

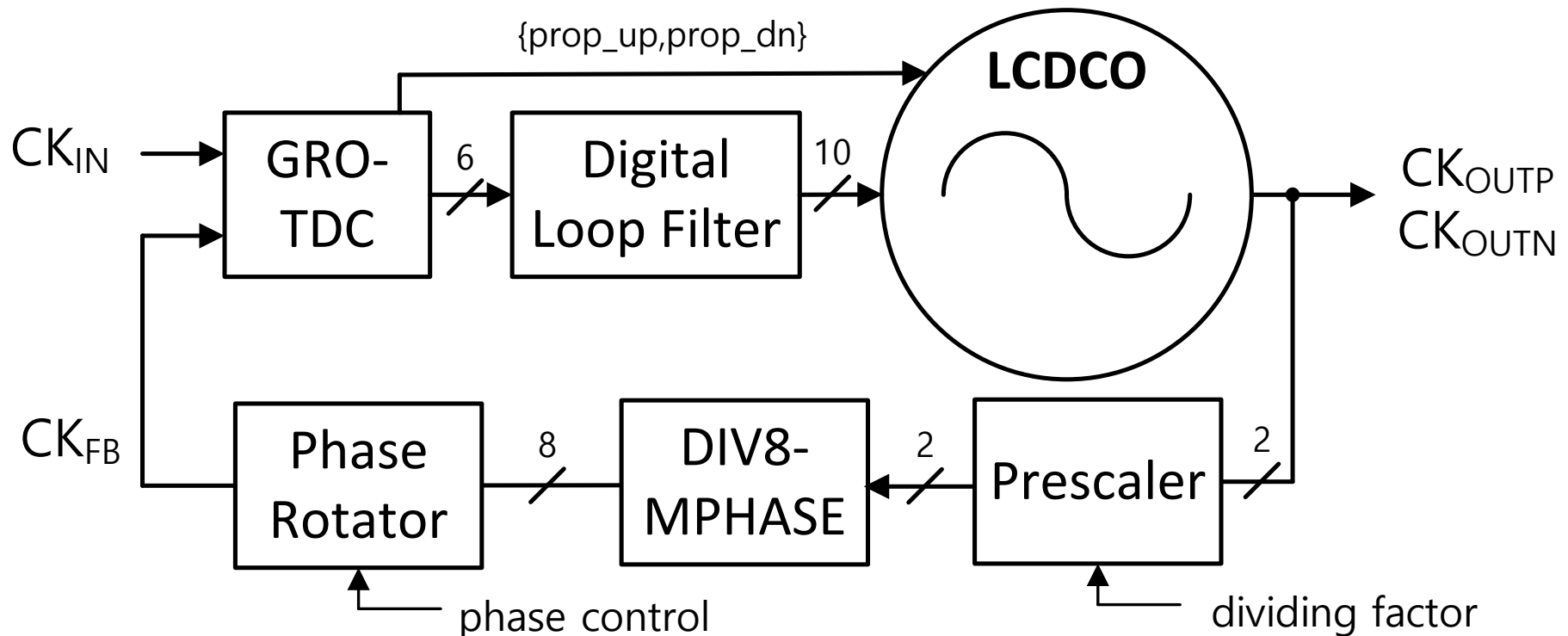
PLL Design and Simulation

Jaeha Kim, Sung-Jun Lee, and Eunseo Kim
Mixed-Signal IC and System Group
Seoul National University
May 19. 2016

PLL Design (Loop Filter Design)

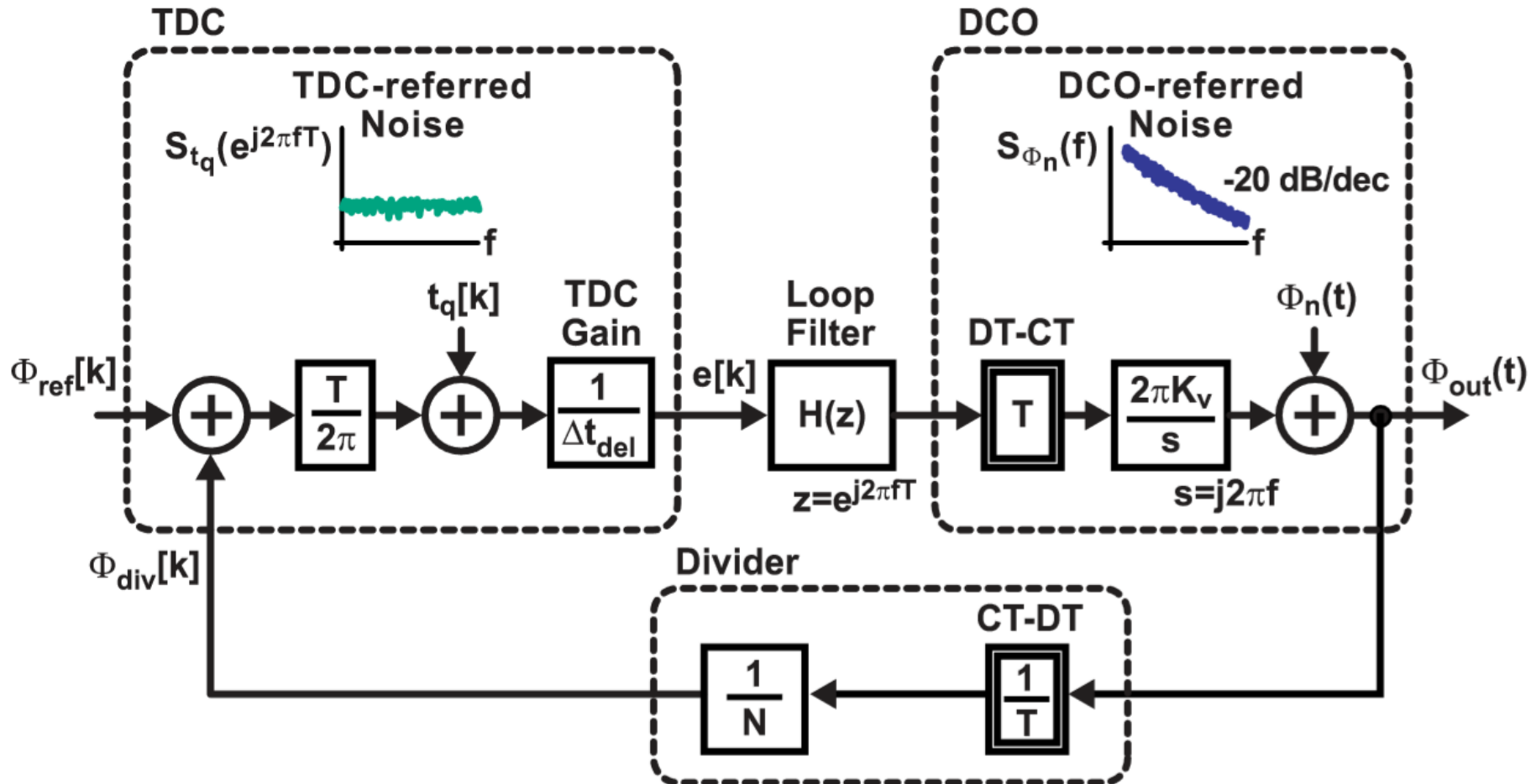
What We Gonna Make is...

