Behavioral Simulation of a High Speed Limit Amplifier with Fast Offset Compensation Using CppSim and the PLL Design Assistant Programs

Ethan Crain and Michael Perrott
http://www-mtl.mit.edu/research/perrottgroup/

July 13, 2008

Table of Contents

Setup....................2
Introduction....................4
A. Limit Amplifier Architecture....................5
B. System Architecture....................5
Preliminaries....................7
A. Opening Sue2 Schematics....................7
B. Running CppSim Simulation....................9
Plotting Time-Domain Results....................10
A. Output signal plots....................10
B. Output signal Eye Diagrams....................13
Plotting Frequency Domain Results....................14
A. RMS Jitter....................14
Examining Nonidealities....................15
A. Intersymbol Interference (ISI)....................15
B. Limit Amplifier Noise....................20
C. Amplifier Non-Linearity....................26
Amplifier non-linearity can influence the offset compensation. Ultimately, non-linearities in the amplifier transfer function can translate to residual output-referred offset. Therefore, we must consider it in this model. To explore this affect:....................26
E. Input Amplitude....................27
F. Feedback Noise and Offset....................27
Design Overview of the Offset Compensation Loop....................31
Key parameters of the limit amplifier and feedback path include:....................31
A. Modeling....................31
B. Determining System Parameters with PLL Design Assistant....................32
Comparison of Simulated and Measured Results....................34
Conclusion....................36

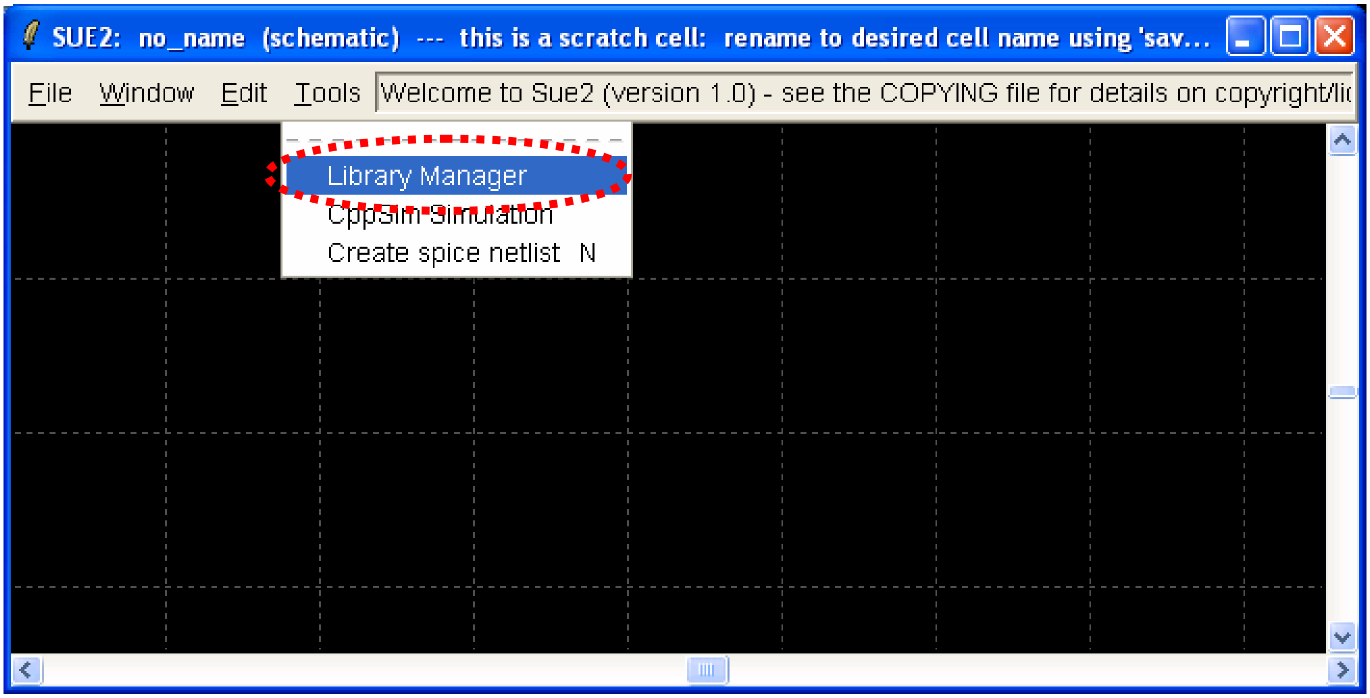
Setup

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

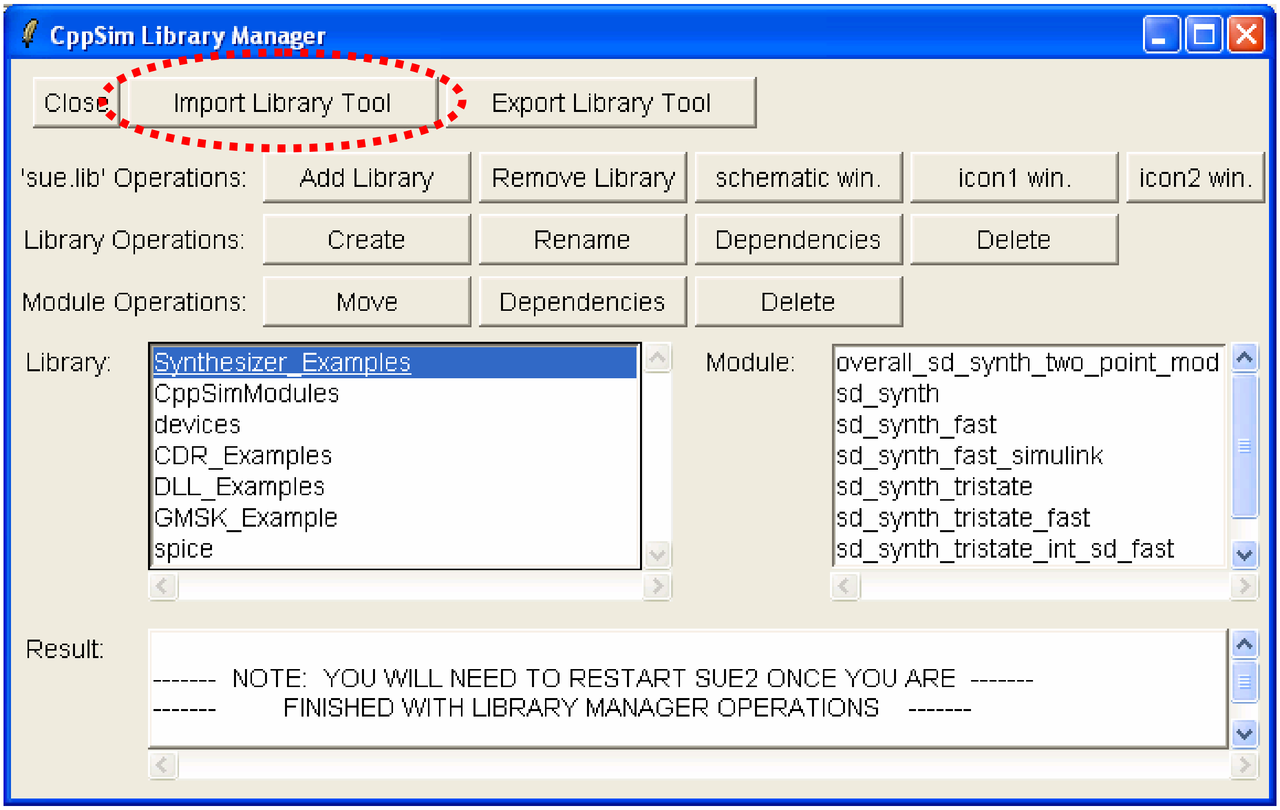
http://www.cppsim.com

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. You should also 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 as it is briefly used in this document.

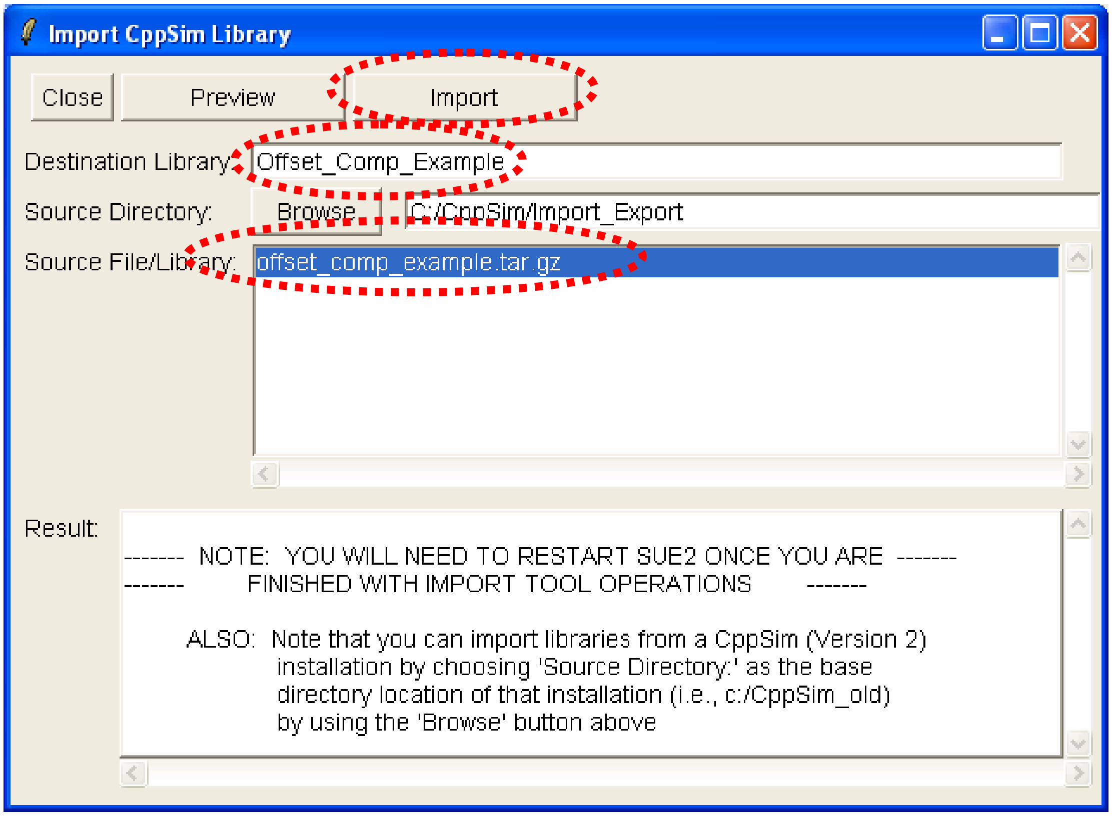
To run this tutorial, you will also need to download the file **offset_comp_example.tar.gz** available at http://www.cppsim.com, and place it in the **Import_Export** directory of CppSim (assumed to be **c:/CppSim/Import_Export**). Once you do so, start up **Sue2** by clicking on its icon, and then click on **Tools->Library Manager** as shown in the figure below.



In the **CppSim Library Manager** window that appears, click on the **Import Library Tool** button as shown in the figure below.



In the **Import CppSim Library** window that appears, change the **Destination Library** to **Offset_Comp_Example**, click on the **Source File/Library** labeled as **offset_comp_example.tar.gz**, and then press the **Import** button as shown in the figure below. Note that if **offset_comp_example.tar.gz** does not appear as an option in the **Source File/Library** selection listbox, then you need to place this file (downloaded from http://www.cppsim.com) in the **c:/CppSim/Import_Export** directory.



Once you have completed the above steps, restart **Sue2** as directed in the above figure.

Introduction

Most significant improvements in performance in increasingly complex communications systems will arise from architectural innovations. These innovations are only possible you can quickly and accurately model and simulate the system under consideration. CppSim, initially designed for simulating phase-locked loops, is a free behavioral simulation package that leverages the C++ language to allow very fast simulation of a wide array of system types. The goal of this tutorial is to expose the reader to a non-PLL based system where modeling with CppSim and the PLL Design Assistant tools enabled an architectural innovation that produces dramatic improvements in performance.

This document explores a fast offset compensation design for use in high gain amplifiers that was presented in [1]. High gain amplifiers require offset compensation to achieve high input sensitivity and low output jitter. Traditional offset compensation designs use a low-pass filter connected in feedback between the amplifier output and its input. Unfortunately, this approach suffers from an undesirable tradeoff between offset compensation settling time and output jitter – a higher compensation bandwidth has the benefit of achieving faster settling time at the expense of higher data dependent jitter. Therefore, current high-speed data-links suffer from extremely long offset settling times in order to satisfy the strict protocol jitter specifications. This tutorial examines the design and operation of a new offset compensation design that dramatically alters the relationship between output jitter and settling time.

---

[1] E. A. Crain, M. H. Perrott, 'A 3.125Gb/s Limit Amplifier with 42dB Gain and 1μS Offset Compensation in 0.18μm CMOS,' IEEE ISSCC Feb. 2005.

A. Limit Amplifier Architecture

The main amplifier is implemented as a cascaded multi-stage limit amplifier to achieve a high gain-bandwidth product [2]. The number of stages selected is a compromise between the conflicting desires for high gain and bandwidth and low power dissipation and input referred noise. Figure 1 shows the top level of the limit amplifier design. The individual stages of the limit amplifier could be implemented in a variety of ways but the resistor-loaded differential pair topology is very amenable to high-speed operation.

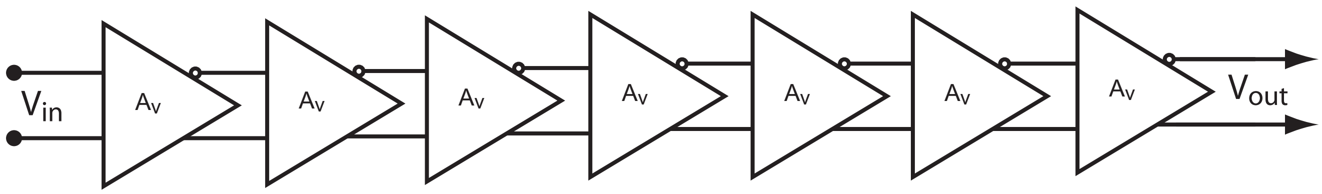


Figure 1: Top level schematic of the limit amplifier.

B. System Architecture

The simplified system architecture of the limit amplifier and new offset compensation is shown in Figure 2. The feed-forward path, which is the data path, consists of a high-speed limit amplifier. The feedback path, which implements the offset compensation, consists of a peak detector followed by an integrator.

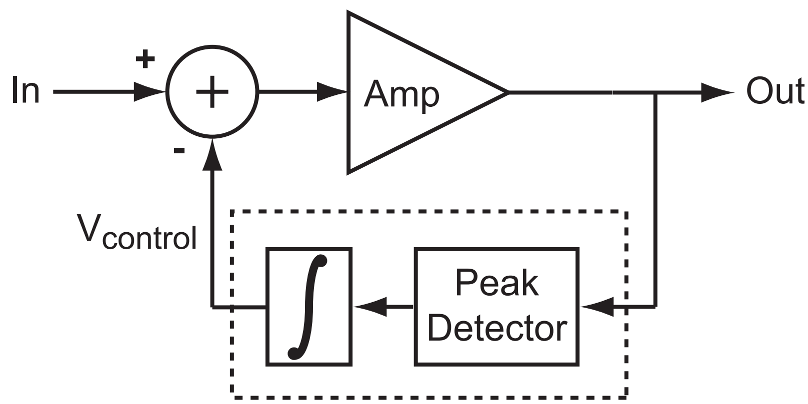


Figure 2: Simplified system architecture.

[2] M. H. Perrott, www-mtl.mit.edu/~perrott/ - Link to 6.976 Open Course Ware site, Lecture 6, pp. 21-25.

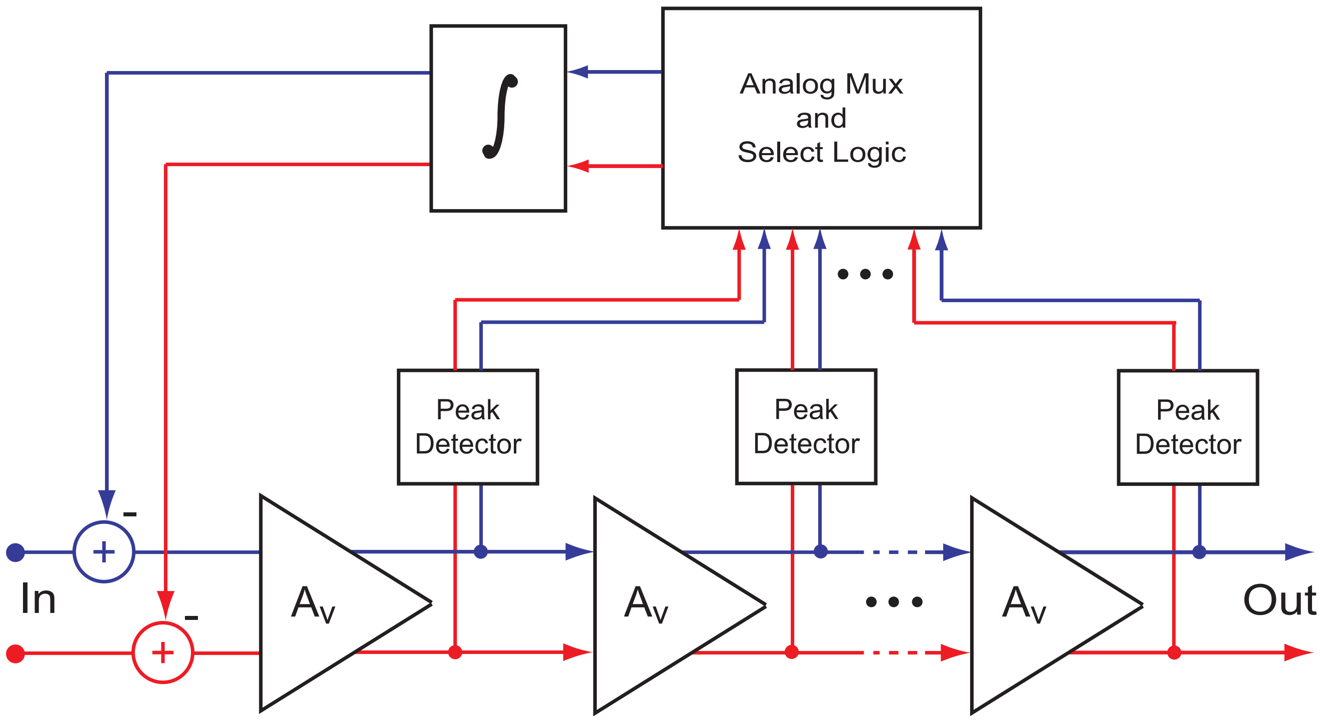
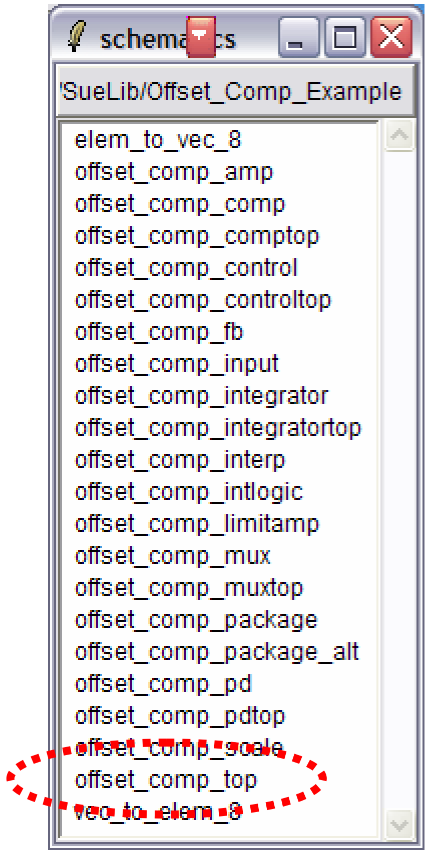


Figure 3: Offset block architecture.

