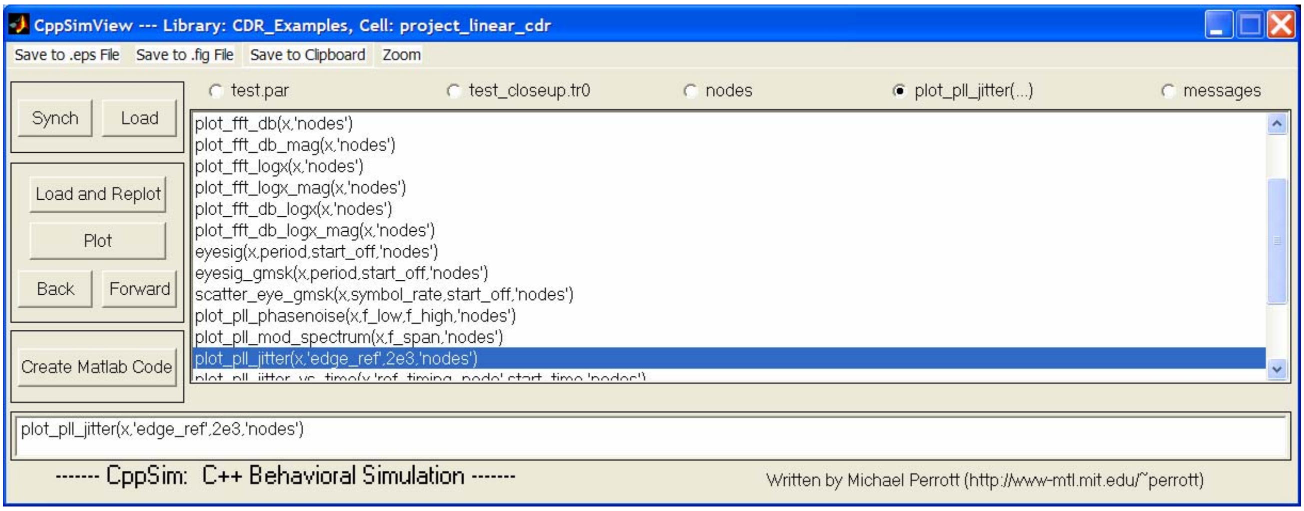
C. Examining the Simulated Step Response

- Now run the CppSim simulation, and then plot signal **vin** within the **test_closeup.tr0** output file to observe the transient response. You should obtain a plot as shown below.



The above plots displays the input voltage to the VCO as a function of time, but a more direct method of observing the transient performance is to examine the instantaneous phase of the output as a function of time. To do so, we change the plotting function to **plot_pll_jitter(…)** as follows

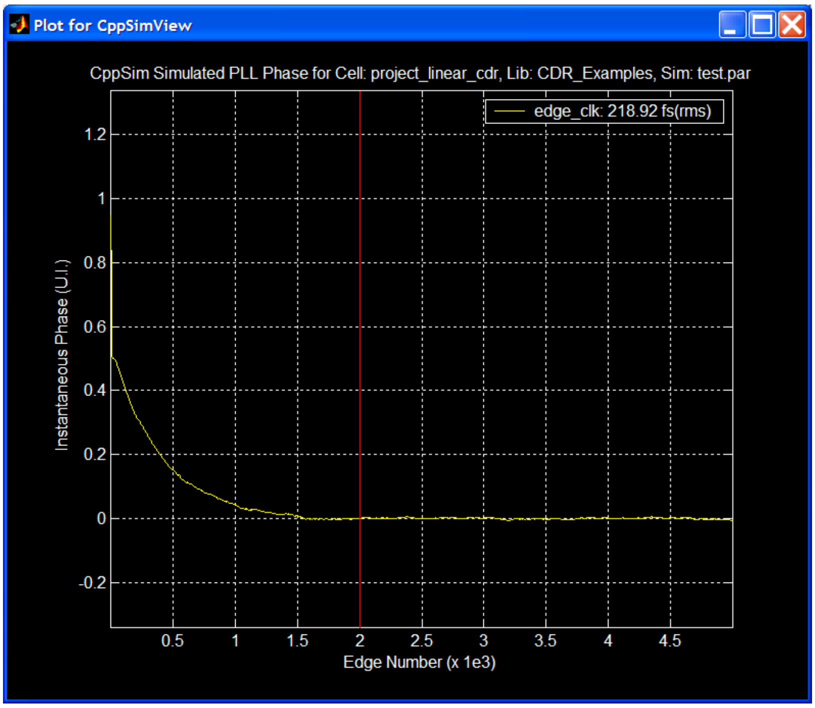
- In CppSimView, click on the **plotsig(…)** radio button, and then click on the **plot_pll_jitter(…)** function
  - In the command line, change **'ref_timing_node'** to **'edge_ref'**, and then change **start_edge** to **2e3**. Press the **Enter** key to update the **plot_pll_jitter(…)** function within the listbox with its new settings. CppSimView should now appear as shown below.



```
plot_fft_db(x,'nodes')
plot_fft_db_mag(x,'nodes')
plot_fft_logx(x,'nodes')
plot_fft_logx_mag(x,'nodes')
plot_fft_db_logx(x,'nodes')
plot_fft_db_logx_mag(x,'nodes')
eyesig(x,period,start_off,'nodes')
eyesig_gmsk(x,period,start_off,'nodes')
scatter_eye_gmsk(x,symbol_rate,start_off,'nodes')
plot_pll_phasenoise(x,f_low,f_high,'nodes')
plot_pll_mod_spectrum(x,f_span,'nodes')
plot_pll_jitter(x,'edge_ref',2e3,'nodes')
plot_pll_jitter_vs_time(x,'ref_timing_node',start_time,'nodes')
```

```
plot_pll_jitter(x,'edge_ref',2e3,'nodes')
```

- Now click on the **nodes** radio button, and then double-click on the **edge_clk** node. The resulting plot should match that shown below. Note that the red line indicates the first edge number for which the jitter of the clock signal is calculated – the calculated value of 218.3 fs (rms) in this case is based on an inadequate number of edge samples to be accurate. We will obtain a more accurate jitter estimate in a following section in this document.
  - One other point about the plot below – the final phase of zero U.I. is not accurate. It should actually be 0.5 U.I. since the CDR aligns the rising edges of its output clock one half-cycle (i.e., 0.5 U.I.) away from the input data transitions under steady-state conditions. This incorrect computation of the phase offset is due to the fact that the **test_closeup.tr0** file ignores the initial samples of **edge_clk** and **edge_ref** signals since this output file begins recording of these signals 1 microsecond into the simulation. To get an accurate estimate of the actual phase offset, it is necessary to use the **plot_pll_jitter(…)** function with **edge_clk** and **edge_ref** signals recorded from the beginning of the simulation (as will be done below). This issue arises due to the fact that the **edgemeasure** blocks have outputs (such as **edge_clk**) corresponding to the time *between* the rising edge transitions of their input signals (such as **clk_out**), and the **plot_pll_jitter(…)** function must *integrate* the information in these signals to construct their phase difference. By removing the initial samples of the **edge_clk** and **edge_ref** signals, we lose key information required to properly construct the *absolute* phase difference between the signals. However, *only* the phase *offset* is in error – the plot produced by **plot_pll_jitter(…)** is still accurate with respect to the phase settling dynamics that it portrays.

