Mixed-Signal Circuit Simulation Guide using Cadence Virtuoso IC6.16

ECE 546 - Advanced Signal Integrity

Contents

1 Introduction

Mixed-Signal circuit design especially for high-speed applications is very intricate and requires validation of certain figures-of-merits (FOMs). This tutorial provides a detailed guide to analysis and simulation of mixed-signal circuits like voltage-controlled oscillators (VCOs) used in clocking circuits for high-speed link applications. A VCO model in Verilog-A is presented and a step-by-step guide to transient reponse and jitter calculation using Cadence Virtuoso IC6.16 is provided.

2 VCO Overview

VCOs are the most important and complex component of the overall PLL/CDR design. The essential idea behind a VCO design is to generate a clock signal based on the Barkhausen criteria for oscillation which states that the magnitude of the VCO transfer function at the oscillation-frequency is 1 while the phase is -180 degrees. Two most popular VCO topologies whose sample architectures can be seen in Fig 4. Due to the superior noise performance we chose to design a LC-Tank base VCO. VCO is the device that generate the target clock. Ideally, its output frequency should be linearly related to the input control voltage. The Laplace transform function of the VCO is derived as follows:

$$\omega_{out}(t) = K_{VCO} v_{ctrl}(t) \tag{1}$$

$$\mathcal{L}[\omega_{out}(t)] = \omega_{out}(s) = K_{VCO} v_{ctrl}(s) \tag{2}$$

$$\phi_{out}(t) = \int_0^t \omega_{out}(\tau) d\tau = \int_0^t K_{VCO} v_{ctrl}(\tau) d\tau \tag{3}$$

$$\mathcal{L}[\phi_{out}(t)] = \phi_{out}(s) = \frac{\omega_{out}(s)}{s} = \frac{K_{VCO} v_{ctrl}(s)}{s} \tag{4}$$

Thus, the Laplace transform function for the VCO is:

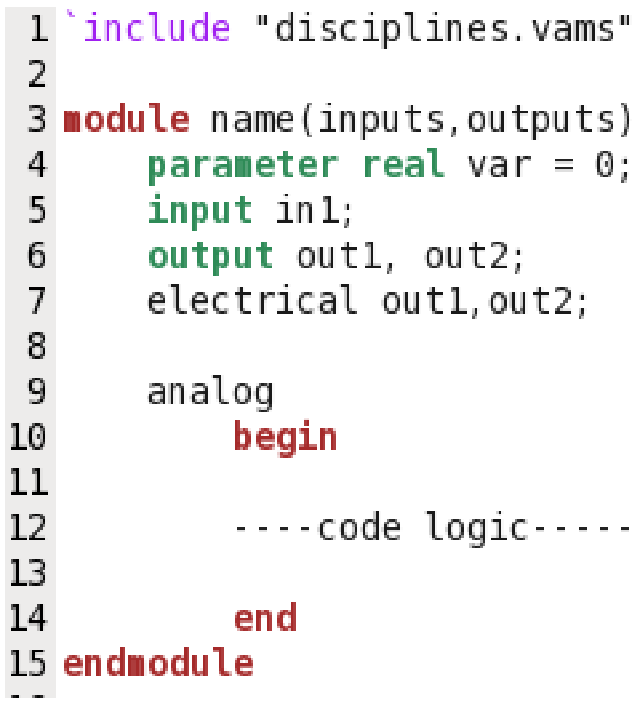
$$H_{VCO}(s) = \frac{\phi_{out}(s)}{v_{ctrl}(s)} = \frac{K_{VCO}}{s} \tag{5}$$

The K_{VCO} is defined as the VCO gain.

3 Verilog-A Overview

Traditionally SPICE is used as a common simulation engine to simulate analog/mixed-signal circuit. However, when simulating large networks the simulation times can become extremely long, thereby limiting the allowed design revisions to the circuit designer. It is very tedious to describe the behavior of a circuit using SPICE unless the complete physical transistor-level structure of the circuit is known to the designer. Furthermore, the SPICE simulation process is very technology dependent in that with technology scaling the SPICE models need to be updated as the older models become obsolete and invalid for accurate simulation. The aforementioned design process has remained virtually the same over the past few decades and even though the digital design synthesis process has progressed significantly by incorporating electronic system-level (ESL) design automation techniques, the mixed signal design process is very slow, laborious and therefore error-prone. Digital design engineers, though work with millions of transistors have been able to automate the design flow, but analog designers have been unable to do so even though most analog circuits only consist of tens of thousands of devices.