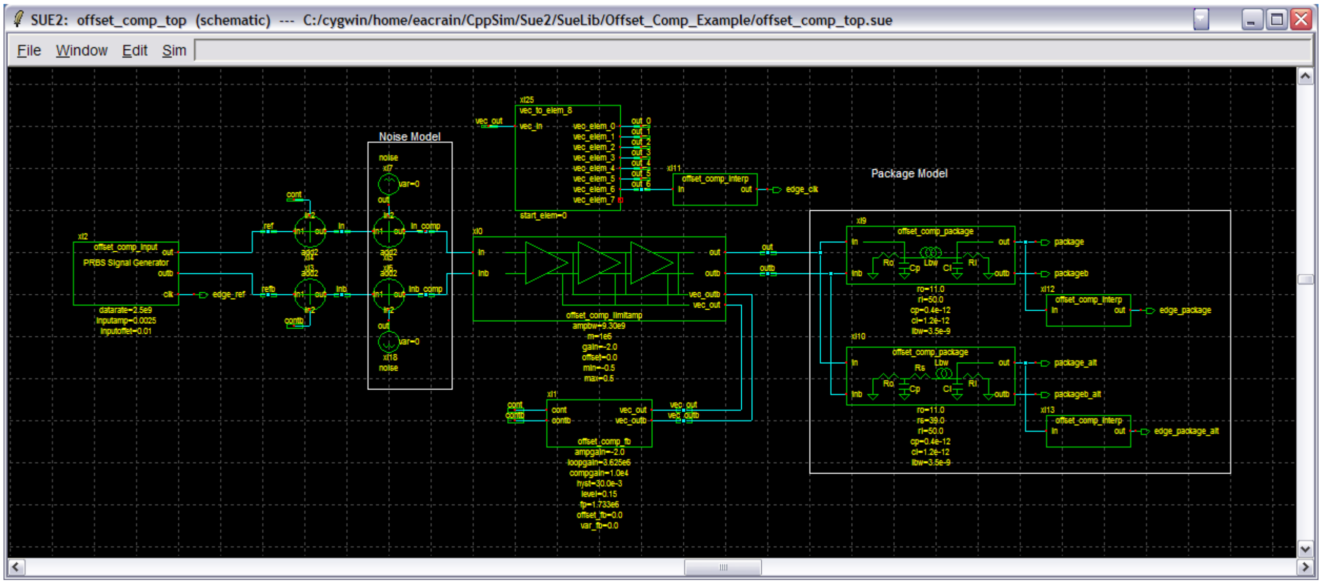
Preliminaries

A. Opening Sue2 Schematics

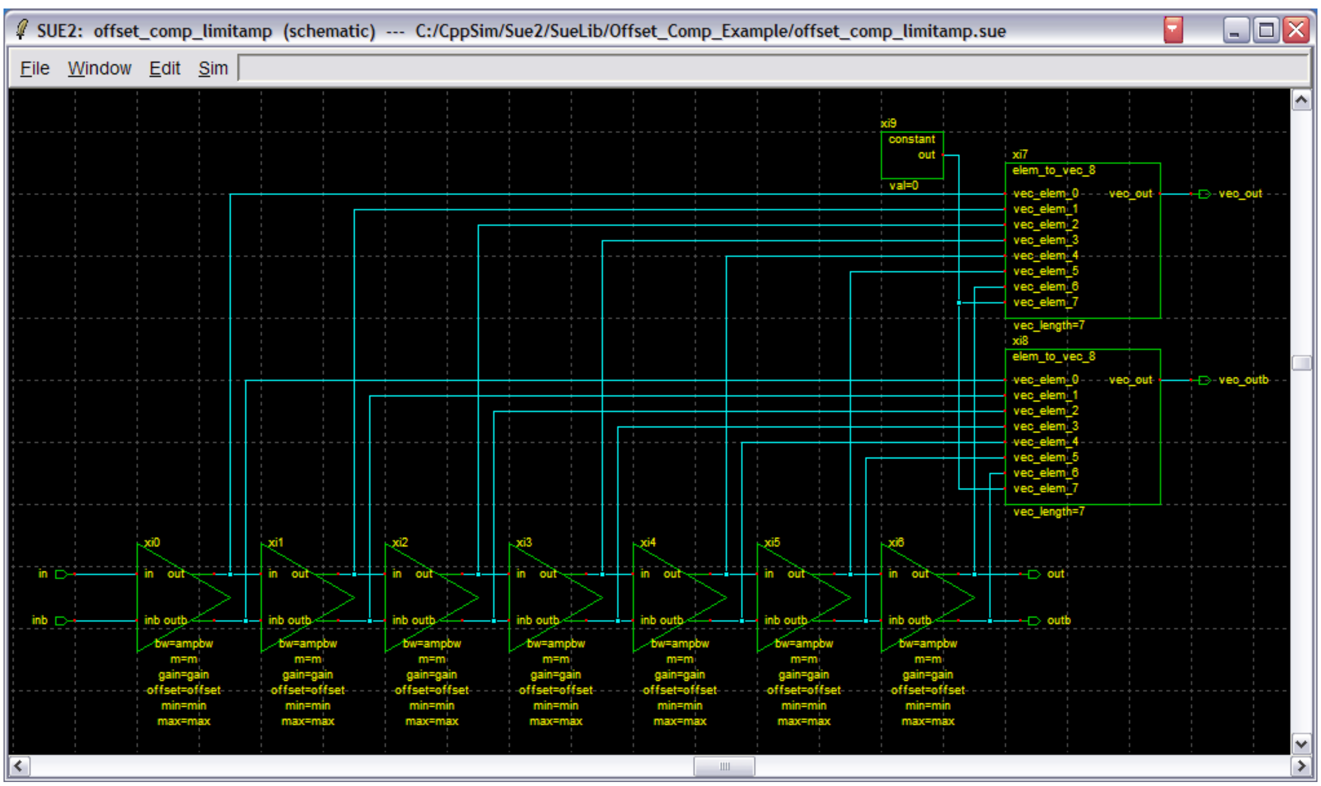
- Click on the Sue2 icon to start Sue2, and then select the **Offset_Comp_Example** library from the **schematic listbox**. The **schematic listbox** should now look as follows:



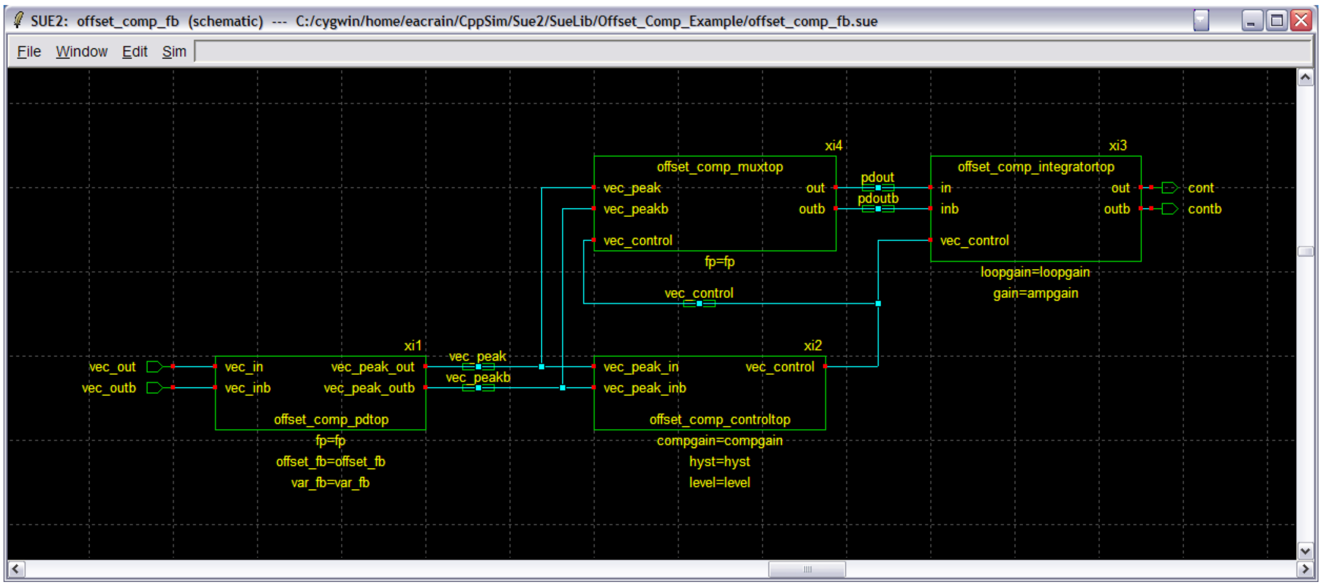
- Select the **offset_comp_top** cell from the above **schematic listbox**. The Sue2 schematic window should now appear as shown below. Key signals for this schematic include
  - **in** and **inb**: limit amplifier differential input signals
  - **out** and **outb**: limit amplifier differential output signals
  - **package** and **packageb**: differential signals at the output of the package model
  - **cont** and **contb**: differential control signals for the offset compensation



- Select the **offset_comp_limitamp** icon within the above schematic, and then press **e** to descend down into its associated schematic. You should now see the schematic shown below. The cell **elem_to_vec_8** allows for buses in vector form. Some key signals in the limit amplifier schematic include:
    - **in** and **inb**: the differential inputs to the limit amplifier
    - **vec_out** and **vec_outb**: vectors for the differential output signals of all amplifier stages

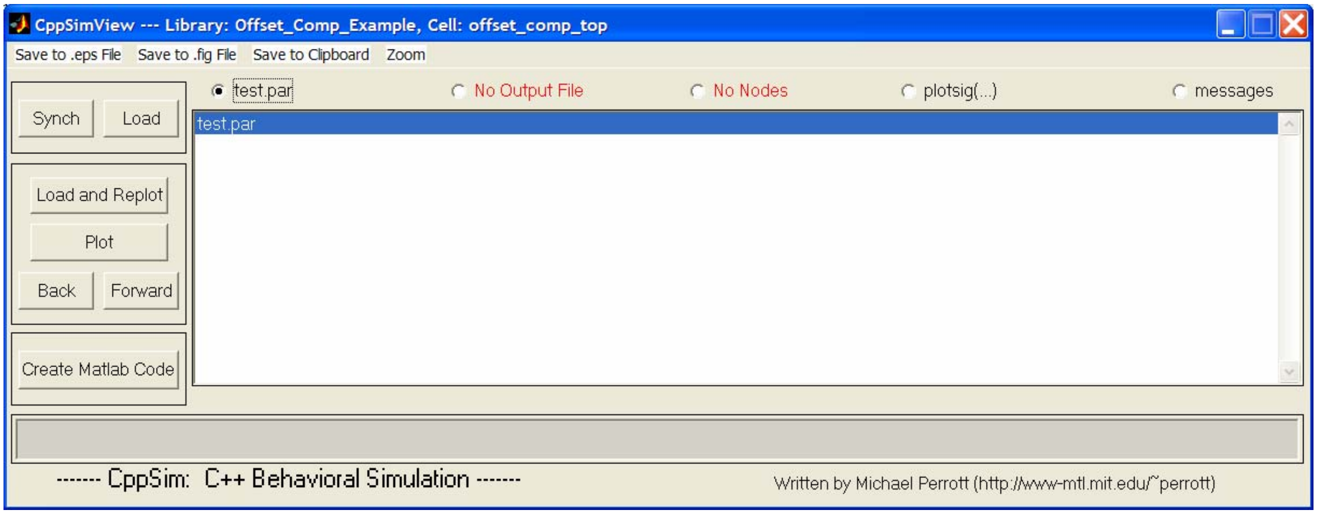


- Select the **offset_comp_fb** icon within the top level schematic, and then press **e** to descend down into its associated schematic. You should now see the schematic shown below. Some key signals in the feedback schematic include
    - **vec_peak** and **vec_peakb**: vectors for the differential output signals of all peak detector outputs
    - **pdout** and **pdoutb**: differential output signals of peak detector that is active in the compensation loop
    - **vec_control** and **vecint_control**: vectors for the peak detector select logic control signals and integrator control signals
- Press **Ctrl-e** to return to the **offset_comp_top** cellview.

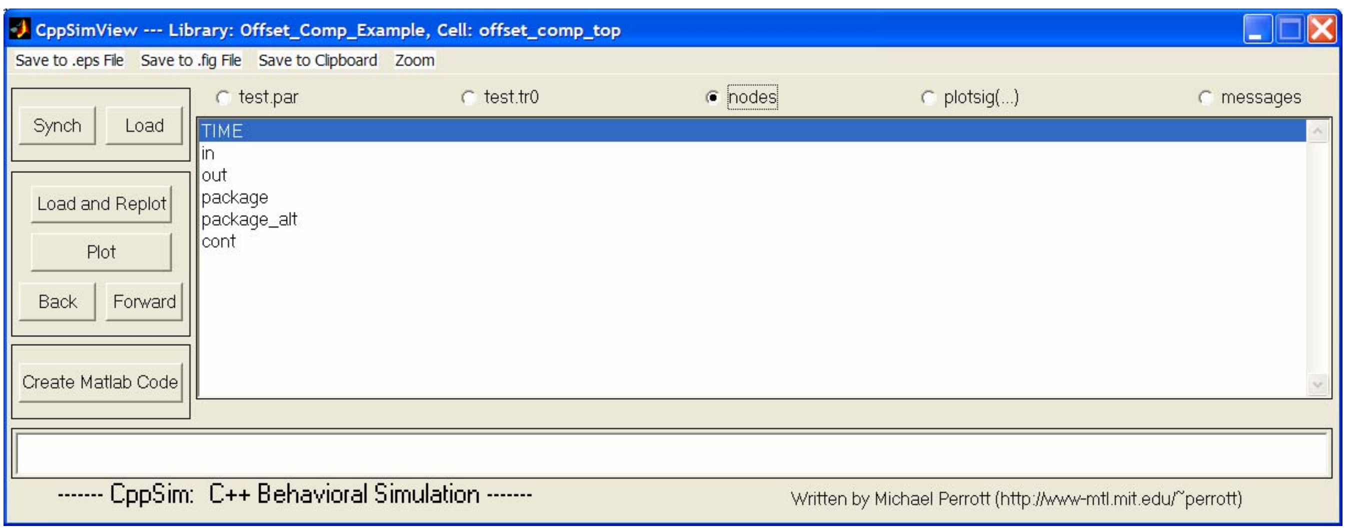


B. Running CppSim Simulation

- 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.