- Frequency synthesizer with phase modulator



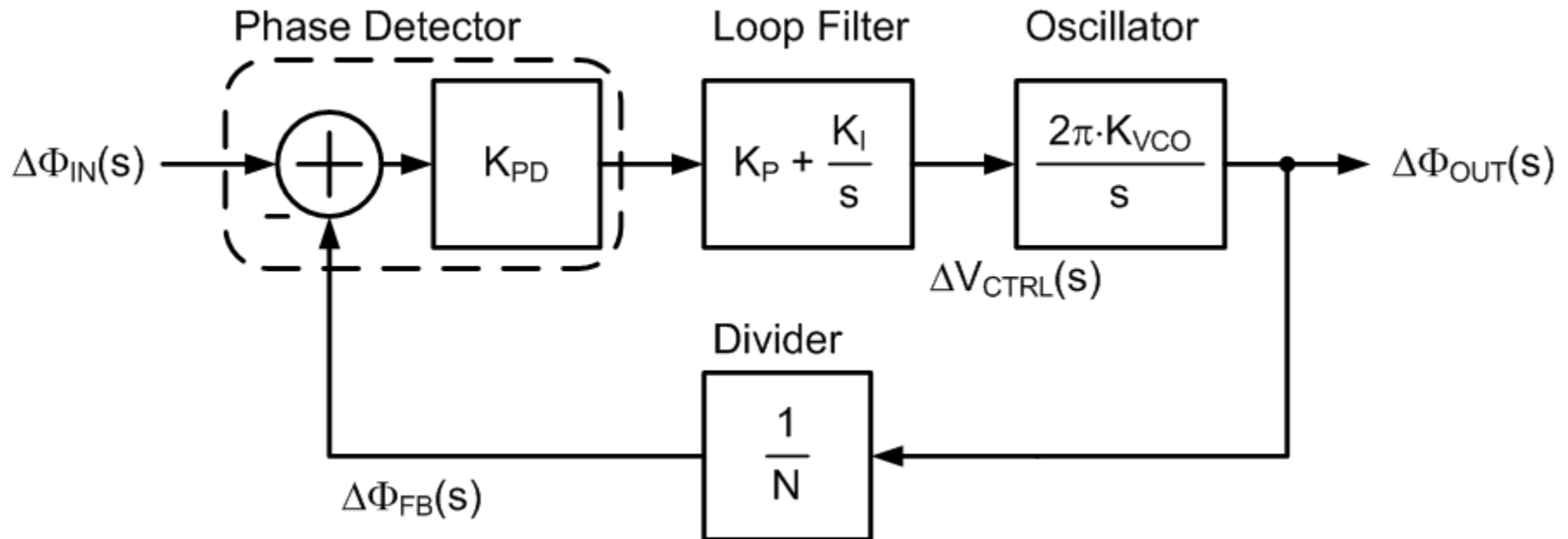
Digital PLL Model

- TDC-referred noise and DCO-referred noise



Loop Transfer Analysis

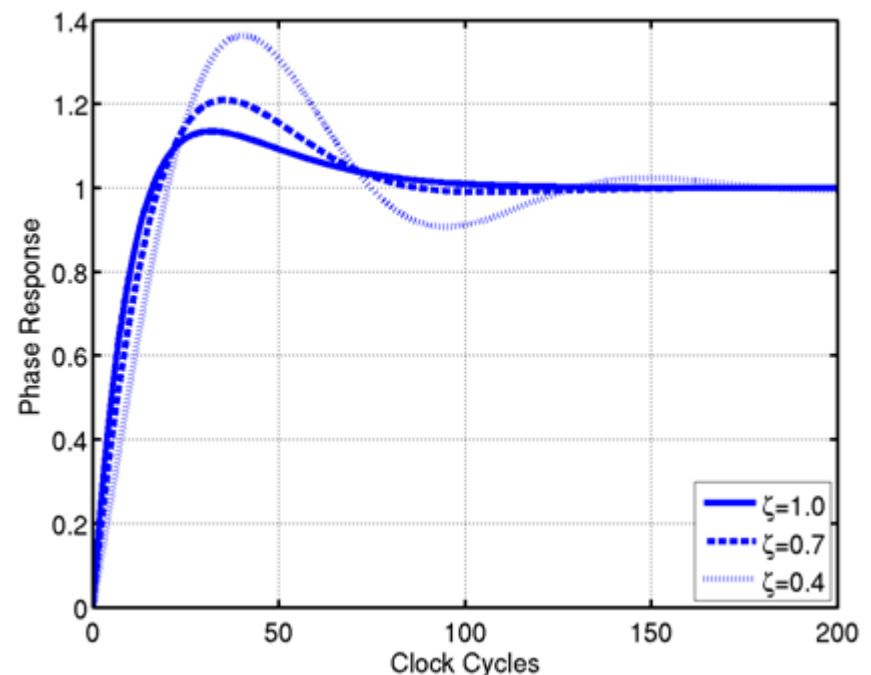
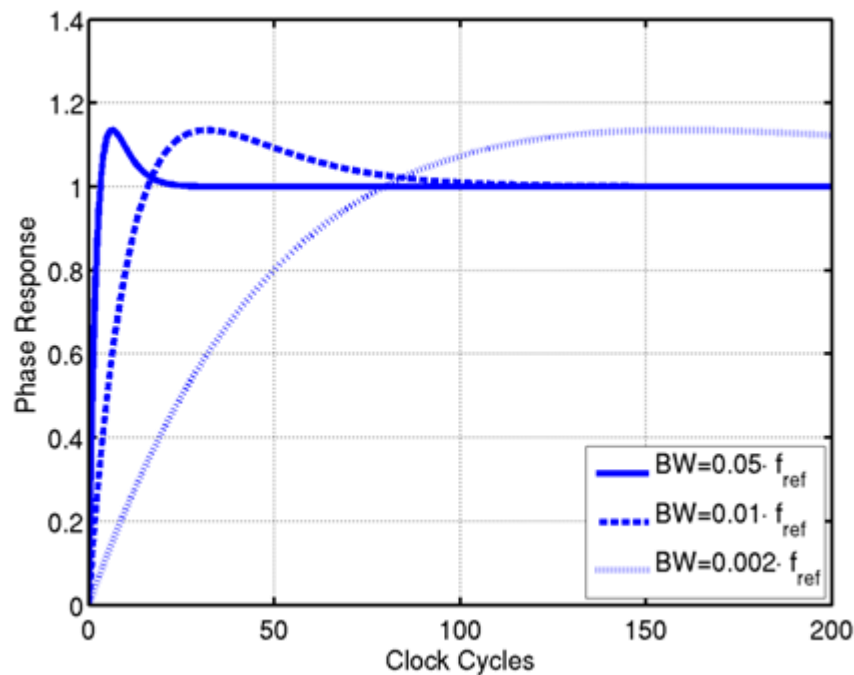
- Approximate s-domain continuous-time model



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

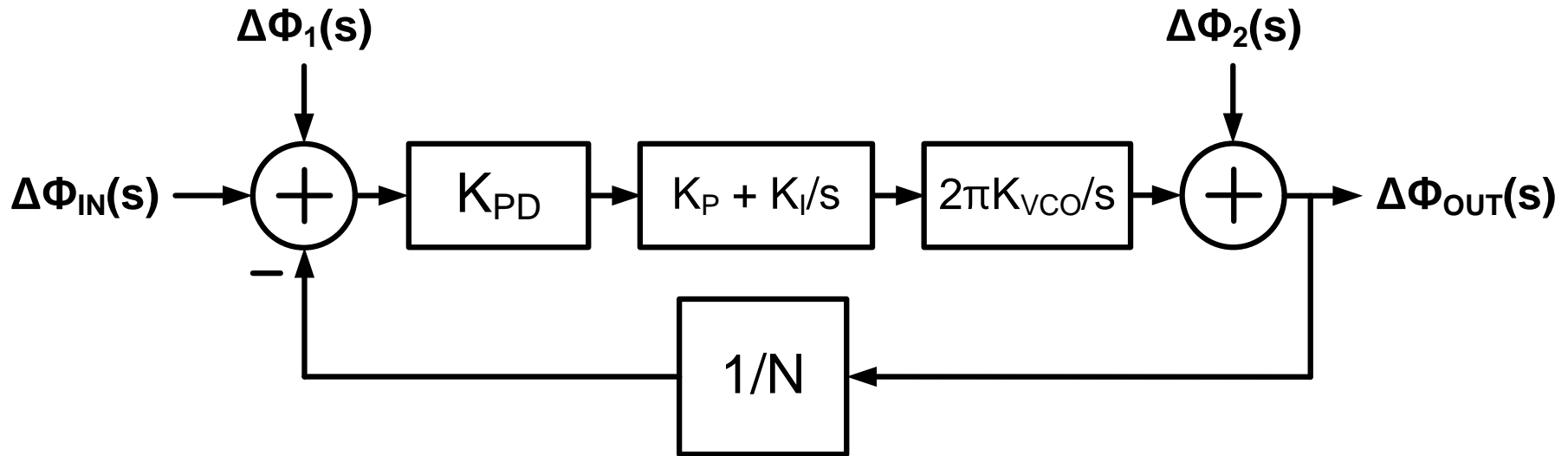
PLL Loop Dynamics

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



Phase Step Responses

Loop Transfer Analysis (2)



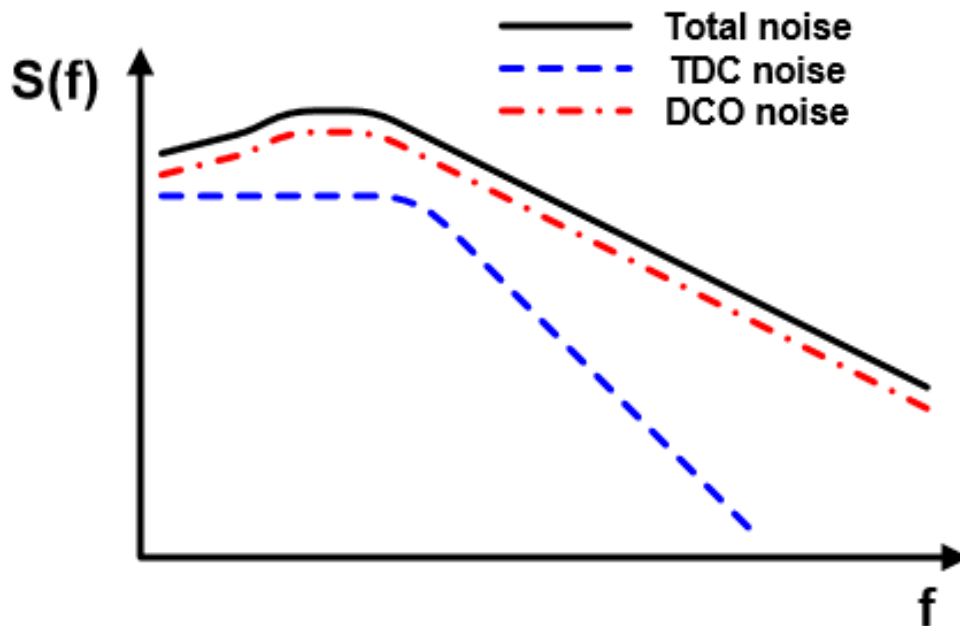
$$H_1(s) = \frac{\Delta\Phi_{OUT}}{\Delta\Phi_1} = N \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

*Low-pass filtered
input reference/TDC noise*

$$H_2(s) = \frac{\Delta\Phi_{OUT}}{\Delta\Phi_2} = N \frac{s^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