Verilog-A is a high-level Hardware Description Language (HDL) used to describe the structure and behavior of analog and mixed-signal systems. It is an extension to the IEEE 1364 Verilog HDL standard and is very powerful in providing fast prototyping capabilities for mixed-signal systems. The key advantage of circuit modeling using Verilog-A is that it provides a single language and simulator ecosystem that can be shared between analog system-level as well as device-level designers. Verilog-Aleverages the superior speed and capacity offered by traditional Verilog and allows event-driven capabilities within analog model simulation, making it an attractive choice when simulating highly complex mixed-signal circuits such as PLLs, CDRs, ADCs, and DACs. The only pitfall of using Verilog-A is that it cannot replace traditional transistor level SPICE simulation completely as it does not have synthesis capabilities like its digital counterpart Verilog. However, at the onset of the design phase, using Verilog-AMS for circuit modeling is very powerful for a mixed-signal circuit/system design engineer as it offers fast prototyping/verification for behavioral level simulation, thereby expediting the time-to-market for the system. Verilog-A is a HDL language capable of performing truly behavioral as well as transistor level co-simulation. Cadence has been the front-runner in promoting the language making it an industry standard, and has led the majority of the advancement efforts ever since its release in 2003. A typical skeleton of a Verilog-AMS code is shown in Figure 1 where the main components of a Verilog-A/AMS code are listed.



```
`include "disciplines.vams"

module name(inputs,outputs)
    parameter real var = 0;
    input in1;
    output out1, out2;
    electrical out1,out2;

    analog
        begin

        ----code logic-----

        end
endmodule
```

Figure 1: Verilog-AMS Sample Code

In the first line of the sample code shown in Figure 1, we include the ‘disciplines.vams’ header file. This file is a collection of physical signal types that are commonly used in Verilog-AMS and are thus referred to as ‘natures’. Electrical disciplines consist of ‘voltages’ and ‘currents’ and are used most commonly during mixed-signal system modeling where ‘voltage’ and ‘current’ are ‘natures’. Every Verilog-A component is defined as a ‘module’ and modules are the basic building blocks of any given Verilog-A files as they describe the component being modeled. Ports are the points where connections are made to the given component. Every port is required to have a direction associated with it, and by default in Verilog-AMS language there are three types of ports: **input, output** and **inout**. The keyword **electrical** signifies that the signals associated with the ports described as electrical are of 'voltage' and 'current' natures. Additionally, **analog** is the keyword after which point the Verilog-AMS compiler starts actual modeling as the logic/process starts after the 'analog begin'. Finally, every Verilog-A component code should end with the word **endmodule** as it signifies the point at which the compiler stops parsing of the code.

4 VCO Implementation Using Verilog-A

```
// VerilogA for TestLib, vco, veriloga
`include "constants.vams"
`include "disciplines.vams"
module vco(vin, out);
/* I/O Declarations */
input vin;
output out;
electrical vin;
electrical out;
/* Parameter Declarations */
parameter real Vmin=0;                              // Minimum input voltage
parameter real Vmax=Vmin+1 from (Vmin:inf); // Maximum input voltage
parameter real Fmin=1e9 from (0:inf);    // Minimum output frequency
parameter real Fmax=2e9 from (Fmin:inf); // Maximum output frequency
parameter real Vamp = 1.8 from [0:inf);  // Output sinusoid amplitude
parameter real ttol=1u/Fmax from (0:1/Fmax);// Crossing time tolerance
parameter real vtol = 1e-9;      // Voltage
// Minimum number points per period for update
parameter integer min_pts_update=32 from [2:inf);
// Transition time for square output
parameter real tran_time = 10e-12 from(0:0.3/Fmax);
// Std deviation of phase jitter (UI)
parameter real jitter_std_ui = 0 from [0:1);
/* Internal Variables */
real freq;
real phase;
integer n;
integer seed;
real jitter_rad;
real dPhase;
real phase_ideal;
analog
begin
        @(initial_step)
        begin
                seed = 671;
                n = 0;
                dPhase = 0;
                jitter_rad = jitter_std_ui*2*`M_PI;
        end
        // compute the freq from the input voltage
        freq = ((V(vin) - Vmin)*(Fmax - Fmin) / (Vmax - Vmin)) + Fmin;
        $bound_step(1/(min_pts_update*freq));
        if (freq > Fmax) freq = Fmax;
        if (freq < Fmin) freq = Fmin;
        phase_ideal = 2*`M_PI*idtmod(freq, 0.0, 1.0, -0.5);
        phase = phase_ideal + dPhase;
        @(cross(phase_ideal + `M_PI/2, +1, ttol, vtol)
                  or cross(phase_ideal - `M_PI/2, +1, ttol, vtol))
        begin
                 dPhase = $rdist_normal(seed,0,jitter_rad);
        end
        @(cross(phase + `M_PI/2, +1, ttol, vtol)
                 or cross(phase - `M_PI/2, +1, ttol, vtol))
        begin
                 n = (phase >= -`M_PI/2)&&(phase < `M_PI/2);
        end
        // generate the output
        V(out) <+ transition(n?Vamp:0, 0,tran_time);
end
endmodule
```