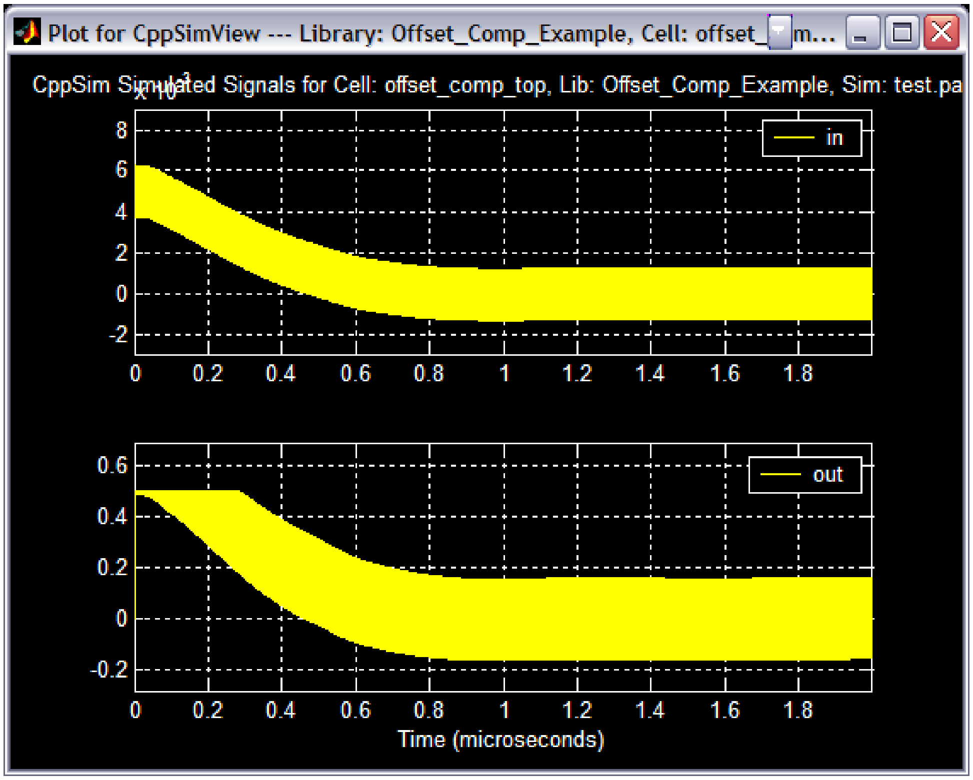
- Within the Emacs window, indicates that **num_sim_steps** is set to 10.0e4 and **Ts** is set to 1/50e9. The sample rate, 1/**Ts**, is set to at least 10 times that of the highest frequency in the simulation. You may then close the Emacs window if you like (cntl-x cntl-c).
- Click on the **Compile/Run** button to run the simulation.
- Click on the **No Output File** radio button and select **test.tr0** as the output file.
- Click on the **No Nodes** radio button to load in the simulated signals. CppSimView should now appear as shown below



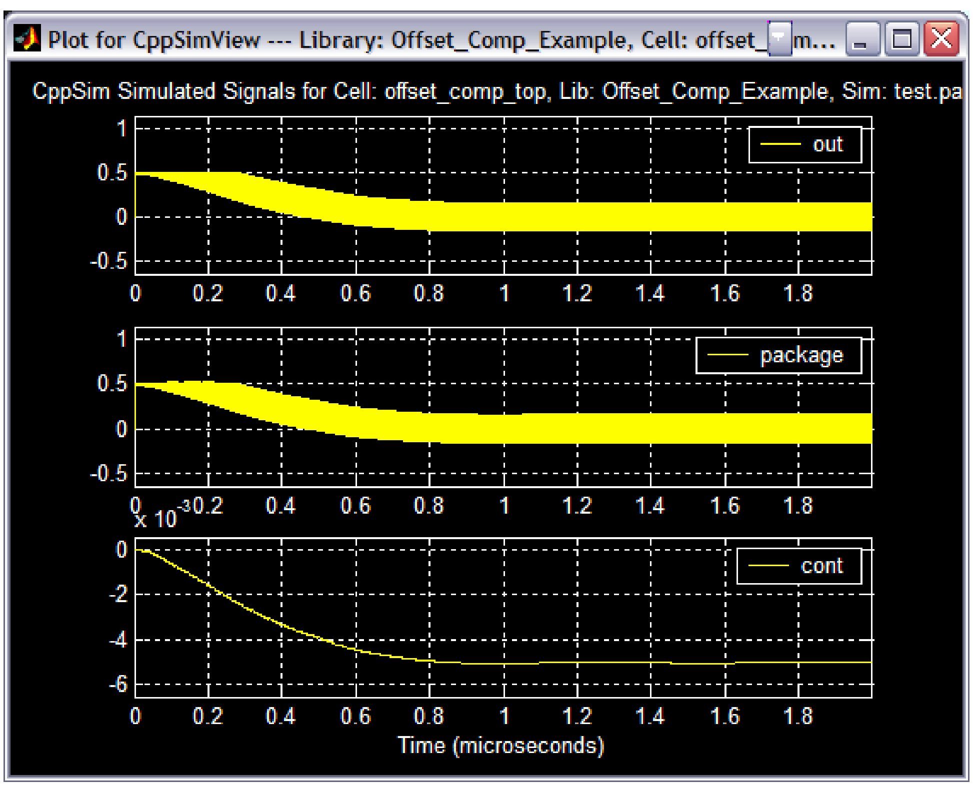
Plotting Time-Domain Results

A. Output signal plots

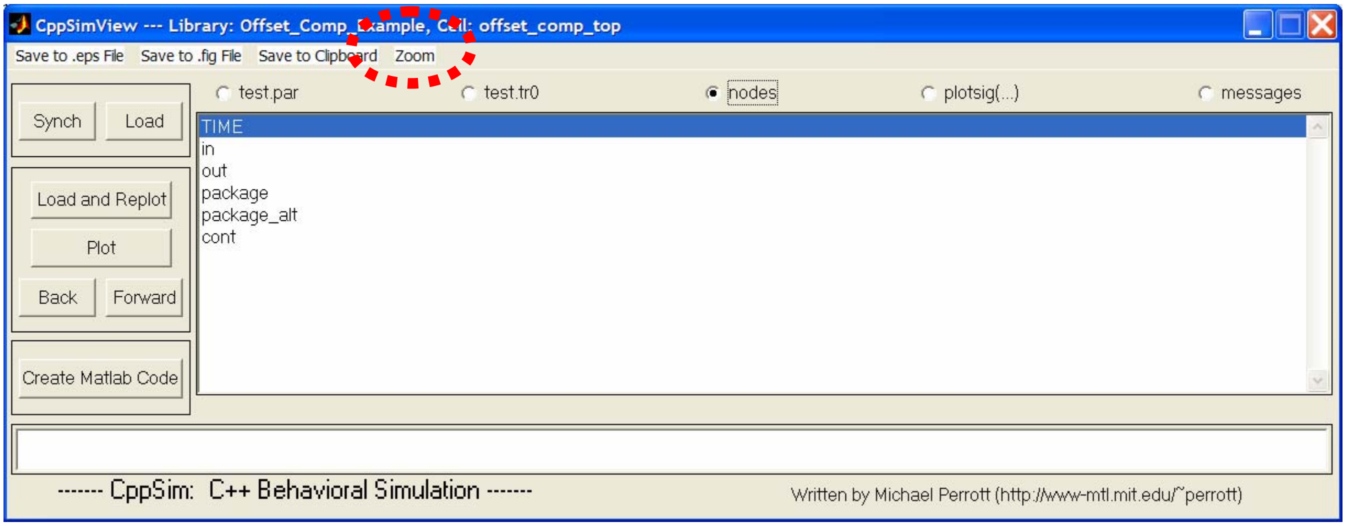
- The input data is a 2.5mVpp PRBS data stream at 2.5Gb/s.
- In the CppSimView window, double-click on signals **in** and **out.** You should see plots of the limit amplifier input and output waveforms as shown below:



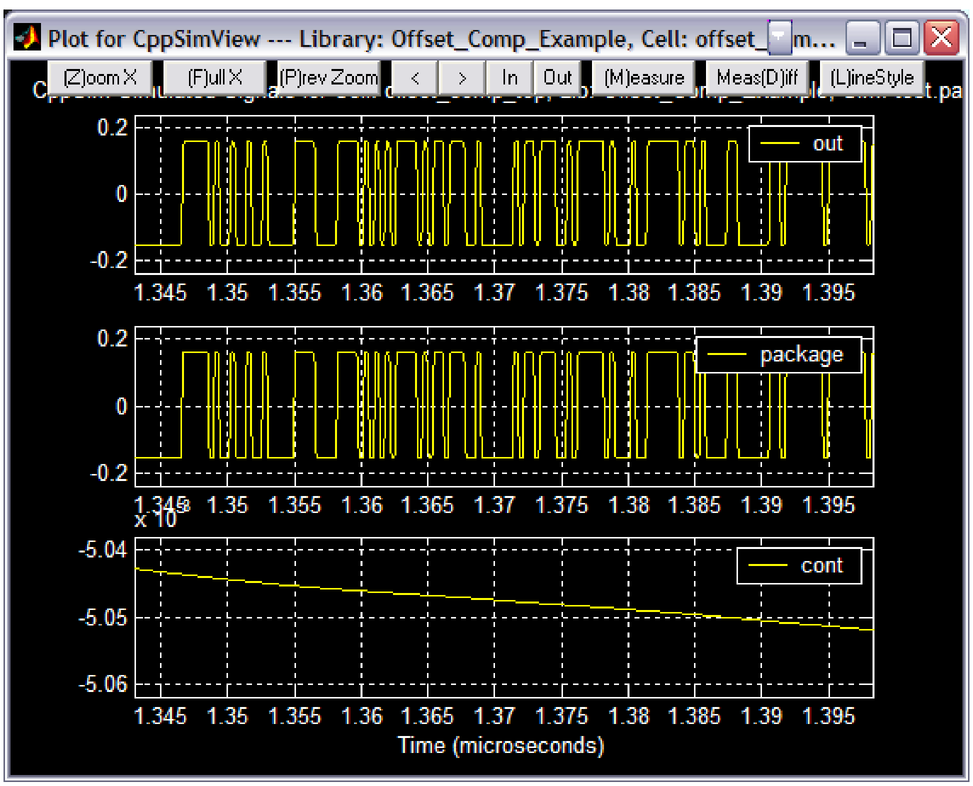
- Now click on the **Reset Node List** button in the CppSimView window, and then double-click on signals **out**, **package** and **cont.** You should see plots of the amplifier output before and after the package model and the feedback control signal as shown below:



- To change the x-axis of the figure (the y-axis automatically scales), hit the **Zoom** radio button on the CppSimView menu-bar, as highlighted in the figure below:

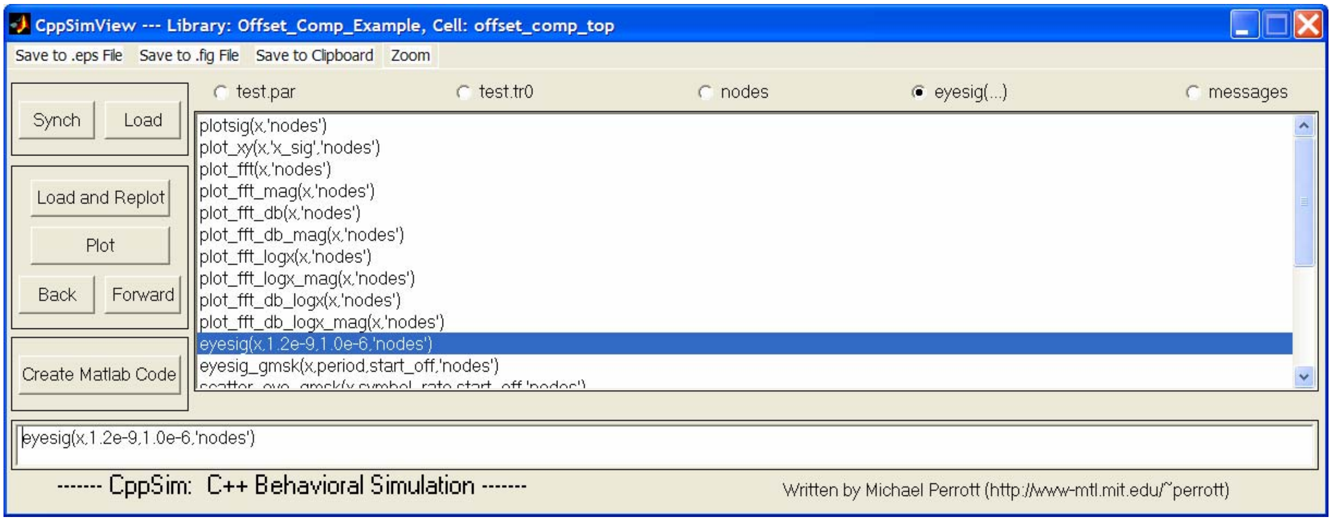


- Next, click the **(Z)oom X** push-button located at the top of the plot figure. Select the desired x-axis range by clicking at the beginning and ending location in any of the plotted signals. The figure will look similar to the figure below. Additionally, you can zoom in and out and pan left and right using the **In** and **Out** and the < and > push-buttons, respectively, located at the top of the plot figure.

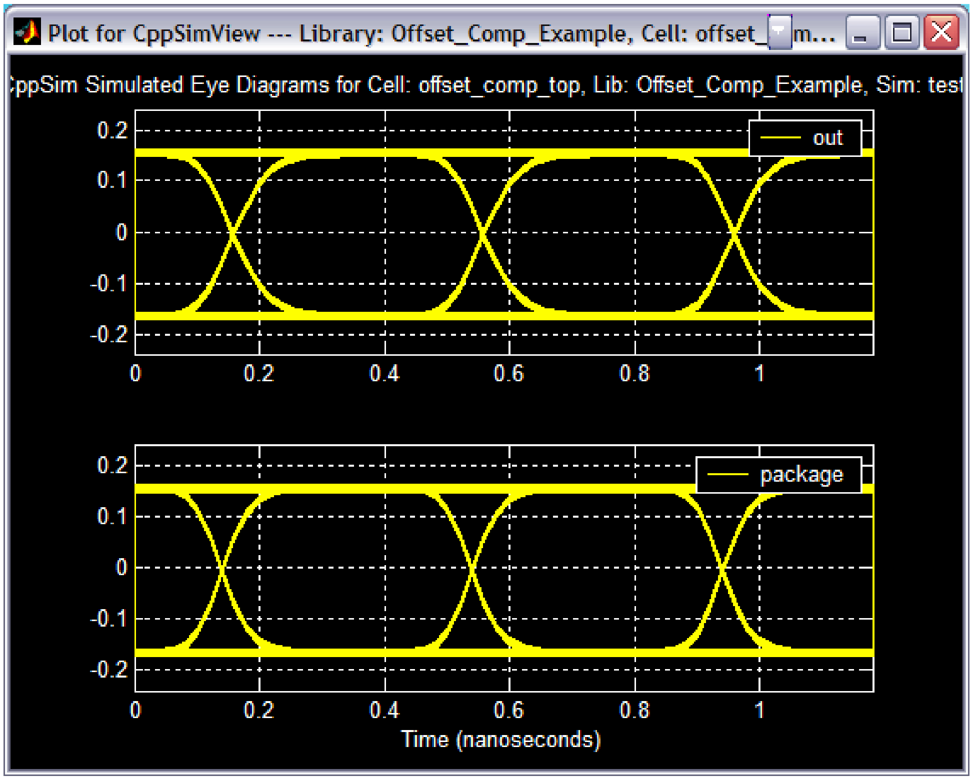


B. Output signal Eye Diagrams

- Click on the **plotsig(…)** radio button in CppSimView and then select the **eyesig(…)** function. Set **period** to be 1.2e-9 (i.e., three symbols long for the output signal waveforms – note that the signal waveforms have twice the symbol period as the instantaneous frequency waveform since they are NRZ signals) and **start_off** to be 1.0e-6 (i.e., 1 microsecond, which is long enough for the offset compensation to settle). Hit **Return** to enter the function into the CppSimView function list. CppSimView should now appear as shown below.



- Clock on the **nodes** radio button, and then double-click on signal **out** and then on signal **package**. The eye diagrams of the ideal limit amplifier output signals before and after the package model should appear as shown below.

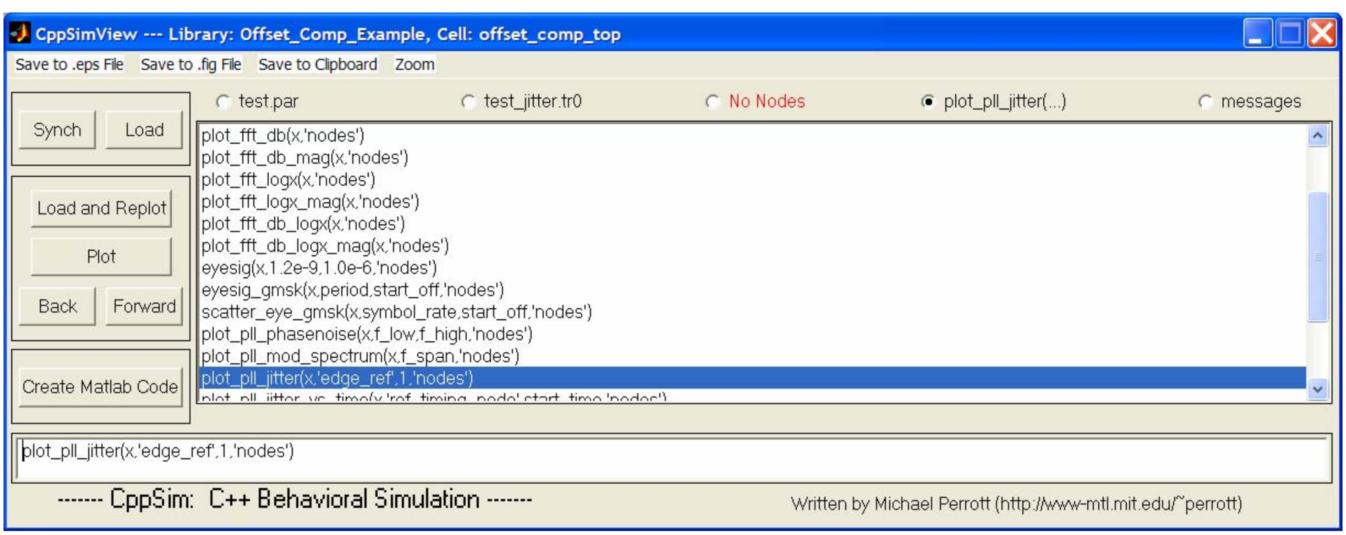


<u>Plotting Frequency Domain Results</u>

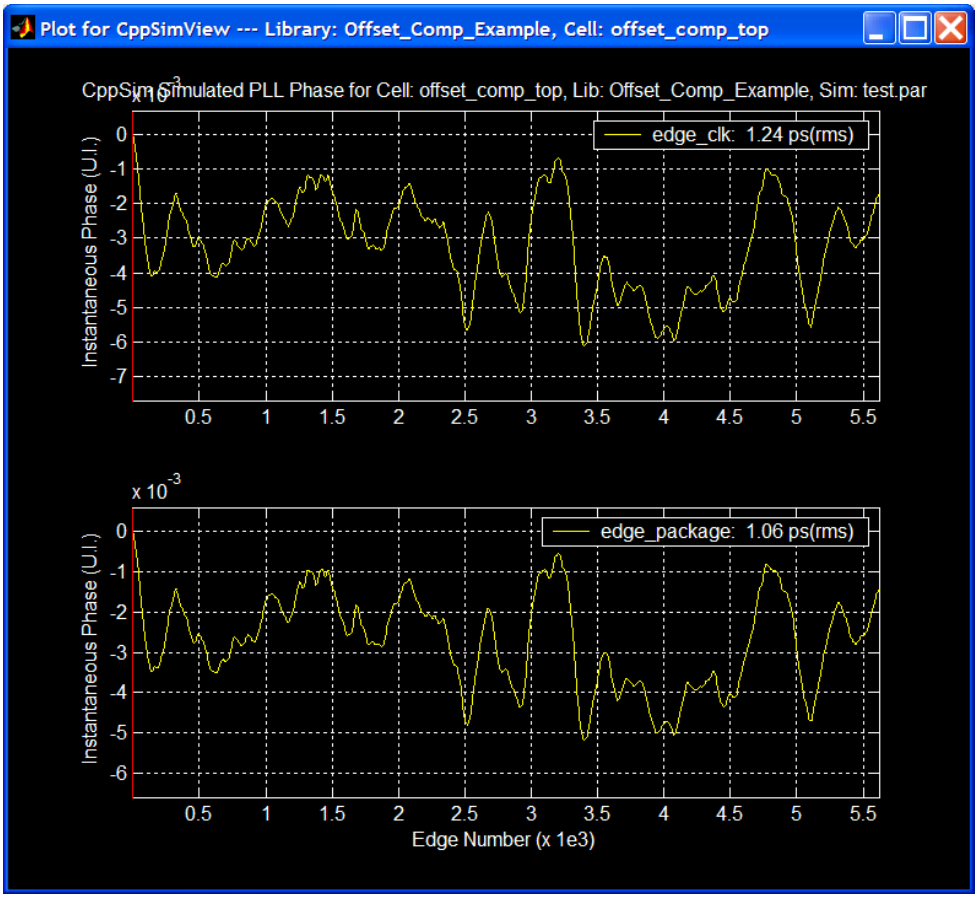
A. RMS Jitter

Now perform the following operations in CppSimView:

- Click on the **test.tr0** radio button and then choose the output file to be **test_jitter.tr0**.
- Click on the **eyesig(…)** radio button and then choose the plotting function to be **plot_pll_jitter(…)**. Set the **ref_timing_node** parameter to edge_ref (this is the interpolated reference clock output signal) and the **start_edge** to 1. Hit **Return** to enter the values into the CppSimView function list. CppSimView should appear as shown below.



