scientific analog

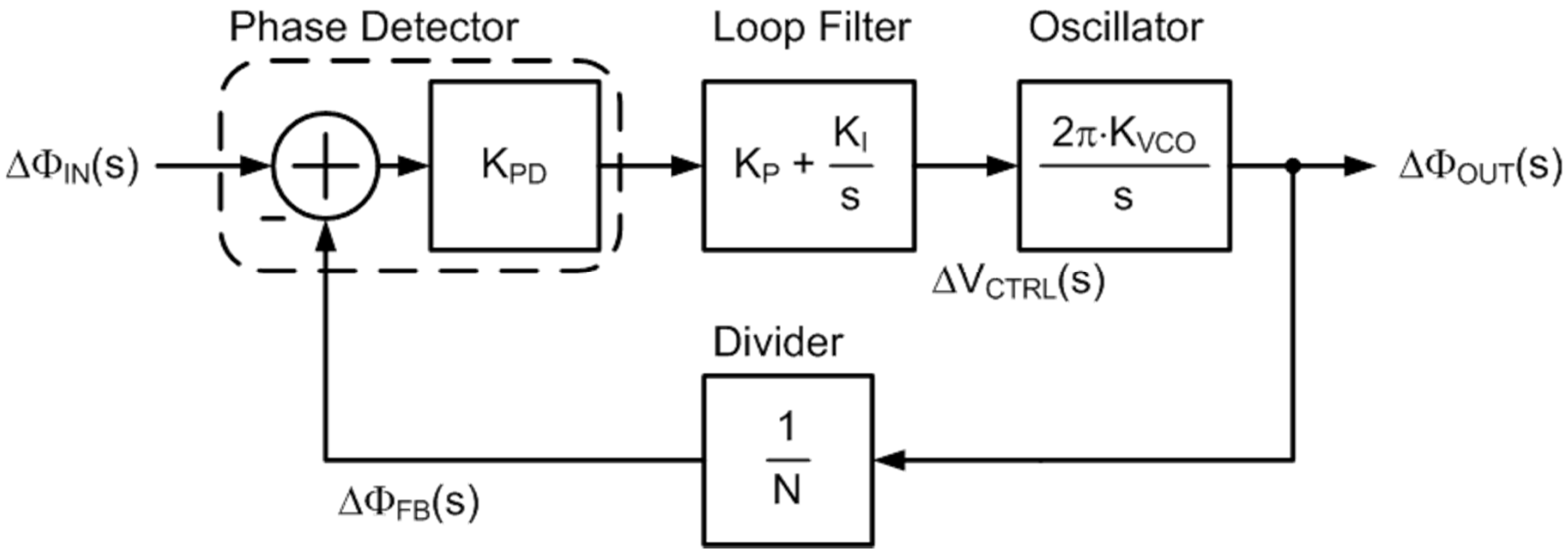
Putting Together: Simulating Digital PLL with *XMODEL*

Scientific Analog, Inc.

July 2017

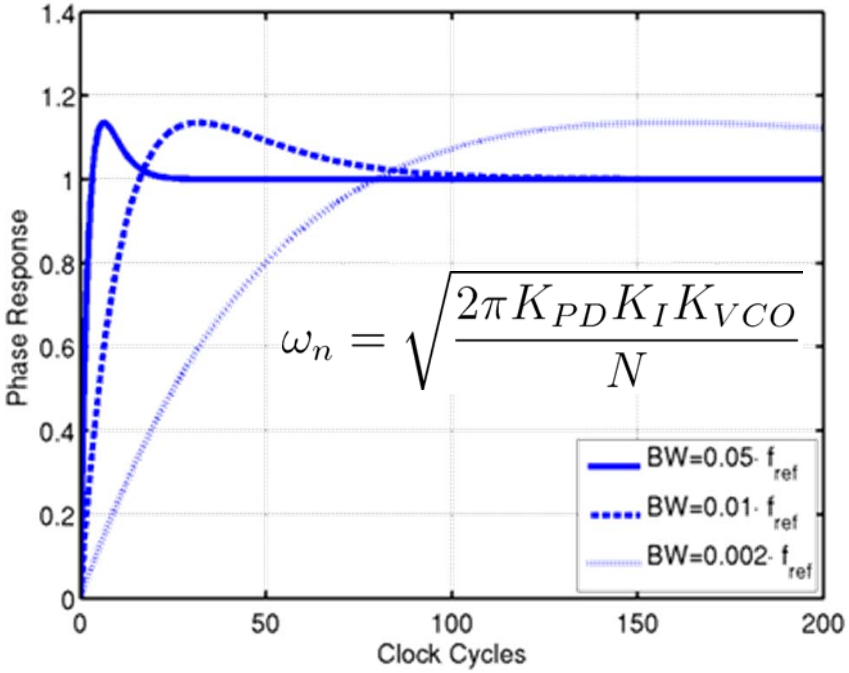
PLL Model

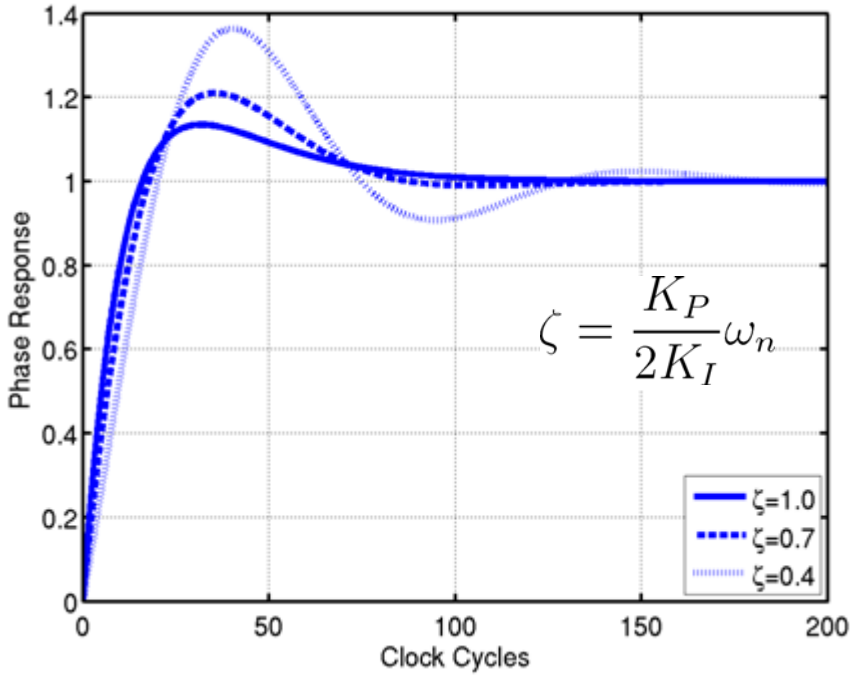
- Approximate s-domain continuous-time model
  - Enables simple analysis on loop dynamics
  - Valid up to ~f_{ref}/10 due to sampling delay [F. Gardner]



PLL Loop Dynamics

$$H(s) = \frac{\Delta\Phi_{out}(s)}{\Delta\Phi_{in}(s)} = N\frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