5 Cadence Simulation Steps for VCO

1. The VCO is the most critical component of a clocking circuit so we try to model using Verilog-A because it allows us to behaviorally estimate the jitter specifications. Create a new library and call it 'TestLib'. Navigate to *File* → *New* → *Cell* – *View* and choose VerilogA in view-type as shown Figure 2. Create a model for the VCO using code shown above and save the file as 'vco'. Only the white-noise jitter is considered in this design and it is modeled by a Gaussian white-noise probability distribution function.

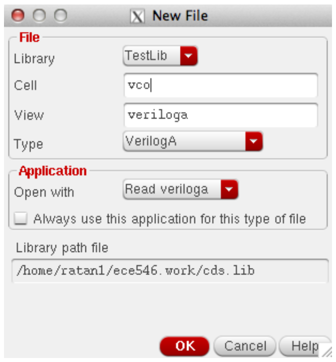

New File
File
Library: TestLib
Cell: vco
View: veriloga
Type: VerilogA
Application
Open with: Read veriloga
Always use this application for this type of file
Library path file
/home/ratan1/ece546.work/cds.lib

Figure 2: VCO Verilog-A Testbench

2. If the code is error-free a pop up window will open asking whether you want to create a symbol for the designed block. Select 'Ok' to create the symbol view for the VCO.

3. Figure 3 shows the 'VCO' testbench schematic. Create a new schematic named 'vco_tb'.

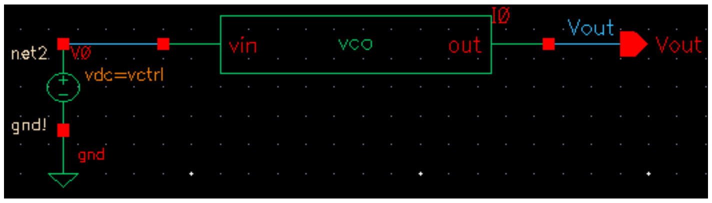
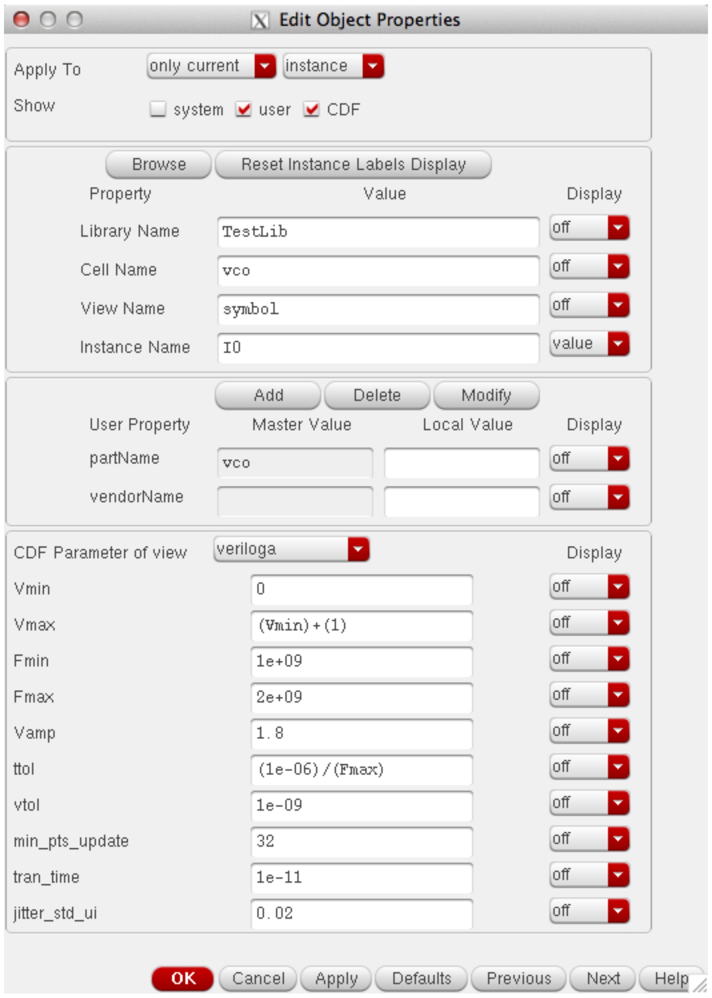