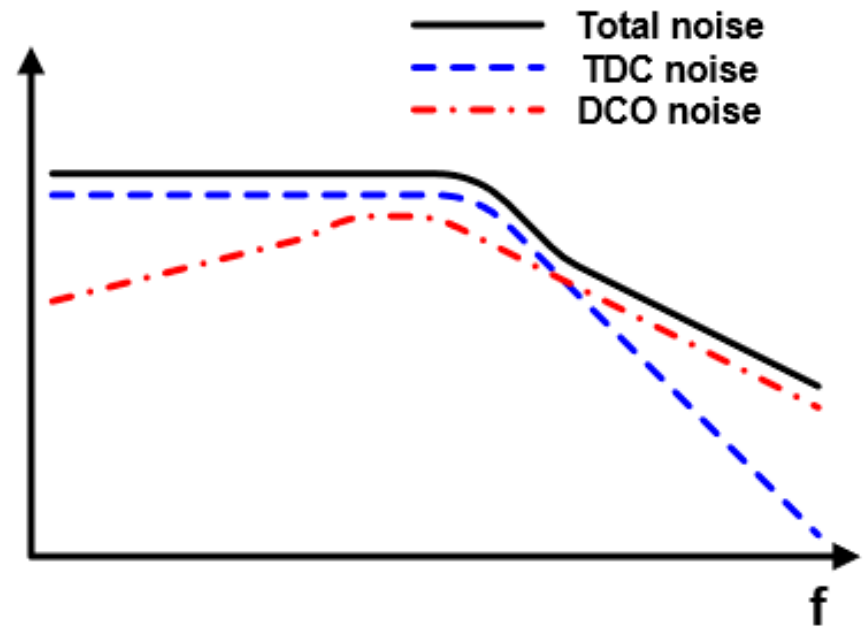
*High-pass filtered
DCO noise*

Noise vs. PLL Bandwidth



Low bandwidth PLL

DCO noise dominant

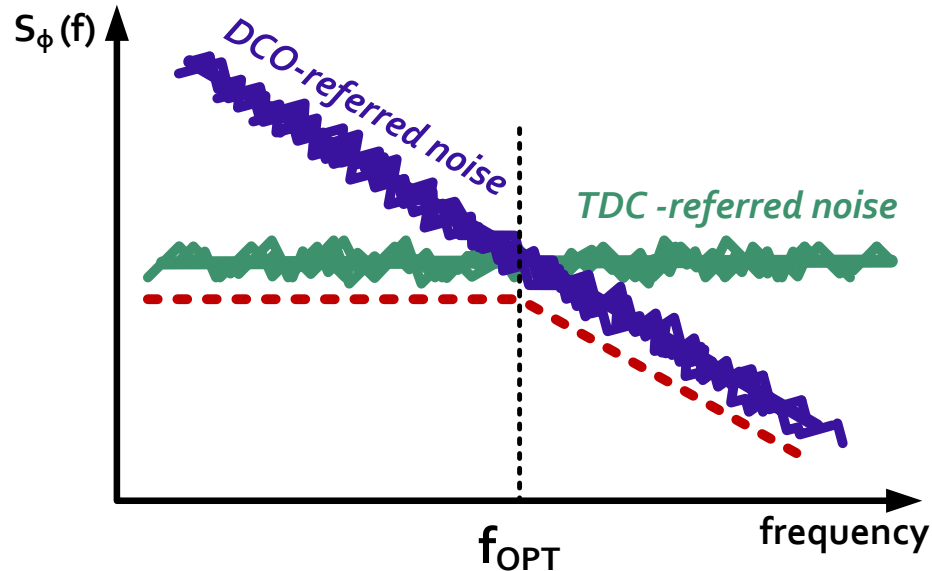


High bandwidth PLL

TDC noise dominant

PLL Bandwidth Tradeoffs

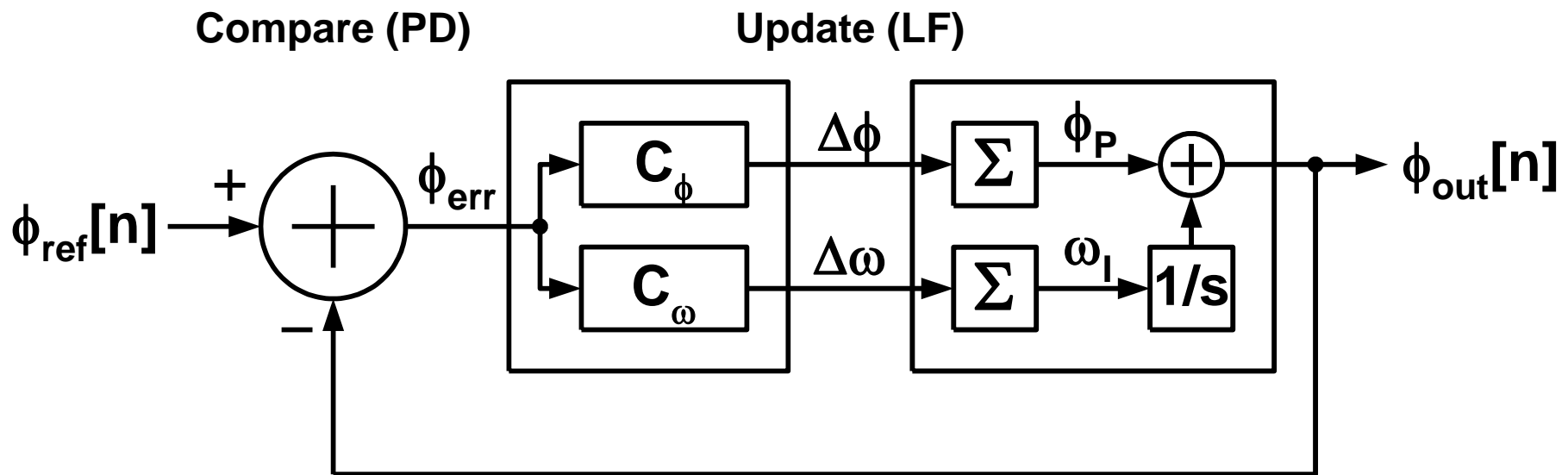
- ▶ With jitter perspective, there's optimal bandwidth we can choose



Type	High BW	Low BW
Input reference / TDC noise rejection		O
VCO noise rejection	O	
Fast acquisition	O	
Jitter integration reduction	O	

Alternate PLL Model

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



$$\frac{\omega_n}{\omega_{ref}} = \sqrt{\frac{\Delta\omega}{2\pi\omega_{ref}\phi_{err}}}$$

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

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,

$$\frac{\omega_n}{\omega_{ref}} = \frac{\sqrt{K_I K_{DCO} K_{TDC}}}{2\pi}$$

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

*Now, your loop filter determines
these design parameters*

Exercise: Digital PLL

- ▶ Consider a 8-GHz PLL with $N=64$, $K_{TDC} = 64$ steps/UI, and $K_{DCO} = 0.0001$; assuming that $K_I=1$ and determine ω_n/ω_{ref} and K_P that yields $\zeta=0.7$

Exercise: Digital PLL

- ▶ Consider a 8-GHz PLL with $N=64$, $K_{TDC} = 64$ steps/UI, and $K_{DCO} = 0.0001$; assuming that $K_I=1$ and determine ω_n/ω_{ref} and K_P that yields $\zeta=0.7$

- ▶ Answer:

$$\frac{\omega_n}{\omega_{ref}} = \frac{\sqrt{K_I K_{DCO} K_{TDC}}}{2\pi} = \frac{\sqrt{1 \cdot 0.0001 \cdot 64}}{2\pi} \approx 0.013 \left(\doteq \frac{1}{80} \right)$$

$$K_P = \frac{4\pi\zeta \omega_n/\omega_{ref}}{K_{DCO} K_{TDC}} = \frac{4\pi \cdot 0.7 \cdot 0.013}{0.0001 \cdot 64} \approx 17.8$$

- ▶ I would choose K_P of 16 (why?)

Digital Loop Filter Model (1)

► dpll:digital_lf:xmodel

```
assign in = {1'bo,up} - {1'bo,dn};
```

```
assign in_ext = {{3{in[6]}}, in};
```

sign extension

```
always @(posedge clk or negedge reset_n) begin
```

```
    if (!reset_n) begin
```

```
        out <= init_value;
```

```
        dsm_out <= 3'bo;
```

initialization

```
    end
```

```
    else begin
```

```
        out <= out + in_ext;
```

```
        dsm_out <= 3'bo;
```

Integrator

```
    end
```

```
end
```

Where's proportional term (K_p)?

Digital Loop Filter Model (2)

- ▶ Since we have *direct* proportional path, setting 'pgain_ctrl'(in dco) is enough
- ▶ We assumed $K_P = 16$ with $K_{TDC} = 64$
- ▶ However, K_{PFD} is ideally 1
- ▶ Therefore, pgain_ctrl should be set to 3'b111
 - ▶ (proportional gain is $1024 = 16 * 64 = K_P * K_{TDC}$)
- ▶ You need 'clk_lf_dco' output

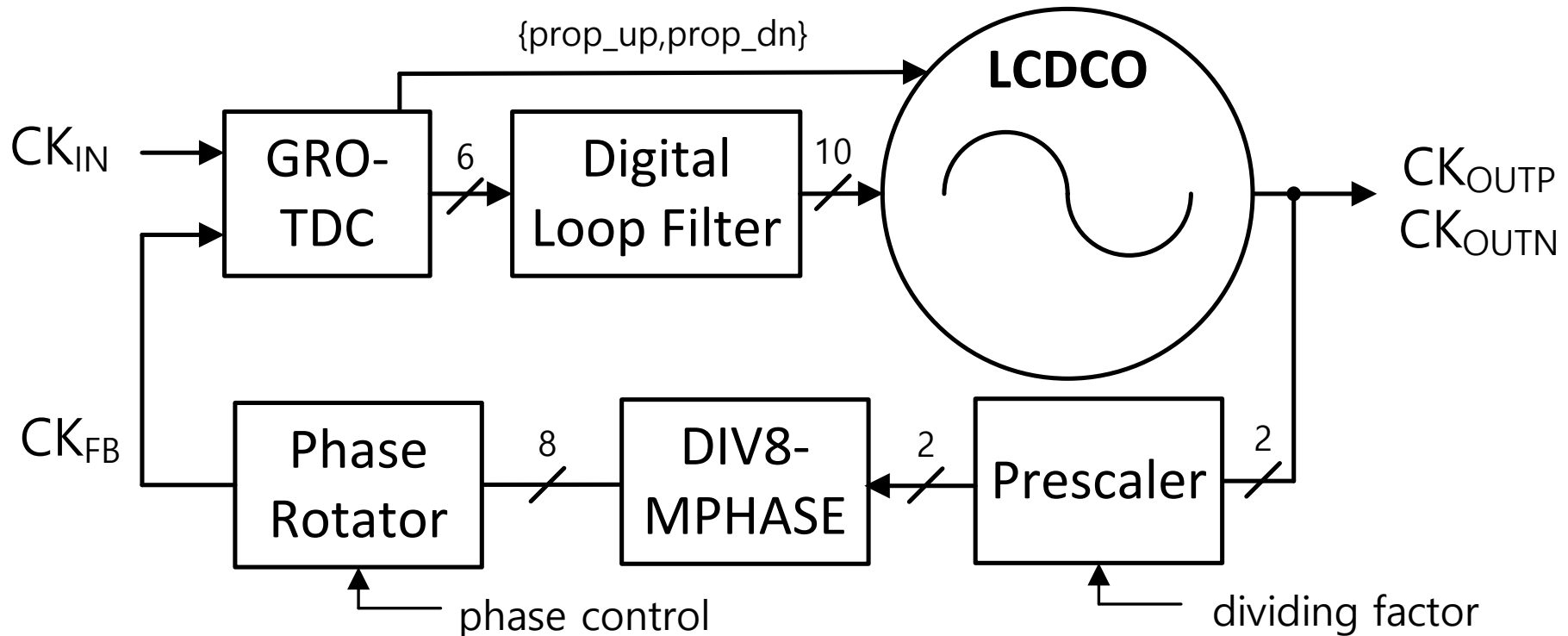
```
always @(clk) begin  
    clk_lf_dco = clk;  
end
```