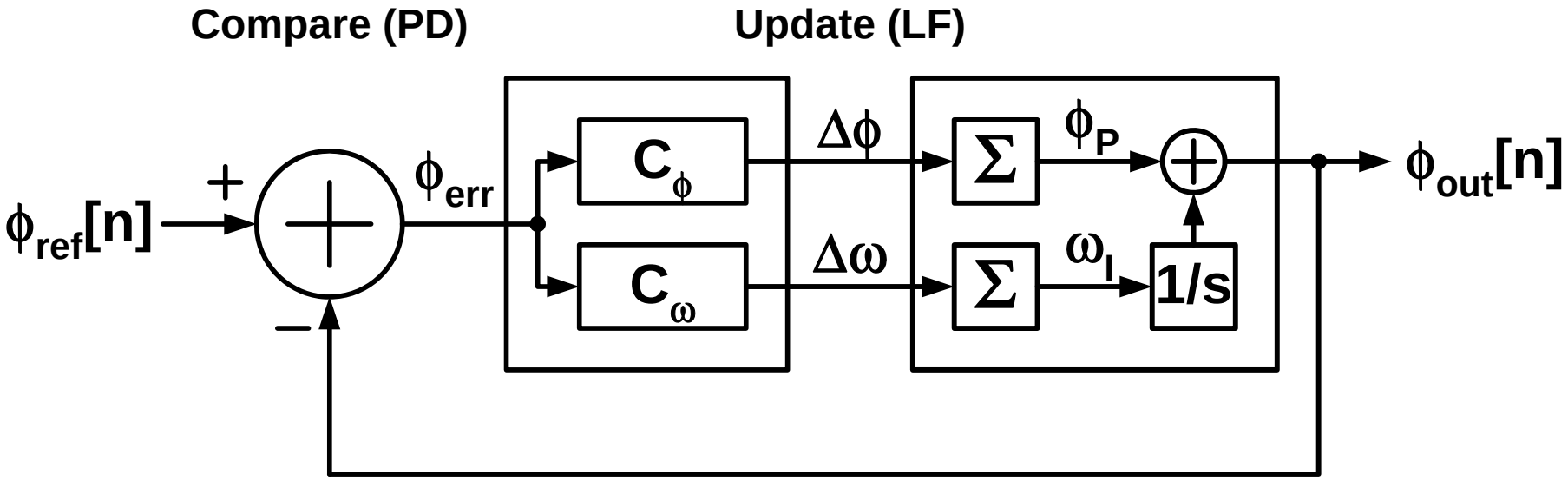




Phase Step Responses

Alternate PLL Model

- Upon the detection of ϕ_{err}, the PLL makes changes to the oscillator phase ($\Delta\phi$) and frequency ($\Delta\omega$)



$$\frac{\omega_n}{\omega_{ref}} = \sqrt{\frac{\Delta\omega}{2\pi\omega_{ref}\phi_{err}}} \qquad \zeta = \frac{\Delta\phi}{4\pi(\omega_n/\omega_{ref})\phi_{err}}$$

Digital Loop Filter Model

- models/digital_lf.sv

```
assign in_ext = {{9{in[3]}}, in};

always @(posedge clk or reset) begin
    if (reset) begin
        acc <= init_value;
        out <= init_value;
    end
    else begin
        acc <= acc + in_ext;
        out <= Kp*in_ext + Ki*acc;
    end
end
```

sign extension

initialization

PI-controller

How would you determine Kp & Ki?

Digital PLL

- ϕ_{err} that causes a unit change in the TDC output is:

$$\phi_{err} = 2\pi / K_{TDC}$$

- Then the resulting $\Delta\phi$ is:

$$\Delta\phi = 2\pi \cdot K_P \cdot K_{DCO}$$

- And the resulting $\Delta\omega$ is:

$$\Delta\omega = K_I \cdot K_{DCO} \cdot \omega_{ref}$$

- Therefore,

$$\frac{\omega_n}{\omega_{ref}} = \frac{\sqrt{K_I K_{DCO} K_{TDC}}}{2\pi} \qquad \zeta = \frac{K_P K_{DCO} K_{TDC}}{4\pi(\omega_n/\omega_{ref})}$$

Exercise: Digital PLL

- Consider a 1.5-GHz PLL with N=16, K_{TDC} = 50 steps/UI, and K_{DCO} = 0.004%; assuming that K_I=1 and determine ω_n/ω_{ref} and K_P that yields ζ=1

Exercise: Digital PLL

- Consider a 1.5-GHz PLL with N=16, K_{TDC} = 50 steps/UI, and K_{DCO} = 0.004%; assuming that K_I=1 and determine ω_n/ω_{ref} and K_P that yields ζ=0.7~1
- Answer:

$$\frac{\omega_n}{\omega_{ref}} = \frac{\sqrt{K_I K_{DCO} K_{TDC}}}{2\pi} = \frac{\sqrt{1 \cdot 0.004 \cdot 0.01 \cdot 50}}{2\pi} \approx 0.0071$$

$$K_P = \frac{4\pi\zeta\omega_n/\omega_{ref}}{K_{DCO}K_{TDC}} = \frac{4\pi \cdot (0.7 \sim 1) \cdot 0.0071}{0.004 \cdot 0.01 \cdot 50} \approx 31 \sim 44.7$$

  - I would choose K_P of 32 (why?)

