PHASE LOCKED LOOP

Mixed Signal design flow

Submitted by

Chandrasekhar Vishak
Gupta Sunil

Table of Contents

Abstract:

Analog/mixed signal design is seeing a pattern shift in design flow from bottom-up to top-down, which makes the realization of complex designs more convenient and feasible. Analog HDLs and behavioral libraries enable system designers to quickly write a block-level system model that can easily be simulated to optimize chip performance early in the design cycle. Because it's written in a standard HDL, this system design can be employed as a live specification to pass down to the transistor-level designer or out to a design subcontractor in a language that they and their tools will understand. Once in the hands of the transistor-level designers, each block can be logically decomposed and simulated, in increasing levels of detail, until the final transistor design is reached. Such a methodical approach can shave weeks or months off of the design cycle, helping companies meet their time-to-market deadlines.

New analog designs often include substantial mixed-signal content that needs accurate interfacing and interaction between the analog and digital portions. Older SPICE tools and techniques force designers to develop analog and digital subsystems in isolation. If subsystems aren’t joined until IC layout, they can’t be tested together until the silicon returns from fabrication. That is an extremely expensive time to discover an inverted bus signal or a faulty interaction between the analog and digital portions. Thus developing the design in a common platform makes the approach more pragmatic.

To gain a thorough understanding of transistor-level behavior, any differences between transistor-level design and its upper-level behavioral model need to be closely examined. This is the ideal time to calibrate the model to the transistor design. If the upper-level model is a library part, the designer can use the built-in test bench to automatically stimulate and characterize the transistor-level design. If the upper-level model is a custom model, the designer can build a test bench and/or use an optimization tool, to match the model to the transistor-level behavior. Any differences must be understood completely and resolved. Once the analog and digital designs are complete, they must be tested to verify that they work together before going to layout and fabrication.

The main accomplishment in this work has been to model a Phase Locked Loop in a top-down mixed signal design flow, and to fine-tune the behavioral description to be consistent with the simulation results of a structural level implementation. This not only helps make sure that the interaction between the mixed signal and purely digital components of the overall circuit is exactly as expected, but it also reduces the simulation time of such a complex design which can otherwise take hours to simulate. VHDL-AMS is used as the mixed signal language for modeling the circuit. Schematic level implementation and corresponding simulations have been done using Cadence Analog Artist. The design development has been initiated on a mixed signal simulation tool that is freely available to the designer and has features capable of implementing a design of moderate complexity. Educational Version of SystemVision, a Mentor Graphics tool, has been chosen for this. Its VHDL/VHDL AMS and spice support makes it a powerful tool for implementing mixed signal designs. The schematic support enables the designing and customizing of test benches conveniently adapt to the requirements of any design under test.

Introduction:

Phased locked loop is a closed loop circuit which produces an output signal that is in phase with input waveform and the frequency of the output signal is a multiple of (or equal to) the input signal. It finds wide application in the field of communication, wireless systems, digital circuits and disk drive electronics. Typical uses are in tasks like jitter reduction, skew suppression and frequency synthesis. A basic PLL has been designed and implemented in a top-down mixed signal design flow.

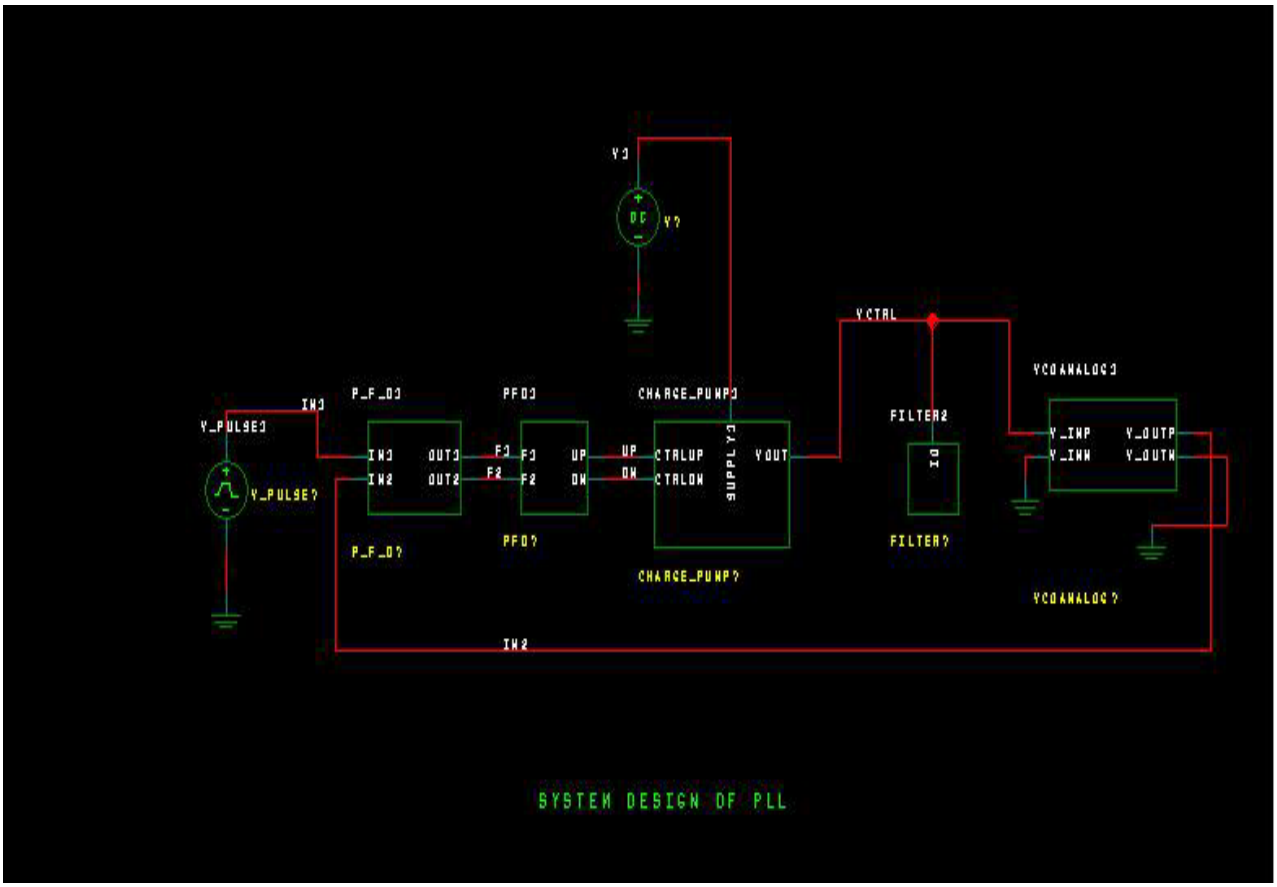


Figure.1 System level view of the PLL

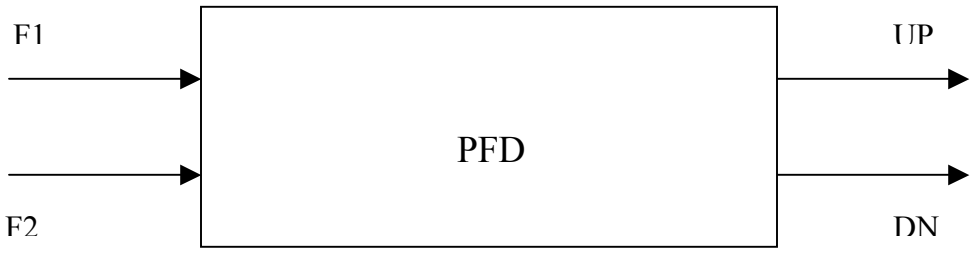
The digital modules have been written in VHDL and the analog parts have been written in VHDL AMS. The behavioral design has been calibrated against a schematic level design using Cadence Analog Artist. Individual Modules were written in HDL and then simulated using SystemVision Test bench setups. Precise delay models were developed for the blocks that

calibrated the schematic counterpart in cadence. The whole PLL was then simulated and the actual simulation time was reduced from a few hours to a few minutes.

The basic phase locked loop is a feedback system that operates on excess phase of normally periodic signals. The frequency and phase detector serves as an error amplifier which minimizes the phase difference between x(t) and y(t). The loop is considered "locked" if $\Delta\Phi$ is constant with time, a result of which is that the input and output frequencies are equal. In the locked condition the frequency and phase detector produces an output whose dc value is proportional to $\Delta\Phi$. The low pass filter suppresses the high frequency components and gives a dc output that controls the VCO frequency. The VCO then oscillates at a frequency equal to the input frequency and phase difference equal to $\Delta\Phi$. The VCO phase can be regarded as the initial condition of the system, independent of the initial condition in the LPF.

Phase and Frequency Detector:

The PFD was modeled using a structural VHDL code. At a system level the PFD is a pair of D flip-flops with their D inputs fixed at high and the clocks derived from the input signal and the feedback signal from the VCO respectively. The outputs are fed back to provide a clear signal to the flip-flops. The structural blocks were simulated in cadence and their delays were determined. These delays were then used to model the PFD in VHDL.



The block diagram of a phase frequency detector (PFD) is shown in Figure 2. If the frequency of input F1 is less than that at input F2, the PFD produces positive pulses at UP, while DN remains at zero. Conversely if the frequency at input F2 is higher than the frequency input F1, the PFD produces pulses at DN and UP remains at zero. If both frequencies are equal, then the circuit generates pulses at either UP or DN with a width equal to the phase difference between the two inputs.

This type of phase detector is also termed as sequential phase detector. It compares the falling edges of the F1 and F2 waveforms. The pulse width of the F1 and F2 are irrelevant in operation of this PFD. If the falling edge of the F1 leads the F2 falling edge, the UP output the PFD goes high while DN remains low. This causes the F2 frequency to increase, having effect of moving the edges closer together. When the F2 falling edge leads the F1 falling edge, UP remains low while DN goes high for a time proportional to the phase difference between F1 and F2.These pulse width modulated signals control the switched current sources in the charge pump.

The characteristics of the PFD can be summarized as below:

1. A falling edge from F1 and F2 must be present when doing a phase comparison.
2. The absolute width of the F1 and F2 is irrelevant.

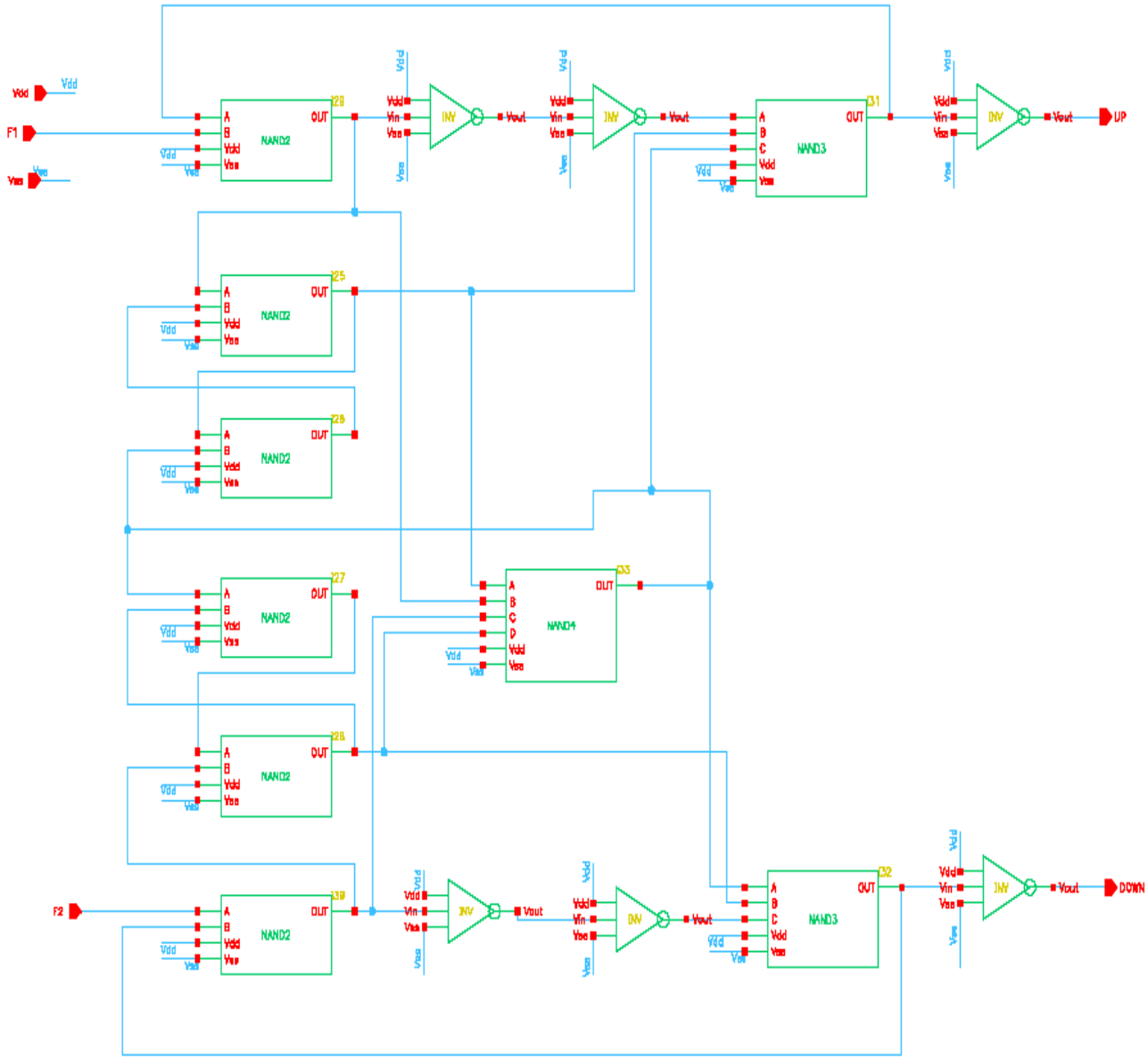


Figure.2 PFD Block Diagram (Schematic Representation)

