In this problem set you will assemble a PLL modelled in Verilog-A in the Cadence Analog Artist environment and simulate it in Spectre. Don’t worry; I’m going to give you the pieces you need but you will have to assemble them and figure out how they work. Spectre allows you to mix Verilog-A behavioral and circuit elements (such as transistors, resistors, and capacitors).

Your PLL will take a 250 MHz square wave input and synthesize a 1GHz square wave output. First, we will assemble the components of the PLL.

- **Phase detector and charge pump.** Create a symbol for this component (pfd_cp) with inputs clkout and refclk and output cpout. Now, create a veriloga view for this component. (When you first create a veriloga view, Cadence brings up the “Model Creator” tools. Please cancel this since we will be just entering the Verilog-A with the text editor.) The Verilog-A behavioral for the PFD/CP is given below (yes, I’m going to make you type it in so that you try to digest it as you type):

```verilog-
module pfd_cp(cpout, clkout, refclk);

// These are standard includes that define the
// electrical discipline and constants, such as `M_PI
`include "constants.h"
`include "discipline.h"

module pfd_cp(cpout, clkout, refclk);

// refclk and clkout are the reference
// and feedback clocks, respectively
input refclk, clkout;
// cpout is the output current of the charge pump
output cpout;

electrical refclk, clkout, cpout;
```
parameter real vdd=2.5;
// charge pump current
parameter real iout=10u;

// transition time for the charge pump current
parameter real trf=200p from (0:inf);
parameter real td=1p from (0:inf);
parameter real ttol=1p from (0:inf);
parameter integer dir=1;

integer state;

analog begin

// This is a simple implementation of the
// three-state sequential phase detector.

// The cross operator generates events
// whenever the expression argument (in this case,
// the voltage on refclk) crosses zero
// with the tolerance ttol.
@(cross(V(refclk)-vdd/2, dir, ttol)) begin
  if (state >-1) state=state-1;
end

@(cross(V(clkout)-vdd/2, dir, ttol)) begin
  if (state <1) state=state+1;
end

// Set the output current of the charge pump
I(cpout) <+ transition (iout*state, td, trf);

end

endmodule

- **Voltage controlled oscillator.** Your VCO component vco will have one input vctrl1 and one output vco_out. The VCO has a range of 250
MHz to 2 GHz for \textit{vctrl} values ranging from 0 to 2.5 V. The Verilog-A behavioral for the VCO is given as follows:

`include "constants.h"
`include "discipline.h"

module vco(vctrl, vco_out);
output vco_out;
input vctrl;
electrical vco_out, vctrl;

// vmin and vmax are the minimum and maximum
// values of \textit{vctrl}, respectively
parameter real vmin=0.0;
parameter real vmax=2.5 from (vmin:inf);
// fmin and fmax are the minimum and maximum
// values of the VCO output frequency
parameter real fmin=250e6 from (vmin:inf);
parameter real fmax=2e9 from (fmin:inf);
// vdd is the supply voltage. The VCO output
// goes between 0 and vdd.
parameter real vdd=2.5;
// tt is the transition time of the VCO output
parameter real tt=0.01/fmax from (0:inf);
// jitter is the rms jitter introduced in the
// VCO, say by substrate or power-supply noise
parameter real jitter=0 from [0:0.25/fmax);
parameter real ttol=1e-6/fmax from (0:1/fmax);

real freq, phase, dt;
integer n, seed;

analog begin

@\texttt{(initial\_step)} seed=-556;

// compute frequency from input voltage
freq=(V(vctrl)-vmin)*(fmax-fmin)/(vmax-vmin)+fmin;
// saturation points for the VCO output
if (freq>fmax) freq=fmax;
if (freq<fmin) freq=fmin;

// phase noise adder
freq = freq/(1+dt*freq);

// phase is the integral of the frequency modulo 2PI
phase=2*M_PI*idtmod(freq, 0.0, 1.0, -0.5);

// jitter is updated twice per period
@((cross(phase+'M_PI/2,+1, ttol) or
   cross(phase-'M_PI/2,+1, ttol)) begin
  dt=1.414*jitter*dist_normal(seed, 0, 1);
  n=(phase >= -'M_PI/2) && (phase < 'M_PI/2);
end

V(vco_out) <+ transition(n?vdd:0, 0, tt);

end
endmodule

**Divider.** Your divider component divider will have one input div_in and one output div_out. The Verilog-A behavioral for the divide-by-four is given as follows:

//-- VerilogA for pll, divider, behavioral

'include "constants.h"
'include "discipline.h"

module divider(div_out, div_in);

input div_in;
output div_out;
electrical div_in, div_out;
parameter real vdd=2.5;
// This is the divide-down ratio for the divider
parameter integer ratio=4 from [2:inf);

// dir=1 for positive-edge-triggered latch
// dir=-1 for negative-edge-triggered latch
parameter integer dir=1 from [-1:1] exclude 0;

// transition time for divider output
parameter real tt=100p from (0:inf);
parameter real td=10p from (0:inf);
parameter real ttol=1p from (0:td/5);

integer count,n;
real dt;

analog begin

@ (cross (V(div_in)-vdd/2, dir, ttol)) begin
  count = count+1;
  if (count >= ratio)
    count=0;
  n=(2*count >= ratio);
end

V(div_out) <+ transition(n? vdd: 0, td, tt);
end
endmodule

1. Design a series RC first-order loop filter for your PLL. Size the capacitance so that $\omega_n = 0.1\omega_{ref}$. Size the resistance so that the damping factor is $\sqrt{2}/2$. Simulate your PLL in Spectre with an input square wave of frequency 250MHz with rise/fall times of 200 psec. To do this, bring up the Analog Artist Circuit Design Environment window (as you do when running HSPICE
Choose Setup → Simulatory/Directory/Host. Set the simulator to “spectre” and OK the form. To set up a transient simulation, choose Analyses → Choose and set the stop time for the transient analysis. Please simulate the PLL long enough to verify that the PLL has correctly locked. Please turn in your simulation plots.

2. Now consider the behavior of your PLL when the input frequency changes abruptly from 250 MHz to 167 MHz. Determine how long it takes your PLL to relock to the new frequency; consider the behavior of this transient. How does this behavior change as $R$ is decreased?

3. Change the jitter variable in your vco model to 100 psec to add this rms cycle (or period) jitter to the VCO. Determine the resulting rms and peak-to-peak cycle jitter at the PLL output. A convenient way to do this may be with a Verilog-A module that monitors the $V_{DD}/2$ crossings of the clk_out waveform. In the Verilog-A notes that we discussed in class, there was an example module period_measure that came close to this functionality. Decrease $\omega_n$ to 0.05$\omega_{ref}$, keeping $\eta = \sqrt{2}/2$, by modifying the loop filter. What effect does this have on the cycle jitter at the PLL output? Please turn in simulation plots justifying your answers.

4. Separate out the PFD and CP, implementing the former with a Verilog-A model and the latter with a circuit implementation. Choose one of the single-ended charge-pump topologies discussed in class, matching the Verilog-A behavioral. Compare the functionality of your design with that simulated in (1). Comment on any effects of charge-pump ripple.