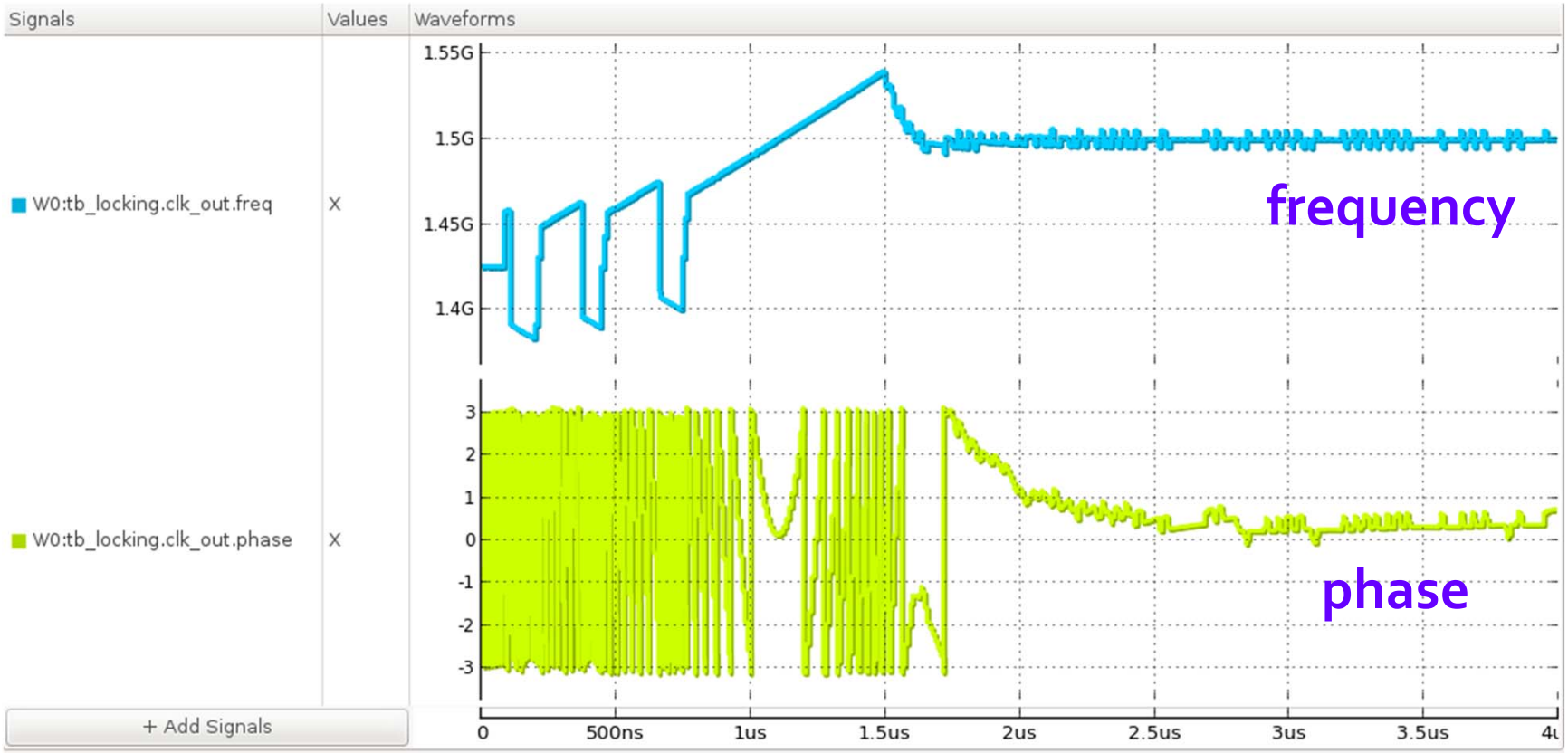
Digital PLL Model

- models/dpll.sv

```
// time-to-digital converter
tdc_gosc     tdc(.clk_ref(clk_ref), .clk_fb(clk_fb), .out(terr));
// digital loop-filter
digital_lf   #(.Kp(32), .Ki(1), .init_value(init_value))
             dlf(.in(terr), .out(dctrl), .clk(clk_lf), .reset(reset));
xbit_to_bit    conn(.in(clk_fb), .out(clk_lf));
// digitally-controlled oscillator
dco_exp      dco(.in(dctrl), .out(clk_out));
// frequency divider (/16)
fdiv16_sync fdiv(.in(clk_out), .out(clk_fb));
```

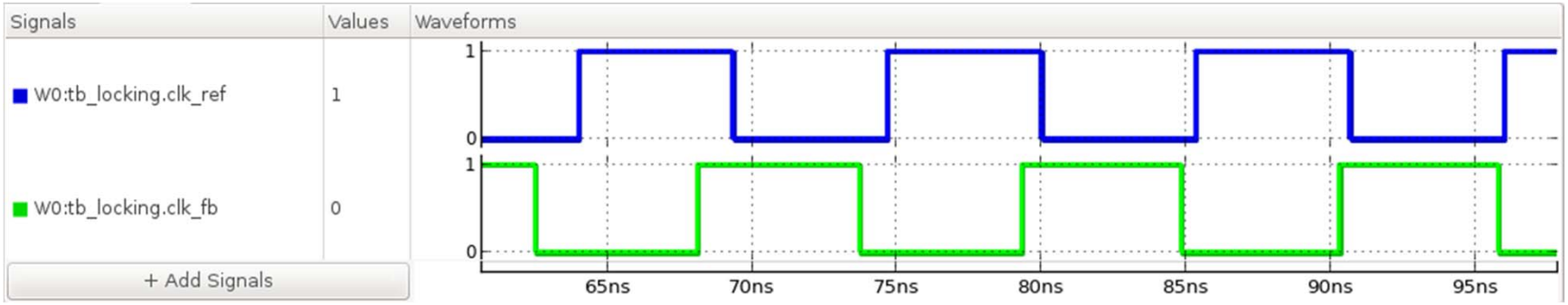
Exercise: Locking Transient

- sim/tb_locking:

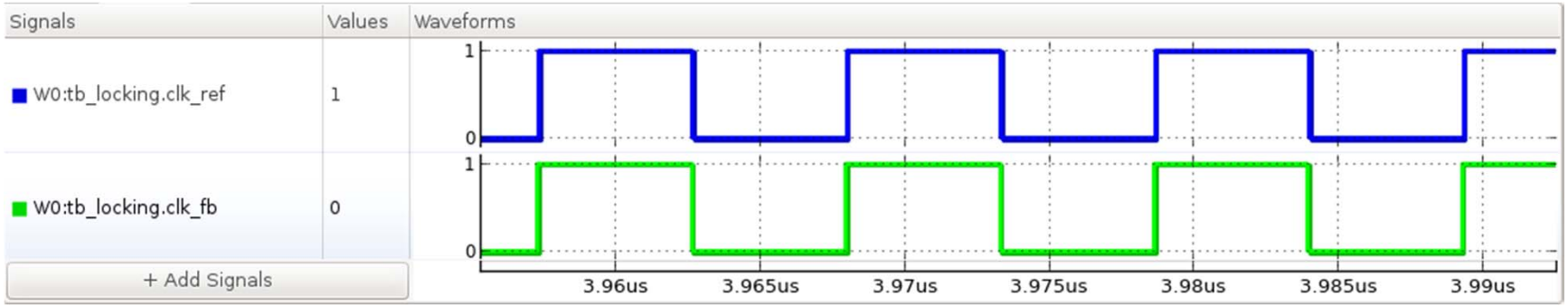


Results

- Before lock



- After lock

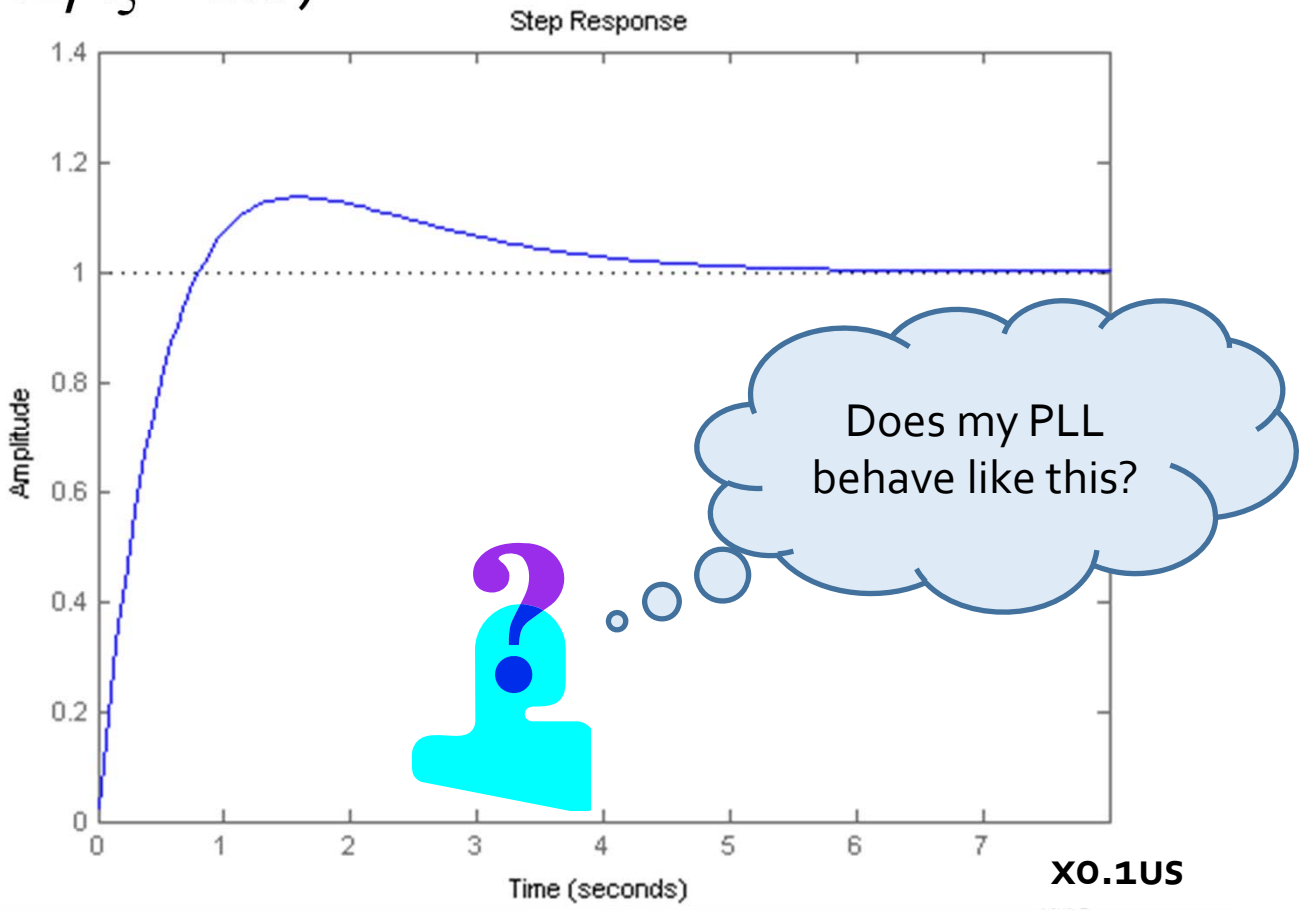


Verifying PLL Behaviors

- You have simulated the PLLs both in transient and in locked state – but is that enough to validate the design?
  - No, we only know that the PLL can lock; but we don't know how well it does
- PLL designers would like to validate the PLL against the desired metric (e.g. bandwidth and damping factor)
  - Measure the phase step response in time domain, or
  - Measure the phase transfer function in frequency domain
- With *XMODEL*, we can do both easily

Phase Step Response in PLL

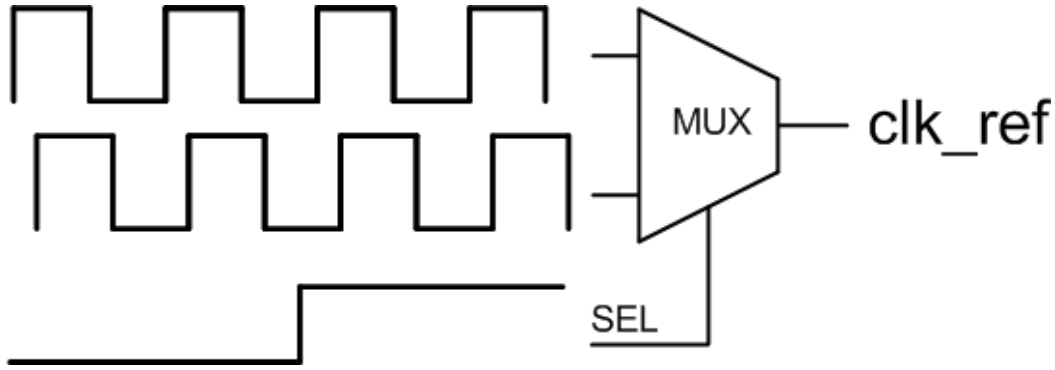
- Expected response from the s-domain model (ω_n = $2\pi \times 2$MHz, ζ = 1.0)



Generating the Phase Step

- How can we generate a step change in the input clock phase?

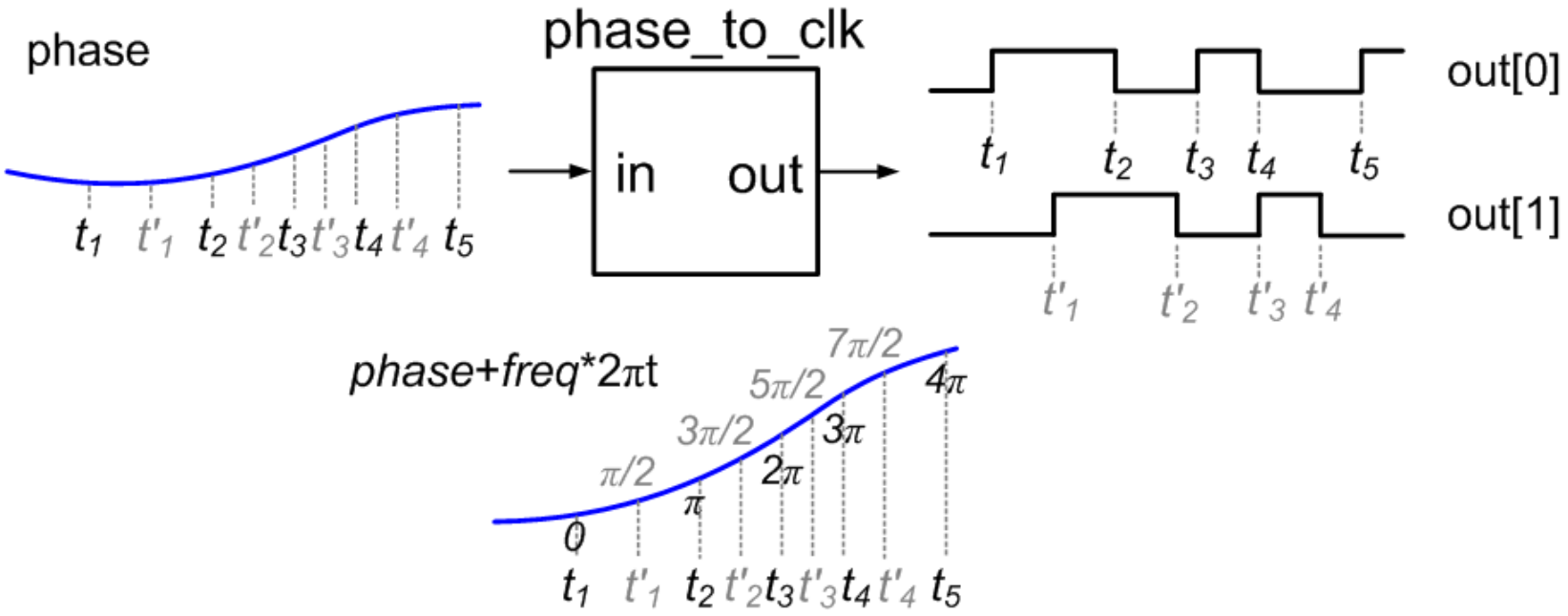
- One possible way is to use a mux….



- The easier way is to use ***variable domain transformation (VDT)*** primitives in *XMODEL*

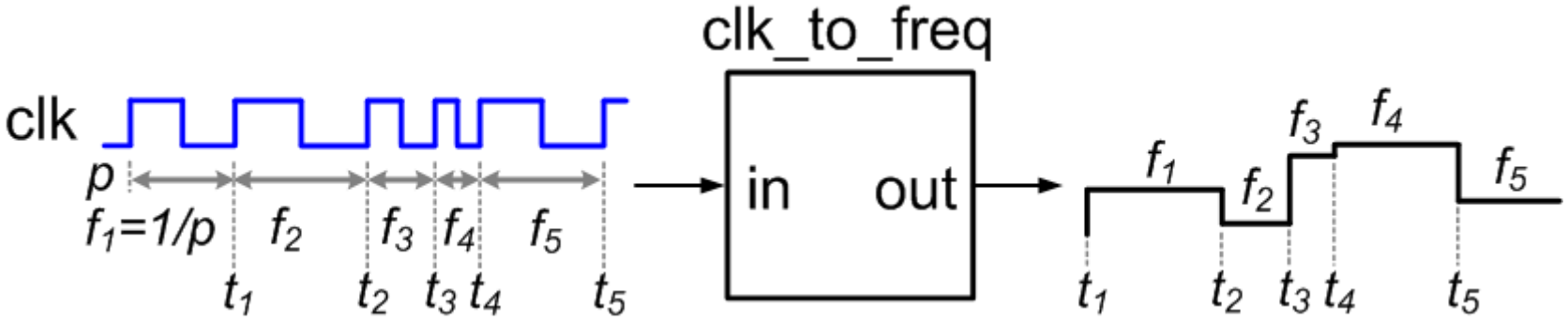
phase_to_clk Primitive

- Takes an xreal-type signal and generates a clock with the corresponding phase
  - It can also generate multi-phase clocks and add noise