Figure 3: VCO Verilog-A Testbench

4. Click on the 'VCO' block and press **q**, a window as shown in Figure 4 will appear. Enter the appropriate value of charge-pump current as per the design objectives.



Apply To: only current, instance
Show: [ ] system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	TestLib	off
Cell Name	vco	off
View Name	symbol	off
Instance Name	I0	value

User Property	Master Value	Local Value	Display
partName	vco		off
vendorName			off

CDF Parameter of view: veriloga

		Display
Vmin	0	off
Vmax	(Vmin)+(1)	off
Fmin	1e+09	off
Fmax	2e+09	off
Vamp	1.8	off
ttol	(1e-06)/(Fmax)	off
vtol	1e-09	off
min_pts_update	32	off
tran_time	1e-11	off
jitter_std_ui	0.02	off

Figure 4: VCO Verilog-A Testbench Variable Setup

5. The VCO circuit is supposed to generate a periodic square-wave output at the desired frequency of interest (as a function of the control voltage) with a certain jitter level which in our case is chosen to be 2% Unit-Interval (UI) of period. From the final output waveform shown in Figure 5 it is clear that the 'VCO' is functioning correctly.

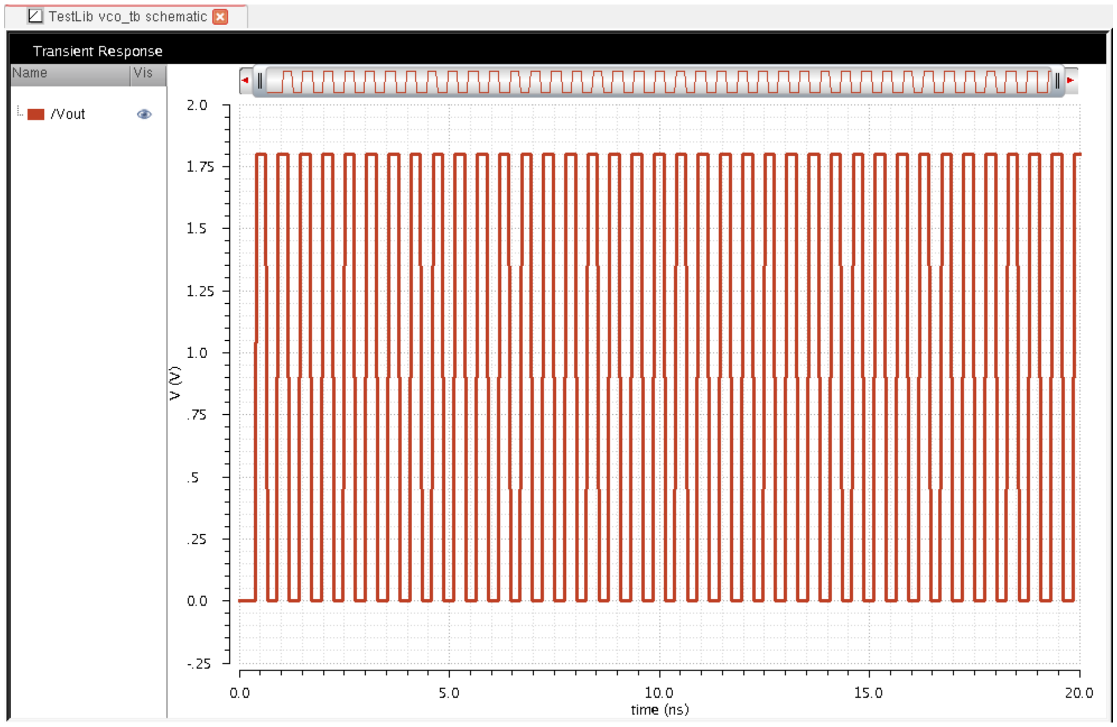
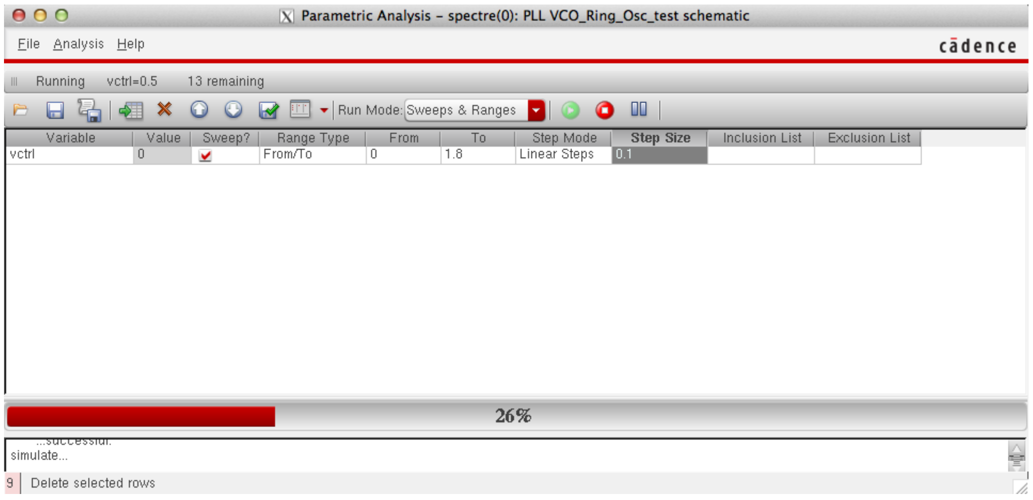


Figure 5: VCO Verilog-A Simulation Output

6. For a VCO, a key figure-of-merit is the control voltage tuning range. Thus, we have to perform a parametric analysis in order to observe the change in 'frequency' as well as K_{VCO} as a function of 'Vctrl'. In order to do so in the ADE setup window click on *Tools* → *Parametric Analysis* and a window like Figure 6 should pop-up. Within the parametric analysis window, when you double-click on the variable box, a drop-down list will show up from which you should pick 'vctrl'.