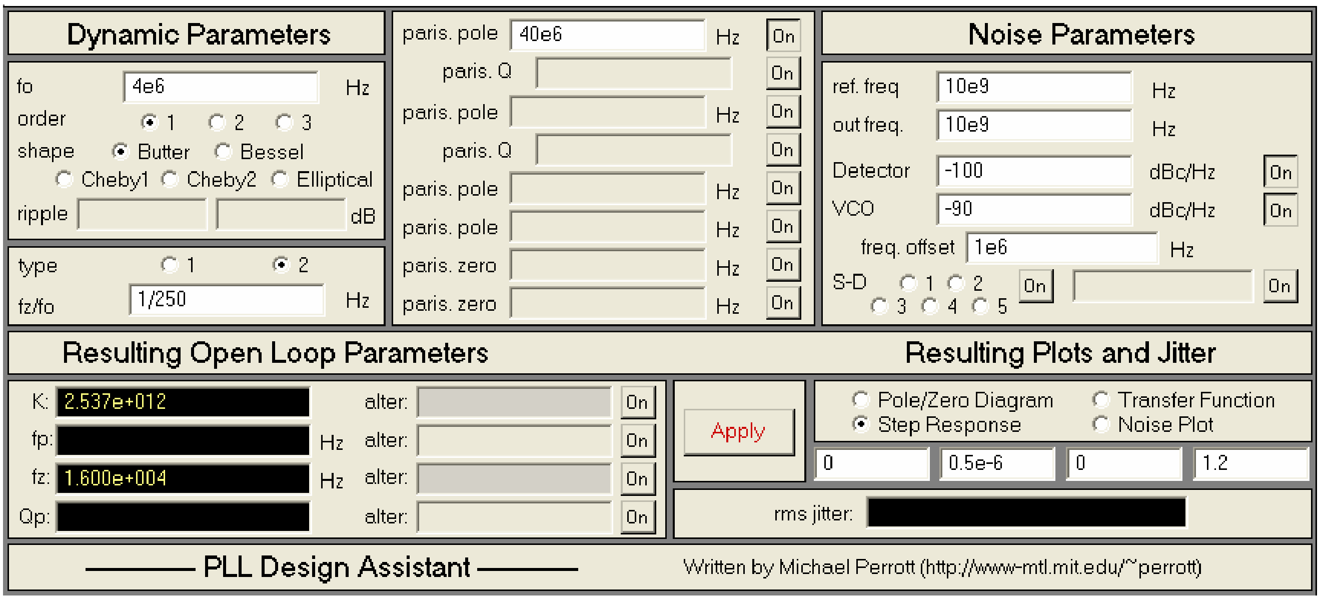


D. Comparing the Simulated and Calculated Step Responses

Let us compare the above CppSim result to the analytical result from the PLL Design Assistant. To do so, return to the PLL Design Assistant and load in its configuration as was saved at the end of the section entitled **Jitter Generation Analysis of a Linear CDR Using the PLL Design Assistant**. Within the PLL Design Assistant, perform the following operations

- Turn off the **alter:** entries for the open loop PLL parameters (i.e., **K** and **fz**),
- Remove the variation of the paris. pole by replacing the vector **[40e6 20e6 80e6]** with **40e6**,
- Click on the **Step Response** radio button,
- Enter axis limits of **[0 0.5e-6 0 1.2]** and then press **Apply**.

The PLL Design Assistant and its resulting step response plot should appear as shown below. Comparison of the calculated step response shown below to the simulated response shown in the previous subsection reveals very good agreement between the two with respect to settling time. (Note that the PLL Design Assistant x-axis is labeled in time, whereas the CppSim x-axis is labeled according to edge number – however, the time span is the same for each of the plots). The simulated response includes the impact of noise and detector non-idealities (such as changing pulse widths at the output of the phase detector as a function of the phase error), and therefore appears more jagged than its analytical counterpart.



Dynamic Parameters

fo 4e6 Hz

order ● 1 ○ 2 ○ 3

shape ● Butter ○ Bessel

○ Cheby1 ○ Cheby2 ○ Elliptical

ripple dB

type ○ 1 ● 2

fz/fo 1/250 Hz

paris. pole 40e6 Hz On

paris. Q On

paris. pole Hz On

paris. Q On

paris. pole Hz On

paris. pole Hz On

paris. zero Hz On

paris. zero Hz On

Noise Parameters

ref. freq 10e9 Hz

out freq. 10e9 Hz

Detector -100 dBc/Hz On

VCO -90 dBc/Hz On

freq. offset 1e6 Hz

S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 2.537e+012 alter: On

fp: Hz alter: On

fz: 1.600e+004 Hz alter: On

Qp: alter: On

Apply

Resulting Plots and Jitter

○ Pole/Zero Diagram ○ Transfer Function

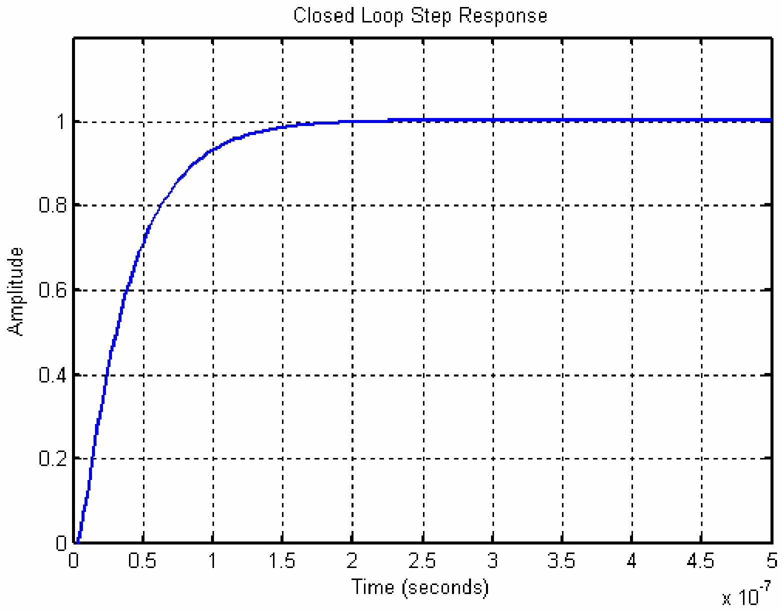
● Step Response ○ Noise Plot

0 0.5e-6 0 1.2

rms jitter:

PLL Design Assistant

Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)



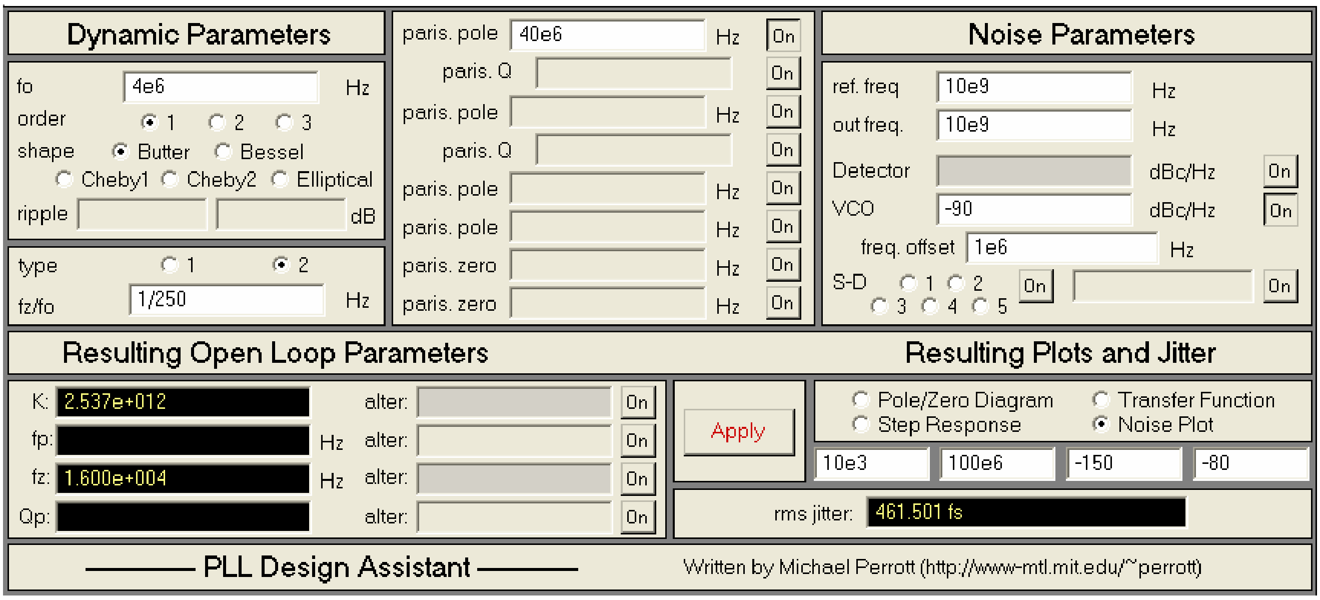
Note that the analytical plot from the PLL Design Assistant reveals that the PLL does not settle to its final value of 1 within the time span examined – the incomplete settling is due to the closed loop, low frequency pole/zero doublet caused by the stabilizing zero**, fz**. If one changes the axis limits so that the time span is increased to 20e-6 seconds, you should observe more complete settling of the PLL toward its final value of 1. It is hard to observe this issue with the CppSim generated plot due to the noise present in the simulation.

E. Comparing the Simulated and Calculated Jitter Generation Performance

Continuing with the PLL Design Assistant analysis, perform the following operations within its interface

- Click on the **Noise Plot** radio button,
- Turn off **Detector** noise (we did not yet include PFD noise within the CppSim simulation, and want to examine the correspondence between the simulated and calculated jitter),
- Click on **Apply**.

The PLL Design Assistant should now appear as shown below. We see that the calculated rms jitter is 461.5 fs, which is higher than the simulated value of 218.3 fs calculated with CppSim in the previous subsection. However, the previous CppSim simulation result was based on a relatively small number of edges – a more accurate jitter estimate can be obtained by looking at more edges from the simulation. Be sure to save the current PLL Design Assistant at this point – we will be returning to it in a later section.