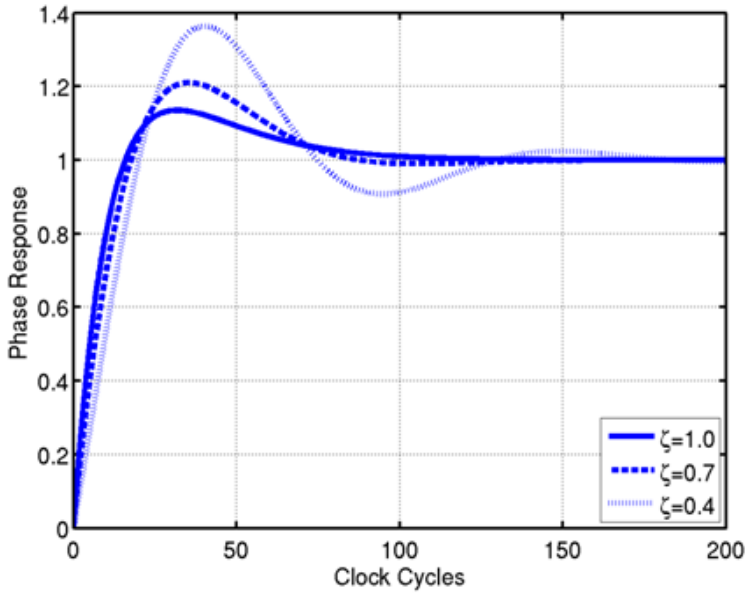
clk_to_freq Primitive

- Measures the frequency of a clock signal and produces the corresponding signal output



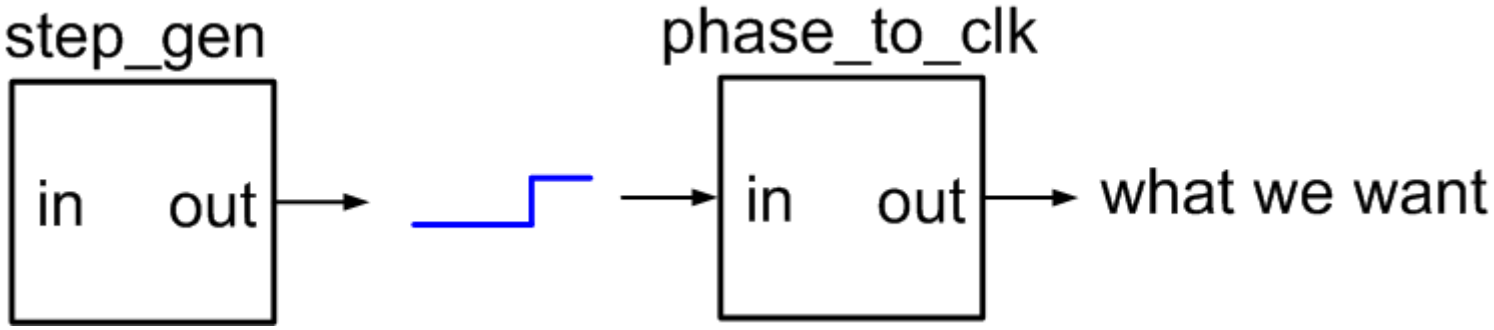
Exercise

- Simulate the phase step response of the PLL
  - The skeleton code is given sim/tb_phasestep/tb_phasestep.sv
- Use a reference clock with a step phase change of 1-rad at t=2us



Hints

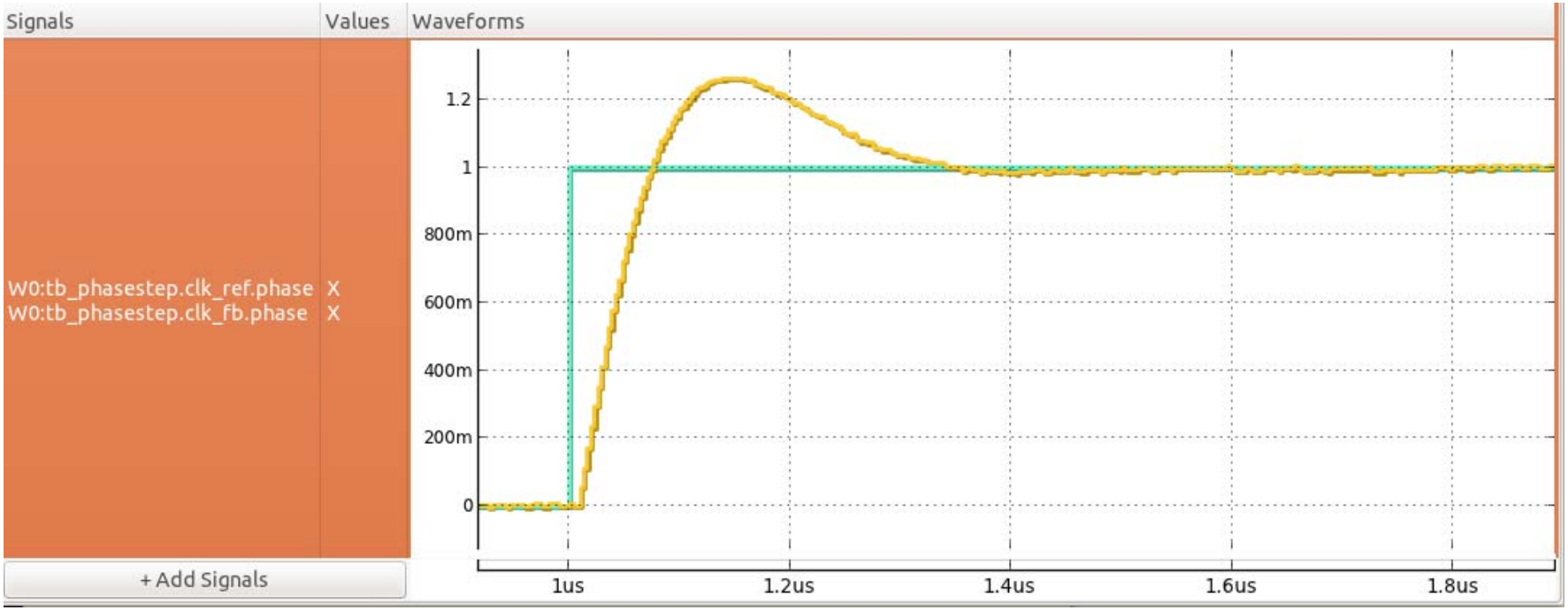
- Use ***phase_to_clk*** primitive to generate the reference clock
- Use ***step_gen*** primitive to generate the step signal