The following section presents the VHDL code used to model this PFD

```
-- PFD module, compares the negative edges of input voltages F1 and F2 and
-- generates capacitor charging and discharging signals UP and DN
----------------------------------------------------------------------

--entity for two input nand

library ieee;
use ieee.std_logic_1164.all;

entity NAND_2 is
port(    x: in bit;
         y: in bit;
         F: out bit);
end NAND_2;

architecture beh of NAND_2 is
begin
         F <= x nand y after 0.12 NS;
end beh;
----------------------------------------------------------------------

--entity for three input nand

library ieee;
use ieee.std_logic_1164.all;

entity NAND_3 is
port(    x: in bit;
         y: in bit;
       z: in bit;
```

```
        F: out bit
);
end NAND_3;

architecture beh of NAND_3 is
begin
        F <= ((not(x) or not(y)) or not(z)) after 0.2 NS;
end beh;
-------------------------------------------------------------------

--entity for four input nand

library ieee;
use ieee.std_logic_1164.all;

entity NAND_4 is
port(   x: in bit;
        y: in bit;
      z: in bit;
        w: in bit;
        F: out bit
);
end NAND_4;

architecture beh of NAND_4 is
begin
        F <= not(x) or not(y) or not(z) or not(w)after 0.3 NS;
end beh;

-------------------------------------------------------------------

--  Structural code for the PFD block
```

```
library ieee;
use ieee.std_logic_1164.all;

entity PFD is
port(    F1: in std_logic;
         F2: in std_logic;
       UP:out std_logic:='0';
         DN: out std_logic:='0'
);
end PFD;

architecture structural of PFD is

--declaring all the components

component NAND_2
port(X,Y:in bit;F:out bit); end component;
component NAND_3
port(X,Y,Z:in bit;F:out bit); end component;
component NAND_4
port(X,Y,Z,W:in bit;F:out bit); end component;

-- stating which library to find them in and which architecture to use

for all: NAND_2 use entity work.NAND_2(beh);
for all: NAND_3 use entity work.NAND_3(beh);
for all: NAND_4 use entity work.NAND_4(beh);


--Declaring local signals required
Signal n1, n2, 3,n4, n5, n6, n7, n8, n9, 10,n11, n12, n13: bit: ='0';
Signal F1BIT, F2BIT: bit;
Signal UPBIT, DNBIT: bit: ='0';
```

```
begin
F1BIT<= TO_BIT (F1);
F2BIT<= TO_BIT (F2);
nand2_one: NAND_2 port map(n12,F1BIT,n1);
nand2_two: NAND_2 port map(n1,n3,n2);
nand2_three: NAND_2 port map(n2,n10,n3);
nand2_four: NAND_2 port map(n10,n5,n4);
nand2_five: NAND_2 port map(n4,n6,n5 );
nand2_six: NAND_2 port map(F2BIT,n13,n6);
n7<= not(n1) after 0.05 NS;
n8<= not(n6) after 0.05 NS;
n9<= not(n7) after 0.05 NS;
n11<= not(n8)after 0.05 NS;
nand4_one: NAND_4 port map(n1,n2,n5,n6,n10);
nand3_one: NAND_3 port map(n2,n9,n10,n12);
nand3_two: NAND_3 port map(n5,n10,n11,n13);
UPBIT<= not(n12)after 0.05 NS;
DNBIT<= not(n13)after 0.05 NS;
UP<= to_stdUlogic (UPBIT);
DN<= to_stdUlogic (DNBIT);
end structural;


------------------------------------------------------------------
    --  end of PFD module
```

Simulation Results (PFD):

The PFD module was tested with a set of two input frequencies. The test setups and the simulation results are shown below in figures 3 and 4:

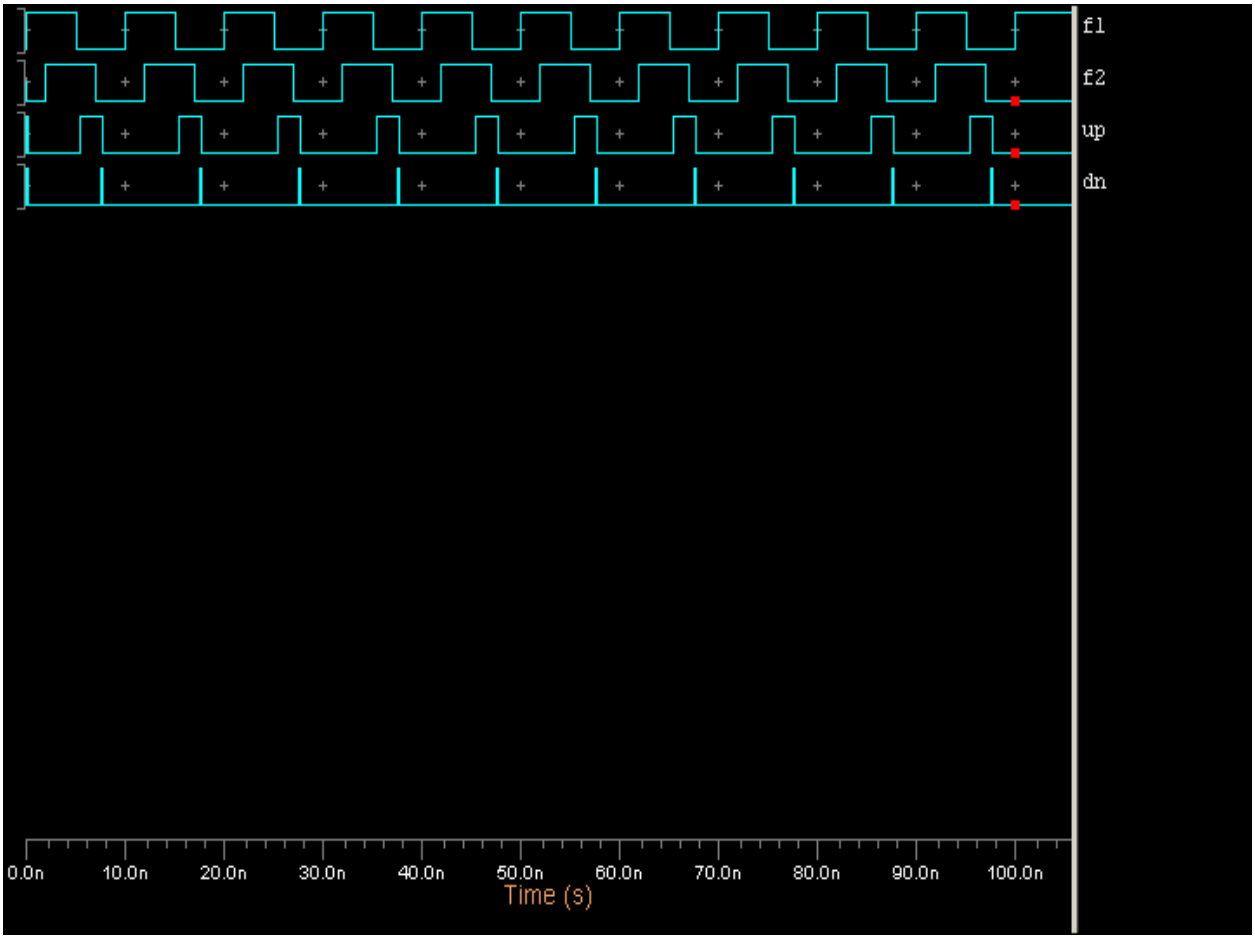


Figure.3 F1 and F2 at the same frequency (100 MHz) with a phase difference of 2 ns.

As expected, since F1 leads F2, a pulse is generated at the UP output with a pulse width equal to the difference in phase between the two inputs. Note here that although spikes are observed for the down signal, they are filtered out by the loop filter and hence don't affect the overall performance of the system.

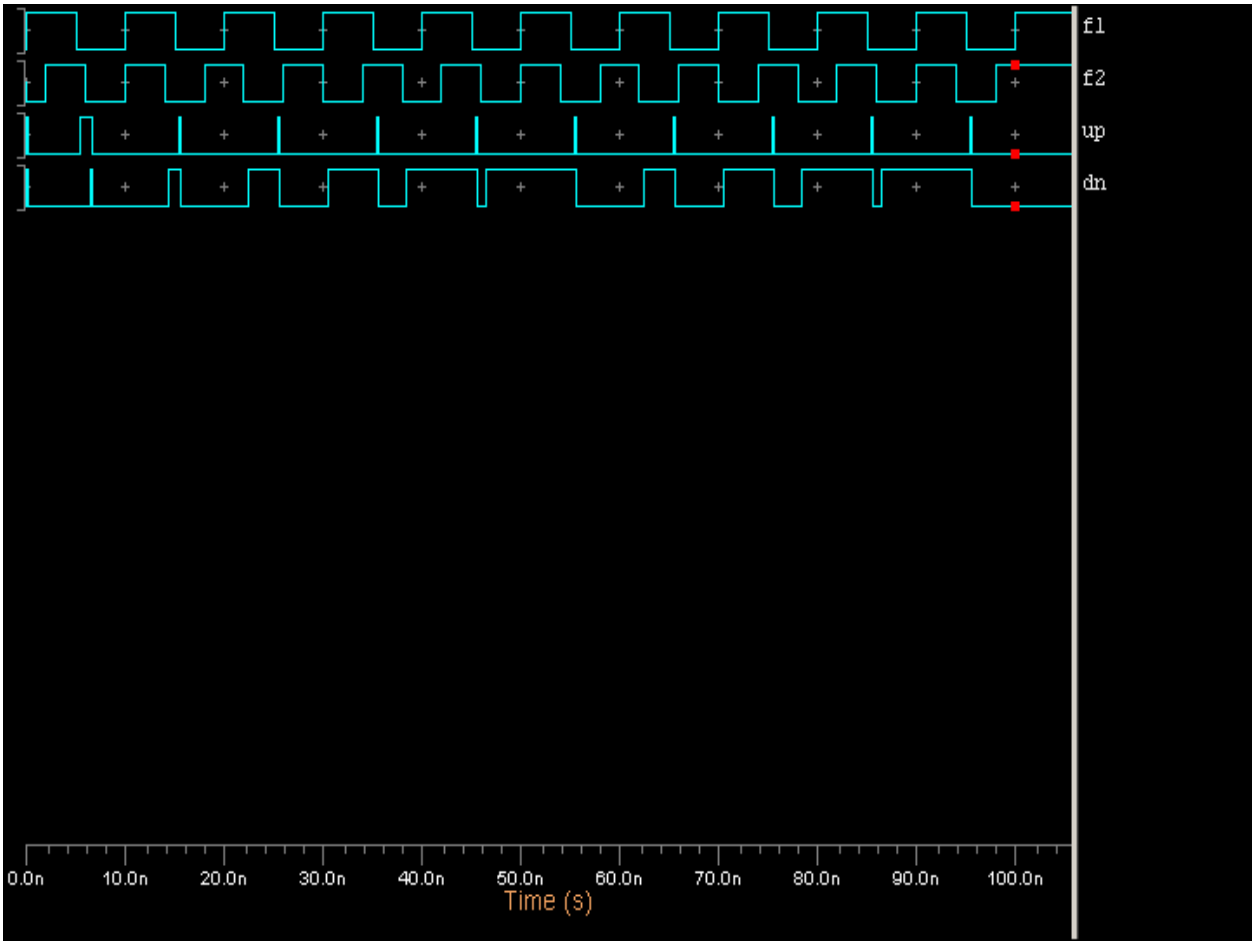


Figure.4 F1 and F2 at different frequencies (100 MHz and 125 MHz resp.) with a phase difference of 2 ns.

Again, pulses are generated at UP and DOWN terminals according to the relative falling edges of the F1 and F2 inputs. These pulses can then be used to suitably charge or discharge a capacitor to generate the VCO control voltage.

Charge Pump:

The Charge pump consists of a pair of switched current sources which either source or sink current pulses to an on-chip loop filter implemented with CMOS. The reference voltage block uses current mirrors and resistively biased current sources to generate bias voltages for transistors that are used to charge and discharge the charge pump capacitors. Thus, the P-bias and the N-bias controls the charging and the discharging current through the capacitors in the low pass filter. The switch block has been implemented as sets of cmos transmission gates with complementary clocks, balanced for equal delay, which are controlled by Up and Down signals from the frequency and phase detector. Charge pump provides an infinite gain for a static phase difference at the input of the PFD i.e., a non-zero (deterministic) difference between the phase of F1 and F2 leads to an indefinite charge build up on the capacitor at the output. Thus, to maintain a constant VCO control voltage, the input phase error must be minimal. When and input and output frequencies are sufficiently close, the PFD operates as a phase detector, performing phase lock. At locking condition the phase difference drops to zero and the charge pump remains relatively idle.

The behavioral model for the charge pump uses the DC resistance of the pull up and pull down blocks in series with the switches to model the behavior of the charge pump. The transistors P0 and N0 in figure 5 remain in saturation for all the times whenever they are on. The DC resistance of these two mos devices are time varying non linear and dependent on the operation point of the transistor. A reasonable approach in that respect is to use the average value of the resistance over the operation range of interest, or even simpler, the average of the values at the end points. Thus, Cadence simulations were run and hand calculations were done to find out the average DC resistance of the Pmos (P0) in the pull up block and the NMOS (N0) in the pull down block during the times in which they are on. A behavioral code of the circuit was then written in VHDL-AMS.