Loop filter clock to
drive dco's clk_lf

PLL Simulation

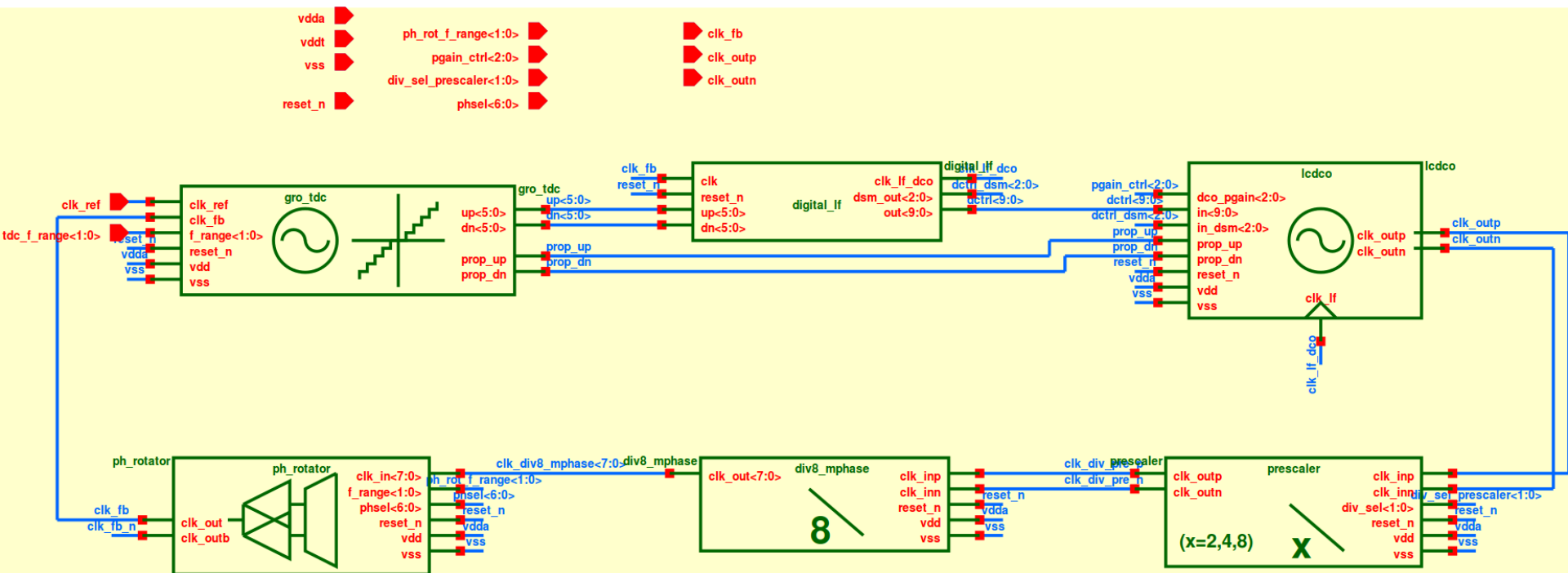
Exercise#2

- ▶ Connects the building blocks
 - ▶ Make your own digital PLL!
- ▶ `dpll:dpll:schematic`



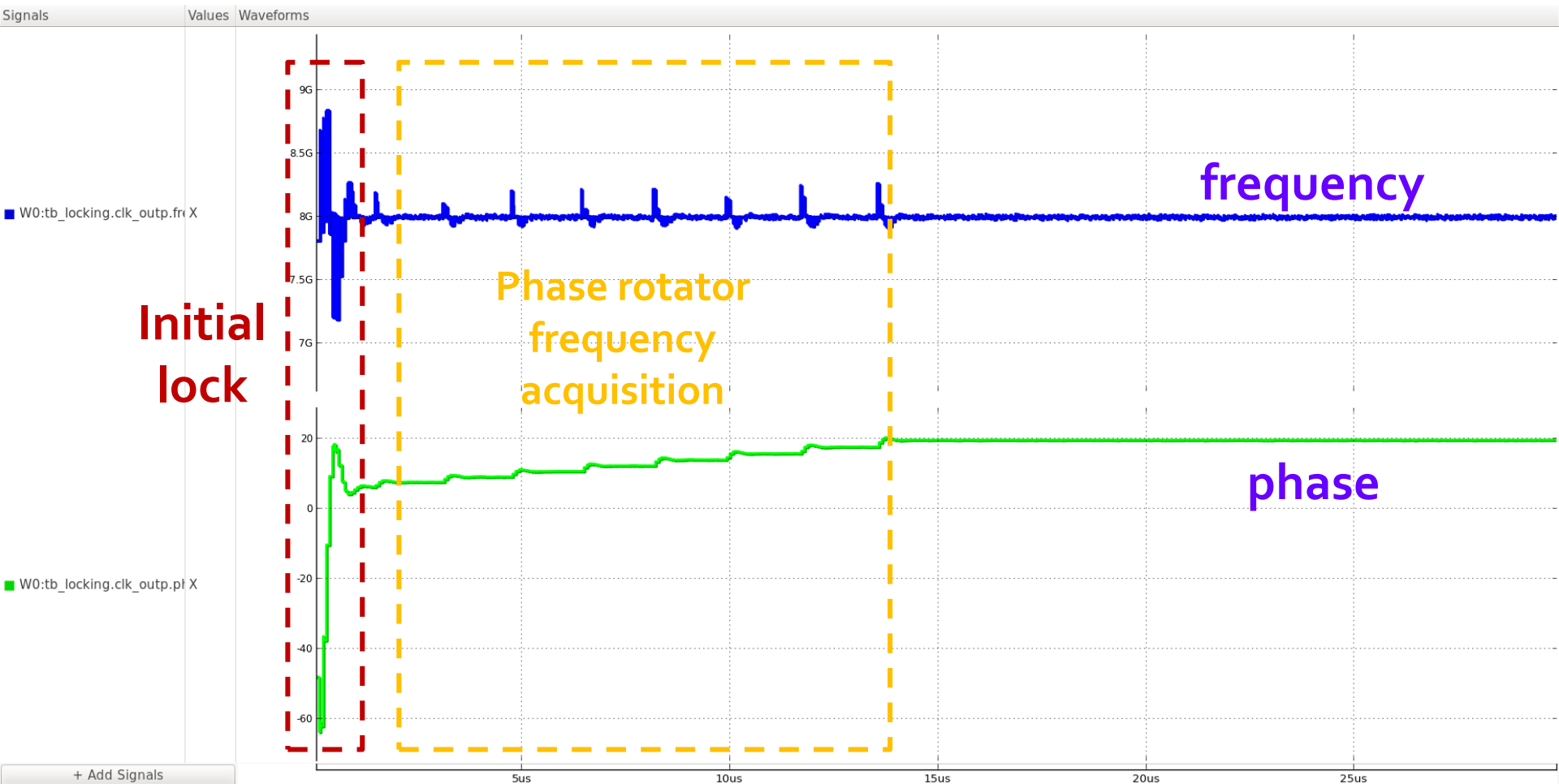
Digital PLL Model

► Completed connection example



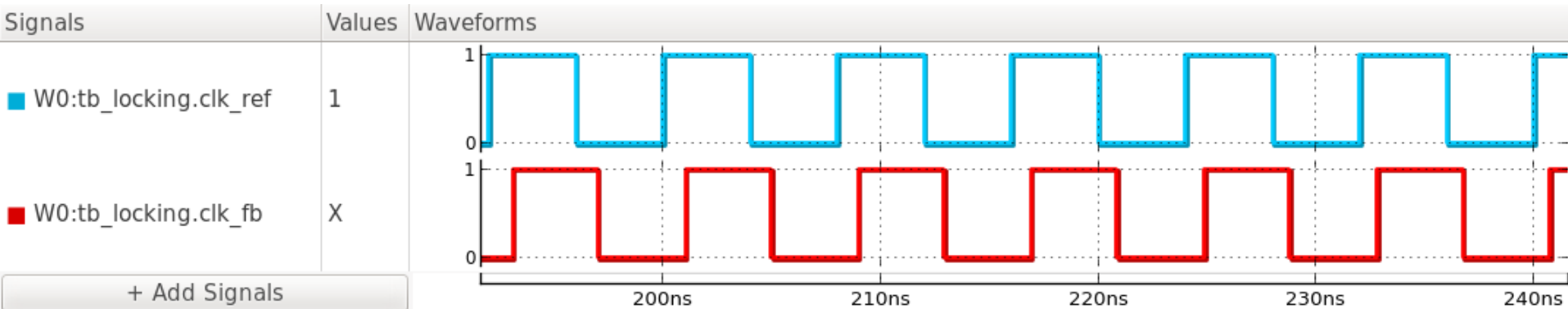
Exercise: Locking Transient

► dpll/dpll/tb_locking:

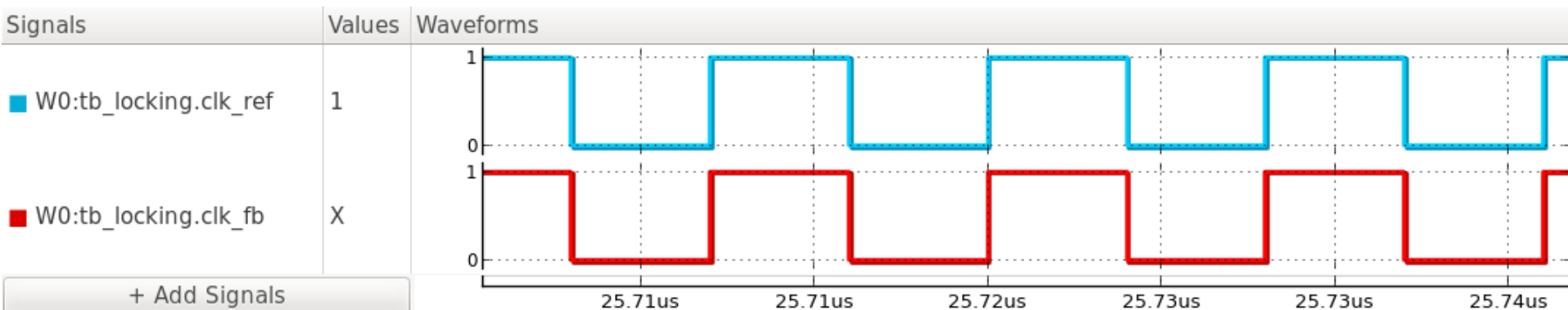


Results

► Before lock

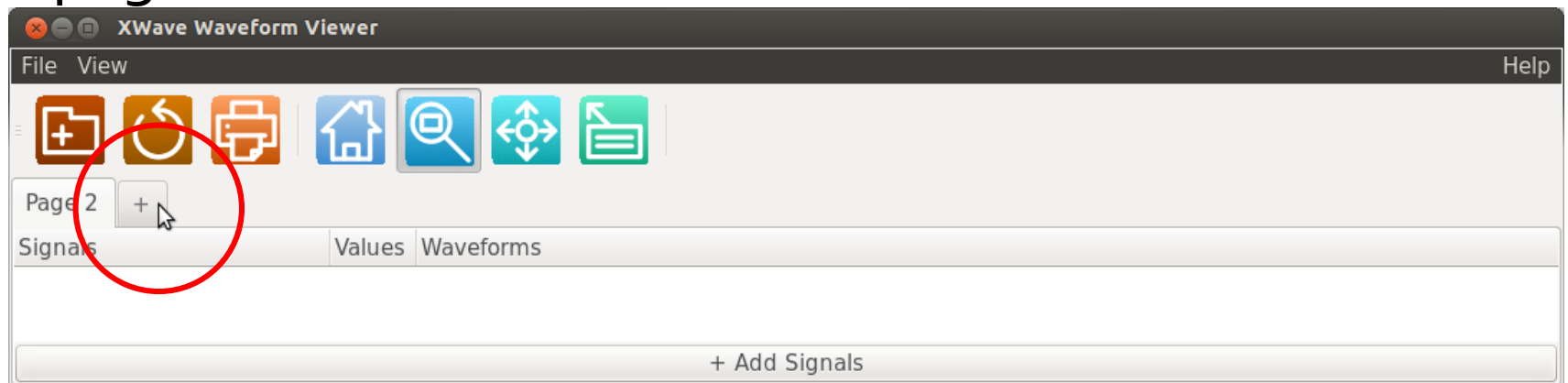


► After lock



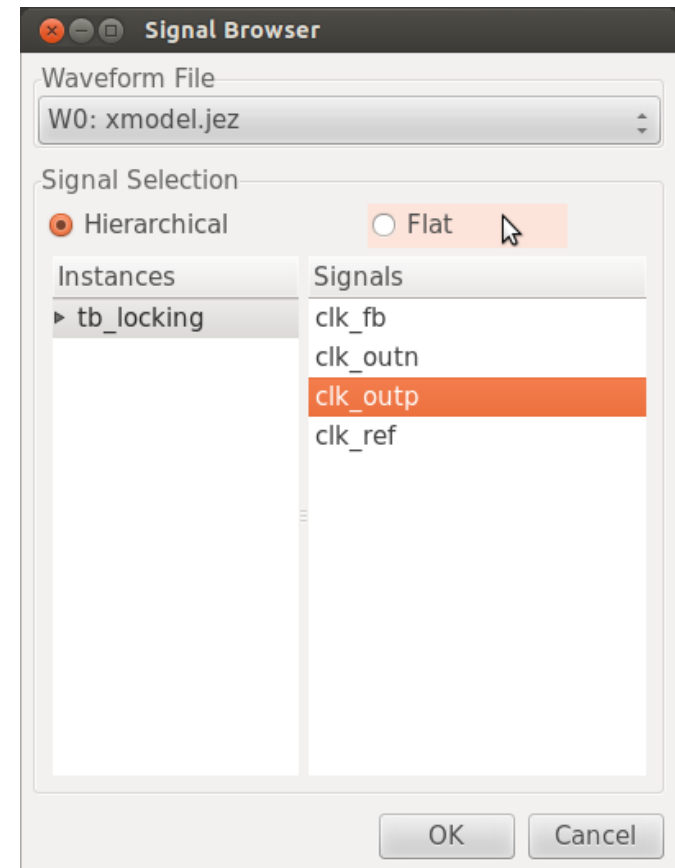
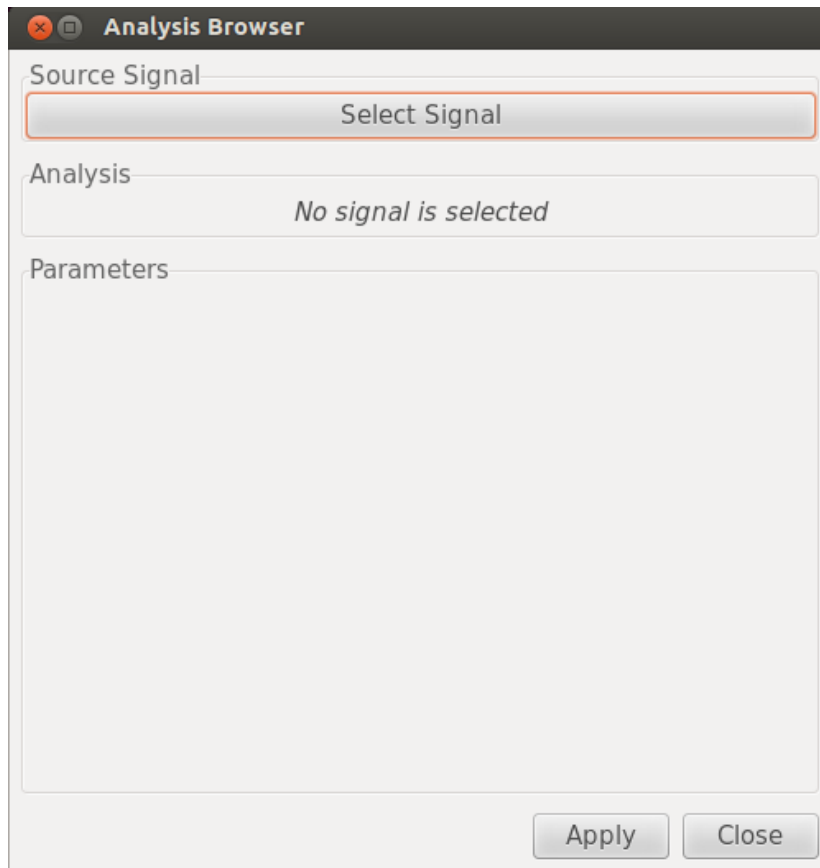
Jitter Histogram Simulation

- ▶ Plotting the jitter histogram is one way to evaluate the PLL noise performance
- ▶ You can evaluate jitter performance in this 'tb_locking' testbench
- ▶ First, remove the waveform page and open analysis page



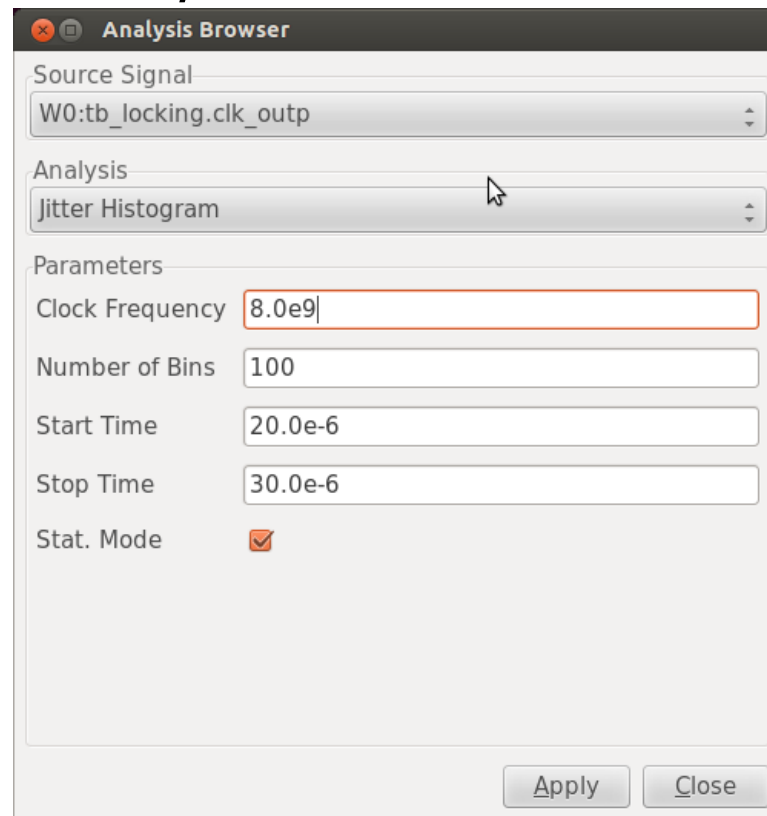
Jitter Histogram Simulation (2)

- ▶ Second, click '+ Add Analysis' button, and select the signal that you want to analyze



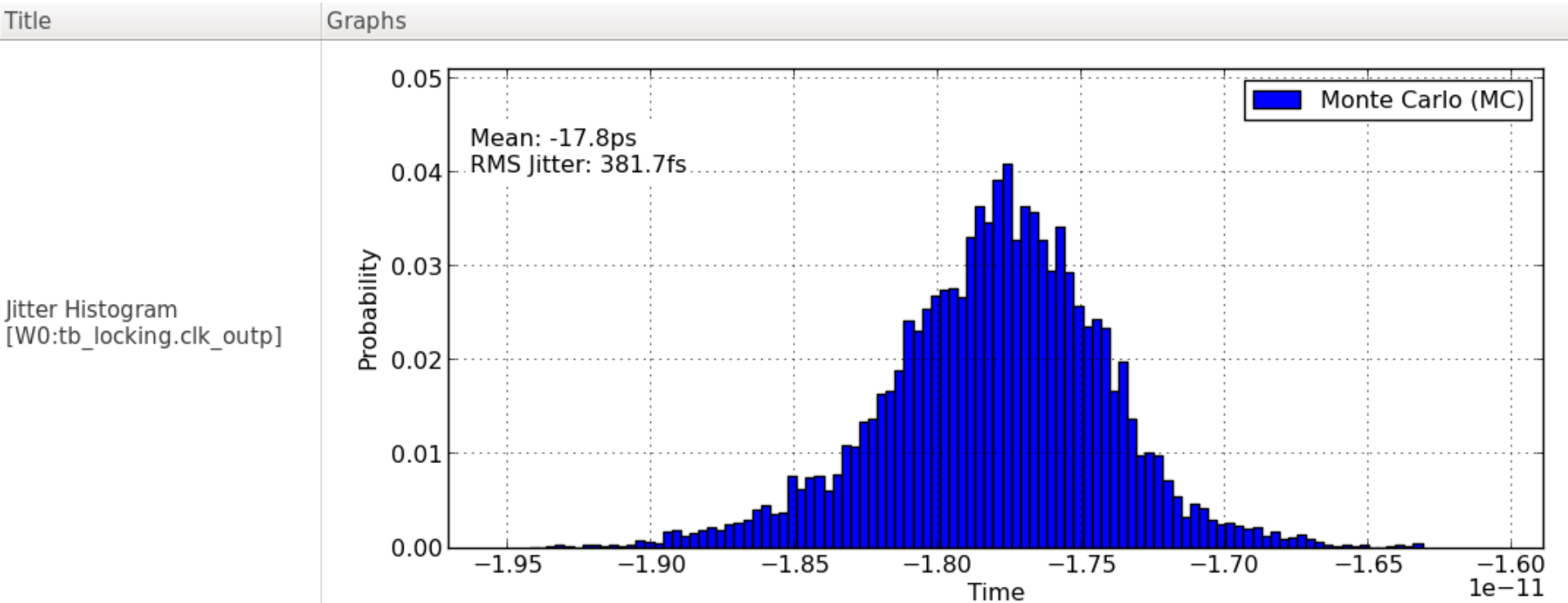
Jitter Histogram Simulation (3)