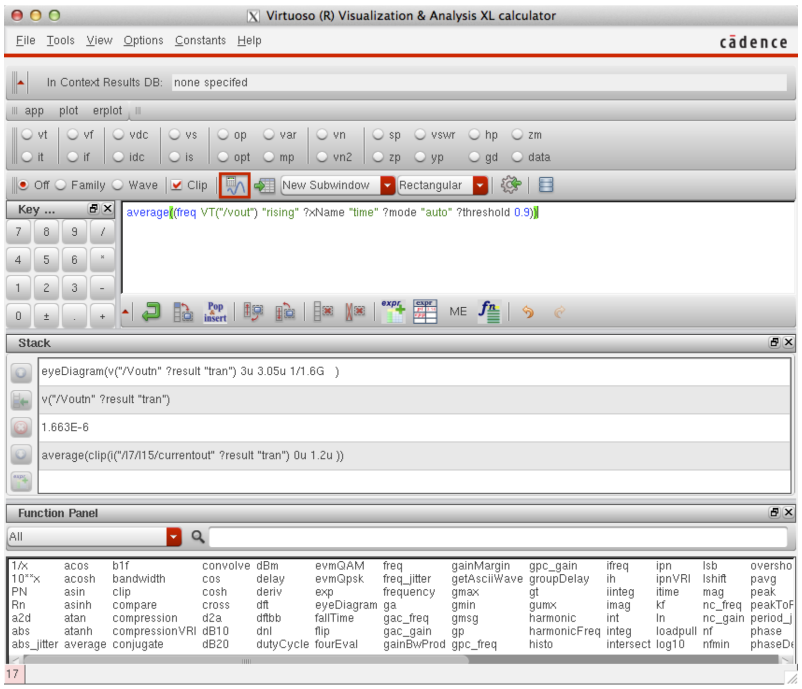

Parametric Analysis – spectre(0): PLL VCO_Ring_Osc_test schematic
File Analysis Help
cādence
Running vctrl=0.5 13 remaining
Run Mode: Sweeps & Ranges
26%
...successful.
simulate...
9 Delete selected rows

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Step Size	Inclusion List	Exclusion List
vctrl	0	✓	From/To	0	1.8	Linear Steps	0.1		

Figure 6: Vctrl Parametric Analysis Setup

To run the parametric analysis, click on the 'Play' within the Parametric-Analysis window. This setup is basically going to run the transient simulation $\frac{To-From}{StepSize}$ times by varying the control-voltage input to the VCO.

7. To plot frequency vs. Vctrl and K_{VCO} vs. Vctrl we need to use the 'Calculator' tool in-built within ADE. Click on *Tools* $\rightarrow$ *Calculator*. The Calculator window as shown in Figure 7 will open up and within it now you should select 'Vt' from the toolbar. The schematic will open up, so within the schematic select the 'vout' node. From the 'Function-Panel' within the Calculator window choose the 'frequency' and 'average' functions to make up the function shown in Figure 7. Now go back to the ADE window, click on the right-pane and select the 'Pick-Outputs' button. A window will pop up so within it select 'Get-Expression' and name it 'freq'. This will bring the expression you just created in the Calculator so that you can plot it. Conversely, you can also click on the 'plot' button shown in the red-box in Figure 7 to plot the expressio; however, doing so makes the title of plot look a little too crammed.


Virtuoso (R) Visualization & Analysis XL calculator
File Tools View Options Constants Help
cādence
In Context Results DB: none specifed
app plot erplot
vt vf vdc vs op var vn sp vswr hp zm
it if idc is opt mp vn2 zp yp gd data
Off Family Wave Clip
New Subwindow
Rectangular
Key ...
Stack
Function Panel
All
17

```
average((freq VT("/vout") "rising" ?xName "time" ?mode "auto" ?threshold 0.9))
```

```
eyeDiagram(v("/Voutn" ?result "tran") 3u 3.05u 1/1.6G   )
v("/Voutn" ?result "tran")
1.663E-6
average(clip(i("/I7/I15/currentout" ?result "tran") 0u 1.2u ))
```

```
1/x      acos    b1f           convolve dBm       evmQAM      freq        gainMargin   gpc_gain     ifreq     ipn      lsb      overshoo
10**x    acosh   bandwidth     cos      delay     evmQpsk     freq_jitter getAsciiWave groupDelay   ih        ipnVRI   lshift   pavg
PN       asin    clip          cosh     deriv     exp         frequency   gmax         gt           iinteg    itime    mag      peak
Rn       asinh   compare       cross    dft       eyeDiagram  ga          gmin         gumx         imag      kf       nc_freq  peakToF
a2d      atan    compression   d2a      dftbb     fallTime    gac_freq    gmsg         harmonic     int       ln       nc_gain  period_j
abs      atanh   compressionVRI dB10    dnl       flip        gac_gain    gp           harmonicFreq integ     loadpull nf       phase
abs_jitter average conjugate   dB20     dutyCycle fourEval    gainBwProd  gpc_freq     histo        intersect log10    nfmin    phaseDe
```

Figure 7: ADE Calculator

8. Repeat the same steps as above to create an expression within the calculator to compute the K_{VCO}. Use the 'deriv' function within the Calculator Function Panel to do so. Finally, click on the 'Play' button within the ADE window to plot frequency vs. Vctrl and K_{VCO} vs. Vctrl curves. Your output should look like Figure 8.