Dynamic Parameters

fo 4e6 Hz
order ● 1 ○ 2 ○ 3
shape ● Butter ○ Bessel
○ Cheby1 ○ Cheby2 ○ Elliptical
ripple dB

type ○ 1 ● 2
fz/fo 1/250 Hz

paris. pole 40e6 Hz On
paris. Q On
paris. pole Hz On
paris. Q On
paris. pole Hz On
paris. pole Hz On
paris. zero Hz On
paris. zero Hz On

Noise Parameters

ref. freq 10e9 Hz
out freq. 10e9 Hz
Detector dBc/Hz On
VCO -90 dBc/Hz On
freq. offset 1e6 Hz
S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 2.537e+012 alter: On
fp: Hz alter: On
fz: 1.600e+004 Hz alter: On
Qp: alter: On

Apply

Resulting Plots and Jitter

○ Pole/Zero Diagram ○ Transfer Function
○ Step Response ● Noise Plot
10e3 100e6 -150 -80

rms jitter: 461.501 fs

PLL Design Assistant

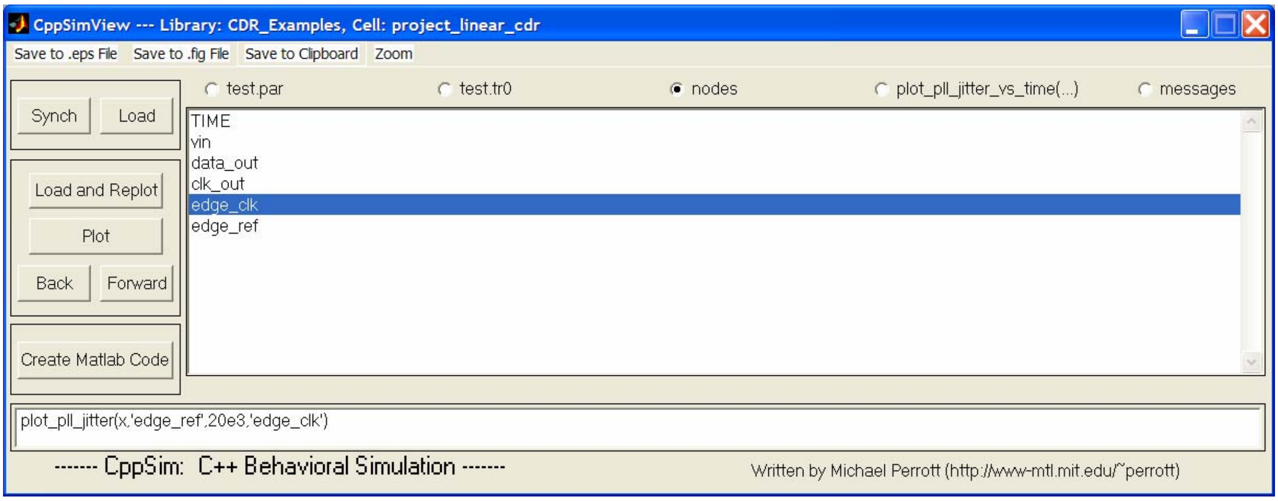
Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)

Returning to CppSimView (synched to **project_linear_cdr**), perform the following operations

- Click on the **test_closeup.tr0** radio button, and select **test.tr0** as the output file in order to look at a longer time duration from the simulation,
- Click on the **No Nodes** radio button to load in the signals from **test.tr0**,
- Run the **plot_pll_jittter(…)** function with **edge_ref** as the reference signal, **20e3** as the **start_edge** value, and **edge_clk** as the signal of interest.

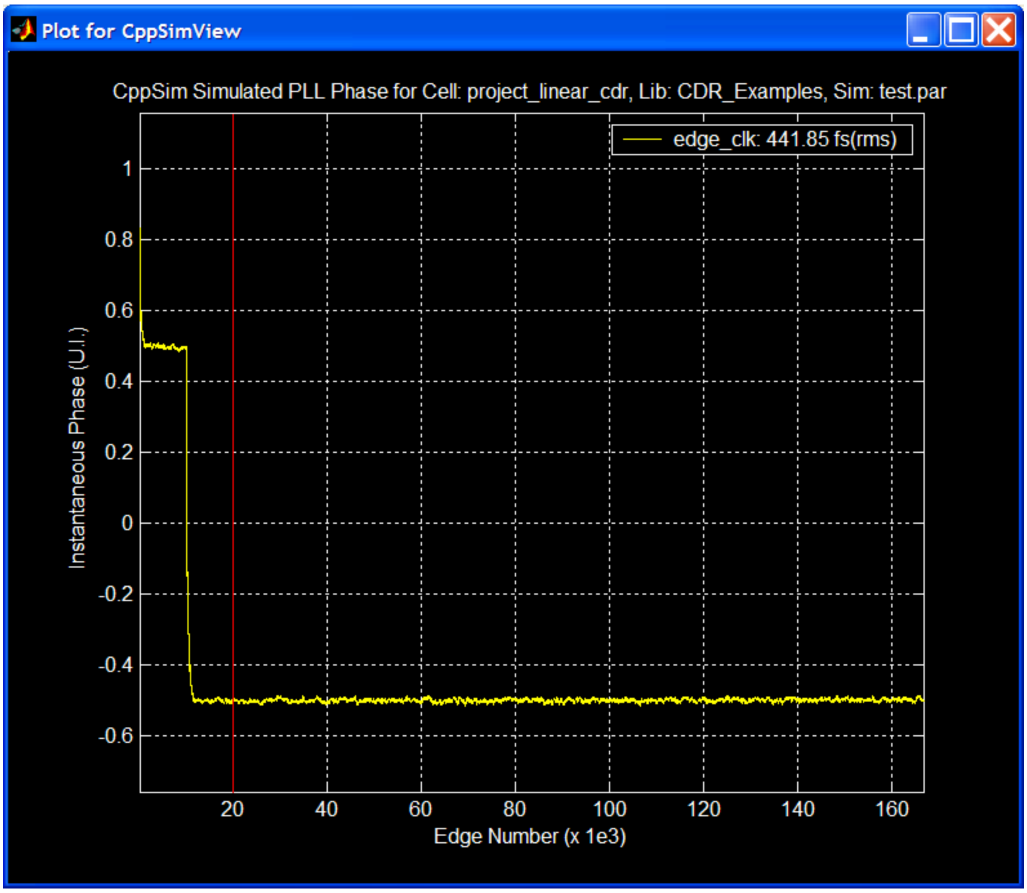
Upon completion of the above operations, CppSimView and the corresponding transient plot should appear as shown below. We see that the jitter estimate is now 444.13 fs (rms), which is fairly close to the calculated value of 461.5 fs from the PLL Design Assistant.

Note that the phase offset shown in the transient plot below is now accurate, which is in contrast to the **plot_pll_jitter(…)** plot previously generated in this document from the **test_closeup.tr0** file. In particular, the steady-state phase is correctly shown to be -0.5 U.I. (which is the same as 0.5 UI since phase wraps every 1.0 U.I.) – this is true since the CDR aligns the rising edges of its output clock one half-cycle (i.e., 0.5 U.I.) away from the input data transitions under steady-state conditions. The reason for the accurate phase offset in this case is that the **test.tr0** file did not truncate the initial samples of the **edge_clk** and **edge_ref** signals as occurred with the **test_closeup.tr0** file. In addition, note that the presence of a step input into the **phase** input of the **signal_source** block does not alter the relative steady-state phase difference between the output clock and the input data transitions – the CDR always aligns that difference to be +/- 0.5 U.I. under steady-state conditions.


CppSimView --- Library: CDR_Examples, Cell: project_linear_cdr
Save to .eps File
Save to .fig File
Save to Clipboard
Zoom
Synch
Load
Load and Replot
Plot
Back
Forward
Create Matlab Code
○ test.par
○ test.tr0
● nodes
○ plot_pll_jitter_vs_time(...)
○ messages
------ CppSim: C++ Behavioral Simulation ------
Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)

```
TIME
vin
data_out
clk_out
edge_clk
edge_ref
```

```
plot_pll_jitter(x,'edge_ref',20e3,'edge_clk')
```



Behavioral Simulation of Jitter Transfer Behavior using CppSim

We now seek to simulate the jitter transfer characteristic of the CDR using CppSim. This task is best accomplished by writing an appropriate Matlab script to automatically post-process simulation results produced by CppSim. However, it is instructive to carry out the operations in CppSimView to get a sense of how the process works. Since CppSimView is a bit cumbersome to deal with in this case, we will limit our examination of jitter transfer to just two jitter frequency points.