- ▶ You can set the parameters to properly plot a jitter histogram
- ▶ frequency = 8 GHz, time interval = 20 ~30 us



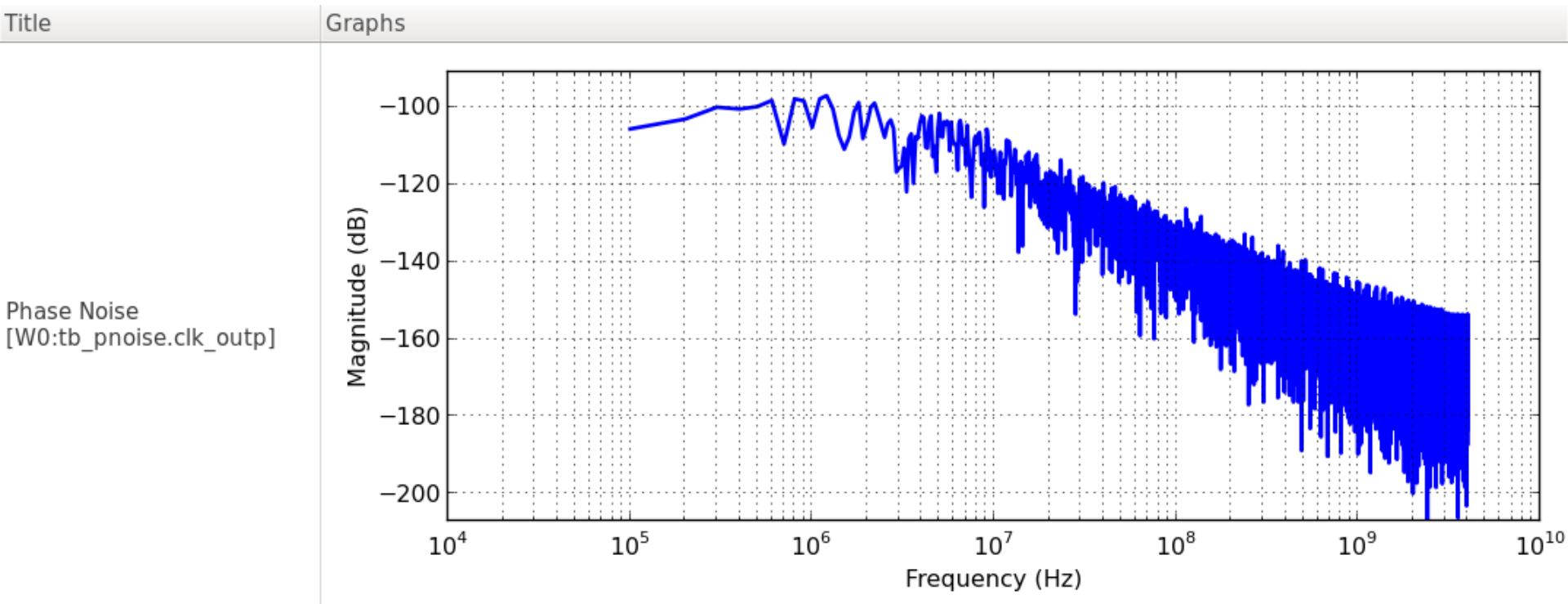
Result

- ▶ Resulted RMS jitter = 381.7 fs



Phase Noise Simulation

- ▶ You can also plot phase noise graph!
- ▶ Directory: `dp11/dp11/tb_pnoise`

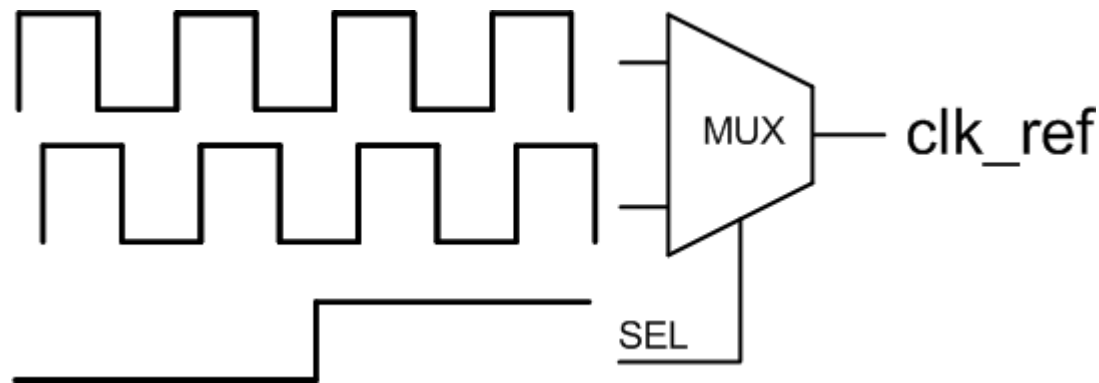


Verifying PLL Behaviors

- ▶ 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

Generating the Phase Step

- ▶ How can we generate a step change in the input clock phase?
- ▶ One possible way is to use a mux....



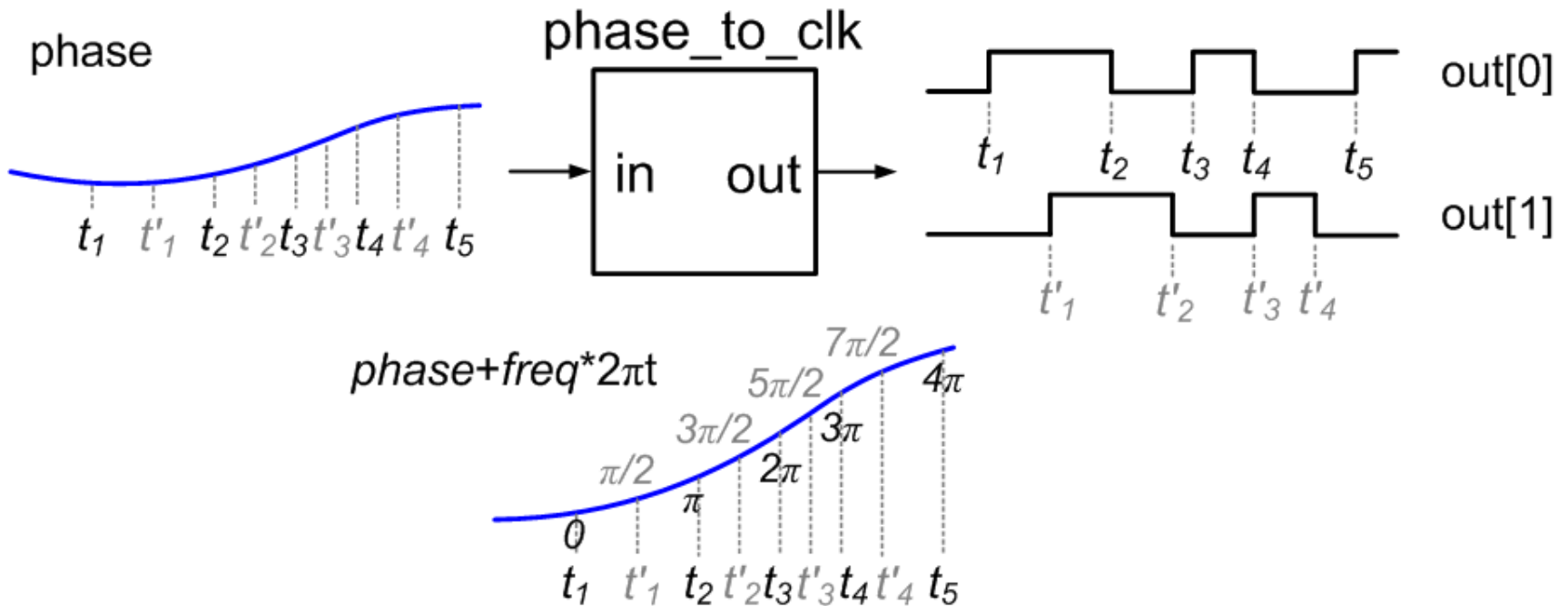
- ▶ The easier way is to use ***variable domain transformation (VDT)*** primitives in XMODEL

XMODEL Primitives: VDT

- ▶ Primitives for variable domain transformation
 - ▶ clk_to_delay
 - ▶ clk_to_duty
 - ▶ clk_to_freq
 - ▶ clk_to_period
 - ▶ clk_to_phase
 - ▶ delay_to_clk
 - ▶ duty_to_clk
 - ▶ freq_to_clk
 - ▶ phase_to_clk
 - ▶ period_to_clk