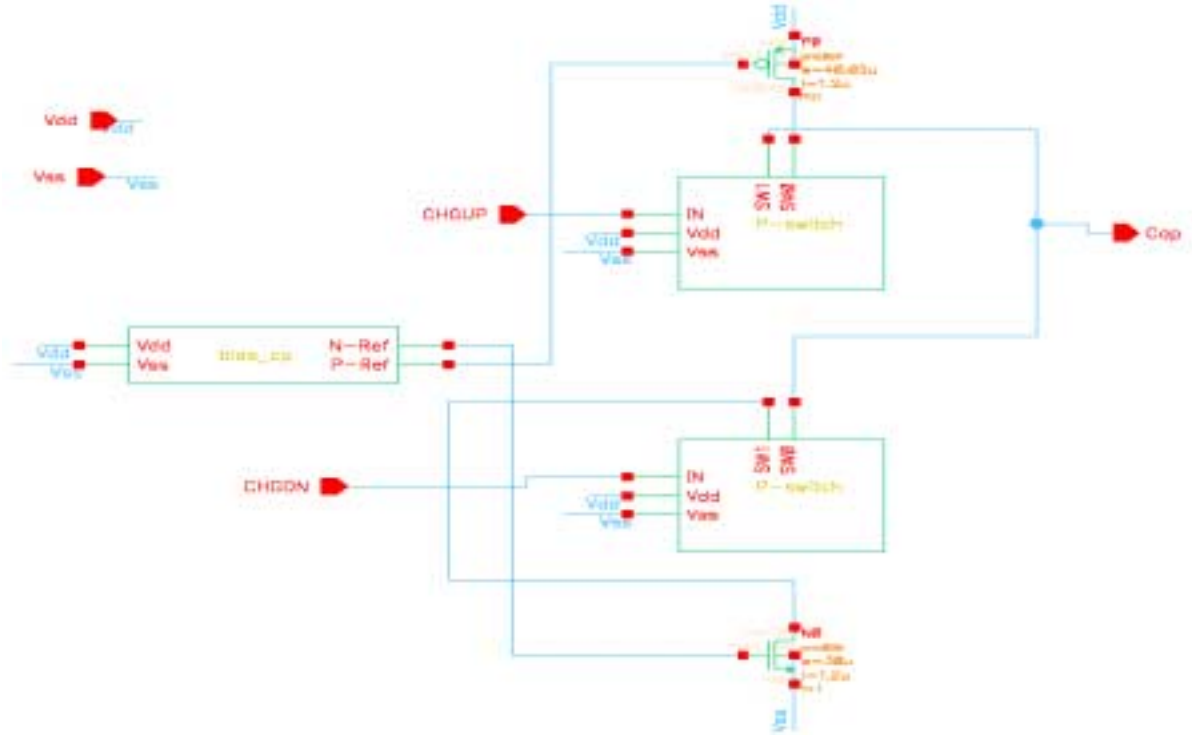


Figure.5 Charge Pump Schematic

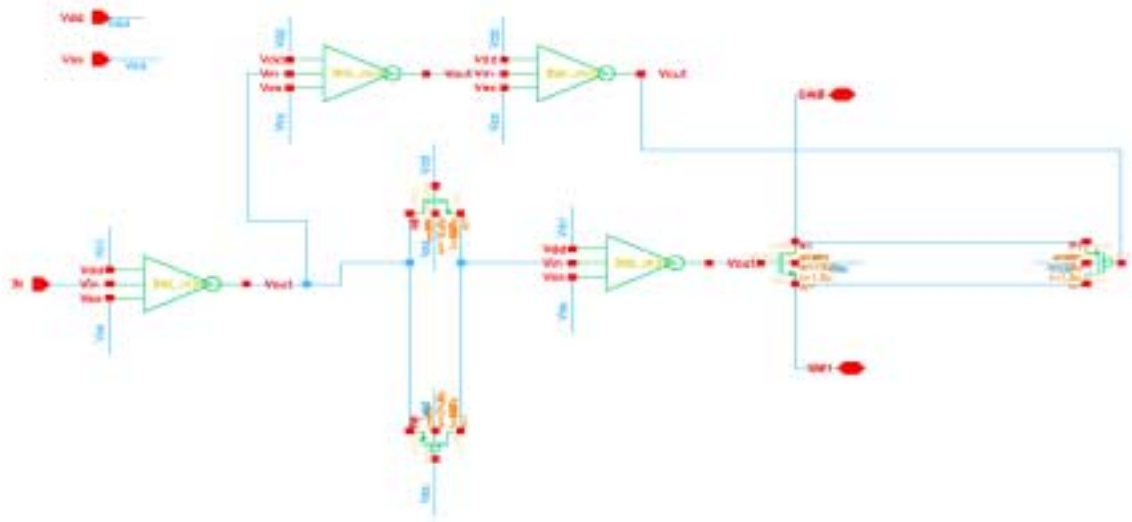


Figure.6 P-Switch Schematic

The following section presents the VHDL AMS code for the charge pump

```
-- Charge pump module, takes in the UP and DOWN signals from PFD and
-- converts them to a single control voltage for the VCO
```

```
-- Resistor Entity

library IEEE;
use IEEE.electrical_systems.all;

entity resistor is
  generic (
    res : resistance);
  port (
    terminal p1, p2 : electrical);
end entity resistor;

architecture ideal of resistor is
 quantity v across i through p1 to p2;
begin
  v == i*res;
end architecture ideal;

------------------------------------------------------------

-- Switch Entity

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.electrical_systems.all;

entity switch is
  generic (r_open     : resistance := 1.0e9;   -- open and close res. of switch
```

```
        r_closed   : resistance := 0.001;
         trans_time : real       := 1.0e-15 ); -- switch transition time
  port (sw_state        : in std_logic:='0';
       terminal p1, p2 :    electrical);
end entity switch;

architecture ideal of switch is
  signal   r_sig : resistance := r_open;
  quantity v across i through p1 to p2;
  quantity r     : resistance;

begin
  DetectState: process (sw_state)

  begin  -- process DetectState
    if (sw_state'event and sw_state = '0') then
      r_sig <= r_open;
    elsif (sw_state'event and sw_state = '1') then
      r_sig <= r_closed;
    end if;
  end process DetectState;
  r == r_sig'ramp(trans_time, trans_time);
  v == r*i;
end architecture ideal;

------------------------------------------------------------------------
-- Charge Pump Entity

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.electrical_systems.all;

entity charge_pump is
```

```
port(
ctrlup,ctrldn:in std_logic:='0';  --inputs
terminal supply1: electrical;  -- Vdd
terminal vout: electrical      -- output voltage
);
end entity charge_pump;

architecture structural of charge_pump is

-- component declaration

component resistor
  generic (
    res : resistance);
  port (
    terminal p1, p2 : electrical);
    end component;

  component switch
  generic (r_open     : resistance := 1.0e9;
           r_closed   : resistance := 0.001;
           trans_time : real       := 1.0e-15 );

  port (sw_state       : in std_logic;
        terminal p1, p2:    electrical);
end component;

for all: resistor use entity work.resistor(ideal);
for all: switch use entity work.switch(ideal);

   terminal p1,p2: electrical;
   begin          -- structural architecture
```

```
 Rup : RESISTOR
 generic map ( RES => 500.0E3 )
 port map ( P1 => supply1,
            P2 => P1 );

  Rdown : RESISTOR
   generic map ( RES => 500.0E3 )
   port map ( P1 => electrical_ref,
              P2 => P2 );

switch_up : switch
    generic map (r_open => 1.0e9,
        r_closed => 0.001,
        trans_time =>1.0e-15
        )
 port map (sw_state => ctrlup,
          p1=>p1,
          p2=>vout);

switch_dn : switch
    generic map (r_open => 1.0e9,
        r_closed => 0.001,
        trans_time =>1.0e-15
        )
 port map (sw_state => ctrldn,
          p1=>p2,
          p2=>vout);

 end architecture structural;
 ------------------------------------------------------------------
    -- end of charge pump module
```

Simulation Results (Charge Pump):

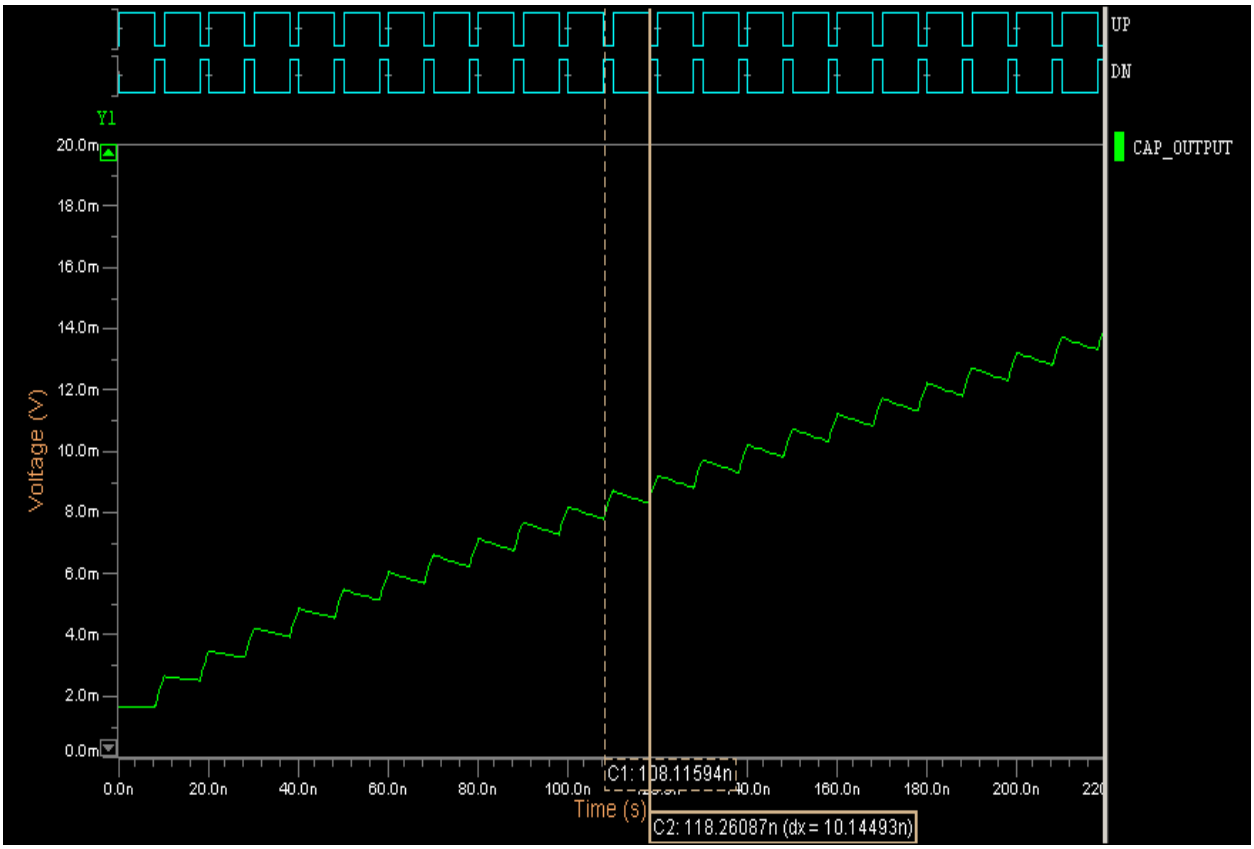


Figure.7 Charge Pump Simulation Results

The figure above shows the simulation results for the charge pump module. The UP and DOWN inputs are given complementary pulses of varying duty cycle, and a loop filter, as described below, is used for the output capacitance. As expected, the capacitance at the output charges whenever UP is asserted and discharges whenever down is asserted.

Loop Filter:

The output of the charge pump has high frequency components that are passed through a low pass loop filter to filter out high frequency terms. The input to the loop filter comes from the output of the charge pump. For slow variations in the phase difference, the filter acts like an integrator averaging output of the PFD. For fast variations, however, the filter looks like a resistive divider without any integration. This allows the loop filter to track fast variations in the

time difference between the rising edges. In the Schematic design a CMOS transmission gate is used as a resistor, and MOS transistors (with Source and Drain shorted) are used as capacitors. These are not ideal components but are adequate for this application. P-diffusion resistors might also be used for the resistor, while the capacitors can be poly-poly capacitors if a two-poly process is available. Alternatively, the filter can be implemented off-chip at the expense of a pad and of possible noise injection. Figure no gives the schematic design for the filter.

The loop time constant of this loop filter R (C1+C2) is chosen to be 5 times the input time period at the center frequency (100 MHz). The two CMOS capacitances where taken to be equal.

The gate capacitance Cox is given by

$C_{ox} = \varepsilon_{ox} . A / Tox$

where $\varepsilon_{ox} = \varepsilon_r . \varepsilon_o = 3.9 \times 8.854 \times 10^{-12}$

Tox = 141 Angstroms (for a typical ami06 0.5 micron process)

Time period of the input wave T = 1/f for a center frequency of 100 MHz

$f = 100 \times 10^6$ Hz, then we get T = 10 ns.

As time constant of the loop filter R(C1+C2) = RC = 5T, we have

RC=50ns.

If R is chosen to be 10Kohms(implemented through a pair of CMOS transistors),

C = 5pF

From above, we get

$C_{ox} = \varepsilon_{ox} . A / Tox = 3.9 \times 8.854 \times 10^{-12} \times A / 141 \times 10^{-10} \Rightarrow A = 2 * 10^{-9} m^2$

For Cox = C/2 (The two CMOS capacitors are in parallel) assuming that the oxide capacitance at the gate is the most dominant of all the capacitances effecting the capacitive effect in the operation of the mos.

$A = W*L = 2 * 10^{-9} m^2$

Choosing L = 3.6 micron and W = 540 micron we get an area close to the analytically calculated value for the filter. Thus these numbers have been used in the design of the loop filter.

The mixed signal model of PLL has this filter implemented as a behavioral block with a resistor and two capacitors. The values of these circuit elements were calibrated by running parallel simulations in Cadence and SystemVision. The test bench set up had the charge pump

and the loop filter blocks next to each other with inputs of varying duty cycle to mimic the in circuit conditions of the actual design.

Again an RC of five times the time period around the central frequency of 100MHz was chosen. Thus RC for this behavioral block was chosen to be 52ns.

This was implemented as a 4K resistor in series with two parallel capacitors with a capacitance of 13 pf each.