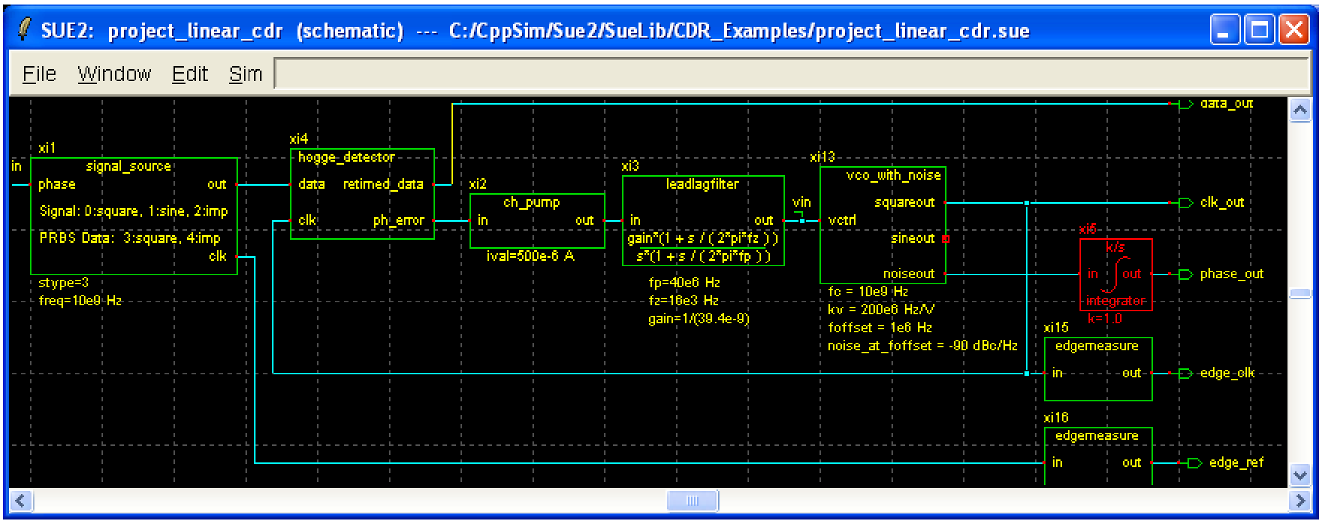
To accomplish this task, we will first modify the Sue2 schematic for the CDR in order to be able to observe the phase of its output clock, and to vary the phase of the incoming PRBS data signal. We will then modify the simulation file, run CppSim, and then use CppSimView to perform the appropriate calculations. Finally, we will go back to the PLL Design Assistant to compare calculated and simulated results.

A. Creating an Output Phase Signal

A convenient way to observe the output phase of the CDR clock is to simply apply the fact that the transfer function model of the VCO from its input voltage to output phase is described by an integration operation, Kv/s. Since the nominal value of the VCO is the same as its steady-state value, 10 GHz, the steady state input to the VCO will conveniently be zero so that its integral will have a steady DC value. By taking advantage of the fact that the **noiseout** terminal of the **vco_with_noise** block corresponds to the VCO input scaled by Kv, we can simply integrate this signal to obtain an estimate of the instantaneous phase of the VCO.

- Select the **integrator cell** within the **icons1** listbox of Sue2, move the mouse into the main Sue2 schematic window, and then place the **integrator** icon to the right of the **vco_with_noise** cell
    - Set the **k** parameter of this cell to 1.0. The units of the phase output will then be UI since **Kv** has units of Hz/V
- Place a wire between the **noiseout** terminal of **vco_with_noise** and the **in** terminal of the **integrator** block,
- Create an output pin named **phase_out**, and wire the output of the **integrator** cell to this pin.

The Sue2 schematic window should look as shown below upon completion of these operations.


SUE2: project_linear_cdr (schematic) --- C:/CppSim/Sue2/SueLib/CDR_Examples/project_linear_cdr.sue

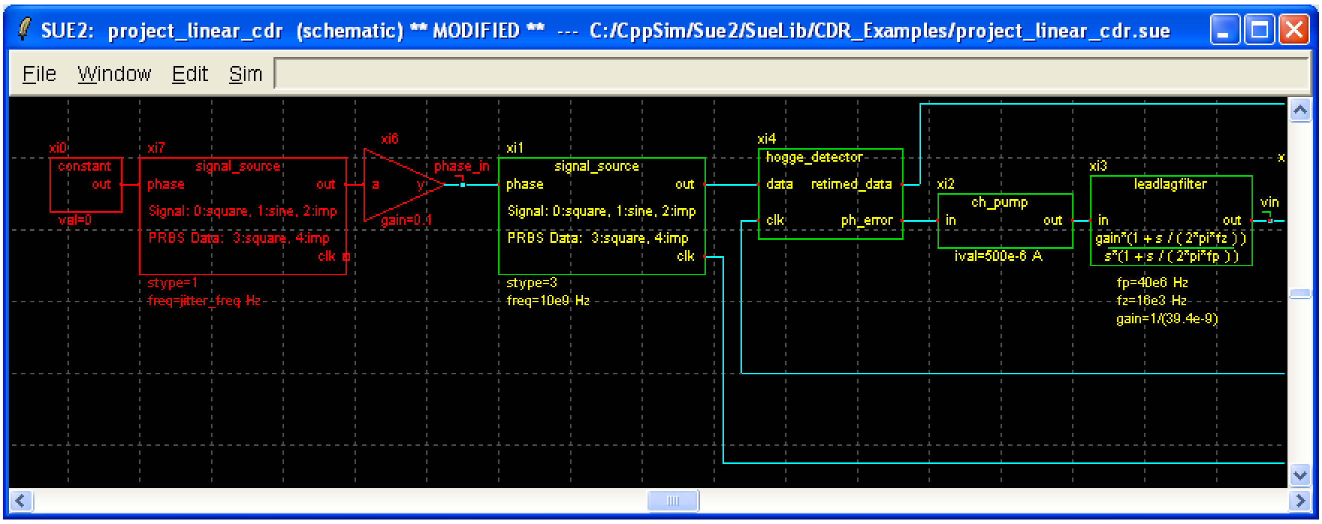
Signal: 0:square, 1:sine, 2:imp
PRBS Data: 3:square, 4:imp
stype=3
freq=10e9 Hz
ival=500e-6 A
gain*(1 + s / (2*pi*fz))
s*(1 + s / (2*pi*fp))
fp=40e6 Hz
fz=16e3 Hz
gain=1/(39.4e-9)
fc = 10e9 Hz
kv = 200e6 Hz/V
foffset = 1e6 Hz
noise_at_foffset = -90 dBc/Hz
k=1.0

B. Creating Input Phase Deviation for the PRBS Data Source

Now that we have a measure of output phase, let us create a deviation source for the input data stream that feeds the CDR. We must limit the phase deviation to a small enough value that bit errors do not occur. It is reasonable to assume that a peak-to-peak deviation of 0.2 UI is adequately small to avoid bit errors from occurring in the simulation at all jitter frequencies of interest.

- Delete the **step_in** cell that feeds into the **phase** terminal of the **signal_source** cell,
- Select the gain block from the **icons1** listbox and then place the cell at the location that the **step_in** cell used to occupy (i.e., such that its output feeds into the **phase** input of the **signal_source** block),
  - Set the **gain** parameter of the **gain** block to 0.1
  - Label the output of the **gain** block as **phase_in** using the **name_net** icon from the **icons2** listbox
- Copy the **signal_source** cell and place the copy at the location to the left of the **gain** block
  - Wire its output to the input of the **gain** block
  - Set its **stype** parameter to 1 (to create a sine wave with peak-to-peak amplitude of 2)
  - Set its **freq** parameter to **jitter_freq**
- Select the **constant** block from the **icons1** listbox, and then place this cell to the left of the newly created **signal_source** block.
  - Wire its output to the **phase** input of the newly created **signal_source** block
  - Set its **val** parameter to 0.

The left portion of the Sue2 schematic should now appear as shown below. Since the phase input of the signal_source block is in units of UI, the configuration below will vary the phase of the PRBS data stream with peak-to-peak variation of 0.2 UI at frequency **jitter_freq**.