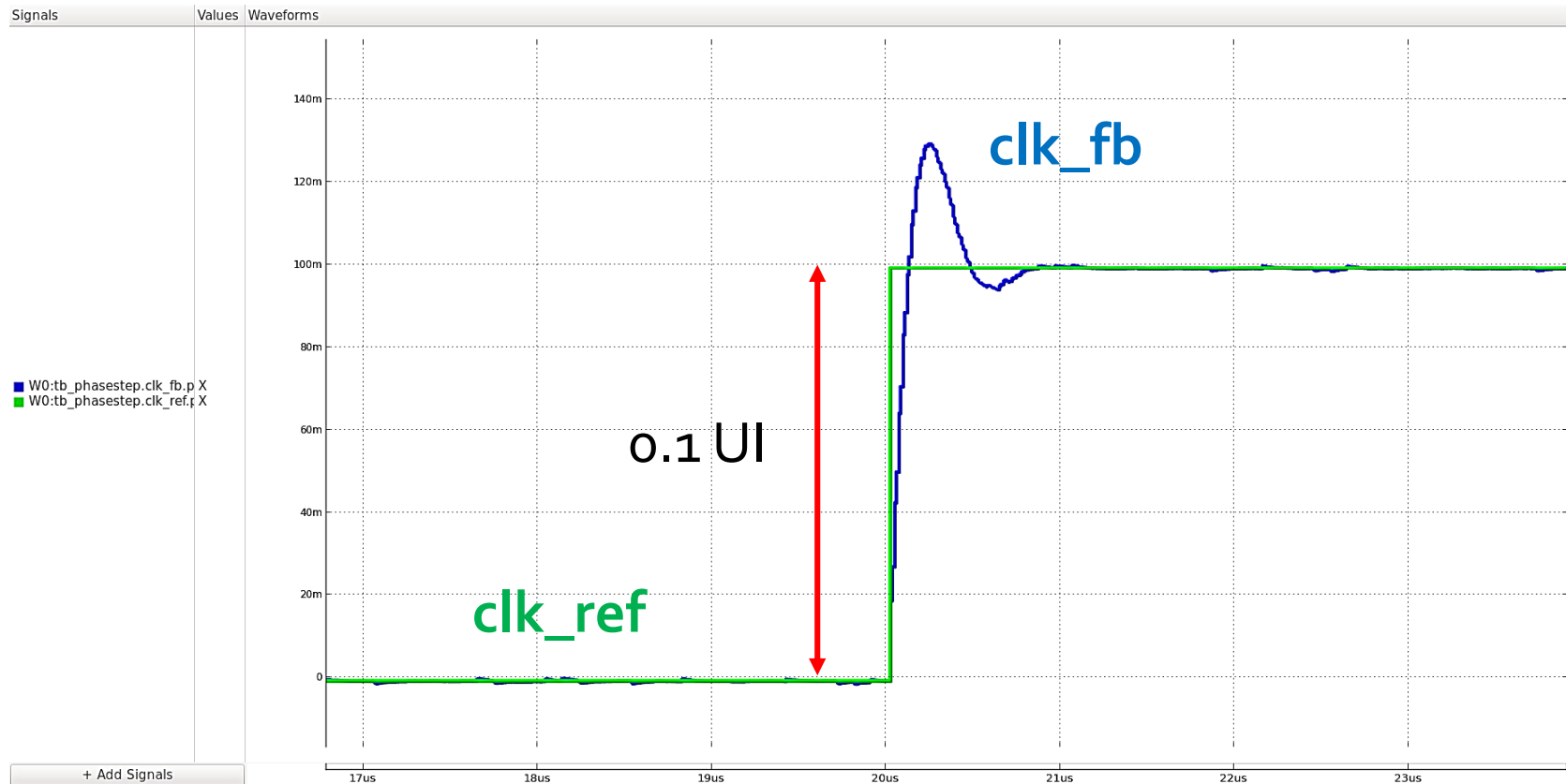
phase_to_clk Primitive

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



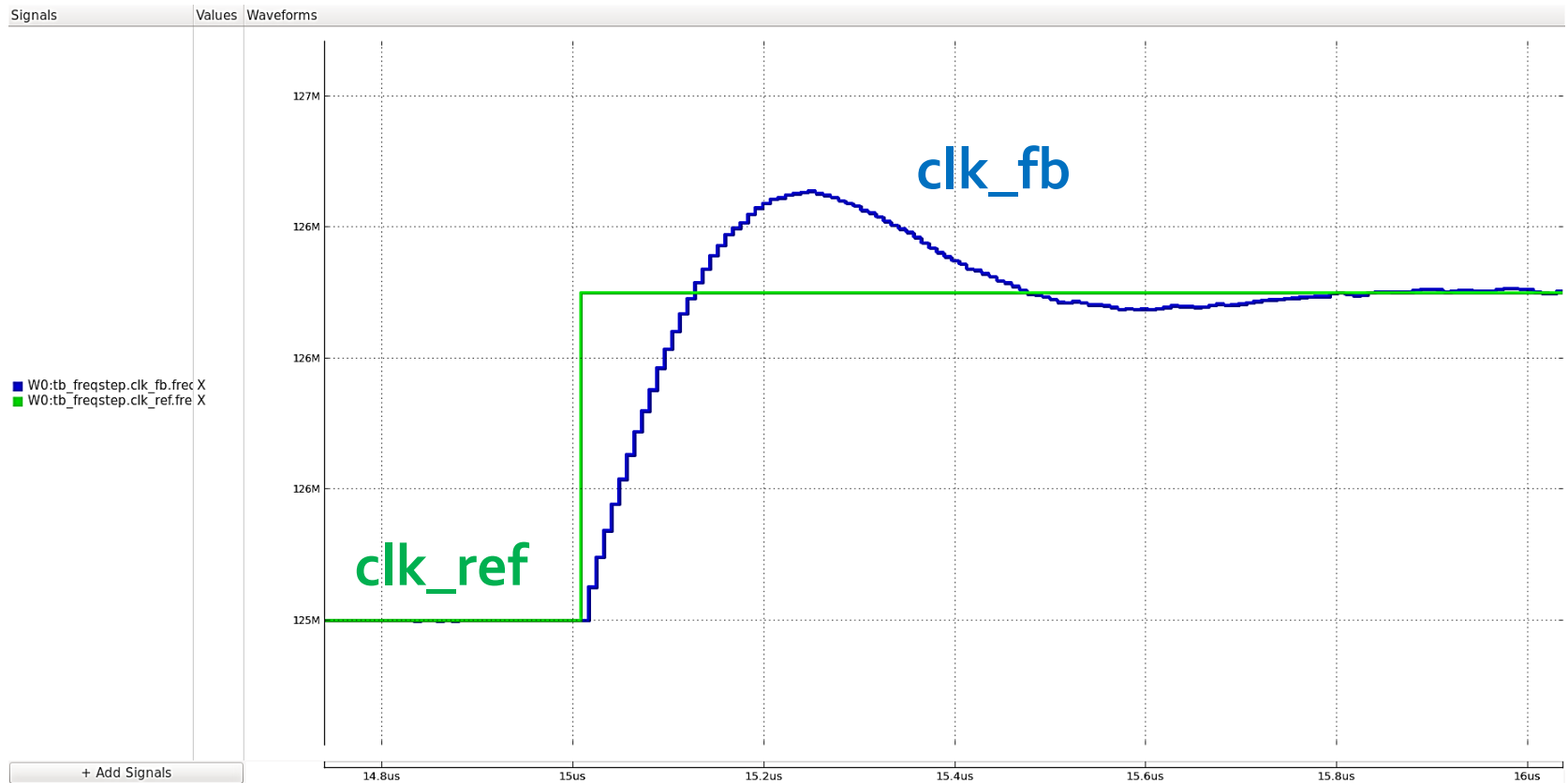
Phase Step Response in PLL

► dpll/dpll/tb_phasestep



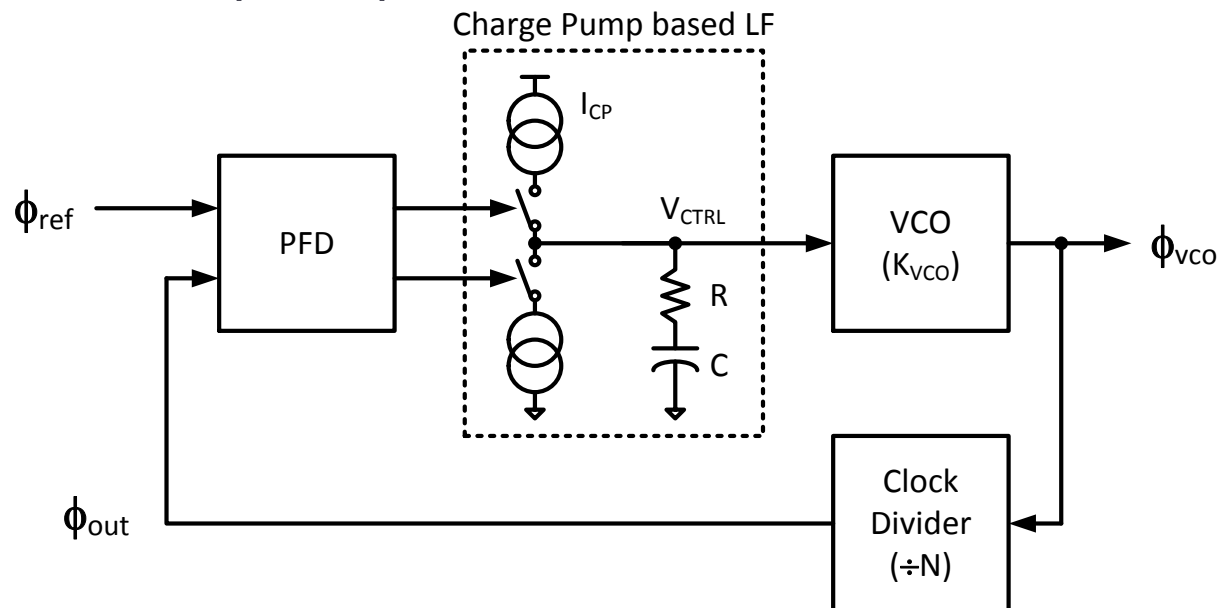
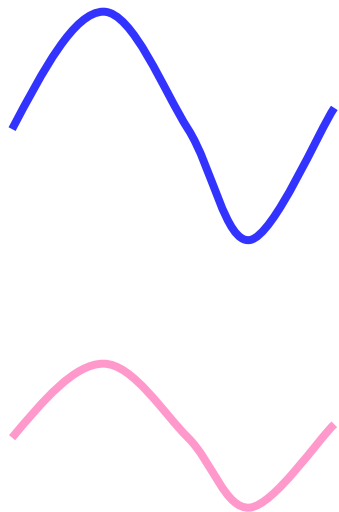
Frequency Step Response in PLL

► dpll/dpll/tb_freqstep



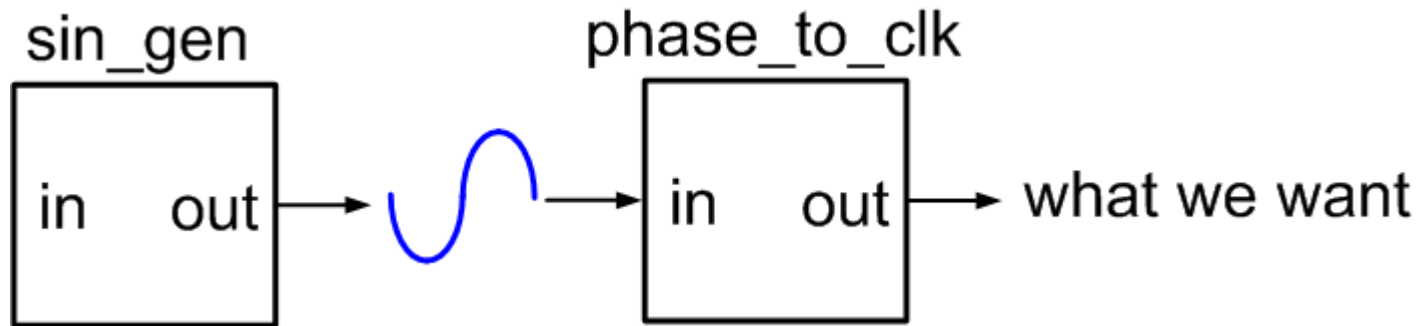
Measuring PLL Jitter Transfer Function

- ▶ More effective way to validate the loop dynamics is to measure its phase-domain 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*