- Click on the **No Nodes** radio button, and then double-click on **edge_clk** and **edge_package**. A plot of the instantaneous phase for the limit amplifier output both before and after the package model should appear, as shown below. The RMS jitter for each signal, in units of mUI (i.e. UI is unit interval, or one data period), is indicated in the legend of each signal.



- The resulting RMS jitter for the limit amplifier output signal, **out**, is 1.24pS, or 3.10mUI.
- The resulting RMS jitter for the limit amplifier output after the package model, **package**, is 1.06pS, or 2.65mUI. (Note: the number of time steps was increased from 10e4 to 50e4 to show that the phase error does settle).

Examining Nonidealities

A. Intersymbol Interference (ISI)

Let us first examine the influence of inter-symbol interference on the output jitter of the limit amplifier. For the initial design, we considered a 2.5Gb/s PRBS input data stream. The minimum amplifier bandwidth (-3dB roll-off frequency) to achieve acceptable data-dependent jitter is approximately 3.0GHz. From above, the RMS jitter is 743fS, or 1.86mUI, at the output of the package model. From page 207 of [3], the required bandwidth per stage of the limit amplifier is:

[3] Lee, Thomas H. *The Design of CMOS Radio-Frequency Integrated Circuits*, New York, NY: Cambridge University Press, 1998.

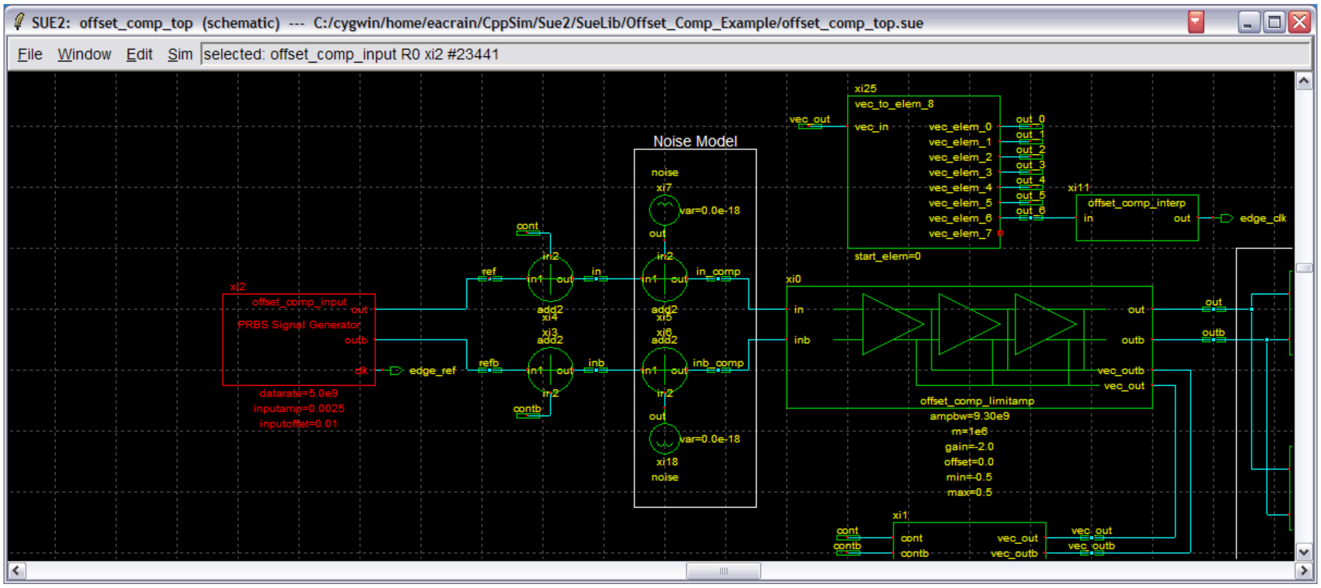
$$f_{-3dB,tot} = f_{-3dB,stage} \cdot \sqrt{2^{1/n} - 1}$$

where $f_{-3dB,tot}$ is the cumulative -3dB roll-off frequency of the entire limit amplifier and $f_{-3dB,stage}$ is the -3dB roll-off frequency of each stage. Solving for $f_{-3dB,stage}$:

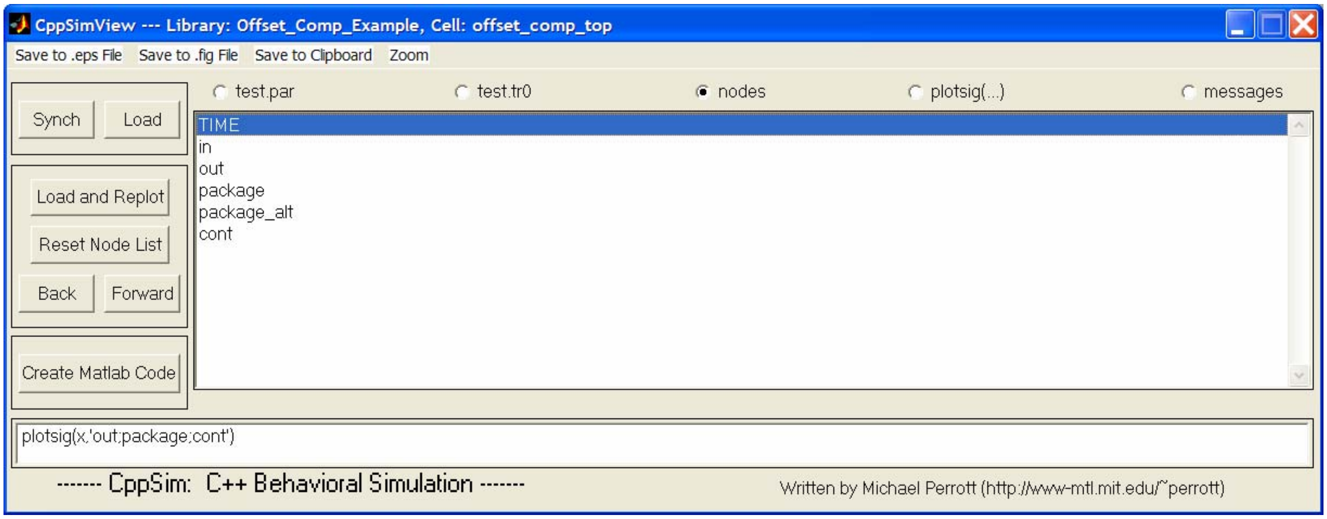
$$f_{-3dB,stage} = \frac{f_{-3dB,tot}}{\sqrt{2^{1/n} - 1}}$$

Since $f_{-3dB,tot}$ equals 3.0GHz and n equals 7, $f_{-3dB,stage}$ is approximately 9.3GHz. We will now increase the data rate to 5.0Gb/s without changing the bandwidth of the amplifier.

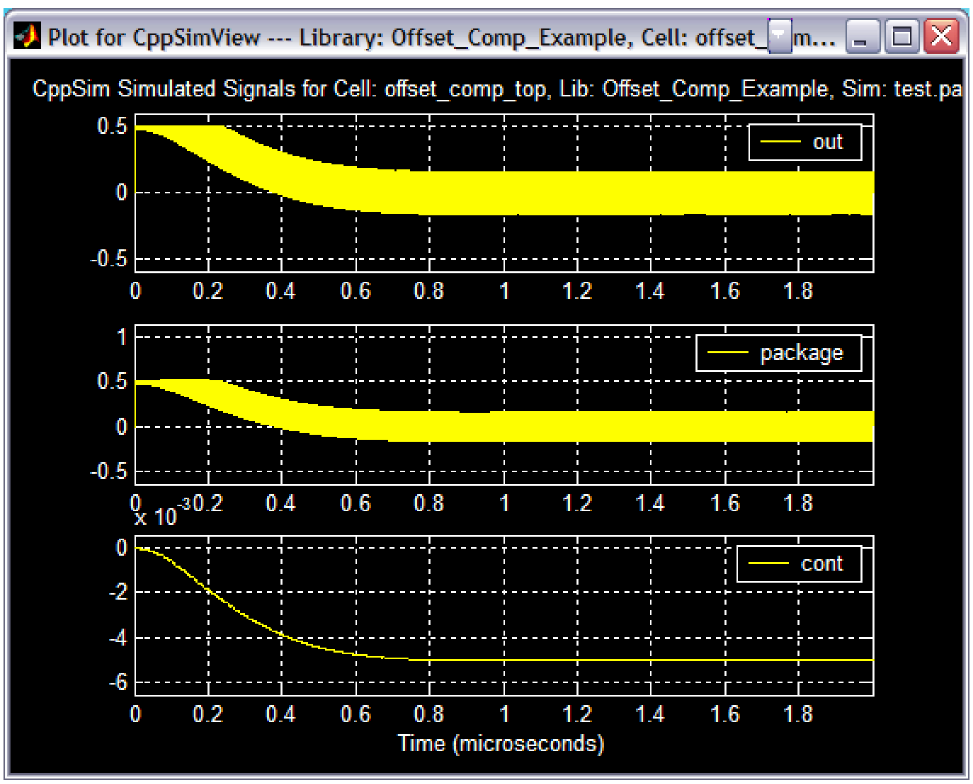
- Click on **Edit Sim File** radio button in the CppSim Run Menu window.
- Change **num_sim_steps** to 20.0e4 and **Ts** is set to 1/100e9. You may then close the Emacs window if you like (cntl-x cntl-c).
- Double click on the **offset_comp_input** symbol in Sue2 and change **datarate** to 5.0e9.
- Either type **Ctrl-S** or click on **File->Save** in Sue2 to save the changes.



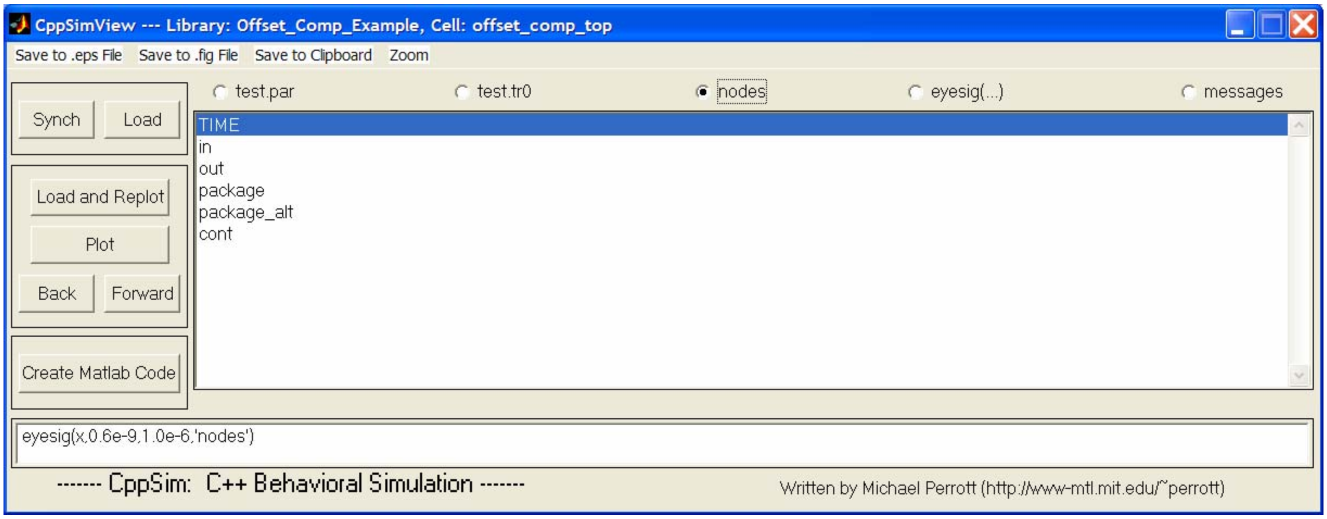
- Click on the **Compile/Run** button to run the CppSim simulation.
- Click on the **test_jitter.tr0** radio button and change the output file to **test.tr0**.
- Click on the **plot_pll_jitter(…)** radio button and change the plotting function to be **plot(…)**.
- Click on the **No Nodes** radio button and double click on **out**, **package** and **cont** as shown below.



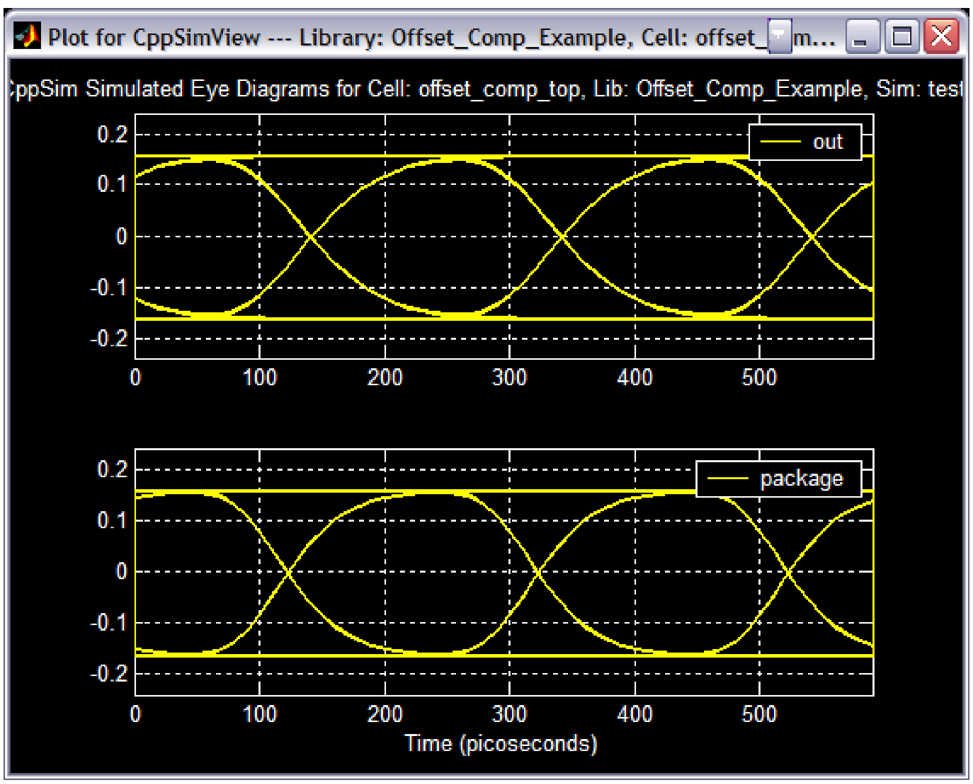
- You should see plots of the amplifier output before and after the package model and the feedback control signal as shown below:



- Next, click on the **plot(…)** radio button and change the plotting function to be **eyesig(…)**.
- Set **period** to be 0.6e-9 and **start_off** to be 1.0e-6. Hit **Return** to enter the function into the CppSimView function list and click on the **nodes** radio button to load in the simulation signals. CppSimView should now appear as shown below:



- Double-click on the **out** and **package** signals to plot their corresponding eye diagrams, which should appear as shown below.



- Comparison of the above eye diagrams to previous ones plotted with a 2.5Gb/s data rate reveals the impact of inter-symbol interference that occurs if the bandwidth is too low. You might consider setting the **ampbw** parameter of the **offset_comp_limitamp** to different values and observe the corresponding impact on the eye diagrams.