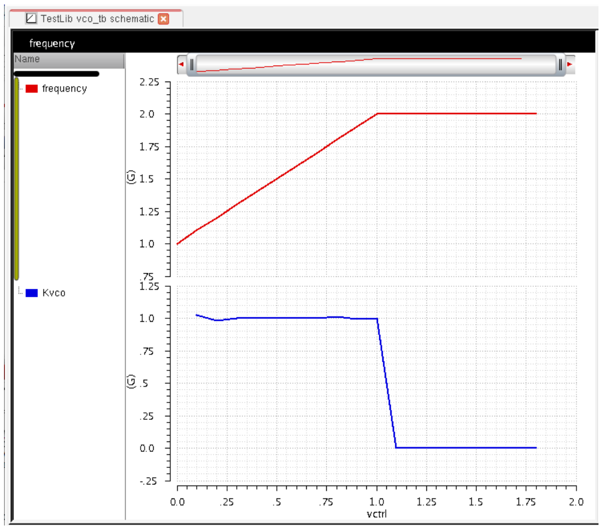


Figure 8: Frequency vs. Vctrl and K_{VCO} vs. Vctrl Simulation Plots

The K_{VCO} calculated in Figure **??** is the value used to calculate the required charge-pump current as well as C_1 and C_2 values from the loop-filter. Depending upon your chosen VCO output frequency you can pick the corresponding K_{VCO} value accordingly.

9. To simulate the jitter at the VCO output during lock-condition, select the *vout* waveform, click on $Measurements \rightarrow EyeDiagram$. Your final output should look like that shown in Figure 9 once you click on 'Plot Eye'. Note that this model is only behavioral so any transistor-level non-idealities are not captured. Nevertheless, behavioral modeling is very powerful in performing rapid prototyping of the clocking circuit core elements and performs a system level noise/timing budget for the design.

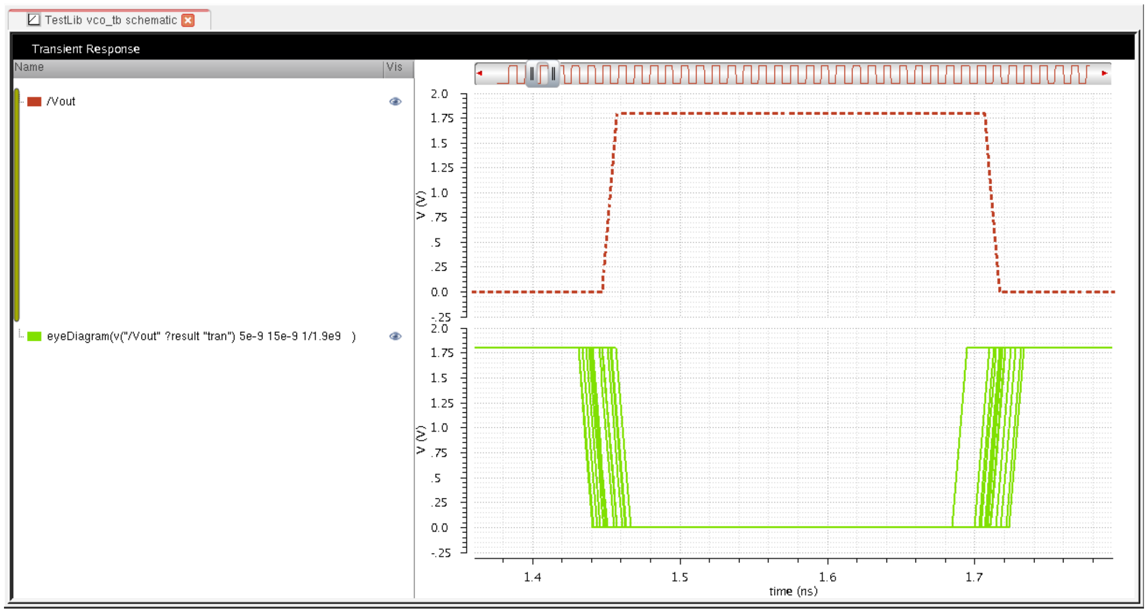


Figure 9: VCO Verilog-A Jitter