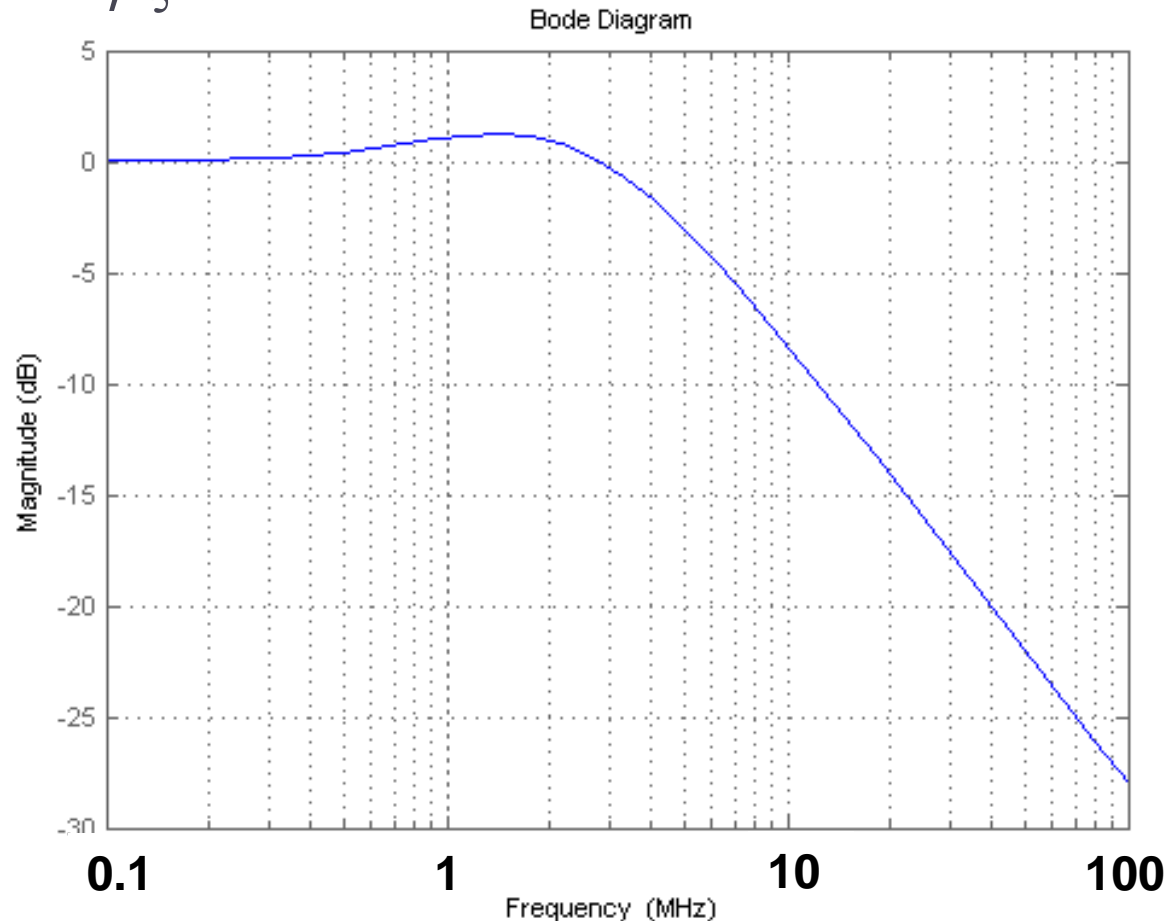


- ▶ Also, the *clk_gen* primitive in XMODEL can also generate a clock with sinusoidal jitter
 - ▶ Use *SJ_amp* and *SJ_freq* parameters to set its amplitude and frequency

Measuring PLL Jitter Transfer Function

- ▶ Expected TF from the s-domain model

- ▶ $\omega_n = 2\pi \times 1.6 \text{ MHz}$, $\zeta = 1.0$

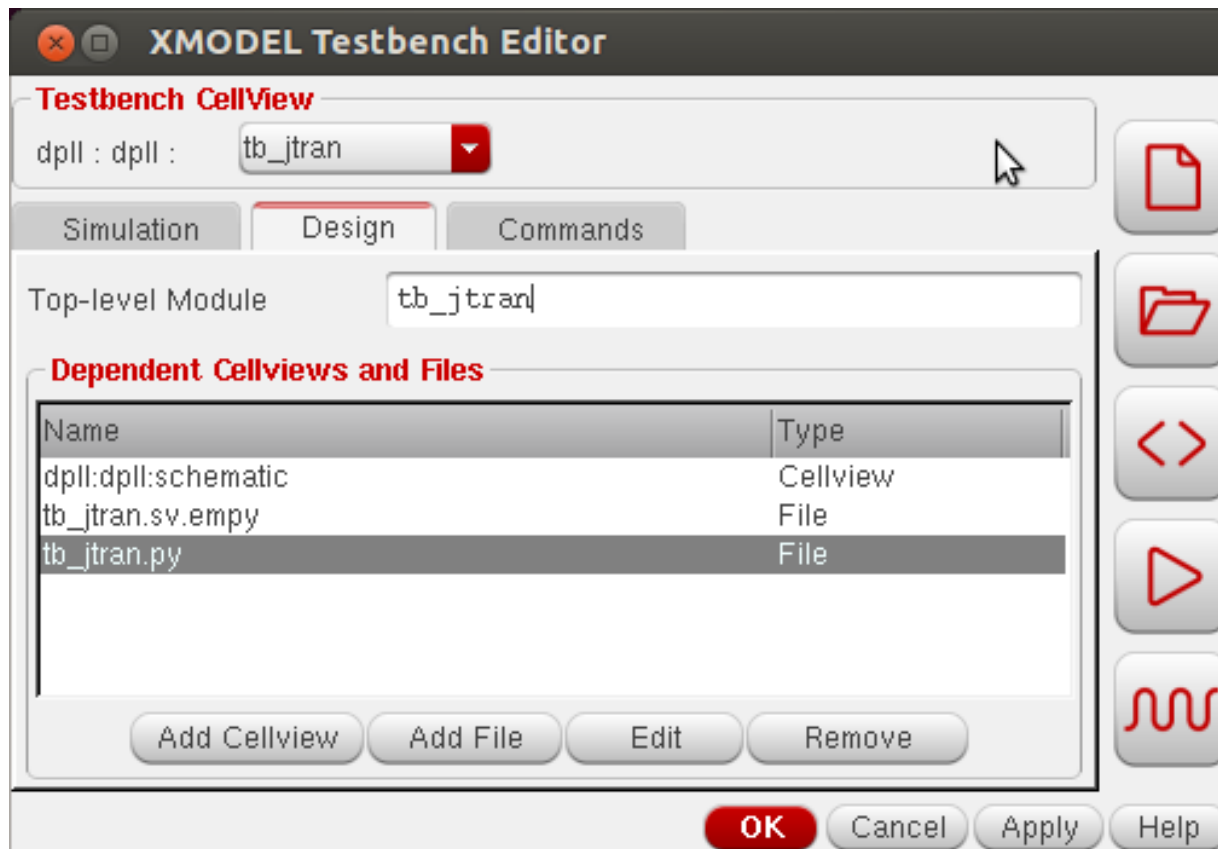


Exercise

- ▶ 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”

Files to Simulate Jtran

- ▶ dpll/dpll/tb_jtran
 - ▶ Comprised of tb_jtran.py, tb_jtran.sv.empty



Answers

- ▶ See tb_jtran.sv.empty and jtran.py located in dpll/dpll/tb_jtran directory
- ▶ tb_jtran.sv.empty is a template testbench in empty:

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 sinusoidal jitter amplitude in UI
parameter real SJ_freq = @sj_freq; // input sinusoidal jitter frequency in Hz
parameter real t_lock = @t_lock; // initial simulation time (PLL locking time)
...
```

// reference clock generator with sinusoidal jitter

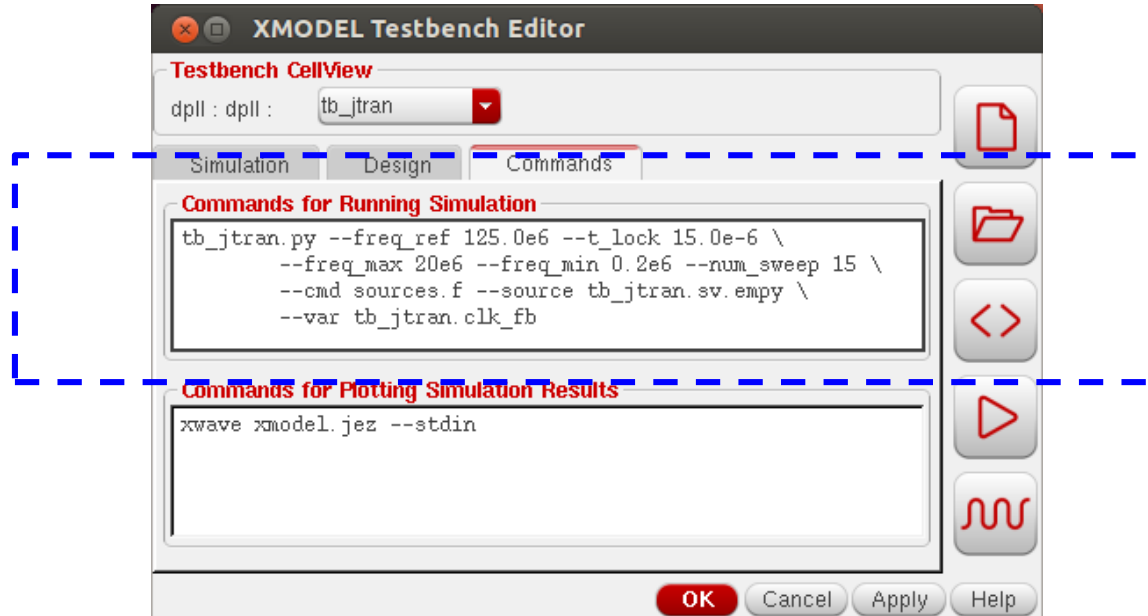
```
clk_gen #(.freq(freq_ref), .SJ_amp(SJ_amp), .SJ_freq(SJ_freq))
        clk_gen(clk_ref);
```

// probing phase of clk_out

```
probe_phase #(.freq(freq_ref), .start(t_lock)) probe_phase(clk_out);
```

Exercise : Jitter Transfer Curve

► Commands



► Available options are :

- --freq_ref : input reference frequency
- --t_lock : estimated locking time
- --freq_max, --freq_min : max. and min. sweep frequency

Results