Bandwidth enhancements can be gained by using some combination of shunt and/or series peaking and neutralization – essentially by adding a zero to the transfer function of the limit amplifier base cells. Each stage of the limit amplifier is modeled as a gain with a single pole:

$$H(s) = \frac{A}{1 + \frac{s}{2\pi \cdot f_p}}$$

The bandwidth enhancement techniques change the transfer function to:

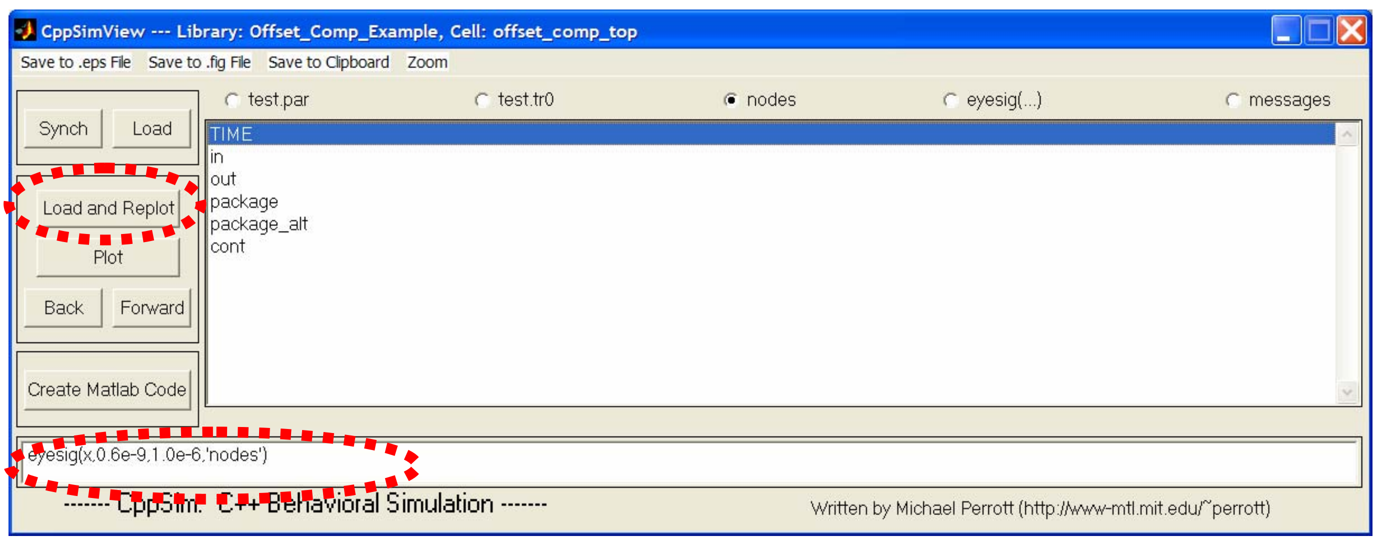
$$H(s)_{enhanced} = \frac{A \cdot (1 + \frac{s}{2\pi \cdot f_z})}{1 + \frac{s}{2\pi \cdot f_p} + \frac{s^2}{(2\pi)^2 \cdot f_p f_z}}$$

Introducing the parameter $m = f_z / f_p$, the transfer function H(s) can be rewritten as:

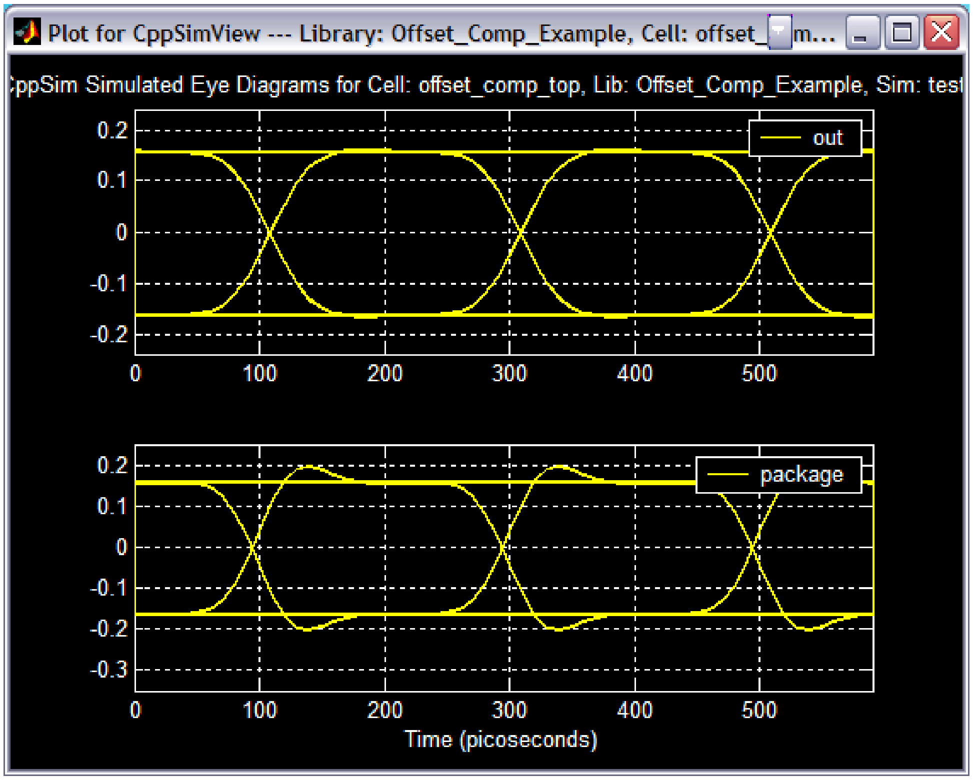
$$H(s)_{enhanced} = \frac{A \cdot (1 + \frac{s}{2\pi \cdot m \cdot f_p})}{1 + \frac{s}{2\pi \cdot f_p} + \frac{s^2}{(2\pi)^2 \cdot m \cdot f_p^2}}$$

As m goes to infinite $H(s)_{enahanced}$ reduces to the original form of $H(s)$. The maximum bandwidth (bandwidth increased by ~1.85x) is achieved for m ~ 1.41 while a maximally flat response (bandwidth increased by ~ 1.72x) is achieved for m = 2.41. Refer to [1] for more information.

- Double click on the **offset_comp_limitamp** symbol in Sue2 and change **m** to 3.10.
- Either type **Ctrl-S** or click on **File->Save** in Sue2 to save the changes.
- Click on the **Compile**/**Run** button to run the simulation.
- Plot eye diagrams of the signals, out and package, again by clicking on **Load and Replot** button.



- The eye diagrams should appear as below.

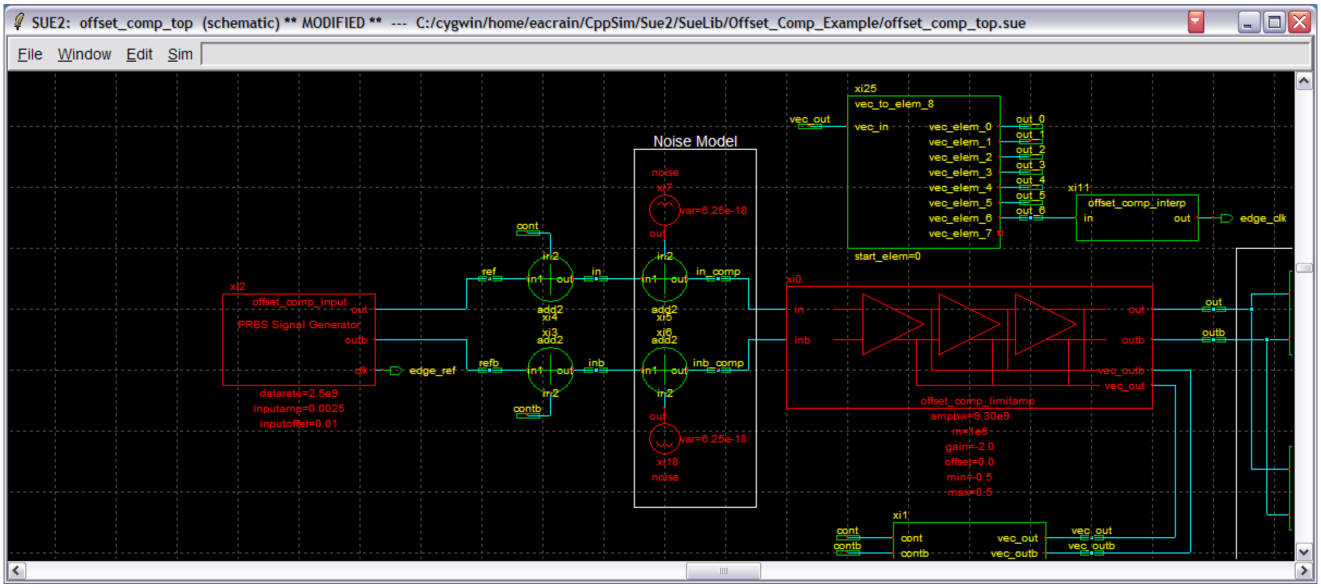


- Comparison of the above eye diagrams to the first ones plotted with a 5.0Gb/s data rate reveals that the enhancement technique has dramatically improved the inter-symbol interference because by increasing the bandwidth by 1.6x (the normalized peak frequency response for m = 3.10 is 1.0 with minimum group delay). Peaking in the package output is due to the large bondwire inductance.
- The reader is encouraged to investigate the affect of further decreasing m on the quality of the eye diagrams.

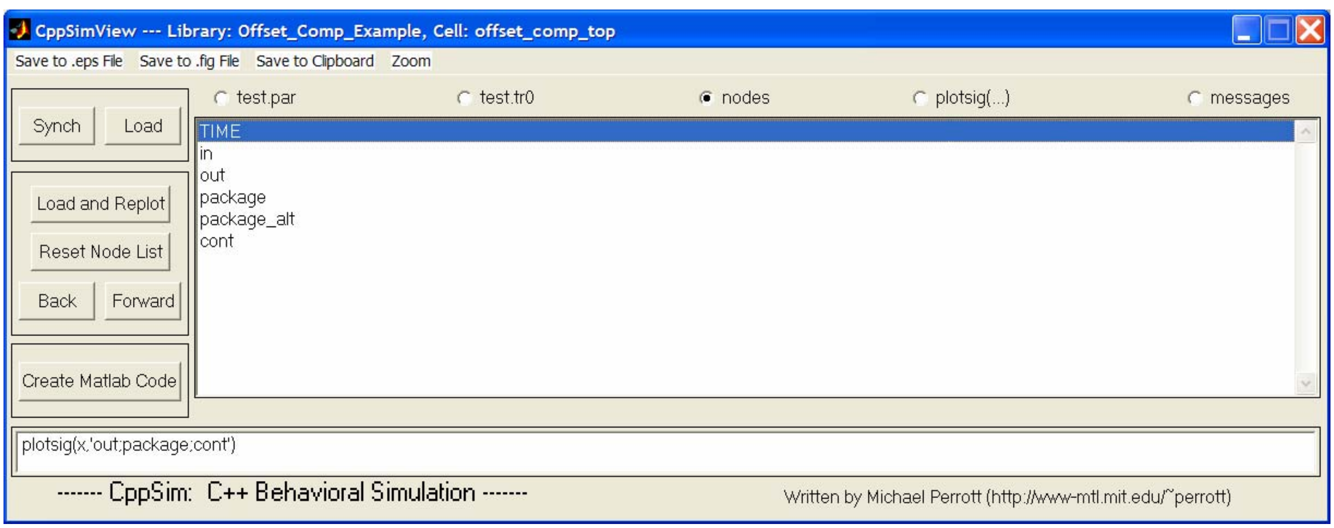
B. Limit Amplifier Noise

A significant non-ideality that the simulations have ignored to this point is noise. Let's now examine the affect of noise on the performance of the limit amplifier.

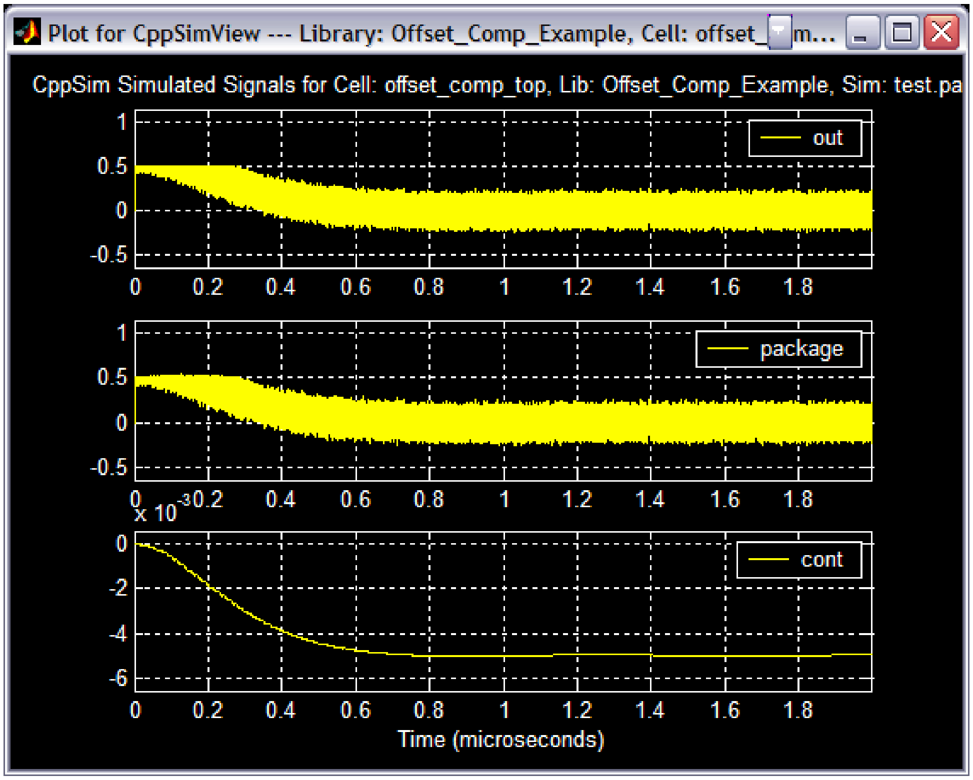
- Click on the **Edit Sim File** radio button in the CppSim Run Menu.
- Change **num_sim_steps** to 10.0e4 and **Ts** is set to 1/50e9. You may then close the Emacs window if you like (cntl-x cntl-c).
- Double click on the **offset_comp_input** symbol in Sue2 and change **datarate** to 2.5e9.
- Double click on the **offset_comp_limitamp** symbol in Sue2 and change **m** to 1.0e6.
- Double click each of the **noise** blocks in the **Noise Model** block and change **var**, the input referred noise variance, to $6.25e^{-18}V^2/Hz$. This is equivalent to a standard deviation of $2.5nV/\sqrt{Hz}$.
- Either type **Ctrl-S** or click on **File->Save** in Sue2 to save the changes.



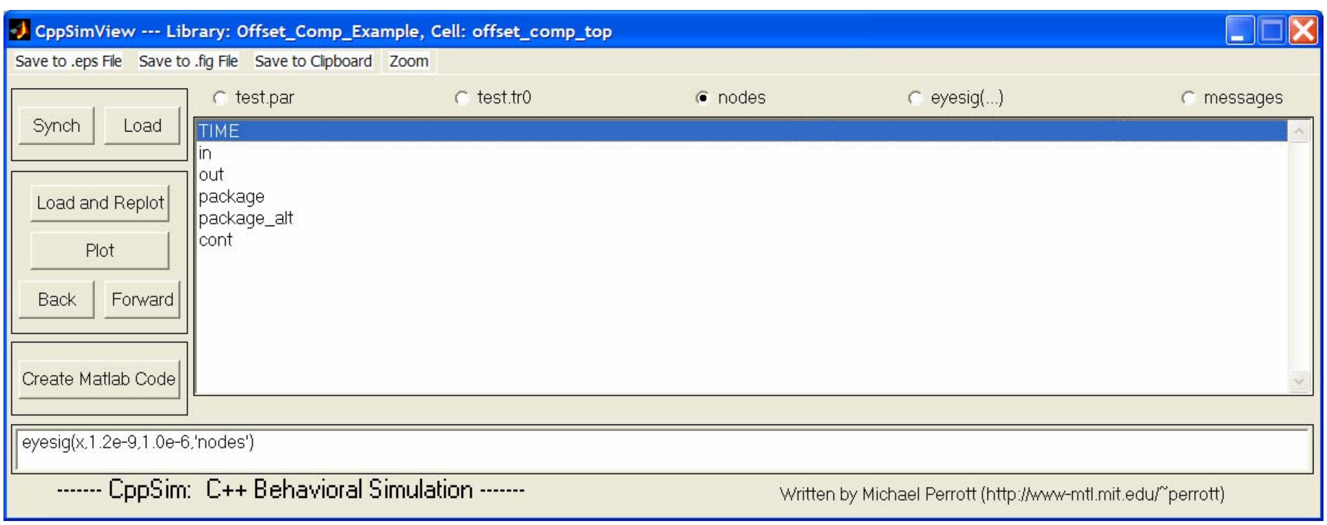
- Click on the **Compile/Run** button to run the simulation.
- Click on the **No Output File** radio button and select **test.tr0** as the output file.
- Click on the **eyesig(…)** radio button and change the plotting function to be **plotsig(…)**.
- Click on the **No Nodes** radio button and double click on **out**, **package** and **cont** as shown:



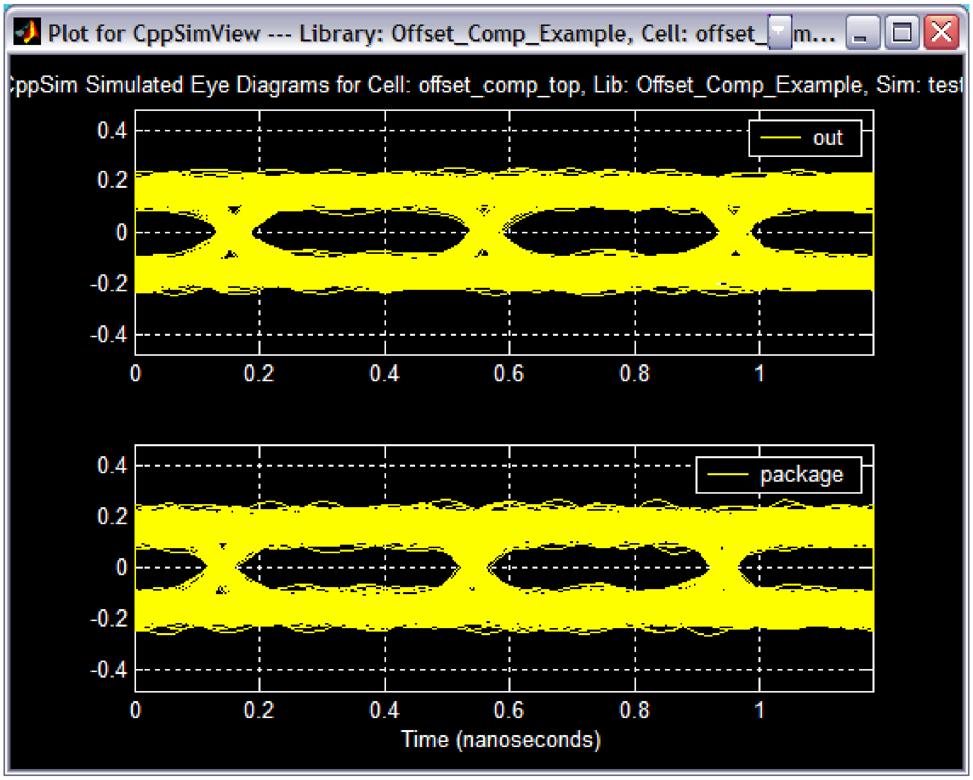
- You should see plots of the amplifier output before and after the package model and the feedback control signal as shown below. Notice that the noise does not significantly change the offset compensation settling behavior.