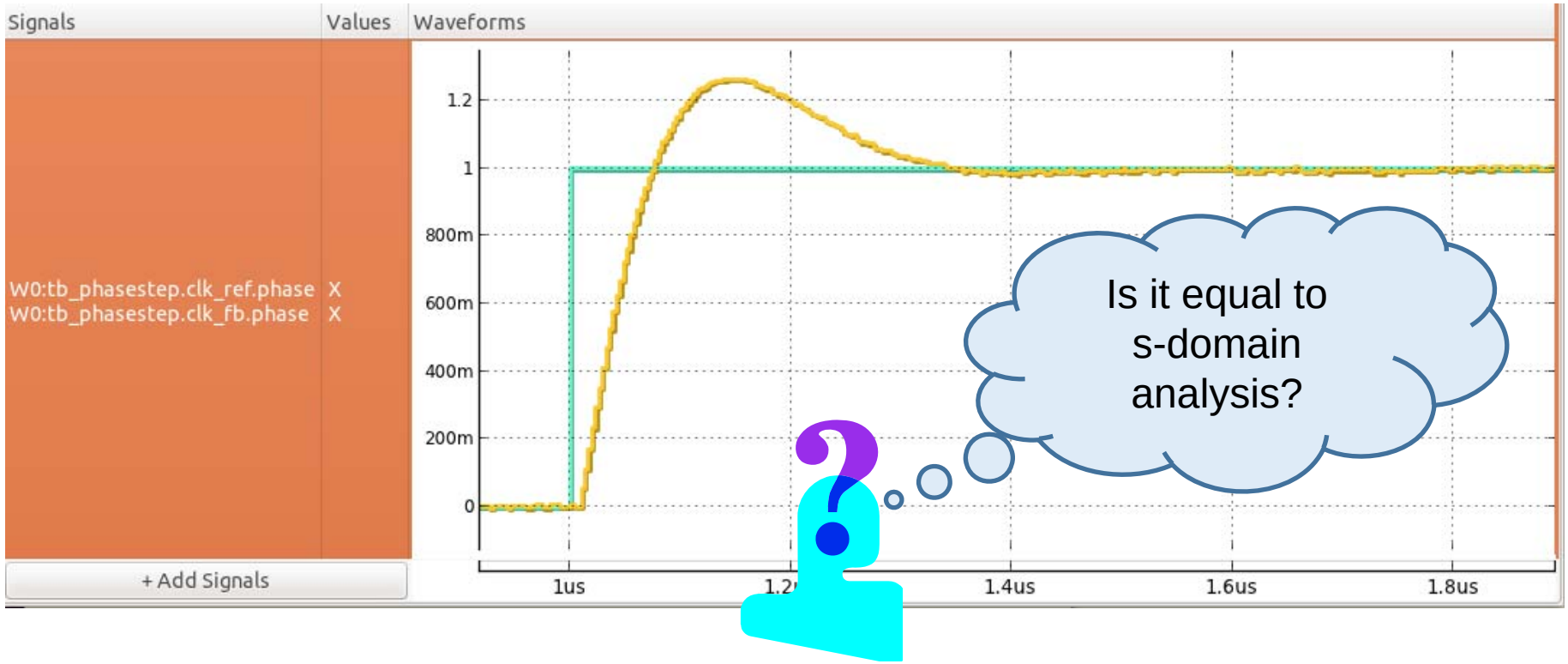
Answer

```
xreal step;
step_gen #(.init_value(0.0), .change(1), .delay(2e-6))  step_gen(step);
phase_to_clk #(.freq(freq_ref)) clk_gen(.in(step), .out(clk_ref));
```

- Step response



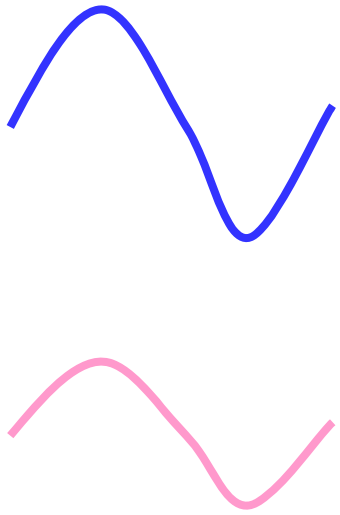
Phase Step Response in PLL

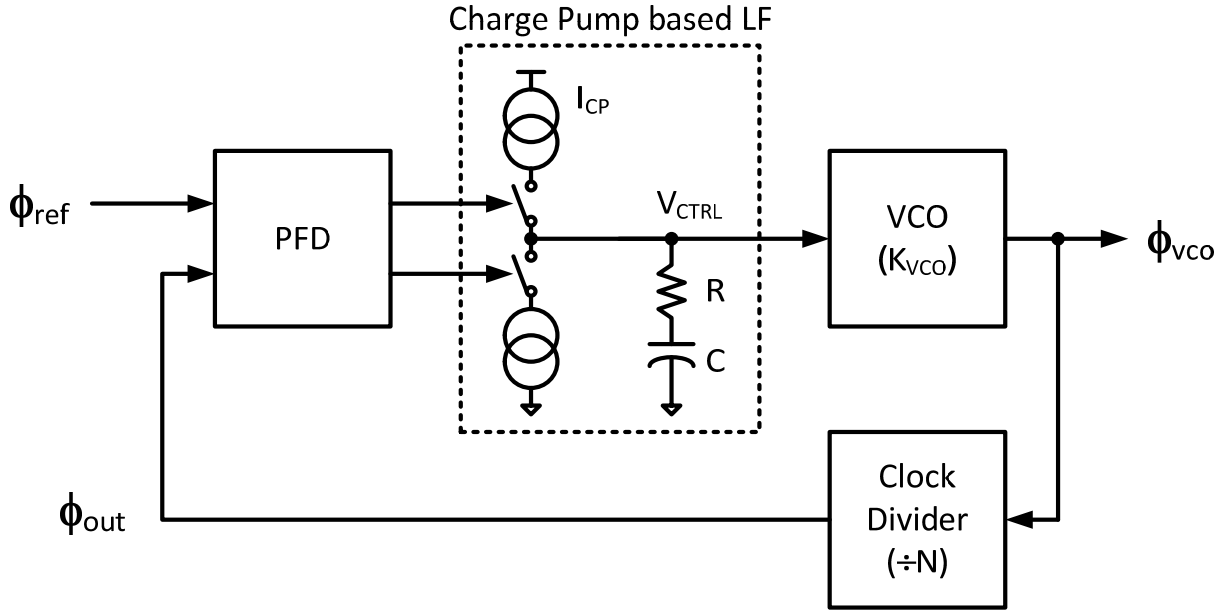


- More effective way to validate the loop dynamics is to measure its phase-domain transfer function

Measuring PLL Jitter Transfer Function

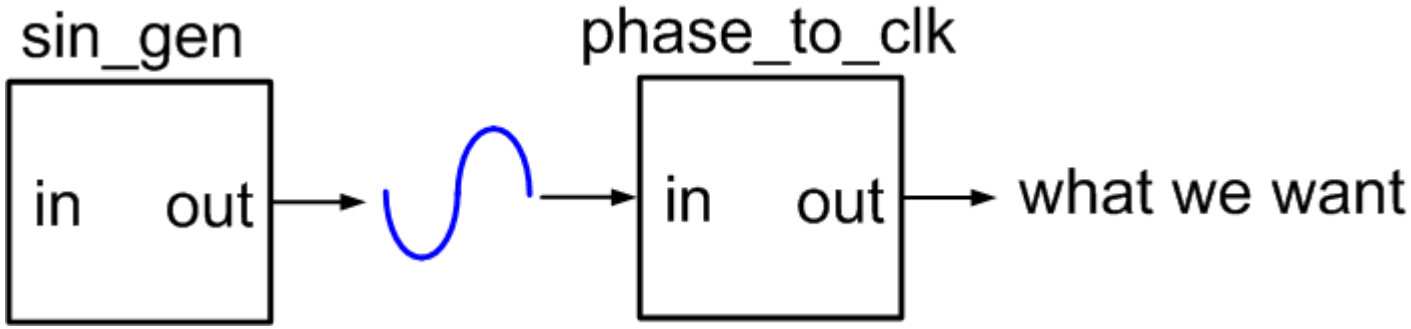
- Apply a sinusoidal jitter (SJ) input and measure the amplitude/phase of the resulting SJ output
  - And sweep the SJ frequency to collect the transfer function