SUE2: project_linear_cdr (schematic) ** MODIFIED ** --- C:/CppSim/Sue2/SueLib/CDR_Examples/project_linear_cdr.sue

C. Modifying the Simulation File and Running CppSim

In CppSim Run Menu, perform the following operations

- Click on **Edit Sim File** and make the following changes to the simulation file within the Emacs window that appears

```
//  Change the number of time steps such that it is 1,048,576 after we cut off the
//  first 100e3 samples (in order to remove transient effects)
//  Note that we choose 1,048,576 since the FFT function requires 2^n samples (n = 20
here)
num_sim_steps: 1148576

// Change the time step to allow better sampling of the key frequencies of interest (4 MHz
// and 40 MHz) in the Fourier Domain
Ts: 1/104.8576e9

// Probe the phase_in and phase_out nodes only
// and change the time period over which the test_closeup output file records signal data
output: test
probe: phase_in  phase_out
output: test_closeup start_sample=100e3
probe: phase_in phase_out

//  Add jitter_freq as a global parameter (note that the nominal value of 1e6 given here
//  is ignored in this case due to the alter statement below)
global_param:  jitter_freq = 1e6

// Alter the jitter_freq so that it hits two frequencies of interest (4 MHz and 40 MHz)
alter: jitter_freq = 4e6 40e6
```

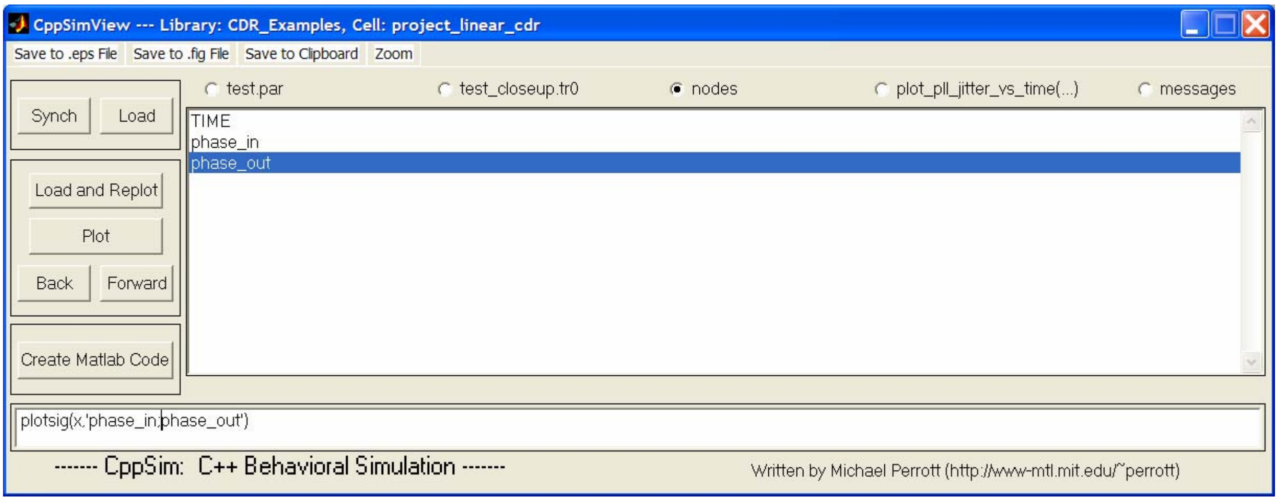
- Click on the Compile/Run button to run the simulation. You should see messages that indicate that both alter runs were simulated.

D. Measuring the Jitter Transfer at the Simulated Frequency Points

Let us now examine the resulting jitter waveforms of the last simulation in the time domain.

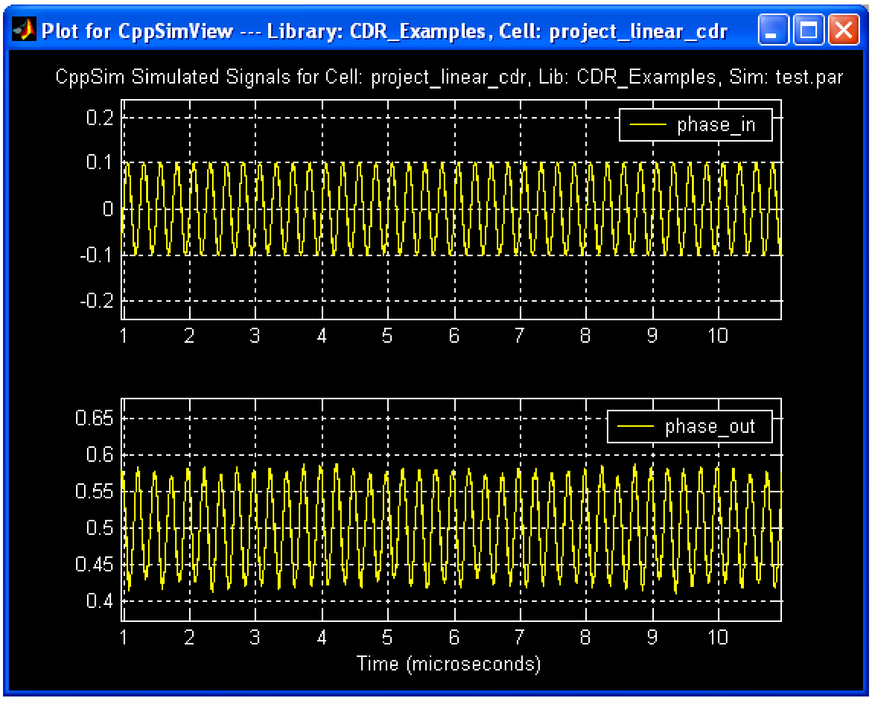
- Click on the **test.tr0** radio button and then select the **test_closeup.tr0** output file
- Click on the **plot_pll_jitter(…)** radio button, and then select **plotsig(…)** as the plotting function
- Click on the **No Nodes** radio button to load the signals into CppSimView
- Double-click on **phase_in** and then double-click on **phase_out**

At this point, CppSimView and its corresponding output plot should appear as shown below


CppSimView --- Library: CDR_Examples, Cell: project_linear_cdr
Save to .eps File
Save to .fig File
Save to Clipboard
Zoom
Synch
Load
Load and Replot
Plot
Back
Forward
Create Matlab Code
○ test.par
○ test_closeup.tr0
● nodes
○ plot_pll_jitter_vs_time(...)
○ messages
------ CppSim: C++ Behavioral Simulation ------
Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)

```
TIME
phase_in
phase_out
```

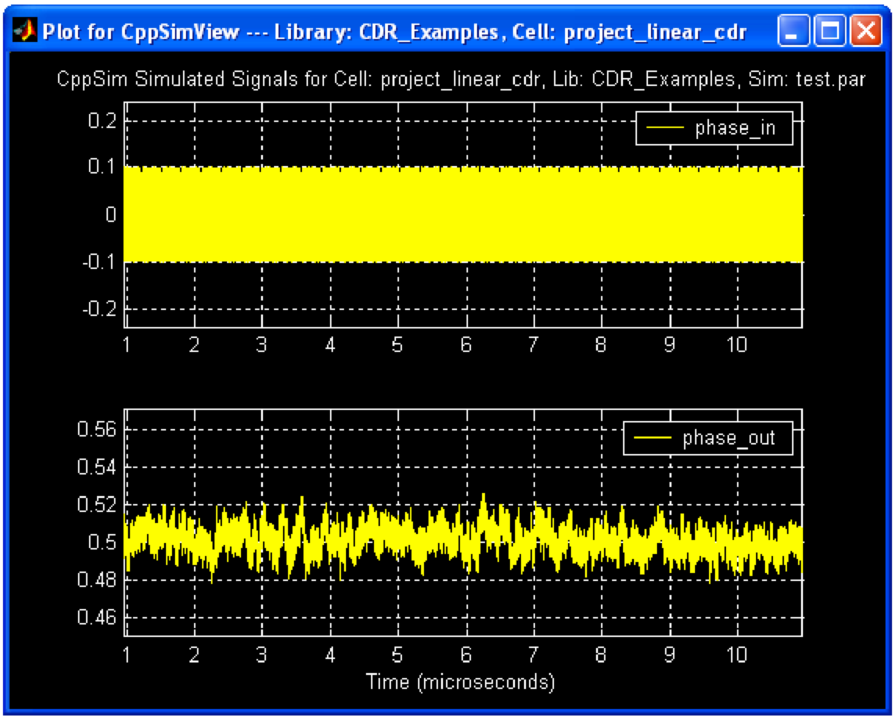
```
plotsig(x,'phase_in;phase_out')
```



Let us repeat the same exercise with the other output file

- Click on the **test_closeup.tr0** radio button and then select the **test_closeup.tr1** output file
- Click on the **No Nodes** radio button to load the signals into CppSimView
- Double-click on **phase_in** and then double-click on **phase_out**

The output plot of CppSimView should now appear as shown below


Plot for CppSimView --- Library: CDR_Examples, Cell: project_linear_cdr
CppSim Simulated Signals for Cell: project_linear_cdr, Lib: CDR_Examples, Sim: test.par
phase_in
0.2
0.1
0
-0.1
-0.2
1
2
3
4
5
6
7
8
9
10
phase_out
0.56
0.54
0.52
0.5
0.48
0.46
Time (microseconds)

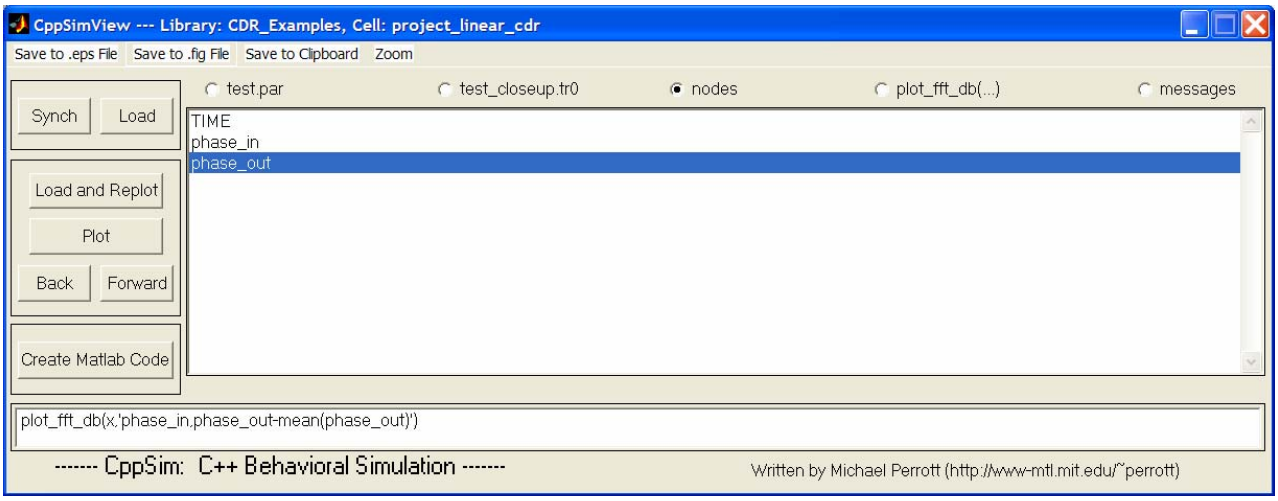
The jitter transfer specification is obtained by comparing the relative magnitude of the input and output jitter at the frequencies of interest (in this case, at 4 MHz and 40 MHz, respectively, in the above two plots). Unfortunately, the above plots reveal that this is a difficult task in the time domain. Instead, let us repeat this exercise in the frequency domain. We begin with the 4 MHz jitter signal.

- Click on the **test_closeup.tr1** radio button and then select the **test_closeup.tr0** output file
- Click on the **plotsig(…)** radio button, and then select **plot_fft_db(…)** as the plotting function
- Click on the **No Nodes** radio button to load the signals into CppSimView
- Double-click on **phase_in** and then double-click on **phase_out**
    - Modify the resulting plot expression within the command line so that it reads

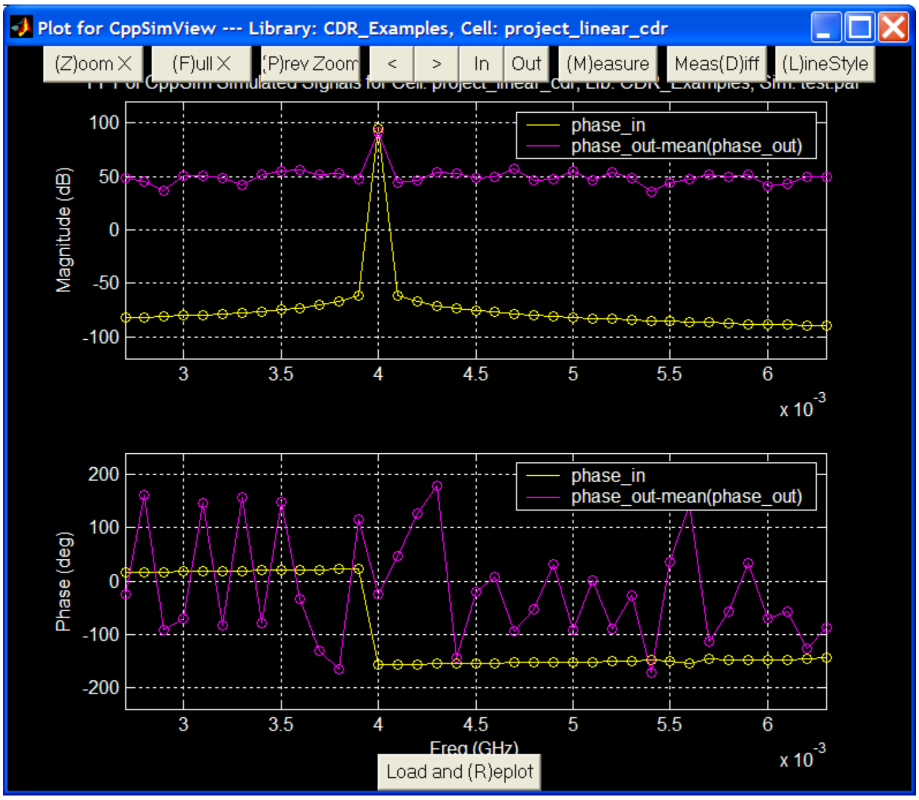
      plot_fft_db(x,'phase_in,phase_out-mean(phase_out)')

        - Note that the change from *semicolon* to *comma* in the above expression places both signals on the same subplot, and the subtraction by the mean of **phase_out** removes the overpowering effect of the DC component of **phase_out**
    - Press the **Enter** key to view the resulting plot
- Click on **Zoom** in the top banner of CppSimView
- In the plot window, use zoom and pan operations to focus on the frequency region around 4 MHz (i.e., 4e-3 with respect to the GHz axis in the plot)
- Click on (L)ineStyle to show the calculated points from the FFT operation

At this point, CppSimView and its corresponding output plot should appear as shown below.



```
plot_fft_db(x,'phase_in,phase_out-mean(phase_out)')
```



To measure the jitter transfer value at 4 MHz, we simply need to compare the magnitude of the FFT values of **phase_in** and **phase_out** at this frequency. We can determine this difference using the **Meas(D)iff** command in the plot window

- Click on the **Meas(D)iff** button
- Left-click on the upper signal point between the two signals, ‘phase_in’ and ‘phase_out-mean(phase_out)’, at 4 MHz in the Magnitude subplot. You can continue left-clicking until you are confident you have indeed selected the top point
- Right-click on the under signal point between the two signals, ‘phase_in’ and ‘phase_out-mean(phase_out)’, at 4 MHz in the Magnitude subplot. Again, you can continue right-clicking until you are confident that the proper point is selected
- Left-click to end the **Meas(D)iff** operation and display the result

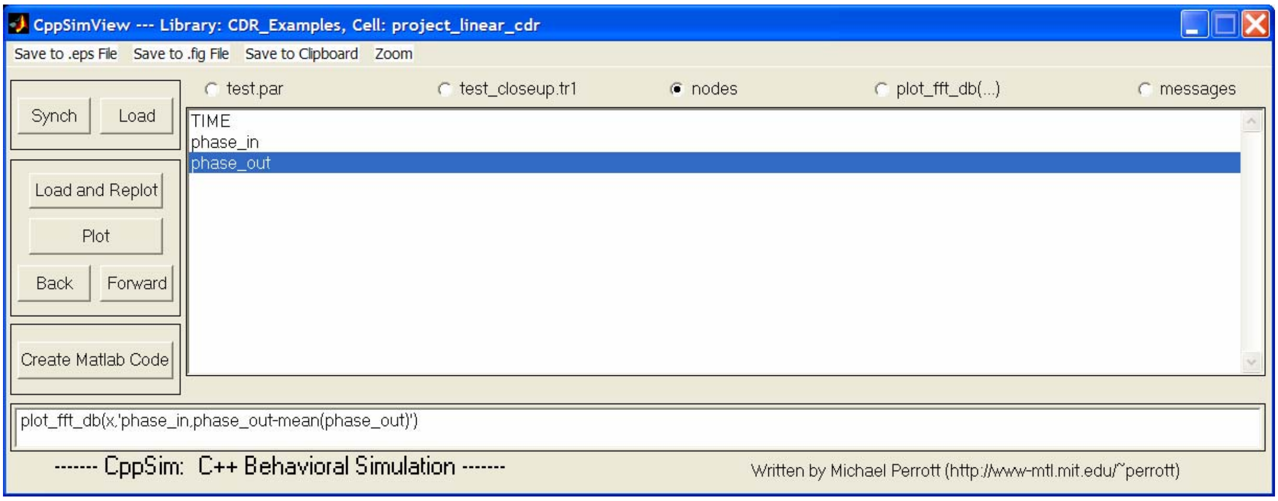
- The resulting Delta Y-Value should approximately equal -2.4 dB

Let us now repeat this exercise for the 40 MHz jitter signal

- Click on the **test_closeup.tr0** radio button and then select the **test_closeup.tr1** output file
- Click on the **No Nodes** radio button to load the signals into CppSimView
- Double-click on **phase_in** and then double-click on **phase_out**
  - Modify the resulting plot expression within the command line so that it reads

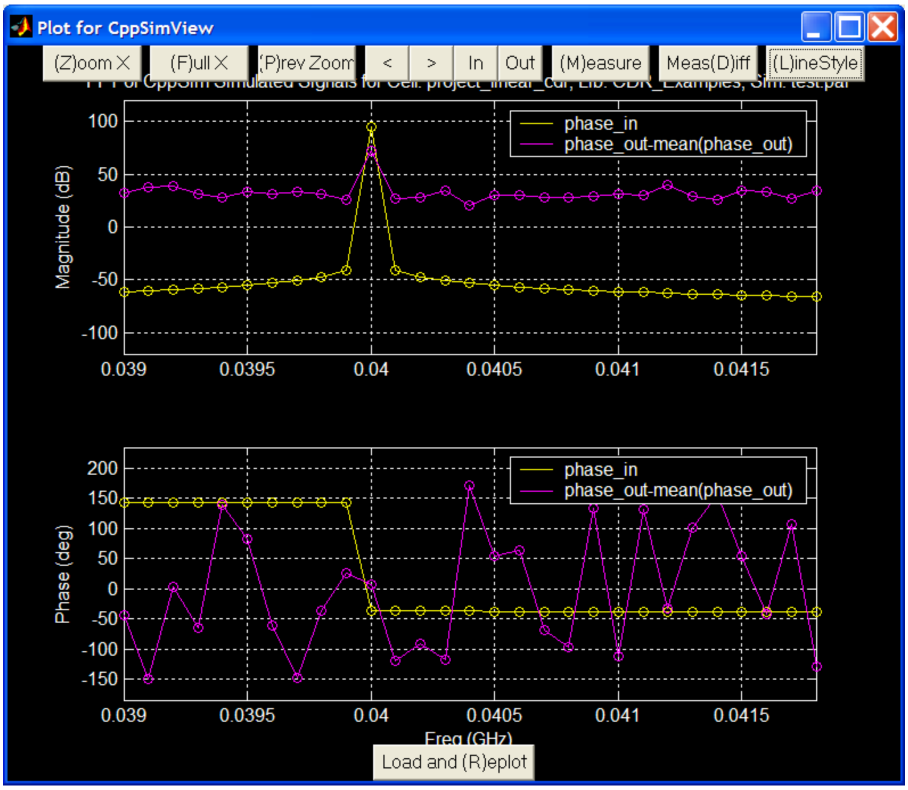
    plot_fft_db(x,’phase_in,phase_out-mean(phase_out)’)
  - Press the **Enter** key to view the resulting plot
- Click on **Zoom** in the top banner of CppSimView
- In the plot window, use zoom and pan operations to focus on the frequency region around 40 MHz (i.e., 0.04 with respect to the GHz axis in the plot)
- Click on (L)ineStyle to show the calculated points from the FFT operation

At this point, CppSimView and its corresponding output plot should appear as shown below.



TIME
phase_in
phase_out

plot_fft_db(x,'phase_in,phase_out-mean(phase_out)')



To measure the jitter transfer value at 40 MHz, we again compare the magnitude of the FFT values of **phase_in** and **phase_out** at this frequency.

- Click on the **Meas(D)iff** button
- Left-click on the topmost signal point at 40 MHz in the Magnitude subplot.
- Right-click on the signal point below the topmost signal point at 40 MHz in the Magnitude subplot.
- Left-click to end the **Meas(D)iff** operation and display the result
    - The resulting Delta Y-Value should approximately equal -22.5 dB

In practice, one would want to characterize the jitter transfer specification over many frequency points beyond just the two considered here. It should be clear that using CppSimView to accomplish this task would prove quite tedious. A better way to accomplish this task is to write a Matlab script to perform the above functions automatically on a large set of transient files (each corresponding to a different jitter frequency as generated by CppSim using the **alter:** statement). It is straightforward to create such a script and have it automatically plot the resulting jitter transfer curve as calculated at each frequency point.

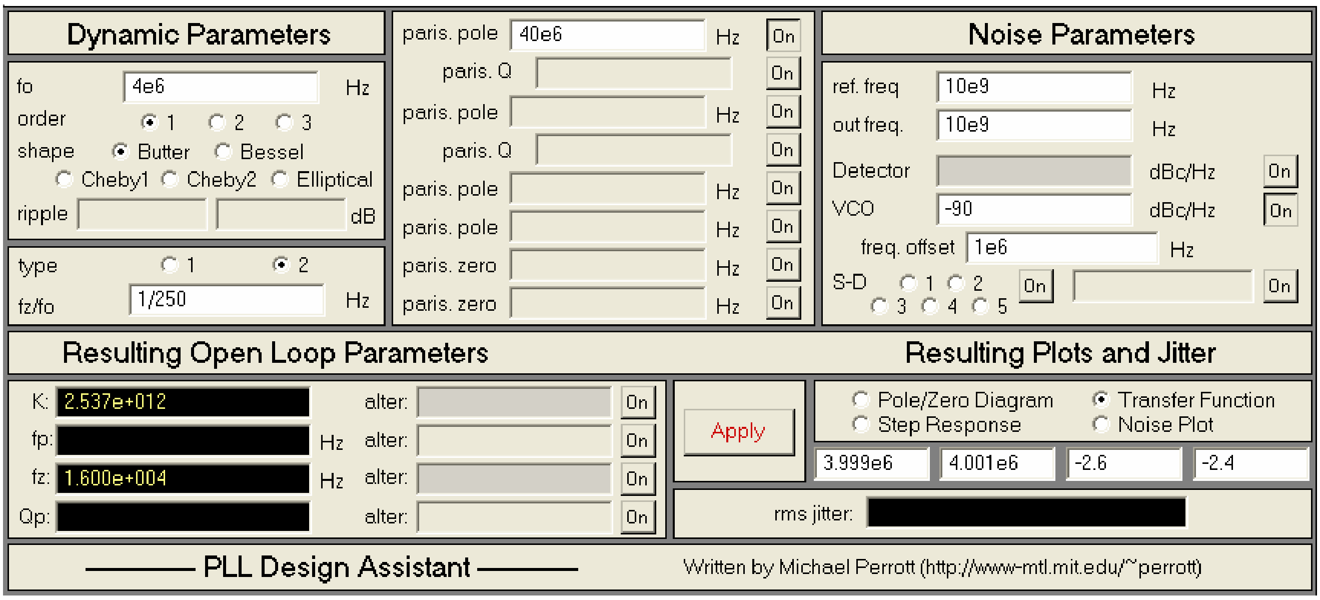
E. Comparing Calculated and Measured Jitter Transfer Values

Let us now return to the PLL Design Assistant to determine the calculated value of jitter transfer at the offset frequencies of 4 MHz and 40 MHz. Be sure to load on the same configuration as entered in the previous subsection **Comparing the Simulated and Calculated Jitter Generation Performance.**

- Within the PLL Design Assistant, click on the **Transfer Function** radio button
    - You should see a plot of the nominal jitter transfer curve spanning 10 kHz to 100 MHz

- Focus on the 4 MHz offset range by appropriately changing the axis limits
    - Set the axis to **[3.999e6 4.001e6 -2.6 -2.4]**

The PLL Design Assistant and its resulting plot are shown below. We see that the calculated jitter transfer value is -2.54 dB, which is reasonably close the simulated value of -2.4 dB as measured above.



Dynamic Parameters

fo 4e6 Hz
order ● 1 ○ 2 ○ 3
shape ● Butter ○ Bessel
○ Cheby1 ○ Cheby2 ○ Elliptical
ripple dB

type ○ 1 ● 2
fz/fo 1/250 Hz

paris. pole 40e6 Hz On
paris. Q On
paris. pole Hz On
paris. Q On
paris. pole Hz On
paris. pole Hz On
paris. zero Hz On
paris. zero Hz On

Noise Parameters

ref. freq 10e9 Hz
out freq. 10e9 Hz
Detector dBc/Hz On
VCO -90 dBc/Hz On
freq. offset 1e6 Hz
S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 2.537e+012 alter: On
fp: Hz alter: On
fz: 1.600e+004 Hz alter: On
Qp: alter: On