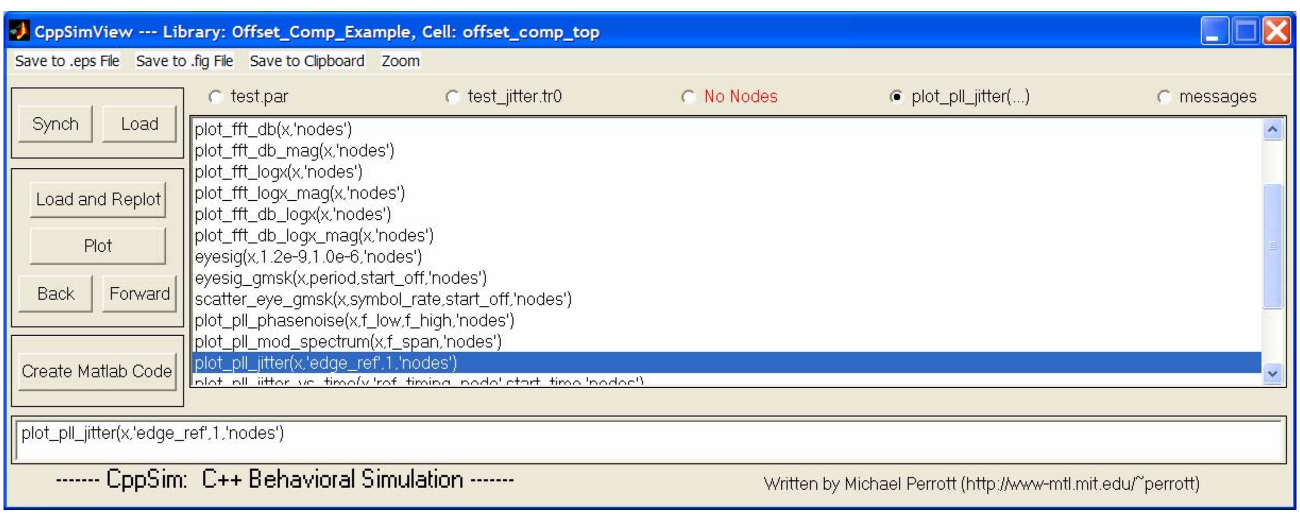
- Next, click on the **plot(..)** radio button and change the plotting fuction to **eyesig(…)**.
- Change **period** to 1.2e-9 and **start_off** to 1.0e-6 and hit **Return**.
- Click on the **nodes** radio button. The CppSimView window should appear as shown below:



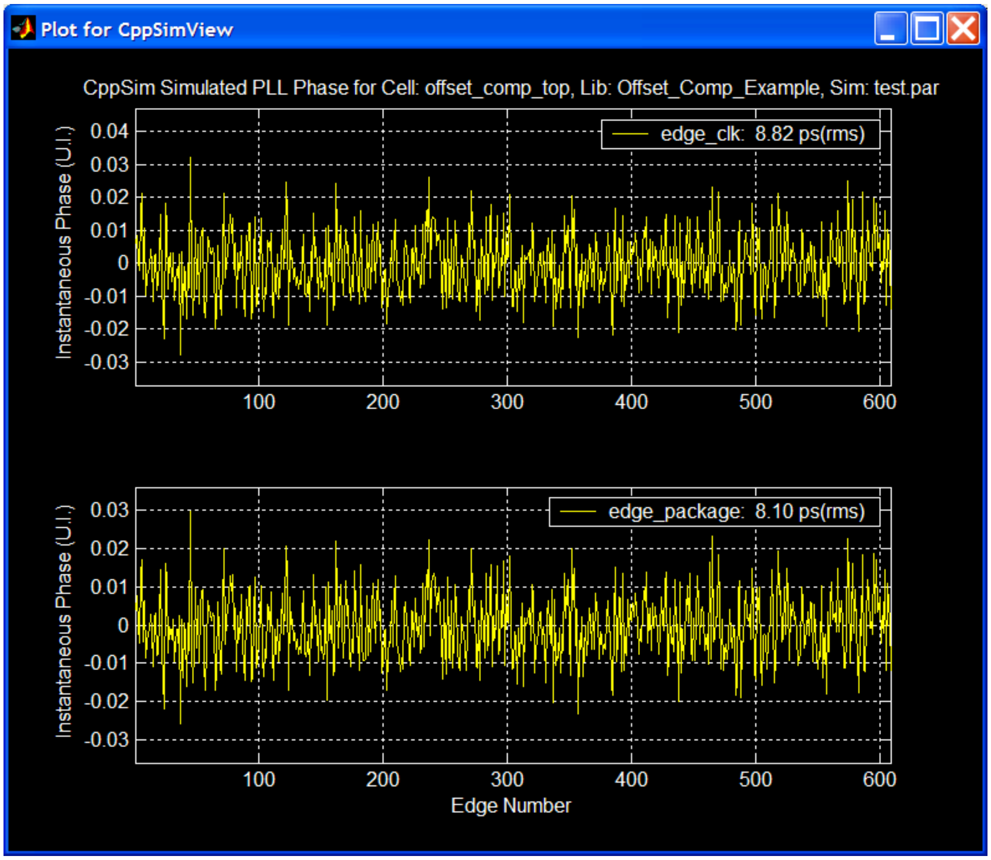
- Double click on **out** and **package** to plot eye diagrams of these signals. The eye diagrams of the limit amplifier output signals with input referred noise added before and after the package model should appear as shown below.



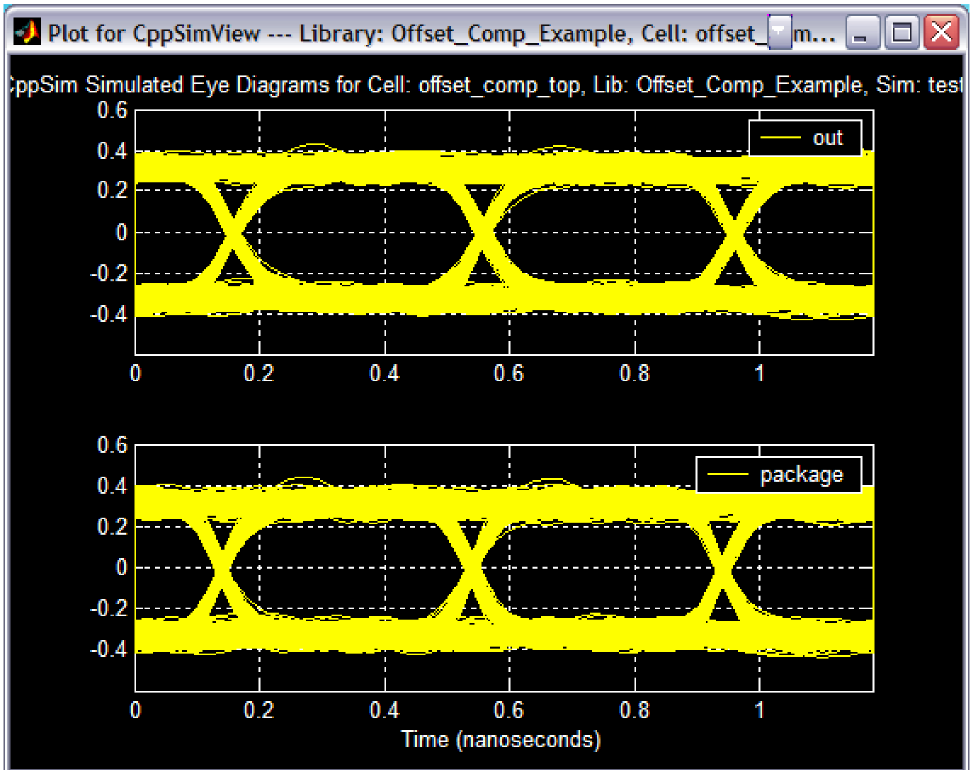
- Comparison of the above eye diagrams to the first ones plotted with a 2.5Gb/s data rate reveals that the noise severely degrades the quality of the eye diagram. To reinforce this, let's examine the RMS jitter of the two output signals and compare to the previous result without noise.
- Click on the **test.tr0** radio button and then choose the output file to be **test_jitter.tr0**.
- Click on the **eyesig(…)** radio button and choose the plotting function to be **plot_pll_jitter(…)**. Set **ref_timing_node** to edge_ref and **start_edge** to 1. Hit **Return** to enter the values into the CppSimView function list. CppSimView should appear as shown below.

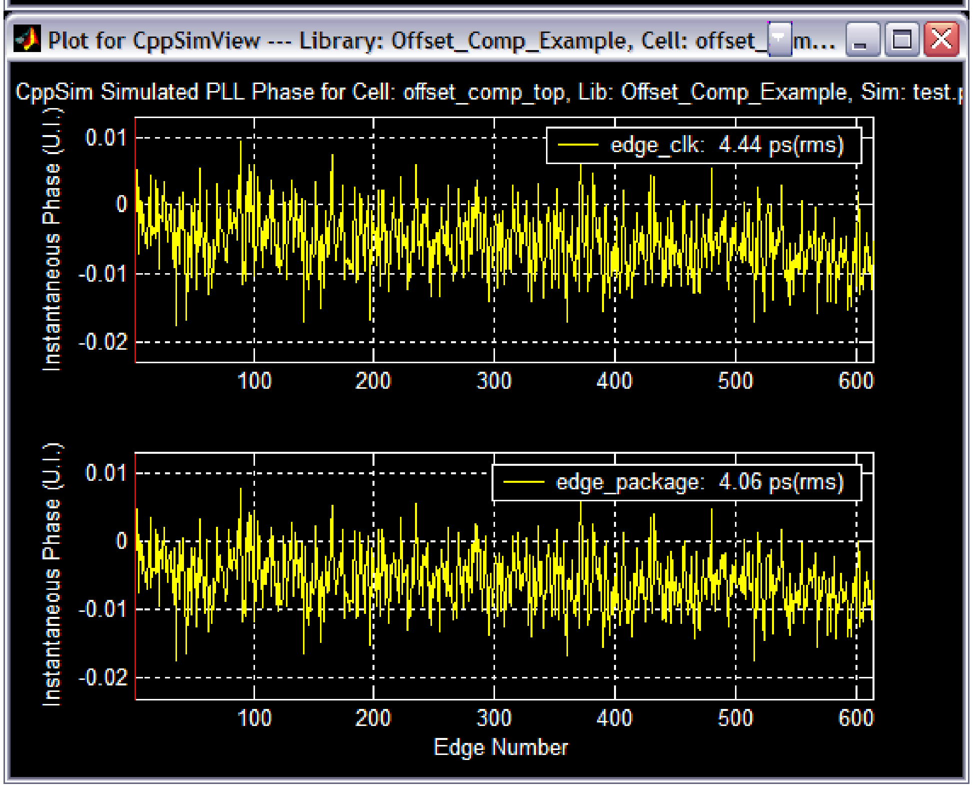


- Click on the **No Nodes** radio button, and then double-click on **edge_clk** and **edge_package**. A plot of the instantaneous phase for the limit amplifier output both before and after the package model should appear, as shown below. The RMS jitter for each signal is indicated in the legend of each signal.



- The RMS jitter of the limit amplifier before and after the package model is 8.82ps and 8.10ps, respectively, compared to 1.24ps and 1.06ps when noise is neglected. Noise severely limits the input sensitivity of the amplifier.
- We can show that the input sensitivity of the amplifier is roughly 5mV if we are to meet the OC48 jitter specifications (<4ps(rms) at 2.5Gb/s) in the eye diagrams and jitter plots below.


Plot for CppSimView --- Library: Offset_Comp_Example, Cell: offset_ m...
ppSim Simulated Eye Diagrams for Cell: offset_comp_top, Lib: Offset_Comp_Example, Sim: test
out
package
Time (nanoseconds)


Plot for CppSimView --- Library: Offset_Comp_Example, Cell: offset_ m...
CppSim Simulated PLL Phase for Cell: offset_comp_top, Lib: Offset_Comp_Example, Sim: test.
edge_clk: 4.44 ps(rms)
edge_package: 4.06 ps(rms)
Instantaneous Phase (U.I.)
Edge Number

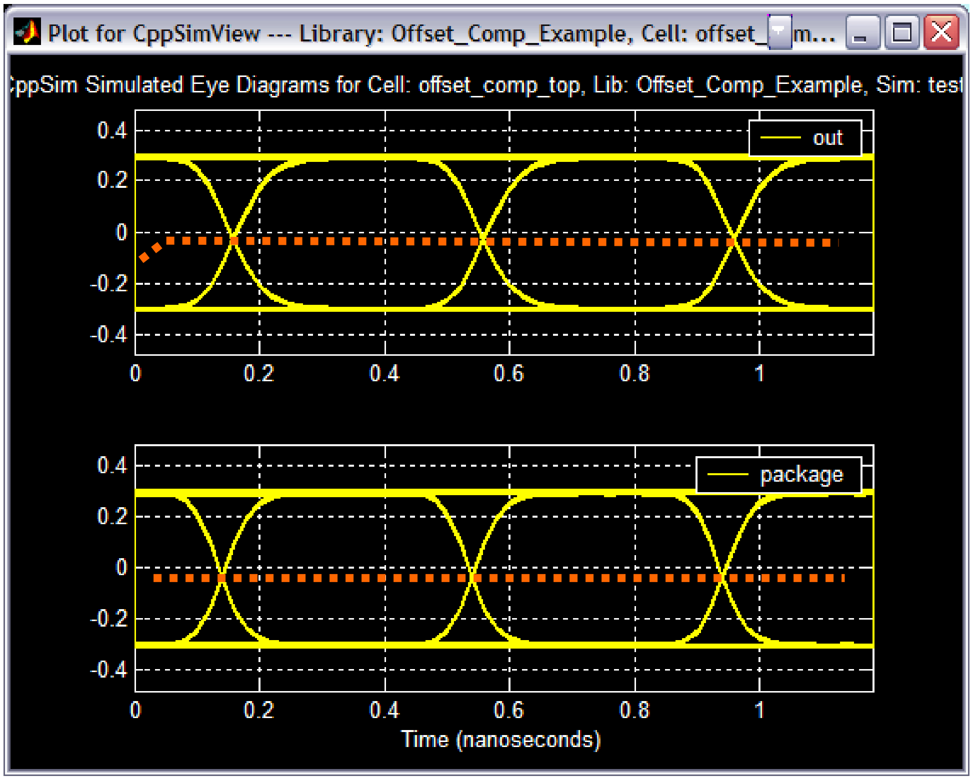
C. Amplifier Non-Linearity

Amplifier non-linearity can influence the offset compensation. Ultimately, non-linearities in the amplifier transfer function can translate to residual output-referred offset. Therefore, we must consider it in this model. To explore this affect:

- Within Sue2, left-click on the **offset_comp_limitamp** block, and press **e** to push into the lower level. You should see the seven-stage limiting amplifier now. Double-click on the **offset_comp_amp** block and then click on the **Edit CppSim Code** button. An Emacs window including the CppSim code for this block should pop up (shown below).
- The Amp class definitions **amp1** and **amp2** have polynomial expressions that define the DC transfer function of the limit amplifier base cell. If the coefficients of the higher order terms are zero then the transfer function is ideally linear. However, if the coefficients are non-zero then the transfer function will have some degree of non-linearity. The coefficients for the higher order terms can be determined by measuring the transfer function of the actual circuit in HSPICE and fitting a curve with MATLAB. Change the variables A1/B1 and A2/B2 in classes amp1 and amp2 from 0 to 0.6 and -3.3 as shown below.

```
module: offset_comp_amp
parameters: double bw double m double gain double offset double min double max
inputs: double in double inb
outputs: double out double outb
classes:
 Amp amp1("offset/2-gain*a+A1*a^2+A2*a^3","offset,gain,Min,Max,A1,A2",
offset,gain,min,max,0.6,-3.3);
 Amp amp2("-offset/2-gain*b+B1*b^2+B2*b^3","offset,gain,Min,Max,B1,B2",
offset,gain,min,max,0.6,-3.3);
 Filter filt1("1 + 1/(2*pi*fz)*s","1 + 1/(2*pi*fp)*s + 1/(fp*fz*(2*pi)^2)*s^2"
,"fp,fz,Ts",bw,m*bw,Ts);
 Filter filt2("1 + 1/(2*pi*fz)*s","1 + 1/(2*pi*fp)*s + 1/(fp*fz*(2*pi)^2)*s^2"
,"fp,fz,Ts",bw,m*bw,Ts);
 Filter filt3("1","1 + 1/(2*pi*fp)*s","fp,Ts",10*bw,Ts);
 Filter filt4("1","1 + 1/(2*pi*fp)*s","fp,Ts",10*bw,Ts);
static_variables:
init:
code:
 amp1.inp(in);
 amp2.inp(inb);
 filt1.inp(amp1.out);
 filt2.inp(amp2.out);
 filt3.inp(filt1.out);
 filt4.inp(filt2.out);
 out = filt3.out;
 outb = filt4.out;
```

- Change the input-referred offset (**inputoffset**) and noise (**var**) to zero, increase the input amplitude (**inputamp**) to 5.0mV and plot the eye diagrams for **out** and **package**.