Generating Sinusoidal Clock Jitter

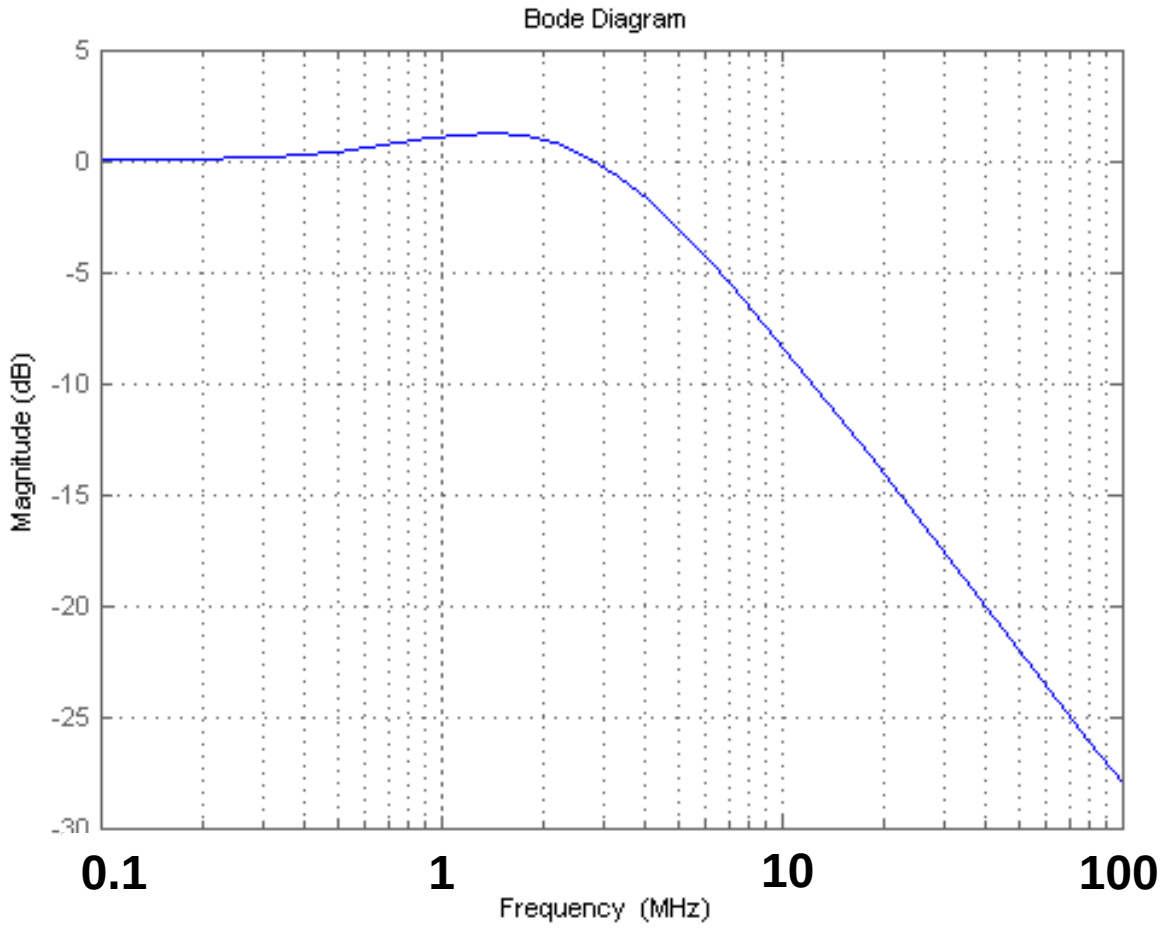
- One way is to combine ***sin_gen*** with ***phase_to_clk***



- Also, the ***clk_gen*** primitive in *XMODEL* can also generate a clock with sinusoidal jitter
  - Use ***SJ_amp*** and ***SJ_freq*** parameters to set its amplitude and frequency

Measuring PLL Jitter Transfer Function

- Expected TF from the s-domain model
  - $\omega_n = 2\pi \times 2\text{MHz}$, $\zeta = 1.0$



Exercise

- Measure the phase-domain transfer function (also called jitter transfer) of the PLL

- Hint: since a set of repetitive simulations is needed, it would be convenient to write an automated script
  - *XMODEL* comes with a Python support library for that purpose, called "*XMULAN*"

Answers

- See tb_tran.sv.empy and jtran.py located in **$XMODEL_HOME/example/bin** directory
- tb_jtran.sv.empy is a template testbench in empy:

Variable fields to be filled by jtran.py script

```
parameter real freq_ref = @freq_ref; // input clock frequency
parameter real SJ_amp   = @sj_amp;   // input SJ amplitude in UI
parameter real SJ_freq  = @sj_freq;  // input SJ frequency in Hz
parameter real t_lock   = @t_lock;   // initial PLL locking time
…
// reference clock generator with sinusoidal jitter
clk_gen   #(.freq(freq_ref), .SJ_amp(SJ_amp), .SJ_freq(SJ_freq))
          clk_gen(clk_ref);

// probing  phase of clk_out
probe_phase #(.freq(freq_ref), .start(t_lock)) probe_phase(clk_out);
```

Running jtran.py

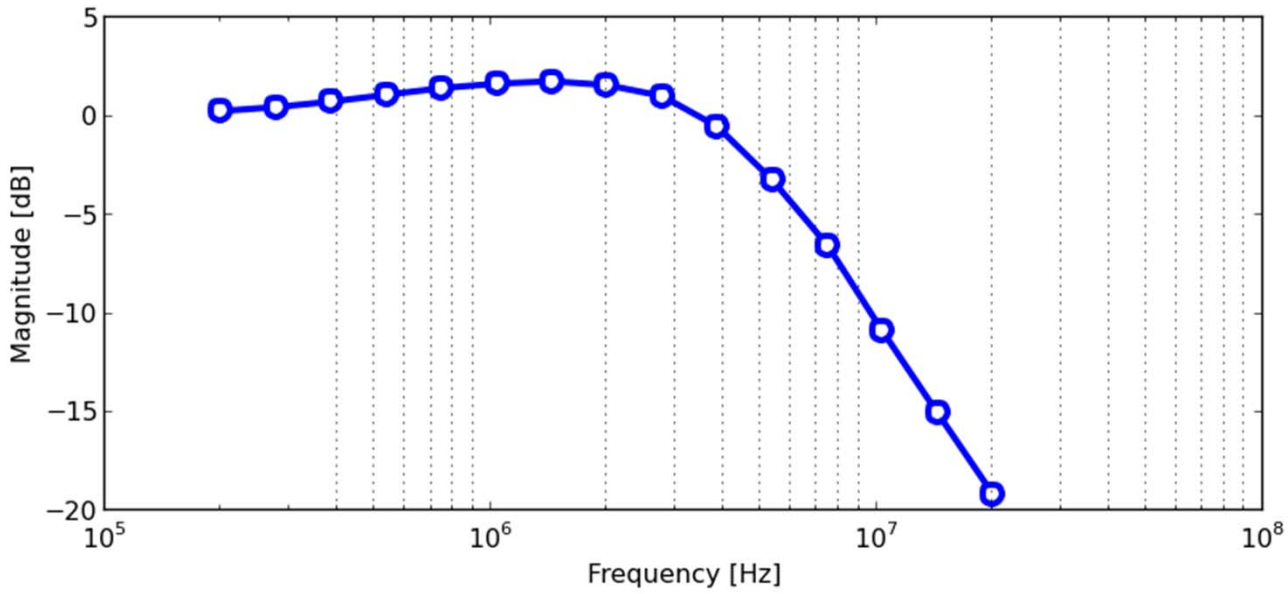
- Example of using jtran.py script:

```
$ ${XMODEL_HOME}/example/bin/jtran.py \
     --t_lock 1e-6 --freq_ref 93.75e6 \
     --freq_max 20e6 --freq_min 0.2e6 --num_sweep 15 \
     --cmd models/dpll.f --source tb_jtran.sv.empy \
     --var tb_jtran.clk_fb
```