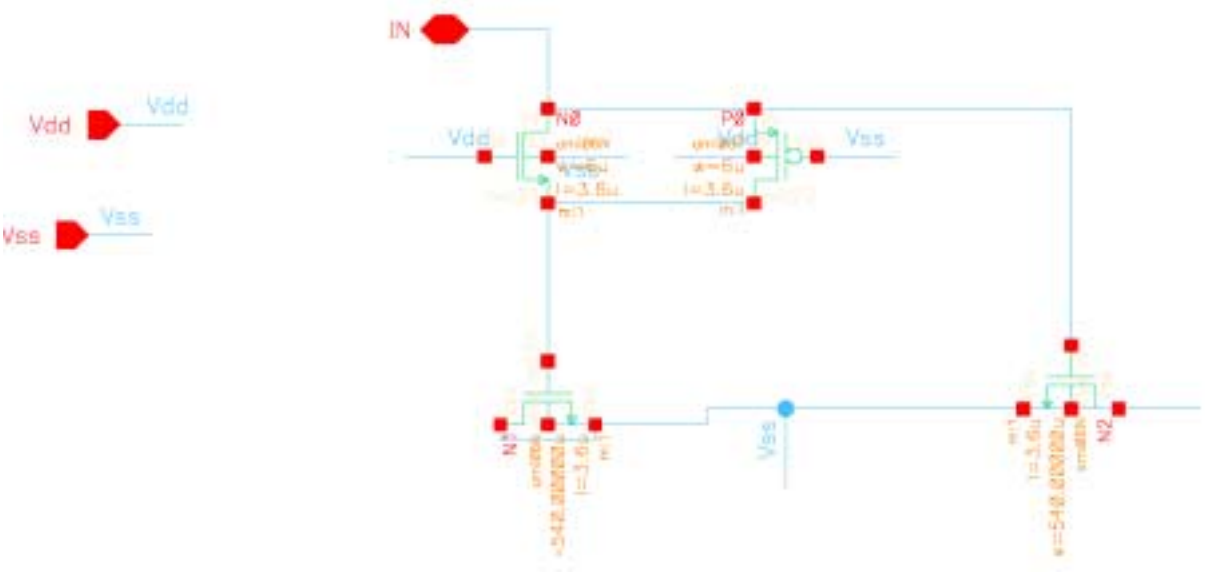


Figure.8 Loop Filter Schematic

The following section presents the VHDL AMS code for the loop filter

```
-- Loop Filter Module, provides the average DC value of the charge pump output
-- as the control input to the VCO
------------------------------------------------------------------

-- Resistor Entity

library IEEE;
use IEEE.electrical_systems.all;

entity resistor is
```

```
  generic (
    res : resistance);

  port (
    terminal p1, p2 : electrical);
end entity resistor;

architecture ideal of resistor is
  quantity v across i through p1 to p2;

begin
  v == i*res;
end architecture ideal;
-------------------------------------------------------------------------

-- Capacitor Entity

library IEEE;
use IEEE.electrical_systems.all;

entity capacitor is
  generic (
    cap           : capacitance;
    v_ic          : real := real'low);  -- quiescent domain voltage

  port (
    terminal p3, p4 : electrical);

end entity capacitor;


architecture ideal of capacitor is
  quantity v across i through p3 to p4;
```

```
begin

  if domain = quiescent_domain and v_ic /= real'low use
    v == v_ic;
  else
    i   == cap * v'dot;
  end use;

end architecture ideal;

--------------------------------------------------------------------
--  Filter Entity

library IEEE;
use IEEE.electrical_systems.all;

entity FILTER is
port(
terminal IO: ELECTRICAL);   -- Inout port of the filter
end entity FILTER;


architecture structural of FILTER is
   terminal P6: ELECTRICAL;

   -- component declaration

   component RESISTOR
      generic( RES : RESISTANCE );
      port( terminal P1 :  ELECTRICAL;
            terminal P2 :  ELECTRICAL );
   end component RESISTOR;
```

```
  component CAPACITOR
    generic( CAP : CAPACITANCE;
             V_IC : REAL:=REAL'LOW );
    port( terminal P3 :  ELECTRICAL;
          terminal P4 :  ELECTRICAL );
  end component CAPACITOR;

  for R2: RESISTOR use entity WORK.RESISTOR(IDEAL);
  for C3: CAPACITOR use entity WORK.CAPACITOR(IDEAL);
  for C4: CAPACITOR use entity WORK.CAPACITOR(IDEAL);

begin  -- structural architecture

  R2 : RESISTOR
    generic map ( RES => 1.0E3 )
    port map ( P1 => IO,
               P2 => P6 );

  C3 : CAPACITOR
    generic map ( CAP => 1.0E-12 )
    port map ( P3 => P6,
               P4 => ELECTRICAL_REF );

  C4 : CAPACITOR
    generic map ( CAP => 1.0E-12 )
    port map ( P3 => IO,
               P4 => ELECTRICAL_REF );

end architecture structural;

----------------------------------------------------------------------
-- end of loop filter code
```

Voltage controlled Oscillator:

The Schematic module of the VCO consists of a 7 stage current-starved oscillator with a buffered output. The output has a two-stage buffer with a stage ratio of e, which enables it to drive the output node with minimum delay. This configuration was chosen because it had a wide range of operation and was verified to operate correctly over the entire range. In fact this VCO design is susceptible to phase noise and power supply noise issues. A differential design will be more beneficial from this pint of view. The design we chose was very feasible to model behaviorally.

VCO simulations were done in cadence over the entire operating range and then the gain was calculated after curve fitting. This analytical gain expression was used to model the VCO in SystemVision. The analog VCO outputs from the module had to be converted to digital bits in the transformation block before they could be fed back to the PFD.

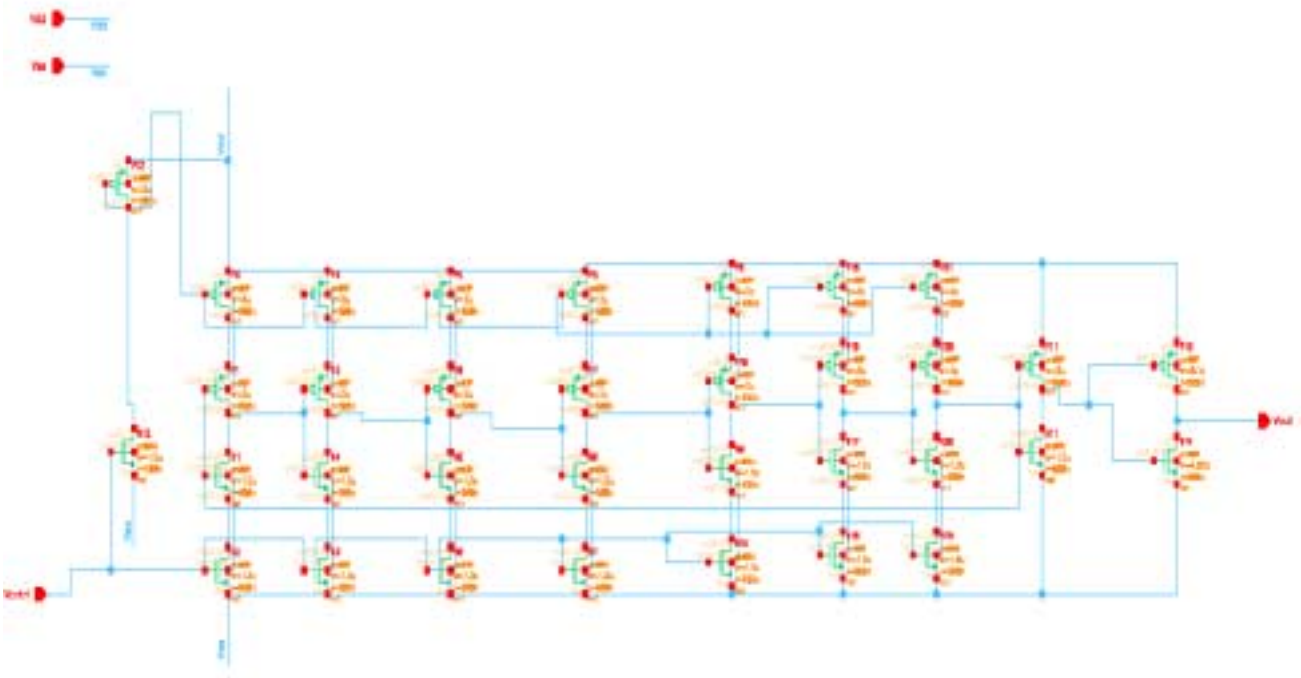


Figure.9 VCO Schematic

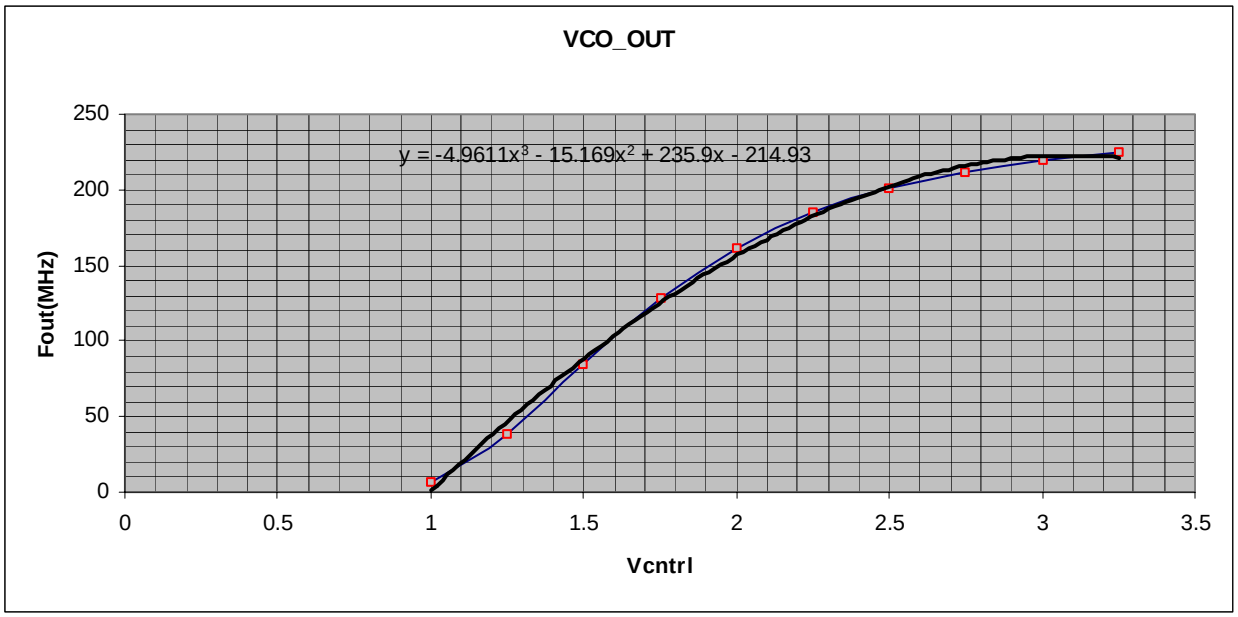


Figure.10 VCO Gain Curve

The curve in blue represents the actual simulation results from Cadence. An equation to fit this curve was calculated and used for the modeling of the VCO in VHDL-AMS. This curve is represented in black in the above figure.

The following section presents the VHDL AMS code for the VCO

```
-- VCO module, takes in control input from the filter and provides a
-- corresponding output frequency

-----------------------------------------------------------------------

-- VCO Entity

library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;
```

```
entity VCOAnalog is
  generic (
    Vcmin       : voltage := 0.0;    -- control voltage mininum [Volts]
    Vcmax       : voltage := 3.3;    -- control voltage maximum [Volts]
    Vout_ampl   : voltage := 1.0;    -- amplitude of output [Volts]
    Vout_offset : voltage := 0.0     -- offset voltage of output [Volts]
    );

  port (
    terminal v_inp, v_inm, v_outp, v_outm : electrical);
end entity VCOAnalog;


architecture behavioral of VCOAnalog is
  quantity vout across iout through v_outp to v_outm;
  quantity vctrl across v_inp to v_inm;
  quantity phi : real;
  quantity vtmp : real;
  quantity vout_temp : real;

begin

  if vctrl > Vcmax use               -- limit range of control voltage
    vtmp == Vcmax;
  elsif vctrl < Vcmin use
    vtmp == Vcmin;
  else
    vtmp == vctrl;
  end use;

  if domain = quiescent_domain use
    phi     == 0.0;
  else
```

```
-- equation derived by curve fitting of the simulation results from cadence

        phi'dot == (-(4.9611*vtmp*vtmp*vtmp) - (15.169*vtmp*vtmp) + (235.9*vtmp) -
(214.93))*1.0e6;


  end use;
vout_temp == Vout_offset + Vout_ampl*cos(math_2_pi*phi);
if vout_temp > 0.0 use
vout == 3.3;
else vout == 0.0;
end use;
end architecture behavioral;
-------------------------------------------------------------------
--end of VCO module
```

Simulation Results(VCO):

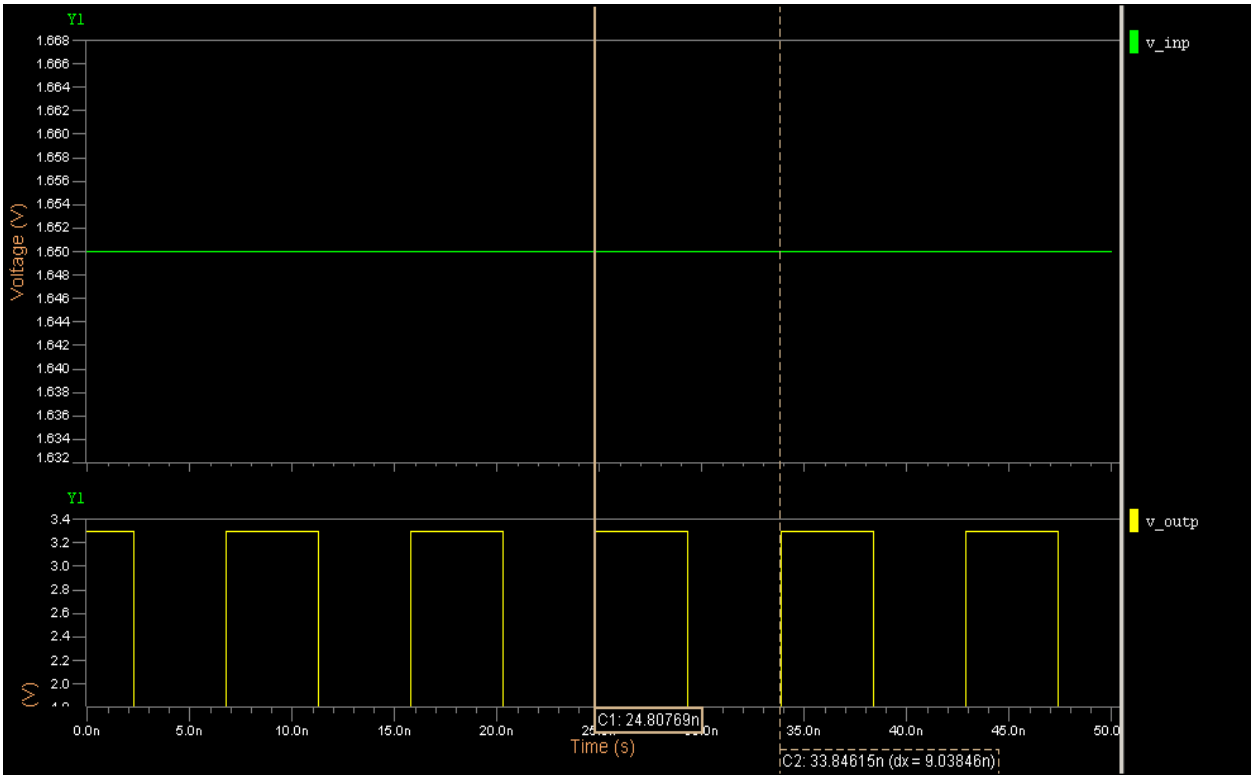


Figure.11 VCO output for Vcntrl = 1.65 V

The above figure shows the VCO simulation results for an input control voltage of 1.65 V. As expected from the equation used for fitting the curve, the output frequency was found to be ~ 110 MHz.

System Integration:

Once all the models for the PLL were tested and calibrated against their schematic counterparts they had to be merged together to perform the system level simulations.
The SystemVision educational version provides a powerful, yet easy-to-use environment for Successive Verification, but with the following restrictions:

Simulator Capacity Limits

- Maximum of 30 analog nodes
- Maximum of 30 analog quantities
- Maximum of 100 digital signals

Schematic Limitations

- Maximum of 10 components for save and open operations
- No packaging capability to PCB layout

Due to these limitations it was not possible to do the complete integrated simulation in the educational version of the tool. The full version of the tool thus had to be used towards the end of the design just to see the entire system run as a whole. To test the entire design a range of frequencies were chosen .The design was tested against the schematic design at the center frequency (100 MHz) and two corner frequencies of 50MHz and 150 MHz.

Comparison of Results: SystemVision vs. Cadence:

The following figures show the input and VCO output waveforms for the various frequencies mentioned above, and also the VCO control voltage to indicate the VCO lock time. The following two significant differences could be observed between the two design flows while simulating the overall PLL module:

1) There was a dramatic improvement in simulation time for the VHDL-AMS module as against schematic level simulations in cadence. While Cadence simulations took a minute of real time for every μs of simulation time, SystemVision took 8 sec for the same.

2) The lock time for the schematic module (~ 5 μs) was much smaller than the lock time for the VHDL-AMS module (~ 20 μs).

3) Since the design in SystemVision is behavioral in nature the locking range for the PLL is much wider than that for the design in Cadence. However, for input frequencies more than 150 MHz, the simulation time taken to lock was appreciably higher (>30μs). The cadence design, on the other hand, does not lock beyond a particular frequency (180MHz) due to VCO and filter constraints. Also since the PFD design has a feedback loop, there is a constraint on the maximum frequency the circuit can be clocked at.

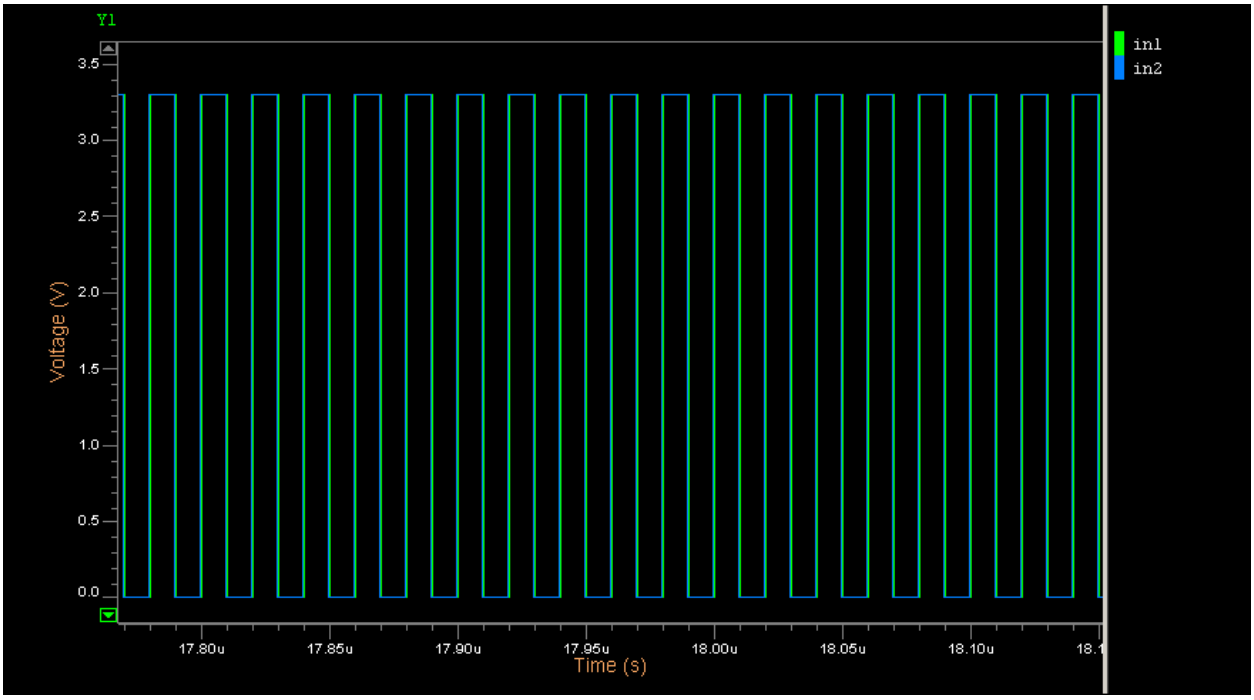


Figure 12 (Zoomed) Input, Output PLL waveforms at lock @ 50 MHz (SystemVision results)

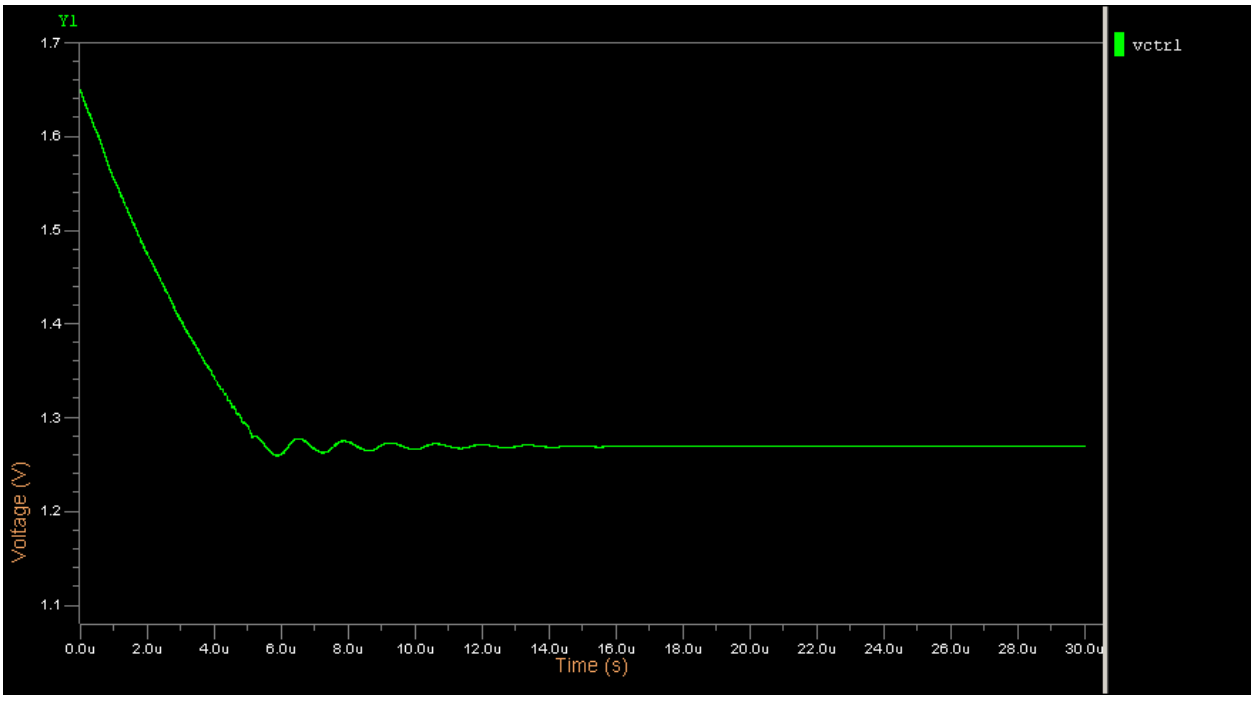


Figure 13 Control Voltage for the VCO at 50 MHz (SystemVision results)

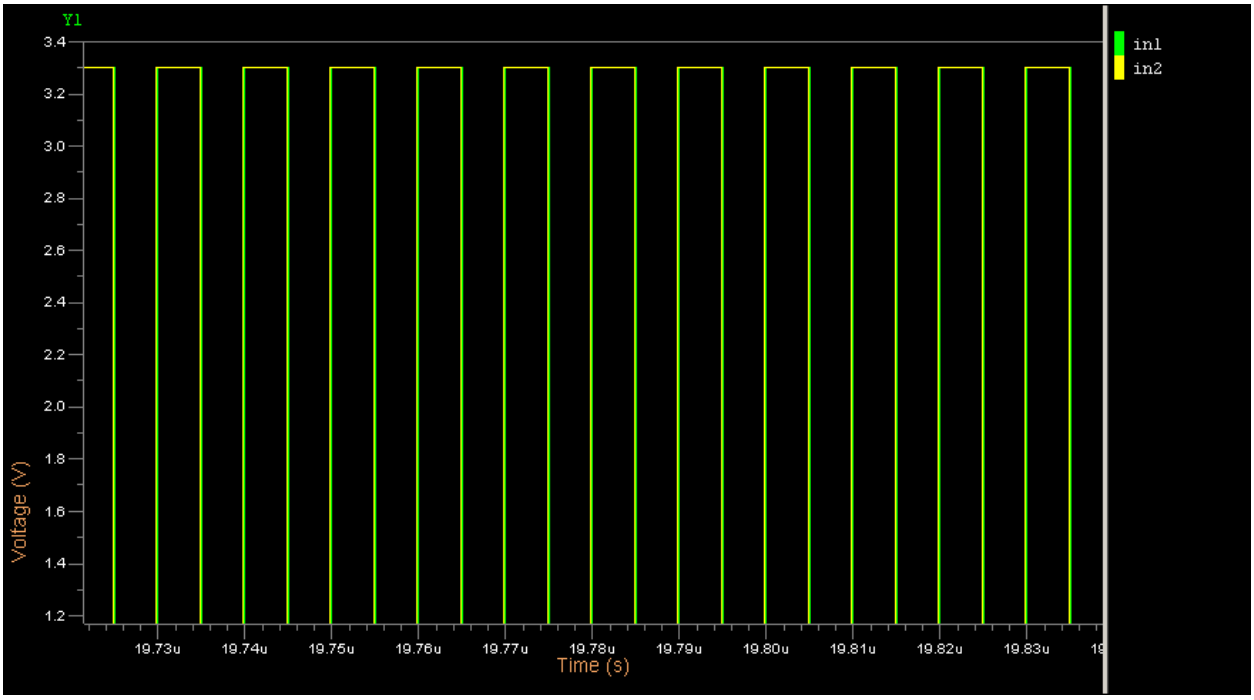


Figure 14 (Zoomed) Input, Output PLL waveforms at lock @ 100 MHz (SystemVision results)

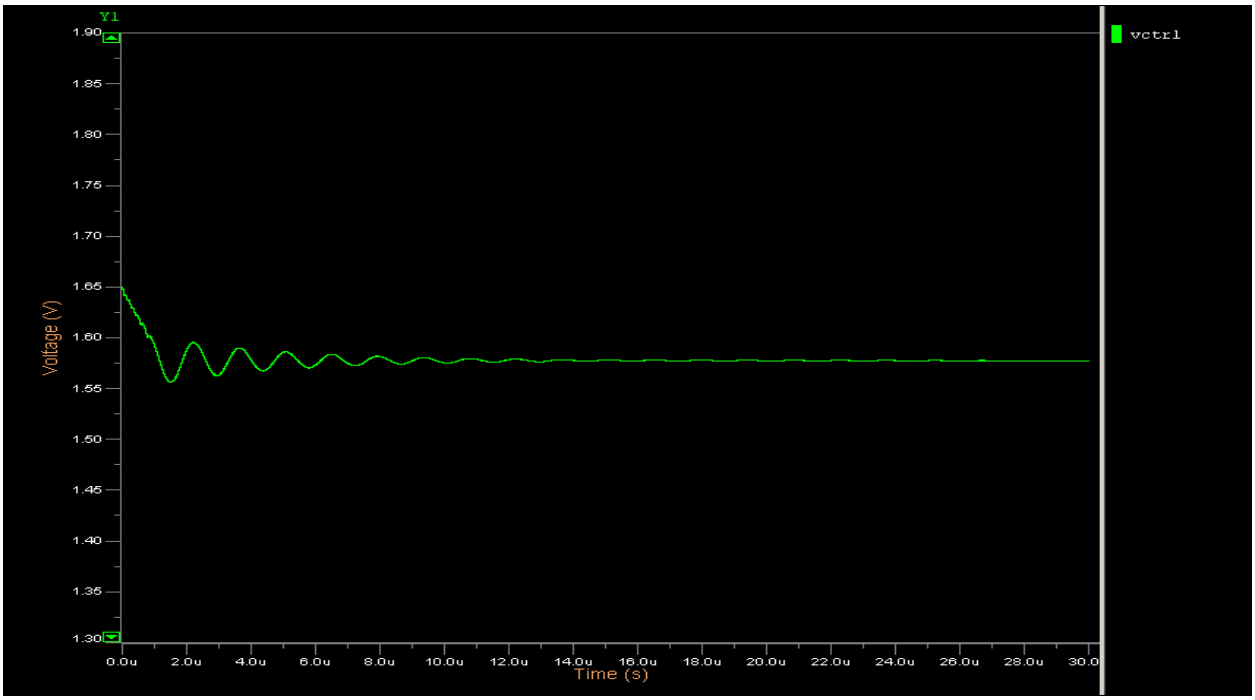


Figure 15 Control Voltage for the VCO at 100 MHz (SystemVision results)

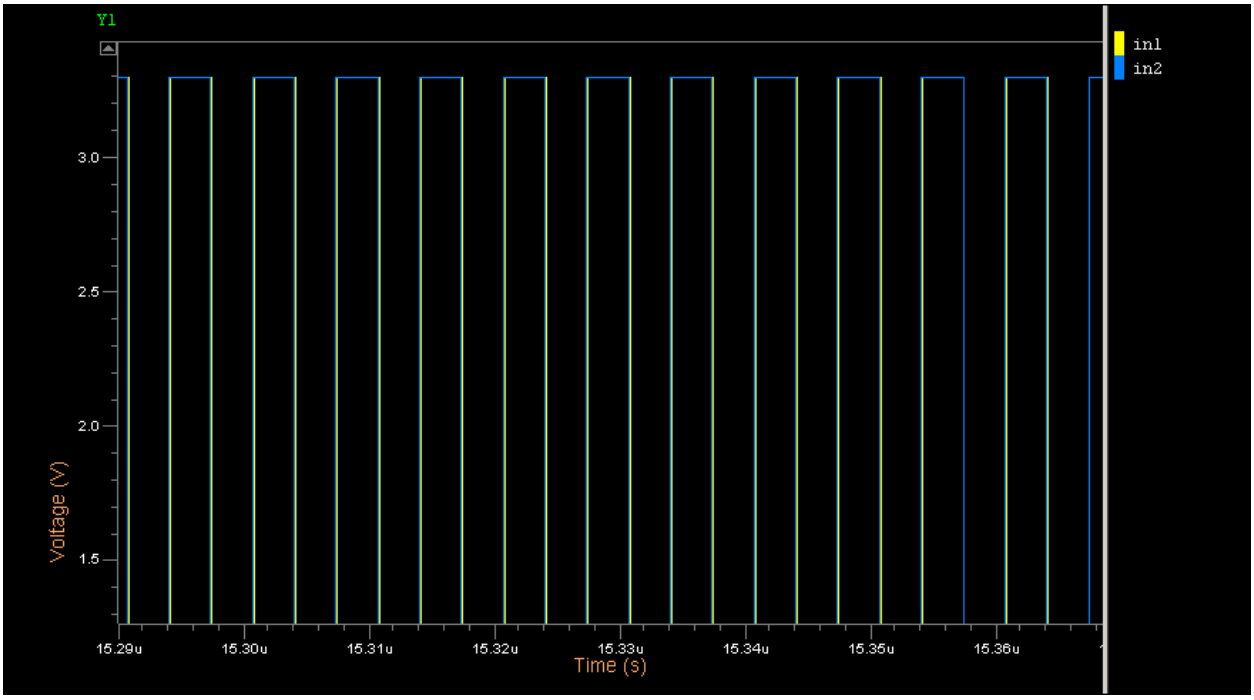


Figure 16 (Zoomed) Input, Output PLL waveforms at lock @ 150 MHz (SystemVision results)

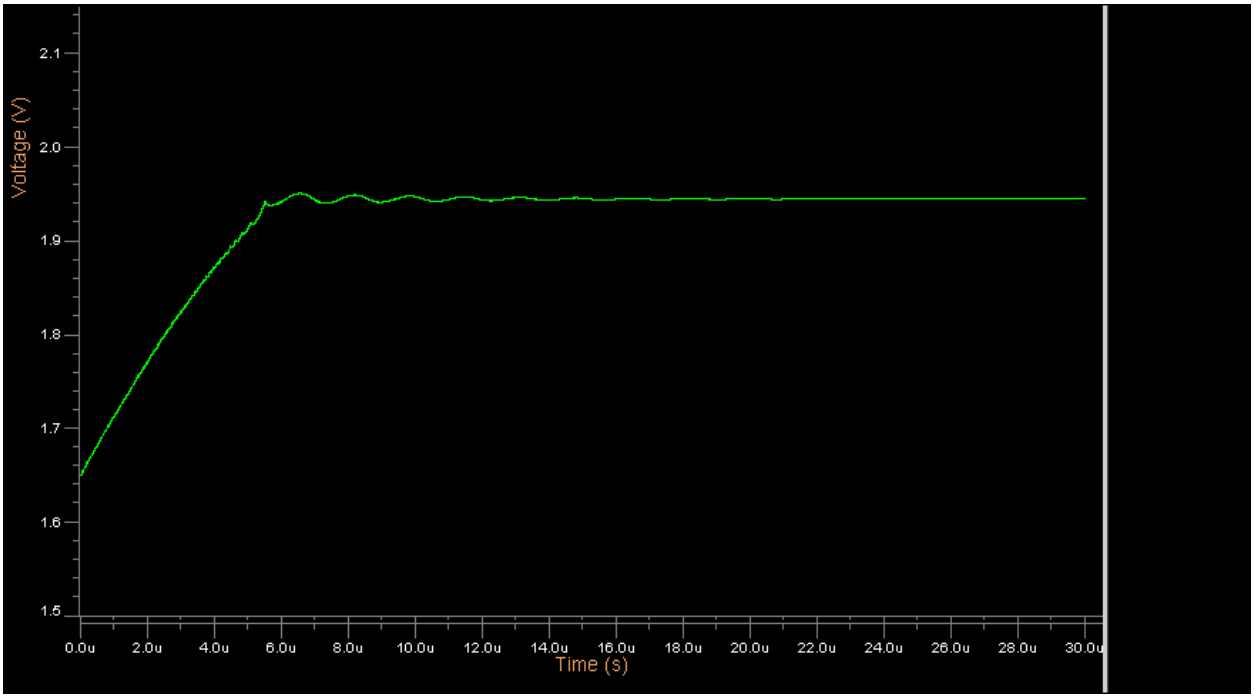


Figure 17 Control Voltage for the VCO (Zoomed) at 150 MHz (SystemVision results)

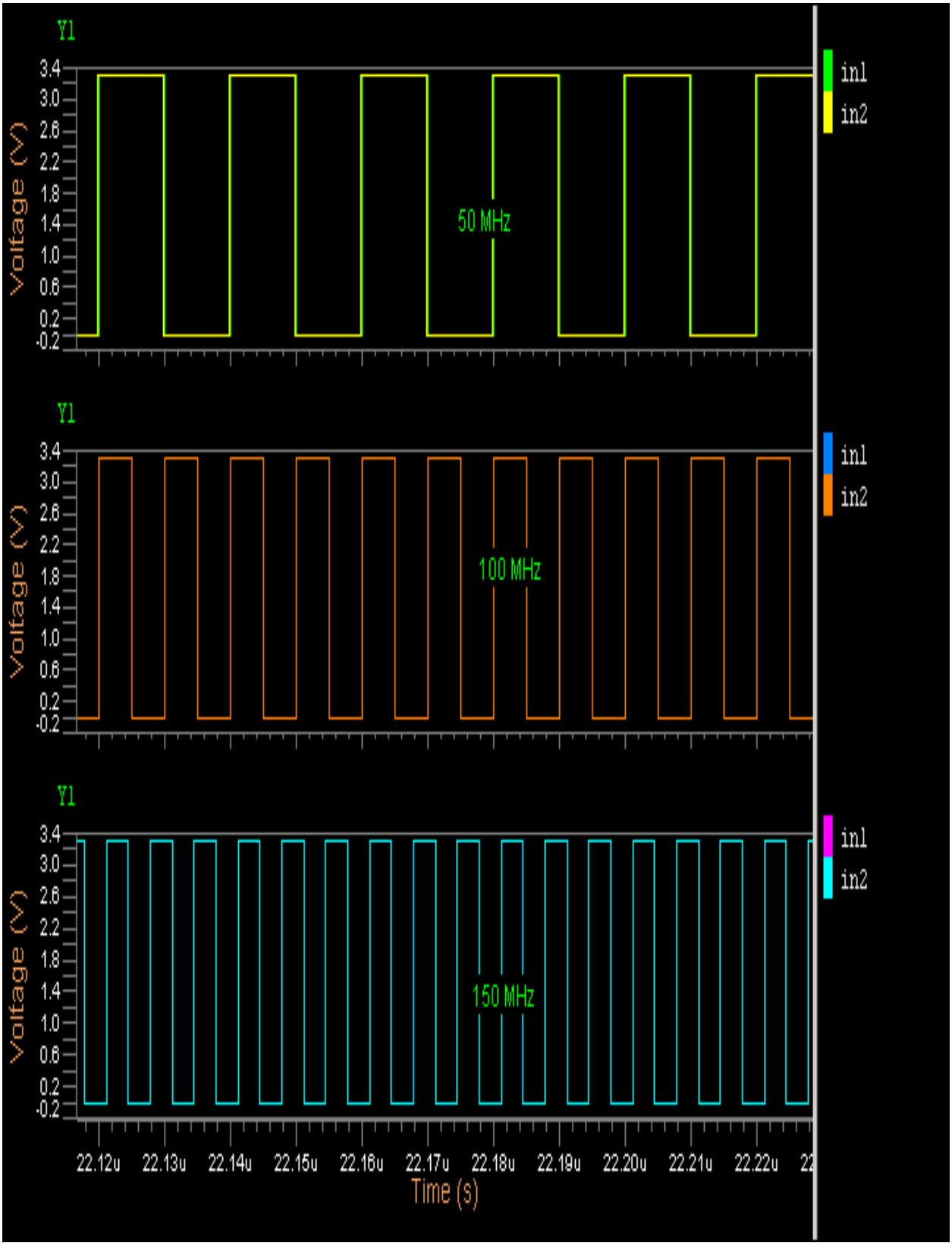


Figure 18 Combined results for the entire operating range of the PLL

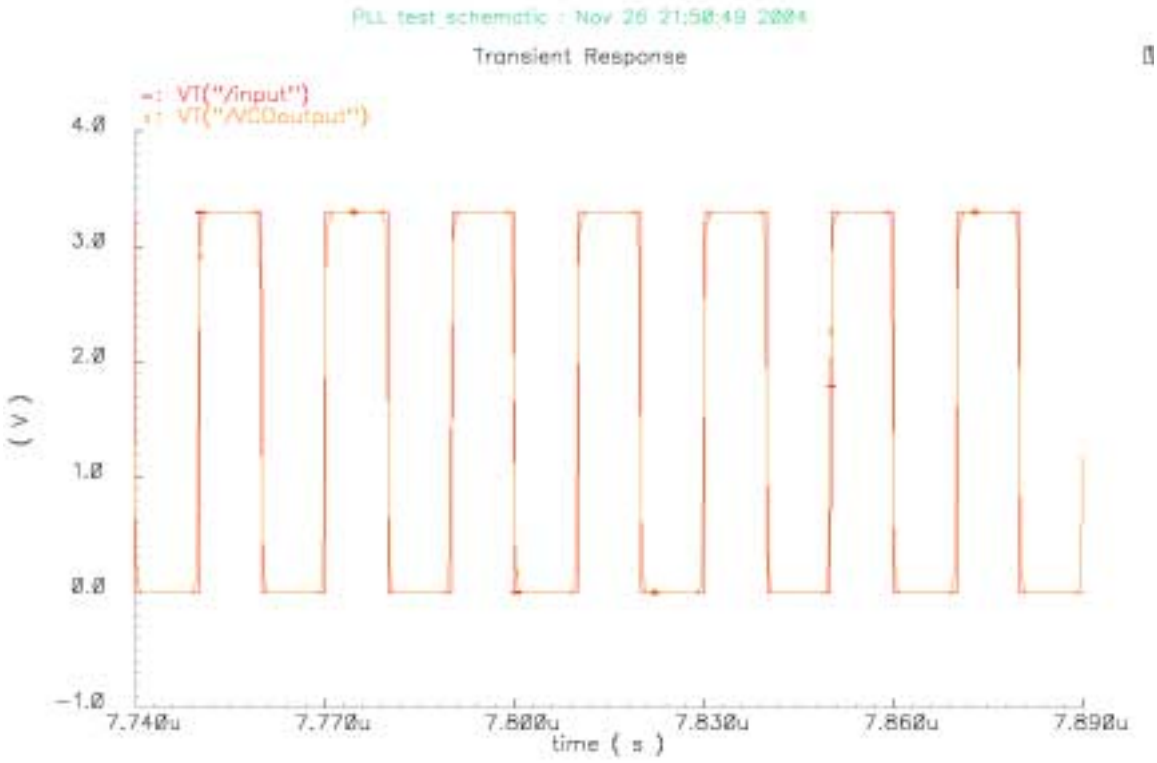


Figure 19 (Zoomed) Input, Output PLL waveforms at lock @ 50 MHz (Cadence results)

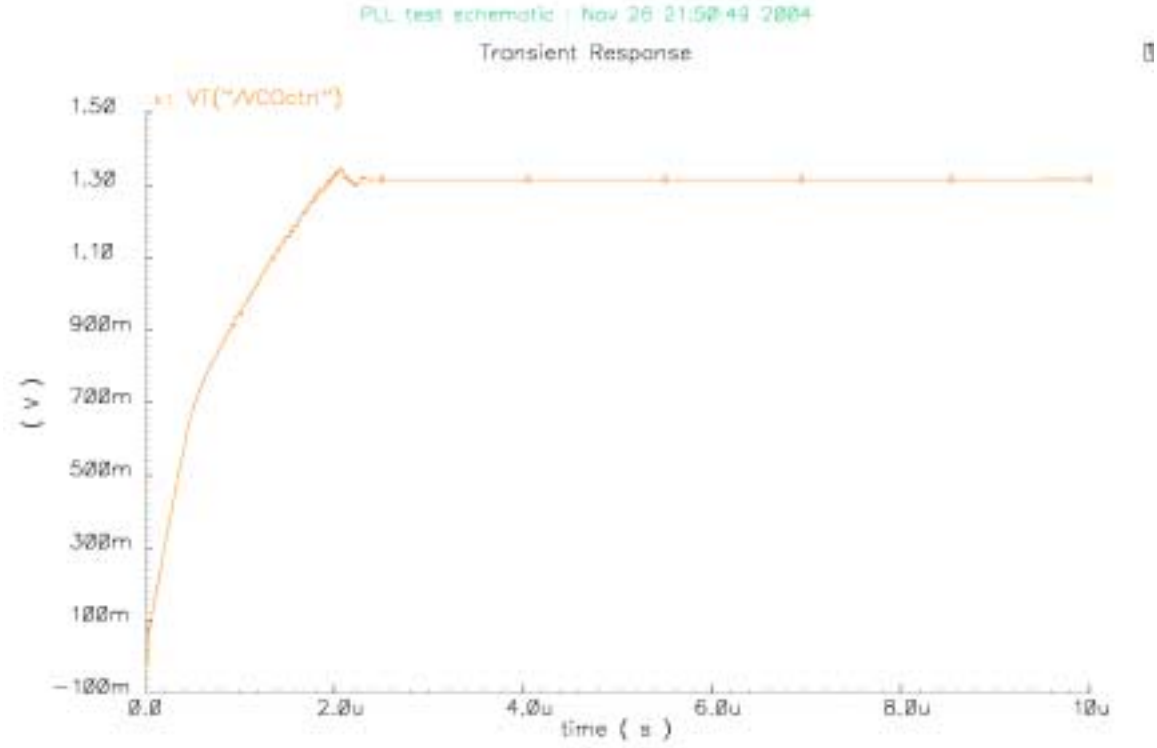


Figure 20 Control Voltage for the VCO (Zoomed) at 50 MHz (Cadence results)

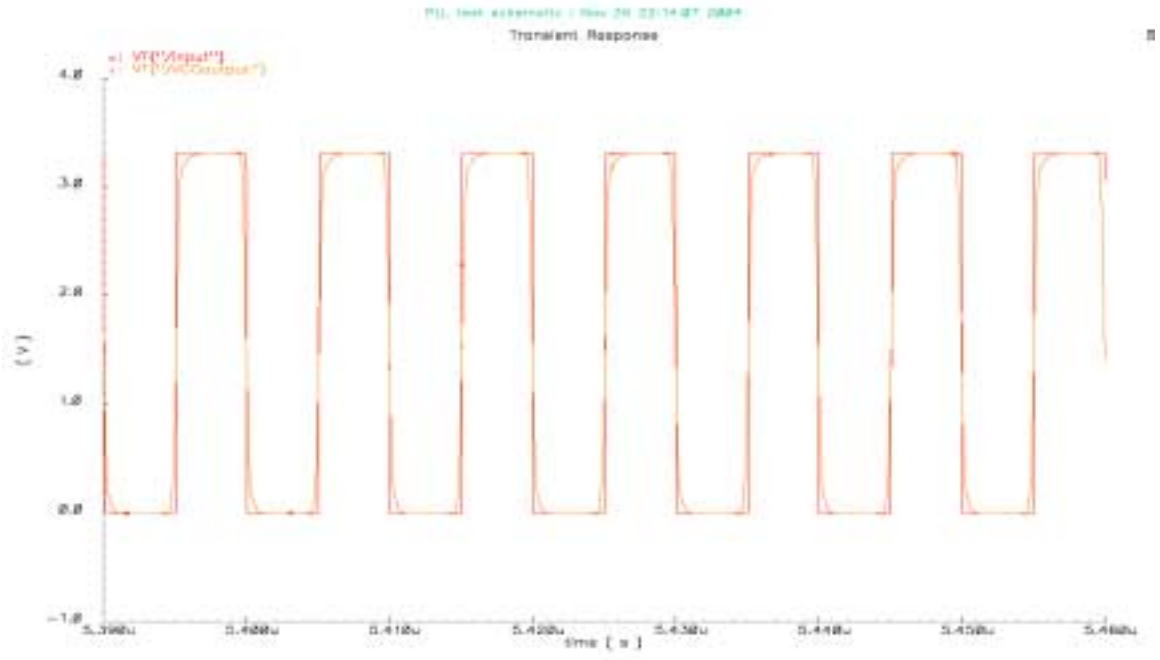


Figure 21 (Zoomed) Input, Output PLL waveforms at lock @ 100 MHz (Cadence results)

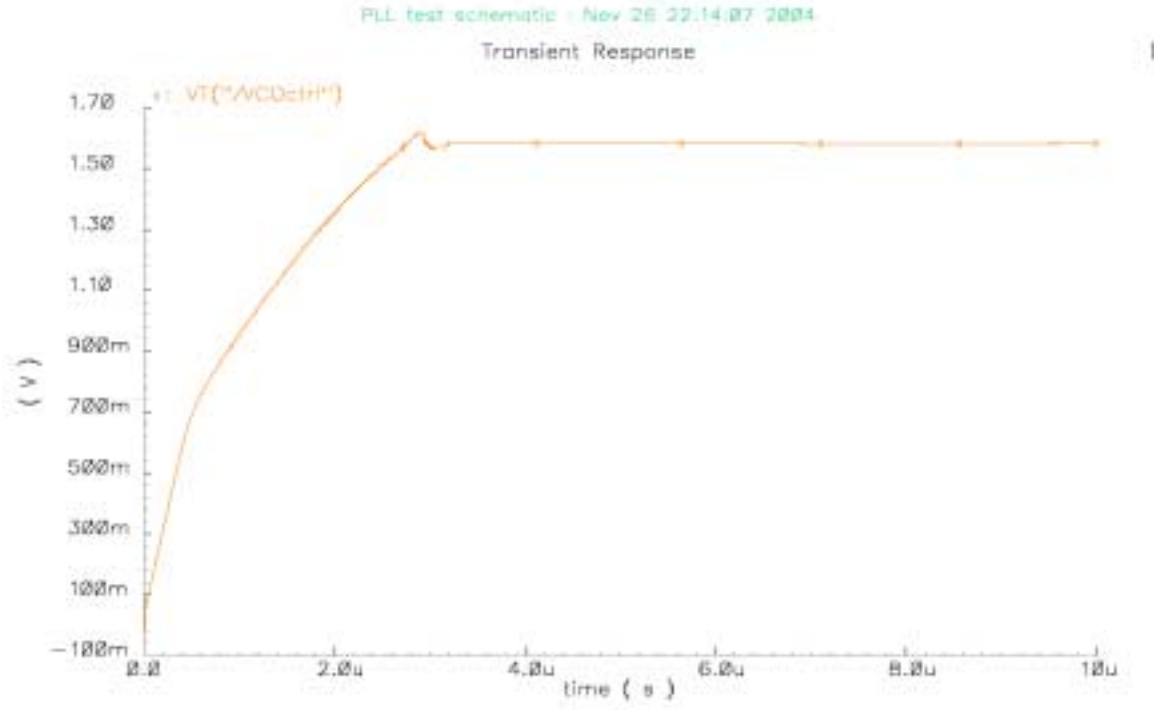


Figure 22 Control Voltage for the VCO (Zoomed) at 100 MHz (Cadence results)

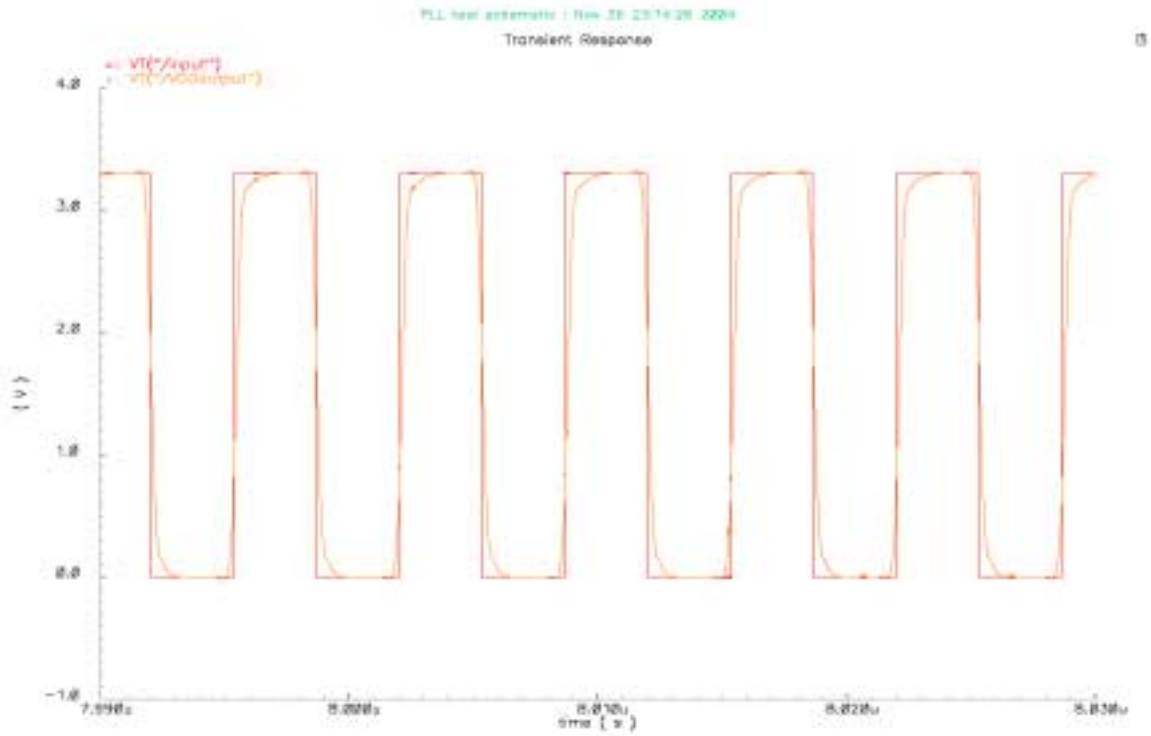


Figure 23 (Zoomed) Input, Output PLL waveforms at lock @ 150 MHz (Cadence results)

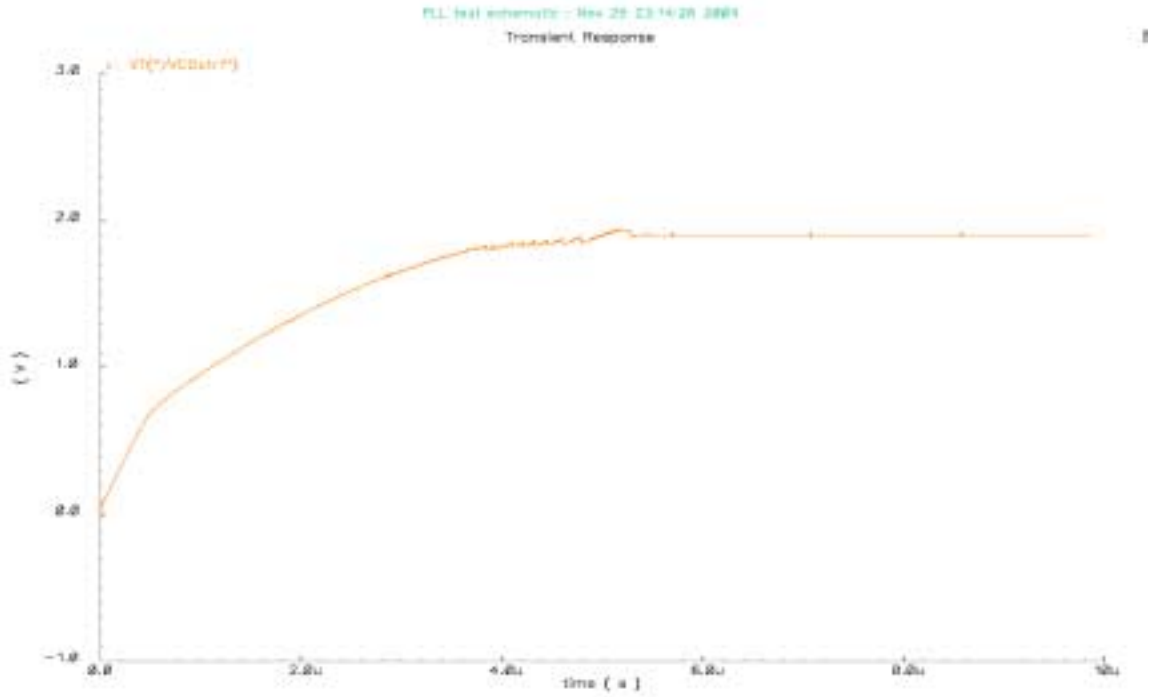


Figure 24 Control Voltage for the VCO (Zoomed) at 150 MHz (Cadence results)

Disturbance Modeling:

Noise in PLL is classified into two categories, which are amplitude noise and phase noise. Amplitude noise is detected and terminated easily. In contrast, phase noise is difficult to identify and express in an equation due to unpredictable characteristics of electronic components. It is, therefore, important to study the characteristic of the phase noise because it affects the system performance and the signal to noise ratio. Phase noise in a phase-locked-loop (PLL) is originated from each electronic component in the PLL itself. The pattern of phase noise is derived from the plot of power spectrum density in frequency domain. By using a reliable phase noise model, the output phase noise due to each noise source is, therefore, predicted correctly by calculating the relation between an input power spectrum density and its closed-loop transfer function. There are four noise sources considered in a PLL, which are generated by reference source, Phase and frequency detector, Charge pump & filter and voltage-controlled oscillator. The figure in the following page shows these noise components.

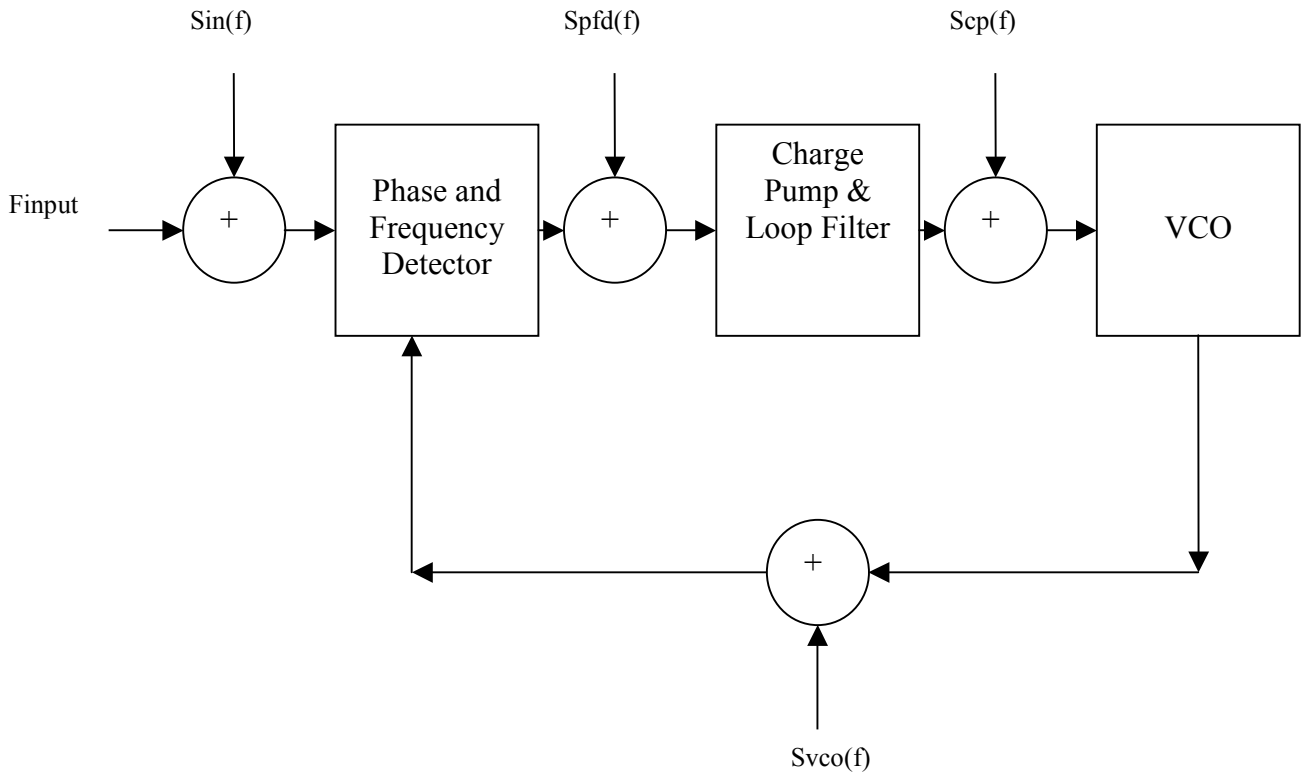


Figure 25 Phase Noise Components in PLL

Setting up a test-bench to model the phase noise would involve coming up with a power spectral density model for all the noise components. To make the task simpler, instead of a random uncorrelated noise signal, a periodic signal was chosen to represent any external disturbance/process variations. A test bench was setup in SystemVision that modeled the addition of this disturbance to the measured signal at the VCO output. Simulations were run with varying amplitudes and frequencies for this signal and the PLL dynamics were observed. The following VHDL-AMS code represents this block.

```
-- The following module describes the disturbance block to model process
-- variations in the VCO output signal


library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

entity disturbance is

  port (
    terminal v_in_p,v_in_n,v_out_p,v_out_n : electrical);
end entity disturbance;

architecture behavioral of disturbance is

quantity vout across iout through v_out_p to v_out_n;
quantity vin1 across v_in_p to v_in_n;
quantity phi : real;  -- phase of the disturbance signal
quantity vin2: real:=1.0; -- disturbance signal
quantity vout_temp : real; --output

begin

if domain = quiescent_domain use
  phi     == 0.0;
  else

          phi'dot == 2000.0e6; -- frequency

end use;

  vin2 == 0.61 * cos(math_2_pi*phi);
  vout_temp == vin1 + vin2 ;

  if vout_temp > 0.0 use
```

```
vout == 3.3;
else vout == 0.0;
end use;

end architecture behavioral;
```

Simulation Results:

Input disturbance signals ranging from 50 MHz to 2 GHz were applied with a VCO output signal of 150 MHz. The amplitude of this disturbance signal was varied from 1.5% to 25% of the amplitude of VCO output. While on the application of lower amplitude signals the PLL was still able to maintain the lock at normal control voltage levels, signals with amplitude greater than ~20% of the VCO output had considerable effect in the PLL performance. The following figure shows this simulation result:

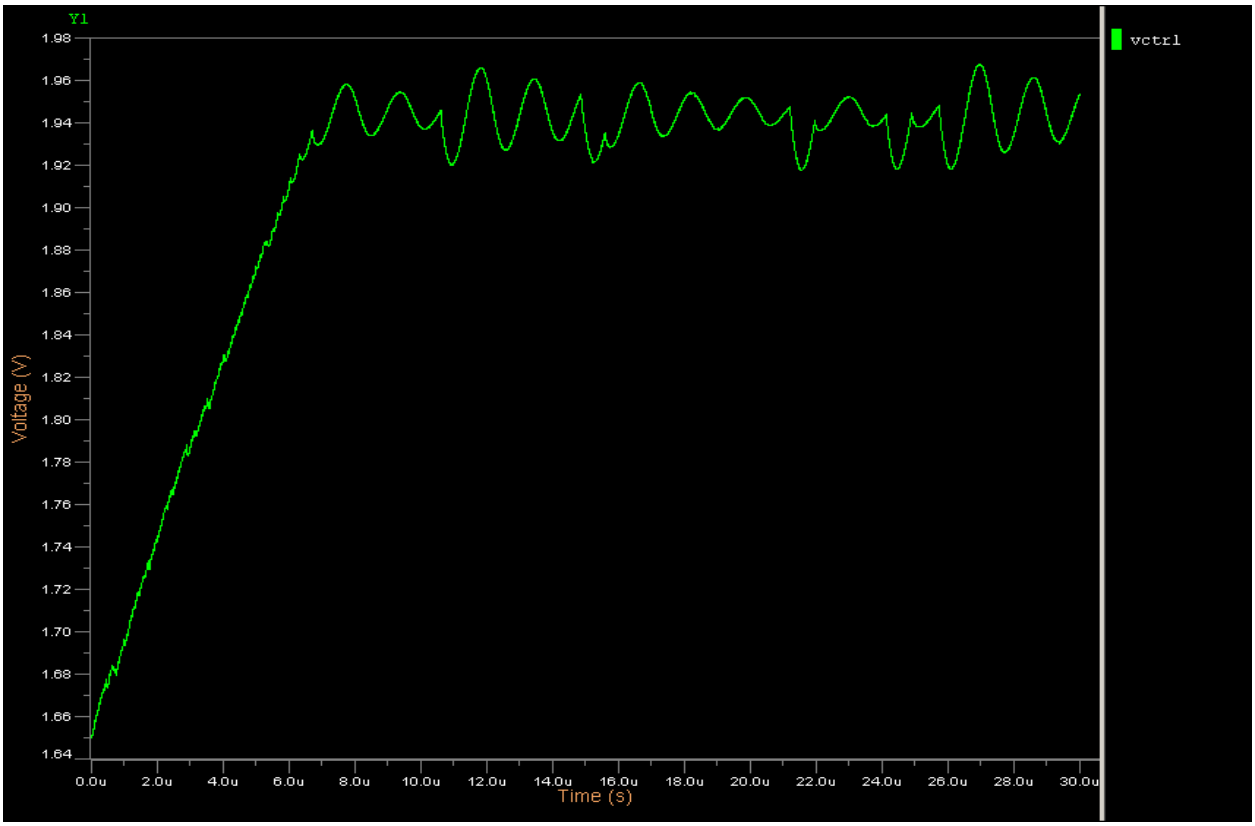


Figure 26 Vctrl for disturbance @ 2 GHz and amplitude of 0.63 (>20% of VCO output)

SystemVision Evaluation:

SystemVision is a powerful mixed mode simulation tool that provides a virtual lab for creating and analyzing analog, digital, and mixed-signal systems. It provides extensive support for industry-standard languages such as VHDL-AMS, SPICE, and C.

The following are the salient capabilities of the language:

- You can simulate VHDL-AMS models, SPICE models, or combinations of both.
- The models themselves can be a mix of SPICE subcircuits and VHDL-AMS language descriptions.
- Design verification of hierarchical schematic and circuit elements could be done through block diagrams and transfer functional blocks using SystemVision.
- SystemVision provides a model library (EduLib) that contains representative devices and effects spanning several technologies. Electrical (analog/digital), mechanical, hydraulic, magnetic, and thermal models are included.
- SystemVision can generate symbols automatically for user-created models.
- Hierarchical design methodologies and multi-page schematics are supported.
- Full mixed-signal design is supported, including both digital and analog buses.
- Design data is organized and managed using a project-oriented approach.

While SystemVision has several capabilities that support convenient implementation of mixed signal design, it also has some implementation issues such as the following:

- Any change made to the port definition of an already created symbol is not updated unless all the previous symbol information is deleted from the SystemVision file structure and a new symbol is created. This creates a lot of confusion while making modifications to a module during the design.
- Many of the error messages are not user friendly.
- Some network computers may require administrative or power user privileges in order to invoke SystemVision. This is because of a dependency on writing the Windows registry.

Conclusion:

A mixed-signal behavioral model of a transistor-level block is an abstract, simulatable representation that exhibits the characteristics of most interest to the designer while suppressing irrelevant physical detail. Suppressing detail speeds simulation. The designer can accelerate simulation runs by replacing one or more transistor-level blocks with behavioral model equivalents. The test plan that judiciously incorporates behavioral models will allow more simulation runs and hence superior coverage. We call this test strategy bottom-up verification. For example, while doing the design of the charge pump module, the impedance seen at the output of the charge pump is an important parameter that will affect the PLL performance. While detailed schematics design of such a charge pump block entails in-depth specifics about transistor sizing and biasing, the alternative behavioral design presents a feasible way around this complex route wherein a lumped impedance model could be used for much faster simulations. Similar behavioral constructs could be used for other blocks making the design simpler, yet as accurate as the structural transistor level design.

In spite of the potential benefits, these barriers slow the adoption of behavioral modeling in the verification flow:

1. Although it is easy to develop idealized system level behavioral models, it is far more difficult to create models that exhibit an appropriate set of physical effects and suppress the rest. System-level models are useful in a top-down design flow, but the more detailed models are required for bottom-up verification. Structural models of a MOS device using VHDL-AMS are probable building blocks for such a design. Spice3 or BSIM3 models, that have numerous parameters, provide accurate transistor behavior representation in various operating ranges.

2. A detailed mixed-signal behavioral model will have many generic parameters to specify such matters as operating frequencies. Each instance of the model is particularized by specifying fixed values for the parameters. An instance of a model used in a bottom-up verification strategy must be tuned, or calibrated, by the appropriate selection of parameter values to match the transistor-level block it is intended to replace.

Although top-down design is the chosen methodology for implementing mixed-signal designs in the industry, design flows in academic settings are influenced more by academic and tape-out deadlines. Therefore designing a transistor level implementation and the corresponding layout on time becomes more important than coming up with a system level behavioral model in the first place. Moderate fabrication costs in an academic setting further encourage the adoption of this kind of a design flow.

With this in mind, the initial design for the PLL was done at a schematic level, and the behavioral model was later developed and calibrated against this schematic implementation, based on the results of the simulations performed.

Future scope of work:

1) A detailed Spice or BSIM model of MOS could be used to write a structural description of the PLL. Such an approach would help the bottom up verification of the design, and would make it more accurate.

2) A detailed noise analysis could be carried out and an accurate phase noise model could be developed. This would help predict correctly the relationship between the input power spectrum density and the closed loop transfer function.

3) The PLL can be made programmable and can be designed to have a wider lock range by making some modifications to the design. This would necessitate the following design extension requirements:
   a) A programmable filter that can be selected based on the range of input frequency.
   b) A frequency meter at the input that tells an approximate value of the input frequency.
   c) A VCO designed to have a large bandwidth.
   d) A divide by N circuit that can be multiplexed to choose the desired operating range.

4) To eliminate power supply noise we can use a circuitry which will generate a Vref(a power supply independent reference generator).A combination of a differential VCO and an op-amp at the output of the differential VCO will further help in reducing the noise problems.

References:

[1] Peter J. Ashenden, Gregory D. Peterson, Darrell A. Teegarden, *The system designer's guide to VHDL-AMS: analog, mixed-signal, and mixed-technology modeling*, Morgan Kaufmann, 2003.

[2] Behzad Razavi, *Design of analog CMOS integrated circuits,* McGraw-Hill, 2001

[3] Neil Weste and Kamran Eshraghian, *Principles of CMOS VLSI design: a systems perspective,* Addison-Wesley, 1985

[4] Ayman Mounir, Ahmad Mostafa, Maged Fikry, *"Automatic Behavioral Model Calibration for Efficient PLL System Verification",* Mentor Graphics, Egypt

[5] Web reference: http://www.elecdesign.com

[6] SystemVision User's Manual, Version 3.2, Mentor Graphics, July 2004