- The eye diagrams demonstrate that the non-linearity can produce a non-zero residual output referred offset voltage even when the offset of the limit amplifier is zero.

E. Input Amplitude

The offset compensation uses multiple control loops to account for large input amplitudes and large input-referred offsets. Compensation will occur from the last unsaturated output and all control loops have the same gain and bandwidth so that the loop dynamics are the same regardless of the input conditions. The reader can confirm that the loop dynamics, specifically the compensation settling time, are constant for all input amplitudes and input offset amplitudes by using the alter command in the Sim file. After adding the variable **inputamp** to the **offset_comp_input** block in the top level, the Sim file syntax to sweep the input amplitude from 5mV to 25mV using a 5mV step is:

```
alter:
inputamp = 5e-3:5e-3:25e-3
```

This can also be done to explore the impact of changing the input-referred offset voltage.

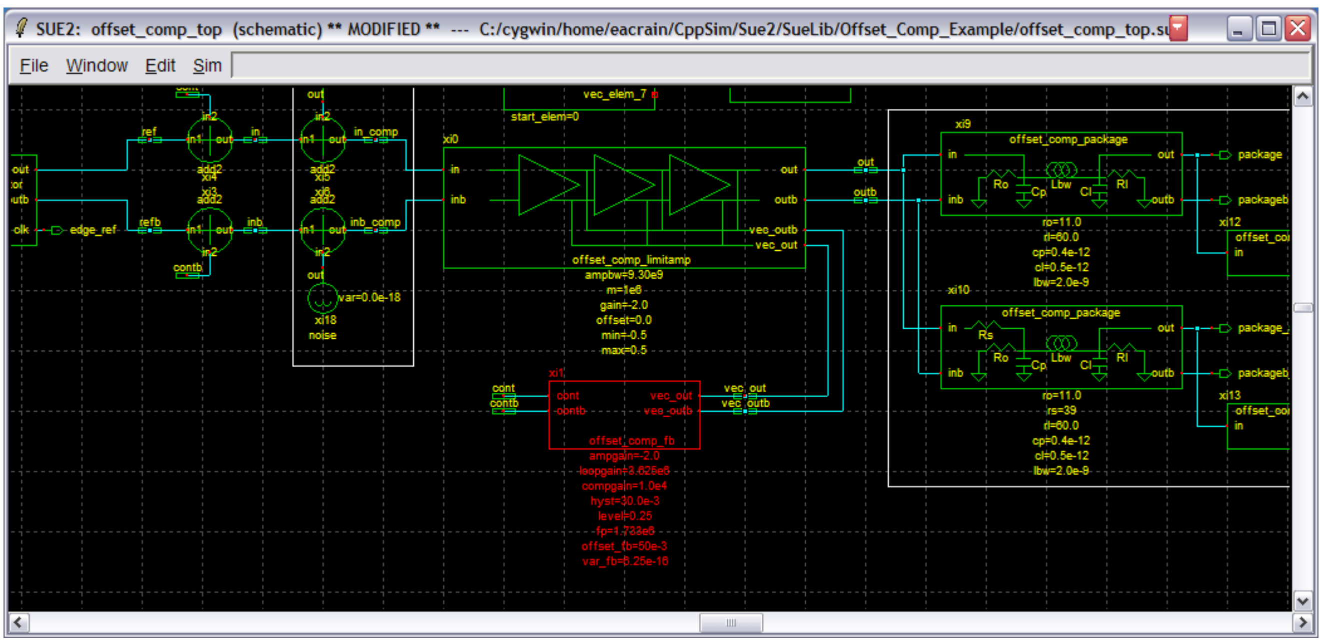
F. Feedback Noise and Offset

There is a significant difference between how the noise and offset introduced by the forward path and that introduced by the feedback path affect the input sensitivity of the amplifier. The noise of each amplifier stage in the forward path is referred to the amplifier input by dividing it by the gain of preceding stages. Therefore, the noise performance of the first several amplifier stages dominates the noise performance. Similarly, the noise introduced by the feedback path is referred to the amplifier input through the gain of the entire main amplifier and is therefore greatly attenuated. The noise performance of the circuitry in the feedback path has virtually no impact on the amplifier input sensitivity.

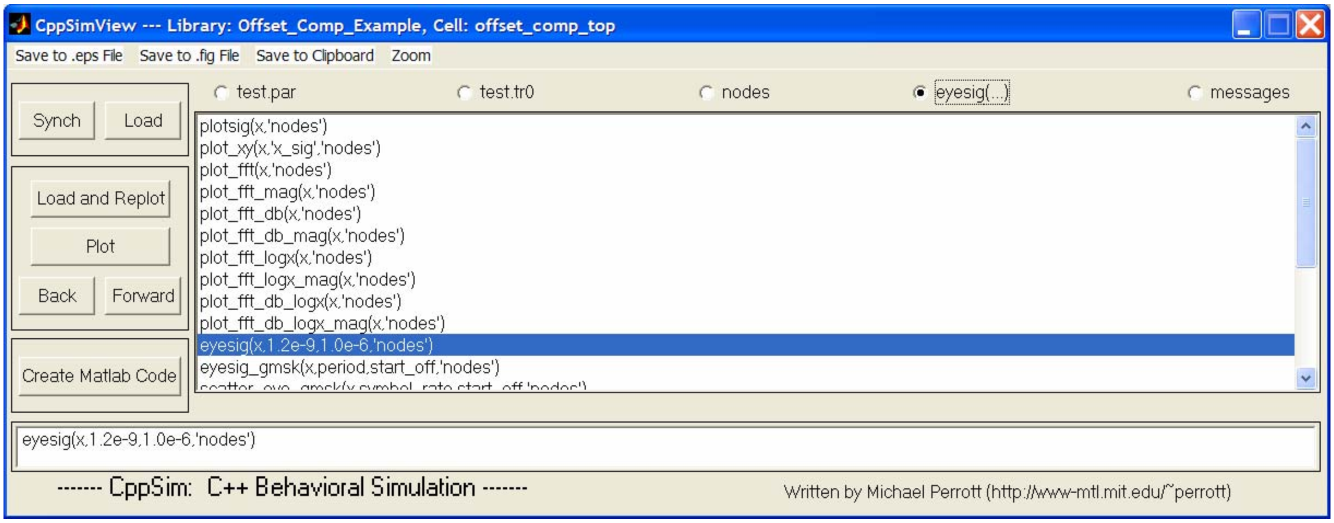
The impact of the offsets introduced by the forward path and feedback path on the amplifier is different. By design, offset introduced by the forward path will be attenuated by the DC loop gain (ideally infinite). However, the offset introduced by the feedback path will be directly referred to the output of the main amplifier. This can be easily seen by referring the offset introduced by the feedback path to the input of the feedback loop and recognizing that this corresponds to the output of the main amplifier.

We can verify these observations with the CppSim model:

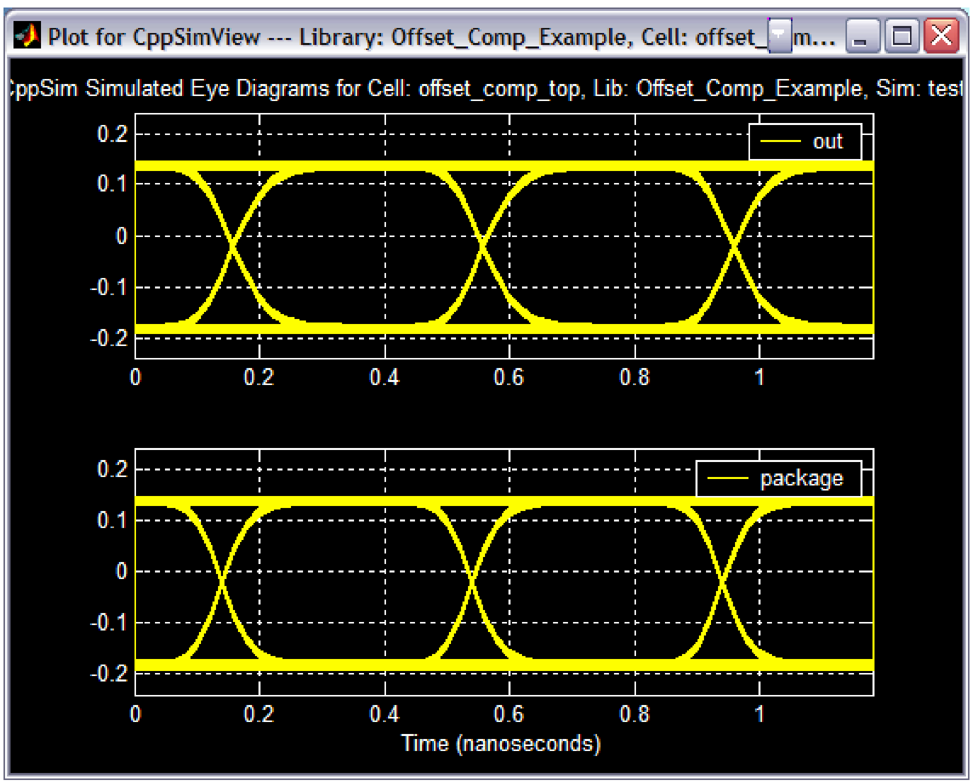
- Double click on the **offset_comp_fb** symbol in Sue2 and change **offset_fb**, the input referred offset of the feedback path, to 50e-3 (50mV – a large value) and **var_fb**, the input referred noise variance of the feedback path, to $6.25e^{-16}V^2/Hz$ (100x the input-referred noise variance of the forward amplifier path). This is equivalent to a standard deviation of $25nV/\sqrt{Hz}$ (also a very large value).
- Either type **Ctrl-S** or click on **File->Save** in Sue2 to save the changes.



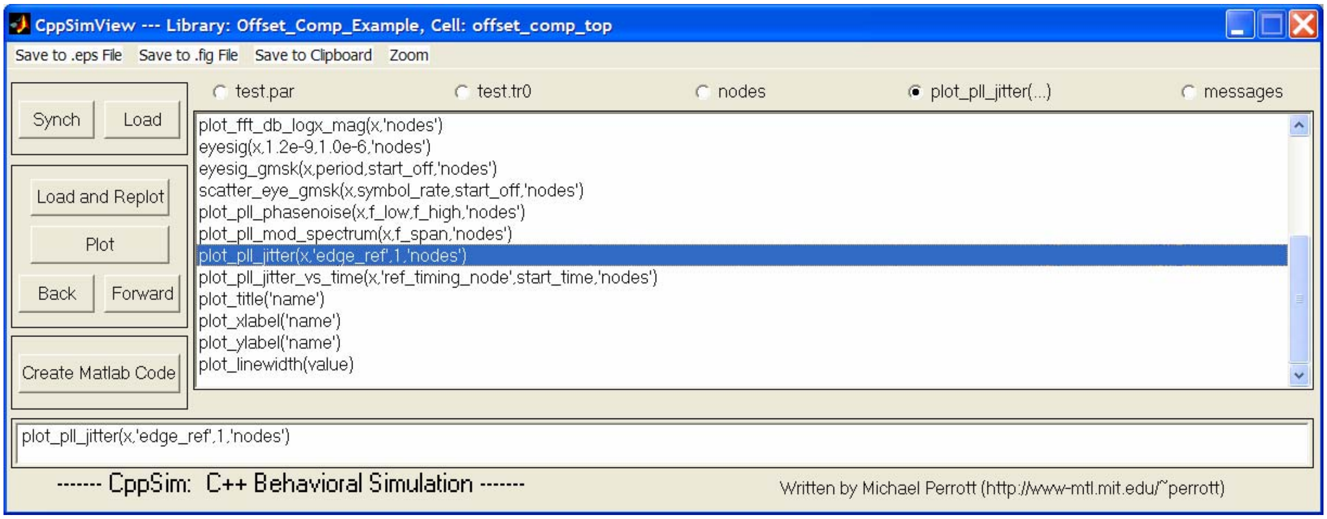
- Click on the **Compile/Run CppSim** button to run the simulation.
- Click on the **plotsig(…)** radio button in CppSimView and then select the **eyesig(…)** function. Set **period** to be 1.200e-12 and **start_off** to be 1.0e-6. Hit **Return** to enter the function into the CppSimView function list. CppSimView should now appear as shown below.



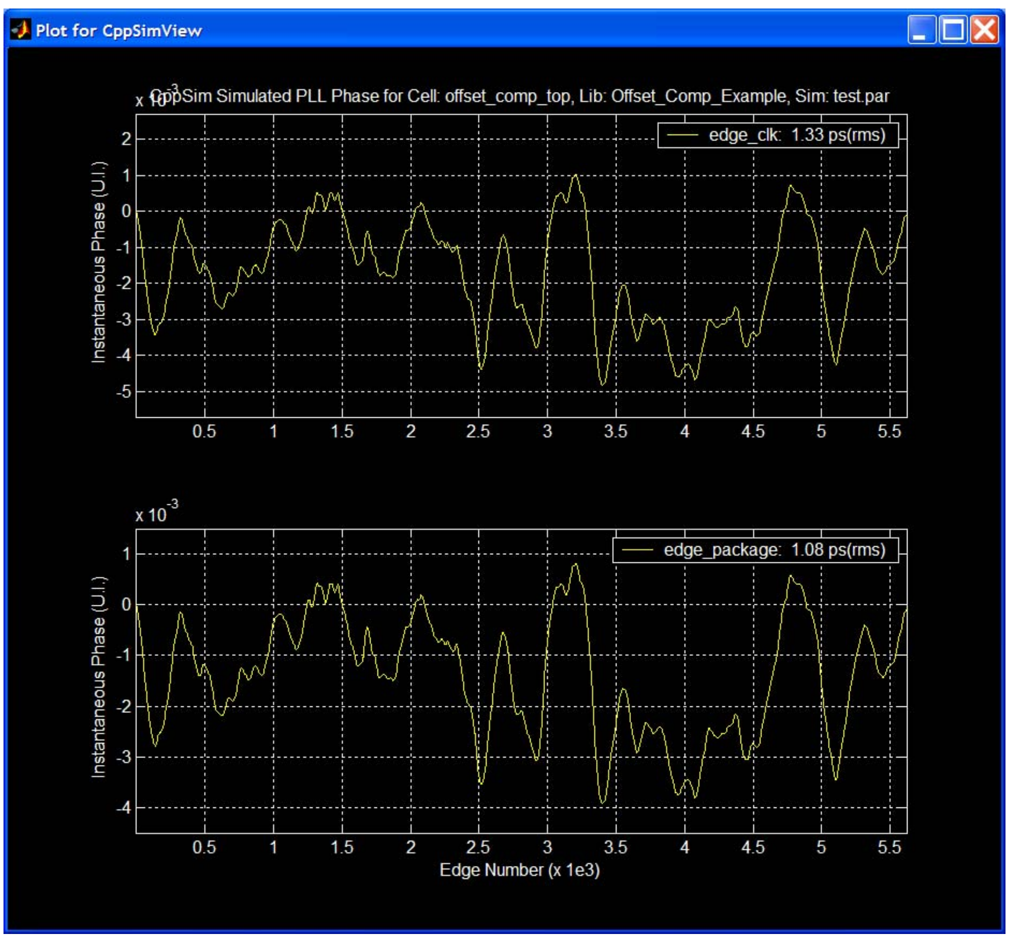
- Clock on the **nodes** radio button, and then double-click on signal **out** and then on signal **package**. The eye diagrams of the ideal limit amplifier output signals before and after the package model should appear as shown below.



- The output-referred offset voltage is approximately 25mV (Note that the nonlinear coefficients on page 24 are set back to zero in this simulation). The offset of the feedback path has been attenuated somewhat.
- Click on the **test.tr0** radio button and then choose the output file to be **test_jitter.tr0**.
- Click on the **eyesig(…)** radio button and then choose the plotting function to be **plot_pll_jitter(…)**. Set the **ref_timing_node** parameter to edge_ref and the **start_edge** to 1. Hit **Return** to enter the values into the CppSimView function list. CppSimView should appear as shown below.



- Click on the **No Nodes** radio button, and then double-click on **edge_clk** and **edge_package**. A plot of the instantaneous phase for the limit amplifier output both before and after the package model should appear, as shown below. The RMS jitter for each signal, in units of mUI (i.e. UI is unit interval, or one data period), is indicated in the legend of each signal.



- The resulting RMS jitter for the limit amplifier output signals, **out** and **package**, increased slightly from 1.24ps and 1.06ps, with no noise, to 1.33ps and 1.08ps with $6.25e^{-16}V^2/Hz$ noise variance in the feedback path (Note that input amplitude is set back to 2.5mv in this simulation). This confirms that noise introduced in the feedback path has significantly less influence than noise introduced in the feed-forward path. (Note: the number of time steps was increased from 1e4 to 5e4 to show that the phase error does settle).

Design Overview of the Offset Compensation Loop

Let us now focus on how the parameters were set for the offset compensation loop used within both the limit amplifier and the feedback path.

Key parameters of the limit amplifier and feedback path include:

- Data rate: 2.5Gb/s
- Limit amplifier bandwidth: 3.0GHz (based on the specified data rate)
- Limit amplifier gain: 42dB
- Compensation settling time: < 1μS (loop bandwidth of ~1MHz)
- Peak detector gain: 0dB
- Simulation time step: Ts=1/50e9 (i.e., the sample rate is set to at least 10 times that of the highest frequency, which is the PRBS input).

A. Modeling

To model the control loop we need to make some simplifying assumptions. First, to eliminate the difficulty of analyzing multiple control loops, we only consider the case when there is one active control loop. In the end, we can extend the analysis to the more general case when there are multiple control loops and test that this assumption is valid in simulation. Further, we can assume that all blocks in the system are linear about a given operating point and make use of LTI modeling techniques. We will now develop models for each of the system blocks.

Each of the amplifiers in the forward amplifier path can be modeled by a DC gain and a single pole, representing the bandwidth of the amplifier. Therefore, the linear model for each amplifier is:

$$H(s) = \frac{A_V}{(1 + s/p_1)}$$

where A_V is the gain and p_1 is the pole at the 3dB frequency. If we consider a cascade of n amplifiers, the aggregate transfer function becomes:

$$H(s) = \left(\frac{A_V}{(1 + s/p_1)} \right)^n$$

Additionally, the peak detector can be similarly modeled by its DC gain, K_1, and a single pole, p_2, and has the same form as the limit amplifier characteristic equation. The final form of the peak detector model is:

$$H(s) = \frac{K_1}{(1 + s/p_2)}$$

The integrator can be modeled as an ideal integrator:

$$H(s) = \frac{K_2}{s}$$

where K_2 is the gain. Putting all of the pieces together, the complete model for the forward amplifier path and the offset compensation is shown in Figure 4 with the noise sources added.

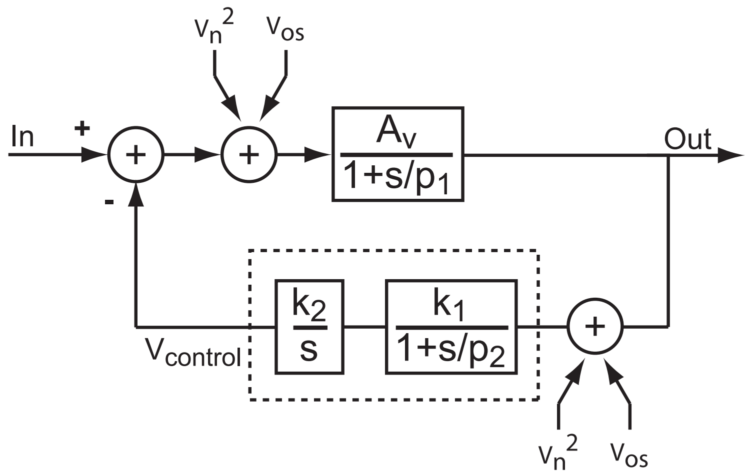


Figure 4: Linearized model of the limit amplifier and offset compensation with noise sources

B. Determining System Parameters with PLL Design Assistant

For the following discussion, we assume that $p_1 >> p_2$. This is valid in this system since p_1 corresponds to the bandwidth of the limit amplifier (3.0GHz), and p_2 corresponds to the bandwidth of the peak detector (~1-5MHz for 1μs compensation time). In a similar fashion to the linear model for a PLL, where the state variable is phase and not the data signal itself, the variable of interest in this system is the offset voltage. To characterize the system response, we define the open loop response to be:

$$A(s) = \left(\frac{A_V}{(1+s/p_1)}\right)\cdot\left(\frac{K_1}{(1+s/p_2)}\right)\cdot\left(\frac{K_2}{s}\right)$$

The PLL Design Assistant is a useful tool that was developed to aid the design of PLL systems and can be downloaded at http://www-mtl.mit.edu/researchgroups/perrottgroup/tools.html. However, with a little imagination this tool can be used to model nearly any linear system. We can think of the system in Figure 4 as a second order, type I PLL with the forward amplifier path corresponding to a high frequency parasitic pole. As stated before, the open-loop pole, p_1, is set to 3.0GHz and we will assume that the gain of the peak detector, K_1, is unity. Then, using the PLL Design Assistant we can specify a desired closed-loop bandwidth, based on the desired compensation settling time and step-response shape, to achieve the optimal settling time.

For example, Figure 5 illustrates the PLL Design Assistant GUI designing the system loop with a Bessel shape and a bandwidth of 1.0MHz. The resulting gain coefficient, K, corresponds to the product $A_V K_2$. The open-loop pole frequency, f_p, corresponds to the pole of the peak detector, p_2. The calculated step-response of the closed-loop system, shown in Figure 6, indicates that the total settling time for the offset compensation is roughly 1μs and that the system is stable.

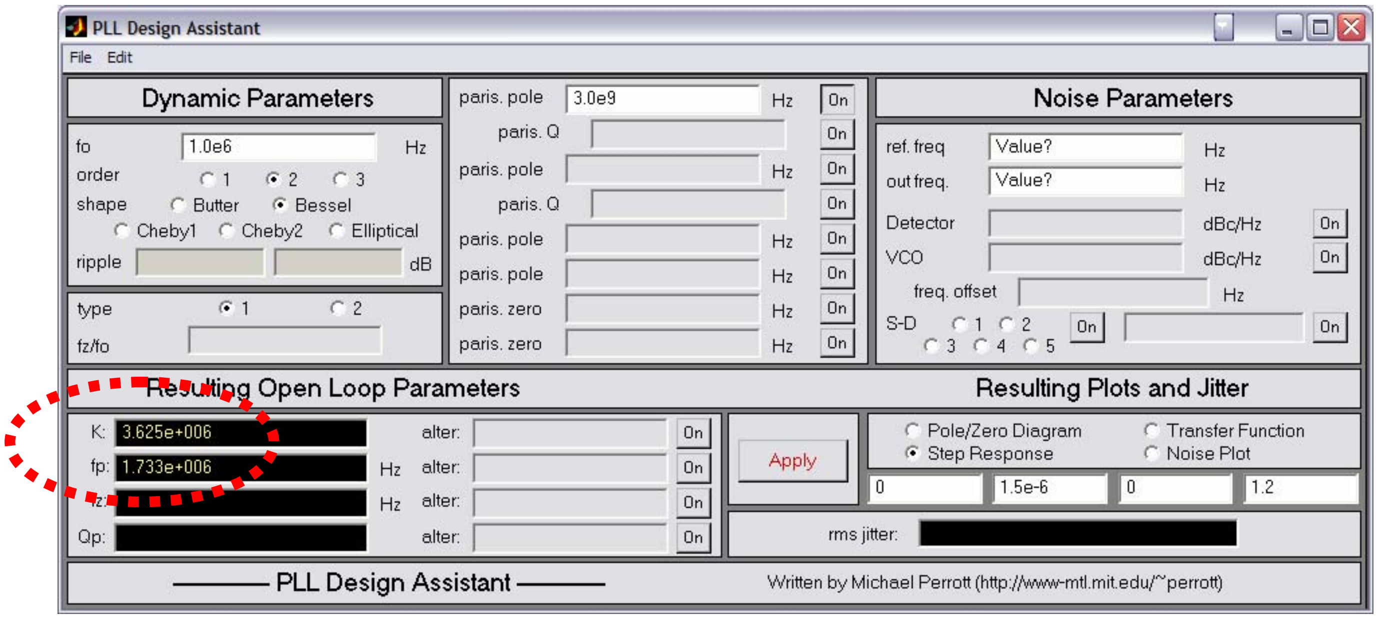


Figure 5: PLL Design Assistant GUI configured for second order, type I linear system

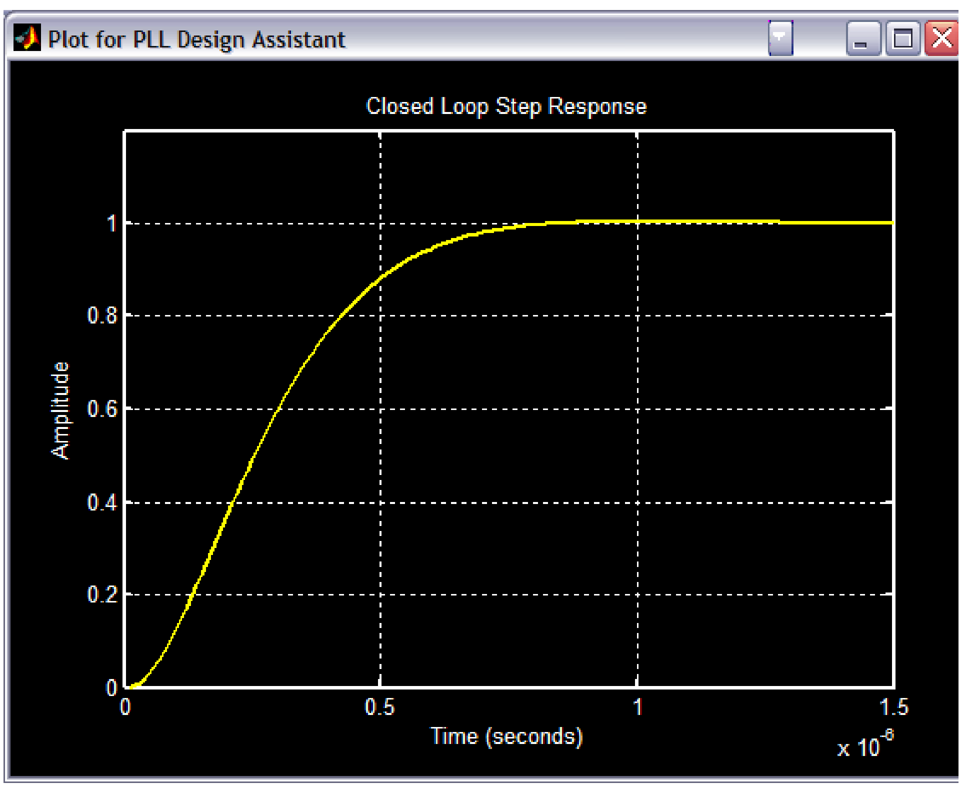


Figure 6: Step response calculated with PLL Design Assistant

Comparison of Simulated and Measured Results

To verify the validity of the proposed behavioral model, we will compare simulated results to measured results. A 7-stage resistor loaded limit amplifier utilizing the proposed offset compensation method was fabricated in National Semiconductor's 0.18 μm CMOS process. The final die size is 1 mm^2 and the total active area is 0.5 mm^2. A die micrograph is shown in Figure 7. The die was wired bonded in a standard ceramic package, which was then soldered to a 5-layer FR4 PC board containing low noise supply voltages and digital interface circuitry. The prototype was tested at data rates up to 3.125 Gb/s with a 2^{31}-1 PRBS input pattern that had input amplitudes from 2.5-50 mVpp. The PRBS input was generated with an HP 70843B 12.5GB/s error performance analyzer driven by an HP 70340A signal generator. The output jitter is measured from eye diagrams of the limit amplifier output using an Agilent 81600A 50GHz oscilloscope. The offset compensation step-response is measured by plotting the feedback control voltage with an Agilent 54832D mixed-signal oscilloscope triggered from the serial I/O input control bits. The final chip dissipates 338 mW from a 1.8 V supply, including the output buffer. The limit amplifier and offset compensation dissipate 113 mW. The total differential gain was measured to be 42 dB.

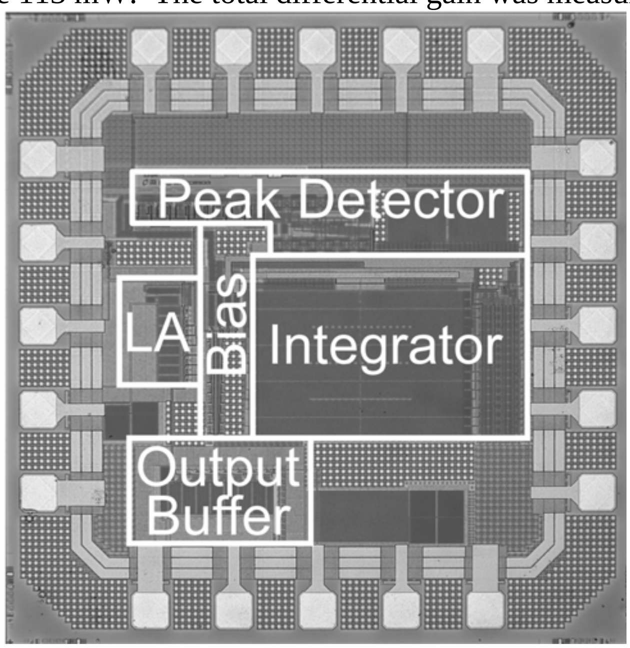
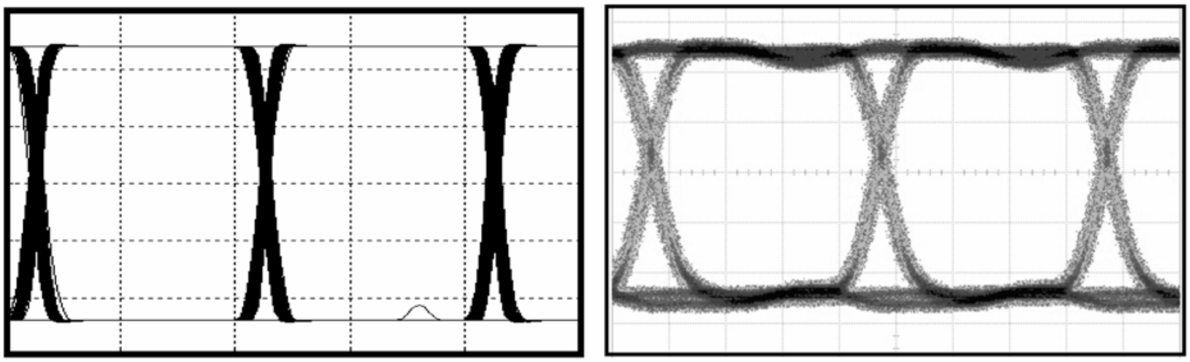


Figure 7: Micrograph of final chip, in National Semiconductor's 0.18μm CMOS process, highlighting the major system blocks

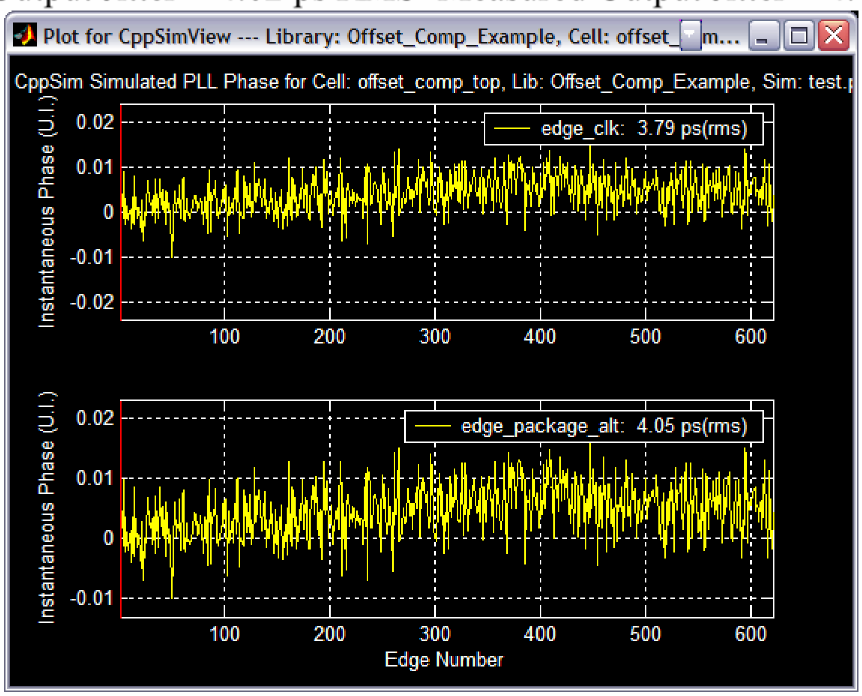
Simulated and measured eye diagrams with a 2.5 mVpp input amplitude at 2.5 Gb/s are shown in Figure 8. The measured RMS jitter is 4.30 ps and the simulated RMS jitter is 4.05 ps at the output of the package (package_alt). The predicted input sensitivity is 2.1 mVpp and the simulated input sensitivity is 2.5 mVpp. The jitter performance, plots shown below, compares favorably with the results predicted in simulation. The differences in the eye diagram shapes at the upper and lower voltage levels are most likely due to transmission-line effects not captured in the behavioral model.



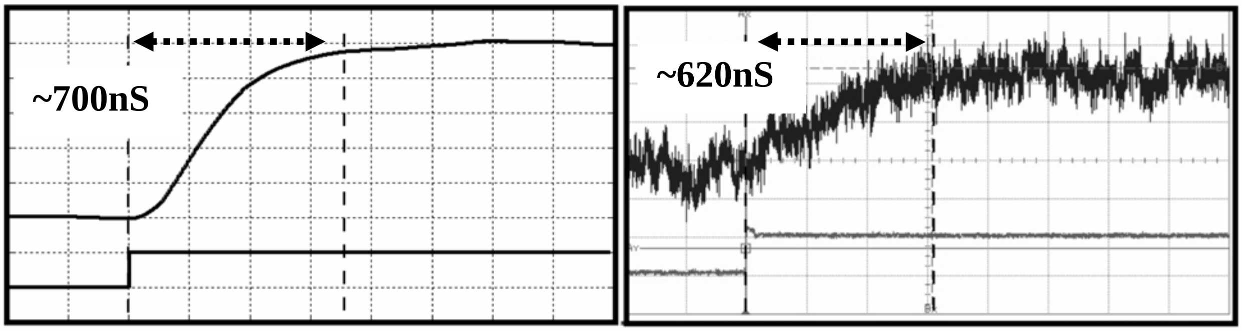
Vertical Scale: 25mV/Division
Horizontal Scale: 100nS/Division

Vertical Scale: 50mV/Division
Horizontal Scale: 100nS/Division

Simulated Output Jitter = 4.02 ps RMS Measured Output Jitter = 4.02 ps RMS



The settling behavior of the control loop can be seen by plotting the feedback control voltage. The simulated (left) and measured (right) control voltage, with the loop bandwidth set to 1MHz (1µS settling time goal). The trigger is plotted at the bottom of each plot (at 400nS) and the control signal is plotted at the top. Besides the noise in the measured plot, the simulated settling response agrees well with the measured result.



Vertical Scale: 1.0mV/Division
Horizontal Scale: 200nS/Division

Simulated Settling Time ~ 700nS

Vertical Scale: 2.5mV/Division
Horizontal Scale: 200nS/Division

Measured Settling Time = 620nS

Conclusion

In this document we have covered the basic issues related to behavioral simulation of a new offset compensation scheme for high gain amplifiers using CppSim. In particular, we have introduced the reader to the tasks of running CppSim simulations, plotting the output signals in the time domain, plotting eye diagrams, plotting instantaneous and average phase error at the output (jitter) and considered the impact of non-indealities such as ISI, noise and offset. Finally, we covered the basic modeling calculations for the system.