Apply

Resulting Plots and Jitter

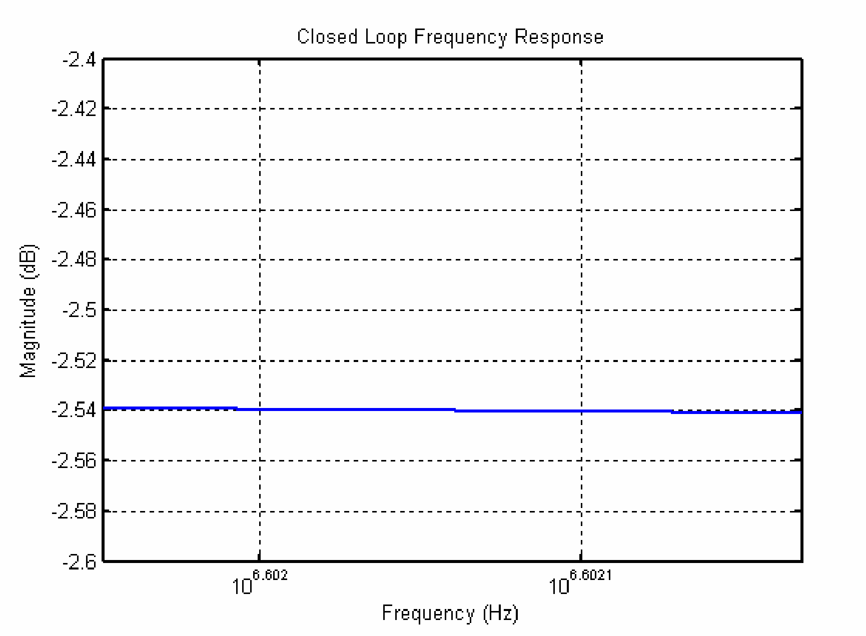
○ Pole/Zero Diagram ● Transfer Function
○ Step Response ○ Noise Plot

3.999e6 4.001e6 -2.6 -2.4

rms jitter:

PLL Design Assistant

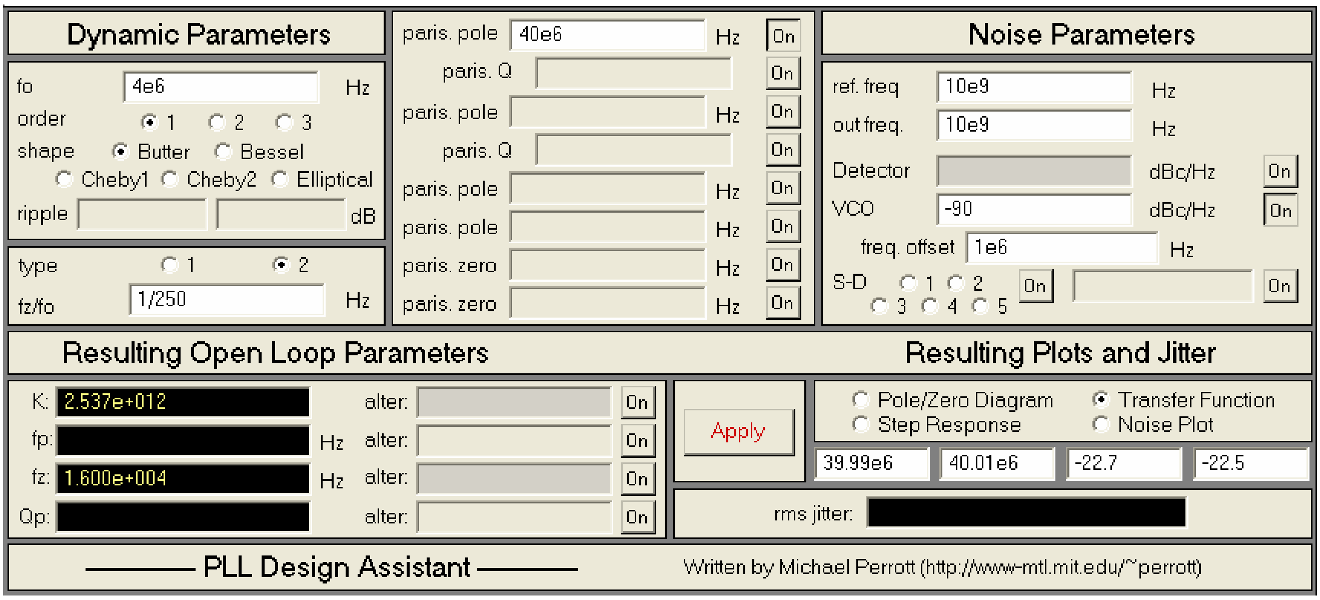
Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)



Now let us focus on the 40 MHz offset range by appropriately changing the axis limits

- Set the axis to **[39.99e6 40.01e6 -22.7 -22.5]**

The PLL Design Assistant and its resulting plot are shown below. We see that the calculated jitter transfer value is -22.54 dB, which is very close the simulated value of -22.58 dB as measured above.



Dynamic Parameters

fo 4e6 Hz

order ● 1 ○ 2 ○ 3

shape ● Butter ○ Bessel

○ Cheby1 ○ Cheby2 ○ Elliptical

ripple dB

type ○ 1 ● 2

fz/fo 1/250 Hz

paris. pole 40e6 Hz On

paris. Q On

paris. pole Hz On

paris. Q On

paris. pole Hz On

paris. pole Hz On

paris. zero Hz On

paris. zero Hz On

Noise Parameters

ref. freq 10e9 Hz

out freq. 10e9 Hz

Detector dBc/Hz On

VCO -90 dBc/Hz On

freq. offset 1e6 Hz

S-D ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 On On

Resulting Open Loop Parameters

K: 2.537e+012 alter: On

fp: Hz alter: On

fz: 1.600e+004 Hz alter: On

Qp: alter: On

Apply

Resulting Plots and Jitter

○ Pole/Zero Diagram ● Transfer Function

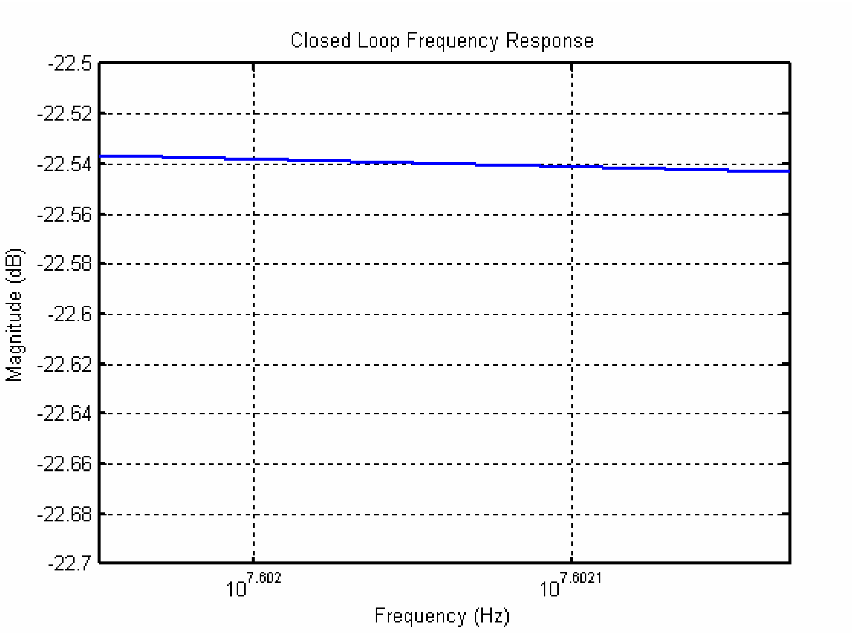
○ Step Response ○ Noise Plot

39.99e6 40.01e6 -22.7 -22.5

rms jitter:

PLL Design Assistant

Written by Michael Perrott (http://www-mtl.mit.edu/~perrott)



Simulation of Jitter Tolerance

Simulation of the jitter tolerance performance of the CDR is, unfortunately, beyond the scope of this document. In practice, it is common to focus only on jitter transfer and generation performance when using simulation to examine a SONET CDR, the reason being that jitter tolerance where 1e-12 bit error rates are sought is very difficult (if not impossible) to directly simulate.

An indirect way to assess jitter tolerance is to examine jitter transfer of the CDR through calculations and behavioral simulation (as done above), and insure that the jitter transfer bandwidth is as high as possible while still meeting the jitter transfer specification across all temperature and process variations. The CDR is best able to track out incoming jitter by having the widest possible bandwidth.

In addition to seeking the widest possible PLL bandwidth for the CDR, SPICE level simulations of the phase detector are run to determine the setup and hold times of its key re-clocking register and the steady-state phase offset of the detector across temperature and process variations. The best jitter tolerance is obtained by having minimal phase offset in the phase detector, and the smallest possible setup and hold time requirements in the re-clocking register. The former issue is achieved by careful design, and the latter issue is addressed by utilizing an adequately fast process technology for the data rates sought after.

One should note that estimation of BER through behavioral simulation is a topic of current interest. It is likely that methods will exist in the next few years to address this issue.

Conclusion

This tutorial covered the basic design principles involved in designing CDR circuits for the OC192 SONET standard. The PLL Design Assistant was utilized to perform high level calculations of the CDR performance under different nominal parameter settings and associated variations. The CppSim behavioral simulator was used to verify the CDR performance within a simulation context. Although this project focused in CDR design only for SONET applications, it has hopefully provided general guidelines of how to approach CDR design for other applications, as well.