- Some options (type 'jtran.py –h' for a full list):
  - --t_lock 1e-6 : run initial transient simulation for 1us
  - --freq_ref 93.75e6 : use reference frequency of 93.75MHz
  - --freq_max, --freq_min, --num_sweep : set the frequency sweep range

Results

```
$ cd sim/tb_jtran
$ make
```

Title	Graphs
Jitter Transfer Curve [tb_jtran.clk_fb]	

Jitter Histogram Simulation

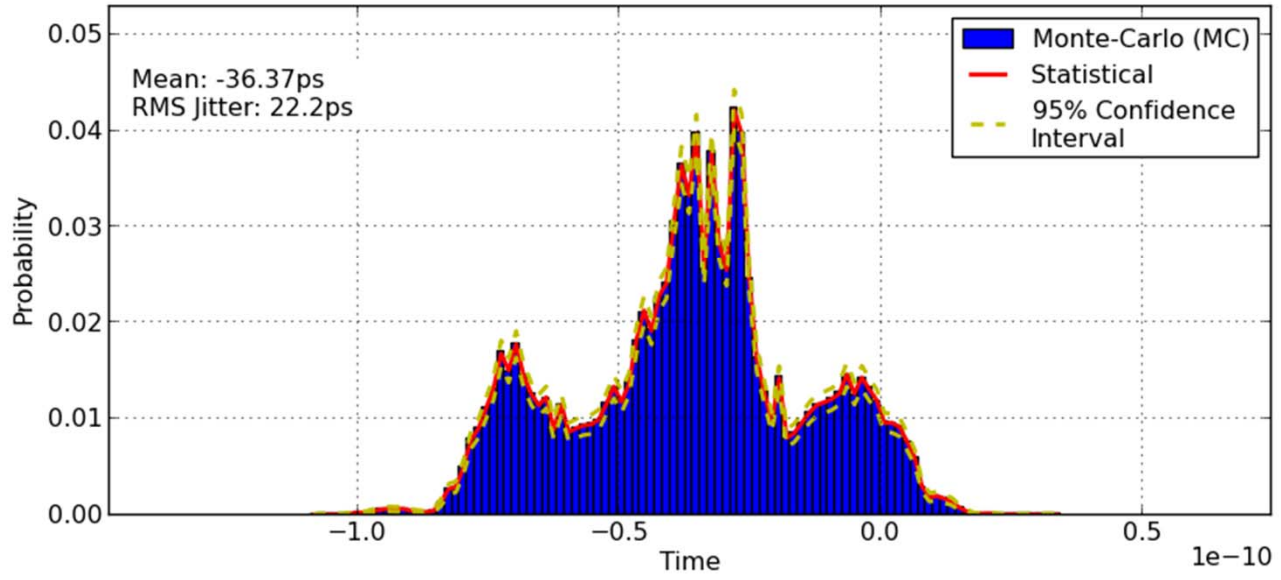
- One way to evaluate the PLL noise performance is to plot its jitter histogram: sim/tb_jhist/tb_jhist.sv

```
$ cd sim/tb_jhist
$ make
```

Title

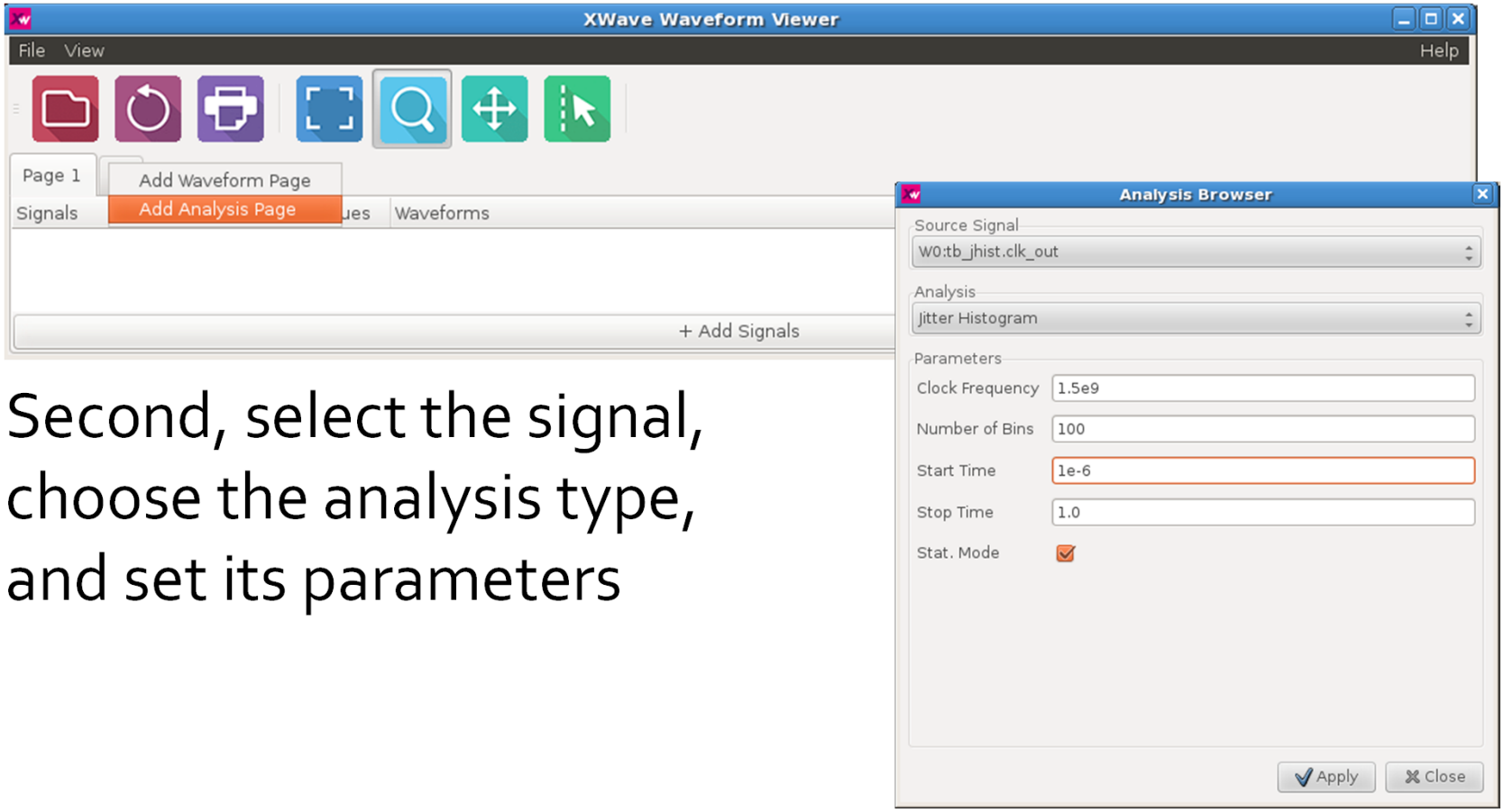
Graphs

Jitter Histogram
[W0:tb_jhist.clk_out]



Plotting Jitter Histogram in *XWAVE*

- First, click "+" tab and select "Add Analysis Page"



- Second, select the signal, choose the analysis type, and set its parameters

Plotting Phase Noise in *XWAVE*

- You can also plot phase noise spectrum in *XWAVE*: