

DESIGN AND IMPLEMENTATION OF A PHASE LOCKED LOOP FOR  
HIGH-SPEED SERIAL LINKS

BY

RUSHABH MEHTA

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Electrical and Computer Engineering  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor José Schutt-Ainé

# ABSTRACT

Recent advances in the semiconductor industry and process technology scaling have increased the demand for fast, robust computing. The thirst for high-processing, low power ICs is ever increasing. This has pushed the demand for high data rates in wireless and wireline communication systems in the multi-Gbps range. With higher data rates, the I/O links need to scale proportionally. However, the I/O channel bandwidth has not scaled appropriately making it the biggest bottleneck in high-speed links. Parallel links have not been able to match this increasing system performance due to issues such as crosstalk, timing skew and packaging costs. Thus there is a need for high-speed serial links. For high-speed transmission of data, there arises a need for high-speed on chip clocking circuits making the use of Phase-Locked Loops (PLLs) imperative.

This thesis includes an overview of high-speed links along with the need for PLLs. An in-depth understanding of PLL theory, loop dynamics and behavioral and transistor level simulation follows. Performance metrics such as phase noise, random jitter and deterministic jitter are discussed. Finally, this thesis concludes with an insight into All Digital Phase-Locked Loops (ADPLLs).

*To my family and friends, for their love and support*

# ACKNOWLEDGMENTS

I have attended UIUC for six long years, four years as an undergraduate student and two years as a graduate student. I have been involved in research for the past three years. This journey was hard and at the same time successful. I encountered countless obstacles which would not be solved if I did not have the support of numerous people along the way. I take this opportunity to thank all the wonderful individuals in my life who have made this journey memorable and fun.

First and foremost, I would like to thank my parents, Ravi Mehta and Amita Mehta, for giving me this opportunity. Without them, I would not have the chance to obtain higher education at such a prestigious university. I also want to thank my brother Rohit Mehta for always being there by my side.

Secondly, I want to thank my advisor Professor José Schutt-Ainé for being the guiding light on my journey. It was under his direction that I took my first circuits course and he trusted my abilities to conduct research under his guidance. His passion for work motivated me to work even harder. As a father figure to me, he has always encouraged and guided me throughout my college journey. He will always continue to inspire me in every aspect of life.

Next, I want to express gratitude to Professor Pavan Kumar Hanumolu for helping me throughout my thesis. His valuable inputs helped me complete my project on time. I am extremely grateful to Professor Steven Franke, Professor Chandrashekhar Radhakrishnan, Professor Songbin Gong, Professor Serge Minin, Professor Jennifer Bernhard, Professor Milton Feng and Professor Elyse Rosenbaum for supporting and mentoring me throughout my undergraduate and graduate career.

I would like to take this opportunity to thank my colleagues and friends in graduate school: Da Wei, Yubo Liu, Sabareesh Ravikumar, Ankit Jain, Xinying Wang, Jerry Yang and Zexian Li for always answering my questions



whenever I encountered problems in my thesis.

A special thanks goes to all my wonderful friends who have made the hardest of moments look the easiest: Rishi Ratan, Maryam Hajimiri, Ishita Bisht, Aarti Shah, Yi Ren, Danny Coombs, Brady Salz, Sarah Shim, Abhiyudai Bhakuni, Rohan Mehra, Anmol Shrivastava, Sakshi Shrivastava, Shashank Saxena, Bhagya Parekh and Akash Kotak.

Lastly, I am very grateful for getting the chance to work with some extremely smart undergraduate students, Alexandra Wleklinski, Ashwarya Rajwardhan and Paprapee Buason.

# TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION . . . . .	1
1.1	Motivation . . . . .	1
1.2	Purpose . . . . .	2
1.3	Outline . . . . .	2
CHAPTER 2	HIGH-SPEED SERIAL LINKS . . . . .	3
CHAPTER 3	PHASE-LOCKED LOOP CIRCUITS . . . . .	7
3.1	What Is a PLL? . . . . .	7
3.2	Building Blocks . . . . .	8
CHAPTER 4	PLL LOOP DYNAMICS . . . . .	14
4.1	Design Procedure and Parameter Calculation . . . . .	16
4.2	PLL Bandwidth . . . . .	16
4.3	PLL Locking Verification . . . . .	17
4.4	PLL Noise Analysis . . . . .	18
CHAPTER 5	PLL DESIGN CHOICES . . . . .	20
5.1	PFD . . . . .	20
5.2	Charge Pump . . . . .	21
5.3	Loop Filter . . . . .	21
5.4	Voltage Controlled Oscillator . . . . .	22
5.5	Divider . . . . .	23
CHAPTER 6	BEHAVIORAL LEVEL SIMULATION . . . . .	24
6.1	Behavioral Modeling . . . . .	24
6.2	PLL Analysis . . . . .	39
CHAPTER 7	TRANSISTOR LEVEL SIMULATION . . . . .	41
7.1	What Is Cadence Virtuoso? . . . . .	41
7.2	Simulator . . . . .	41
7.3	Transient Simulations . . . . .	41
7.4	Cadence Virtuoso: Getting Started . . . . .	41
7.5	PLL Simulation Using Cadence Virtuoso . . . . .	53

CHAPTER 8 DISCUSSION . . . . .	71
8.1 Conclusion . . . . .	71
8.2 Future Work . . . . .	71
REFERENCES . . . . .	73

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

A sudden advancement in the semiconductor industry has resulted in technology scaling which requires faster data rates and necessitates high-speed clocks in the multi-GHz range. With the ever increasing data rates, the I/O links need to scale appropriately. However, the I/O bandwidth has not scaled proportionally and has created a bottleneck in the system performance. Parallel links have not been able to match up with the system performance with issues such as packaging costs, data skew and crosstalk. The rising data rates are pushing the tolerance boundaries for timing skew and crosstalk. Looking at the recent trends, ethernet data rates are advancing from 100 Mbps to 10 Gbps. PCI Express 2 is going from 5 Gbps to 8 Gbps in PCI Express 3. Serial ATA has increased from 1.5 Gbps to 6 Gbps in computing applications. Thus there is a need to condense the parallel buses to serial buses. The serial buses have less crosstalk, data and clock skew as well as packaging costs. This gives rise to the importance of High-Speed Serial Links (HSSL). HSSL are used in modern-day systems. An example is the conversion of the parallel SCSI bus to Serially Attached SCSI (SAS).

With skyrocketing data-rate interfaces, the need for clock frequencies in the multi-GHz range is paramount and requires the use of Phase-Locked Loops (PLLs). PLLs have a wide variety of use in analog, digital and Radio Frequency (RF) communication systems. PLLs are capable of generating high-frequency, low-jitter clocks with minimal timing skew. Besides, PLLs are also used in Clock and Data Recovery (CDR) circuits to reliably recover the data sent from the transmitter.

## 1.2 Purpose

The purpose of this thesis is to provide an in-depth understanding of the theory and working of PLLs. PLL theory will be discussed in detail before moving on to behavioral and transistor-level simulations using Cadence Virtuoso. Finally, figures of merit of the PLL will be documented and a discussion on design improvement will follow.

## 1.3 Outline

1. Chapter 1 provides an introduction to High-Speed Serial Links (HSSL).
2. Chapter 2 provides an overview of the HSSL along with the effects of the channel with increasing data rates and use of serial links in modern-day communication systems.
3. Chapter 3 presents an in-depth analysis of the theory of the different building blocks of the PLL.
4. Chapter 4 describes the loop dynamics of the PLL along with a focus on PLL noise. Finally, a design procedure is documented for calculating the loop parameters required for the PLL to lock with optimum bandwidth (BW), phase margin and settling time.
5. Chapter 5 describes the topologies chosen for the different blocks of the PLL.
6. Chapter 6 describes the procedure for behavioral-level modeling of the PLL using VerilogAMS.
7. Chapter 7 describes the procedure for transistor-level modeling of the PLL using Cadence Virtuoso.
8. Chapter 8 concludes the thesis with an emphasis on future work and an insight into ADPLLs.

# CHAPTER 2

## HIGH-SPEED SERIAL LINKS

A high-speed serial link model is shown in figure 2.1 [1]. Parallel data comes into the serializer converting it to serial data. This data is fed to the Transmitter (TX). The PLL generates the master clock to drive the serializer and the transmitter. The transmitter generates a series of pulses and sends the encoded clock and data across the channel. To counteract the effects of the lossy channel, there is an equalizer present on the receiver side. The received data stream is sent to the Clock and Data Recovery (CDR) circuit. The CDR recovers the clock sent from the TX side and uses that clock to resample and recover the data. The resampled serial data is sent to the deserializer where it is converted back to parallel data.

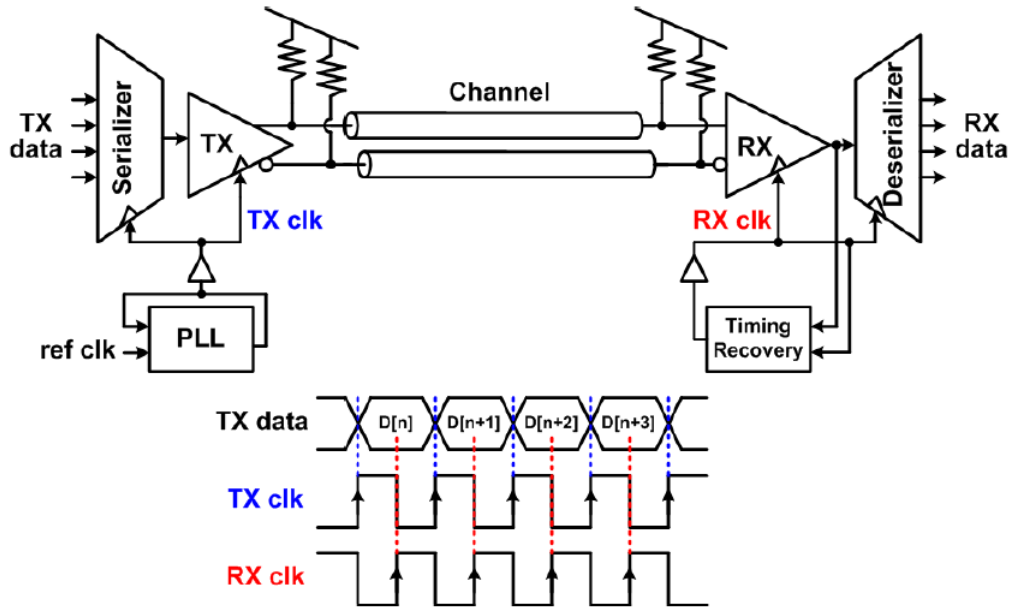


Figure 2.1: HSSL Block Diagram

Data rates are increasing with technology scaling, however the channel BW has not scaled due to many effects. Figure 2.2 shows a roadmap of increasing

data rates and the target channel BW.

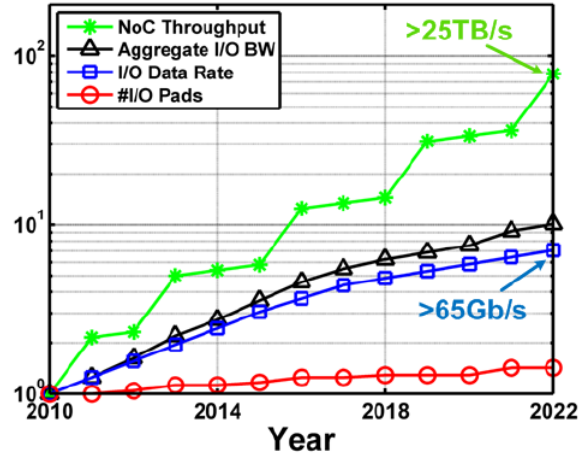


Figure 2.2: Data Rates and Required Channel BW

A typical backplane channel is shown in figure 2.3. The channel is an electrical path between the TX and RX and consists of vias, connectors and PCB traces. The channel and its effects can be modeled by measuring its S-parameters using a Vector Network Analyzer (VNA) or using a computational electromagnetic software such as Ansys HFSS. Once the channel is modeled, there is a need for a high-fidelity, robust communication system that can counteract the effects of the channel, such as crosstalk, substrate loss and impedance mismatch between connectors. Besides, it needs to consume low power and occupy the least possible area [1].

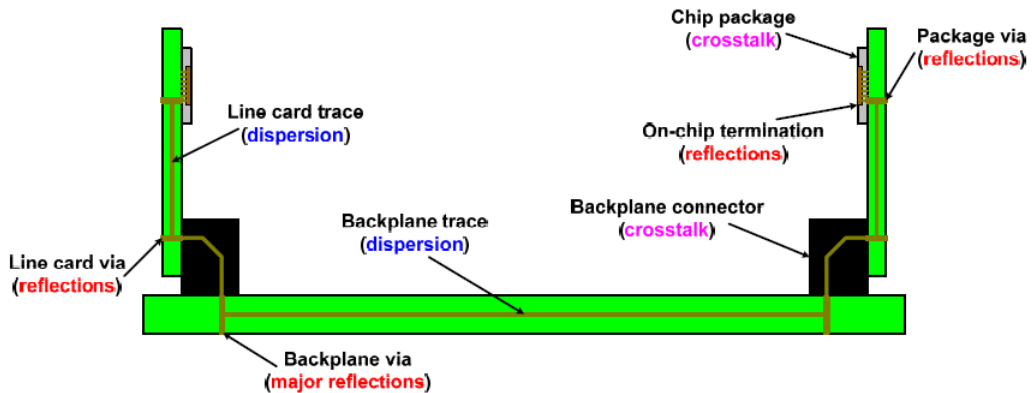


Figure 2.3: Backplane Channel Interface

Figure 2.4 shows the channel response and eye diagrams for different data rates.

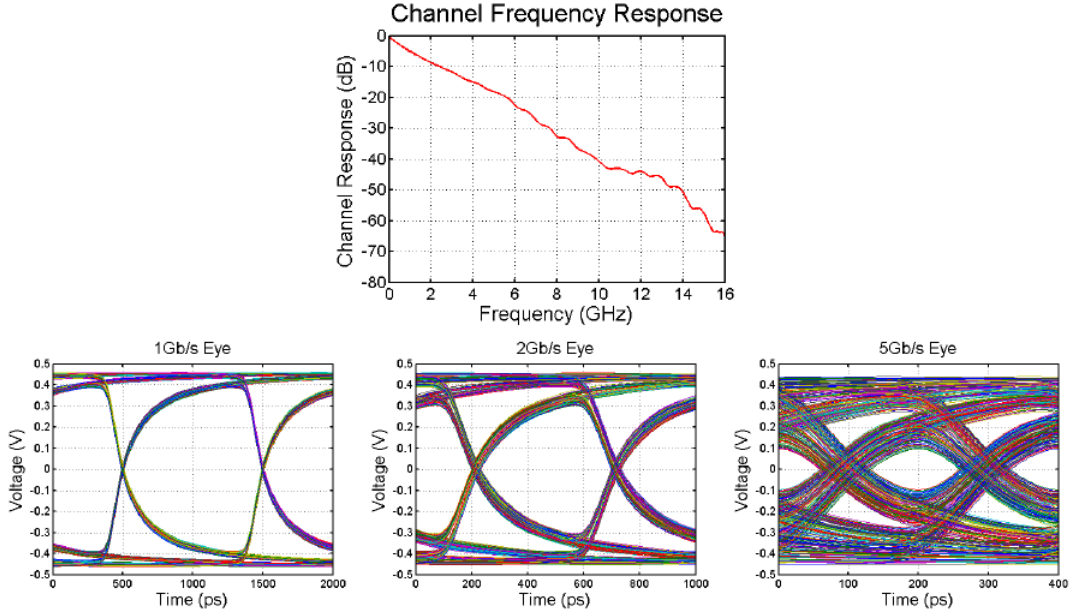


Figure 2.4: Channel Response and Eye Diagrams for Different Data Rates

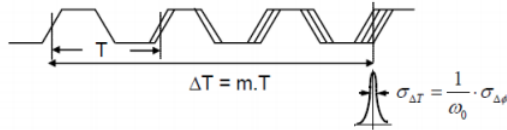


Figure 2.5: Timing Jitter

The attenuation caused due to the many effects in the channel increases with increasing frequencies. Notice how the eye virtually closes for increasing data rates. Thus the signal received after the channel has a lower Signal-to-Noise Ratio (SNR) and higher timing jitter. Figure 2.5 explains the concept of timing jitter. To minimize the degradation of signal quality, it is important to design TX clocks with minimum timing skew and RX with minimum sampling errors to recover the signal with maximum integrity.

Serial links eliminate the use of multiple pins and hence reduce the cost which is an added burden in parallel links. Also, parallel links encounter interference between adjacent channels when data is being transmitted. This effect is called crosstalk and worsens as the data rate increases. Crosstalk can lead to erroneous data reception and is a huge bottleneck for signal integrity. This does not occur in serial links due to the absence of multiple channels. Another effect known as data skew is eliminated in serial links. When data



is transmitted at high speeds across a parallel link interface, it is possible that there arises a difference in the arrival time of the data at the receiver, which potentially leads to data unreliability [2].

Figure 2.6 illustrates the widespread applications of serial links in the consumer electronic industry. Some examples are provided in the following list [1]:

- Processor-to-memory: RDRAM (1.6 Gbps), XDR DRAM (7.2 Gbps), XDR2 DRAM (12.8 Gbps), GDDR5 (8 Gbps), DDR4 (3.2 Gbps)
- Processor-to-peripheral: PCIe (2.5, 5, 8, 16 Gbps), Infiniband (2.5 - 25 Gbps), USB3.1 (10 Gbps)
- Processor-to-processor: Intel QPI (9.6 Gbps), AMD Hypertransport (6.4 Gbps)
- Storage: SATA (16 Gbps), Fiber Channel (25 Gbps)
- Networks: LAN Ethernet (4 x 25 Gbps), WAN SONET (2.5, 10, 40 Gbps), Backplane Routers (2.5 - 25 Gbps)

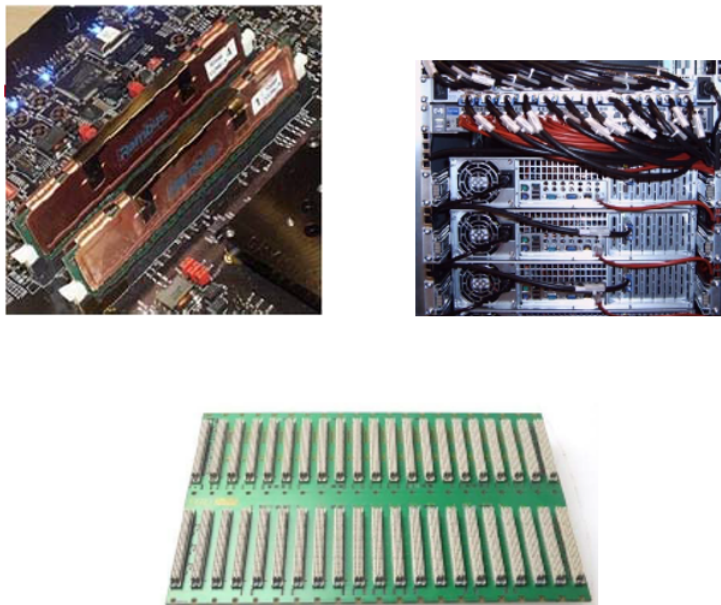


Figure 2.6: Serial Link Applications

# CHAPTER 3

## PHASE-LOCKED LOOP CIRCUITS

### 3.1 What Is a PLL?

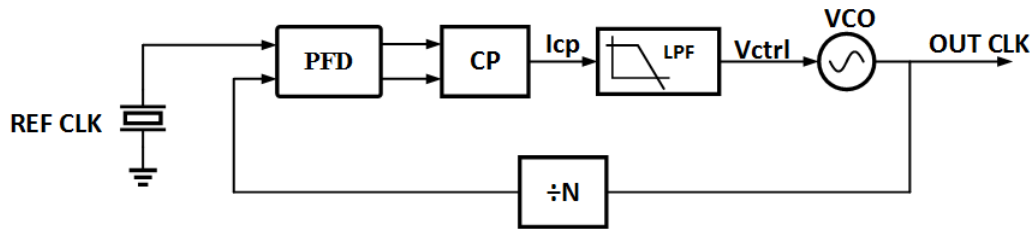


Figure 3.1: PLL Circuit

A Phase-Locked Loop (PLL) is a negative feedback control system that generates a high-frequency output clock whose phase is related to the low-frequency input clock. The need for PLLs is widespread as they find application in analog, digital, RF and communication systems. The low-frequency clock can be generated using a crystal with frequencies up to 200 MHz. Generating a high-frequency clock with high spectral purity is not possible with a crystal and this is where PLLs come into play. They generate a high-frequency clock with as low jitter as possible.

A simple PLL consists of a Phase and Frequency Detector (PFD), Charge Pump (CP), Loop Filter (LF), Voltage Controlled Oscillator (VCO) and a Divider. The operation of the PLL is as follows. The PFD tracks the phase and frequency difference between the reference clock and the divided output clock. The CP outputs a current that is proportional to this difference. These current pulses are fed to the LF which generate a control voltage to be fed to the VCO. The control voltage directs the VCO to generate a high-frequency output clock which is divided through the divider and fed back to the PFD.

The loop is said to be locked when the reference frequency and the divided output clock frequency are matched [3].

## 3.2 Building Blocks

The five main building blocks of the PLL are:

- Phase and Frequency Detector
- Charge Pump
- Loop Filter
- Voltage Controlled Oscillator
- Divider

### 3.2.1 Phase and Frequency Detector

The phase and frequency detector detects the phase and frequency difference between the reference clock and the divided output clock and produces a signal that is proportional to the difference between the two phases. This error signal needs to remain constant with time for the loop to be locked. Thus the PFD serves as an error detector to detect and minimize the error as the lock state approaches.

A three-state PFD is required to track both phase and frequency due to its asymmetrical transfer characteristics. A three-state PFD consists of an UP, DN and RST state. When the reference signal is high, UP goes high. When the divided signal is high, DN goes high. When both the reference and divided signal are high, RST is activated, which resets the UP and DN signal and both go low.

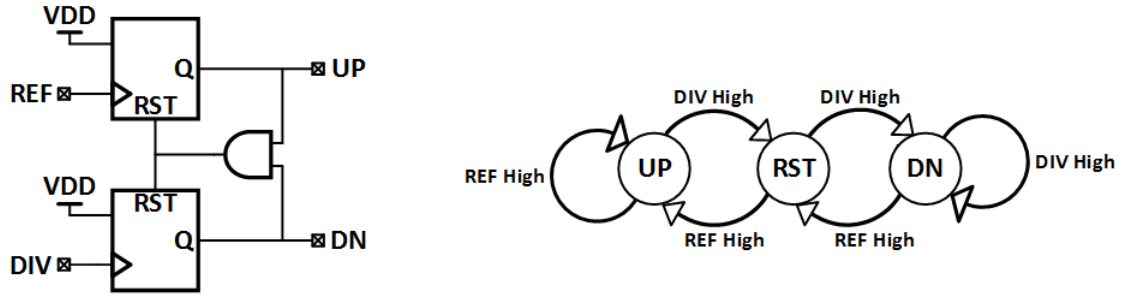


Figure 3.2: PFD Functionality

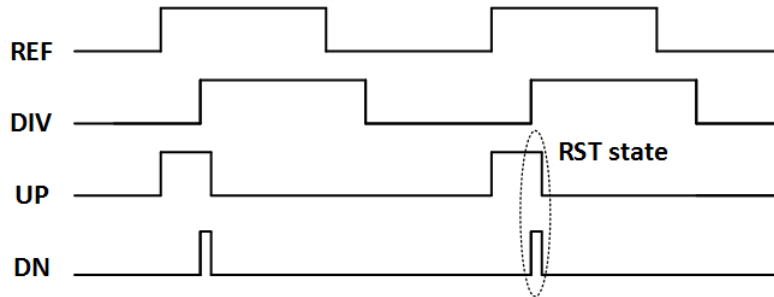


Figure 3.3: PFD Operation

Figures 3.2 and 3.3 describe the operation of the PFD. When the reference clock leads the divided output clock, UP is 1. When the divided output clock leads the reference clock, DN is 1. When both UP and DN are 1, the reset path is activated and pulls the UP and DN pulses to 0. The width of the UP and DN pulses depends on the delay in the reset path. It is required to have a minimum delay before resetting these pulses in order to avoid a condition called “dead zone” which will be discussed in section 3.2.2.

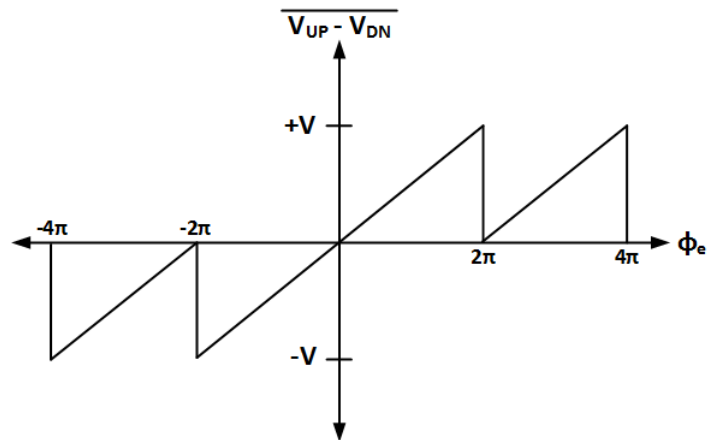


Figure 3.4: PFD Transfer Function

Figure 3.4 shows the transfer function of the PFD with a linear range of  $\pm 2\pi$ . The gain of the PFD is given by:  $K_{PFD} = \frac{V}{2\pi}$ .

### 3.2.2 Charge Pump

The charge pump converts the digital pulses from the UP and DN signals to current pulses which are fed to the loop filter to generate the control voltage.

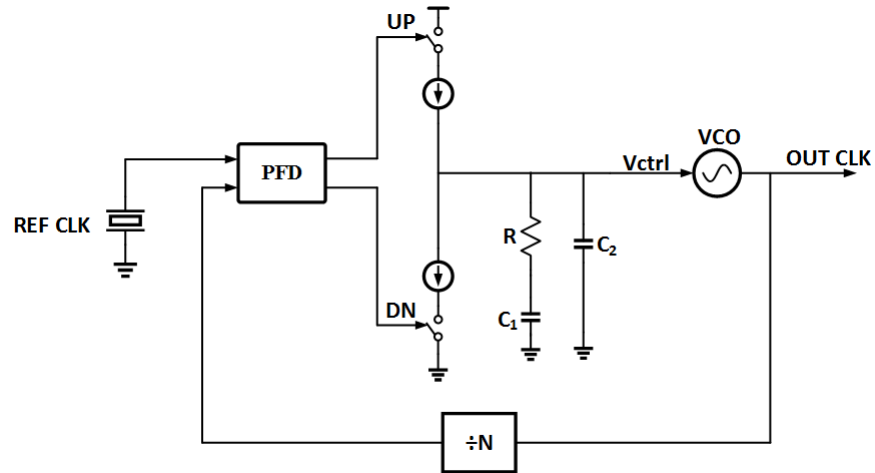


Figure 3.5: Charge Pump PLL

Figure 3.5 shows a block diagram of a charge pump PLL. The charge pump and capacitor  $C_1$  form an integrator. This introduces a pole which makes the loop unstable. Hence the resistor is added to stabilize the loop by introducing a proportional path.

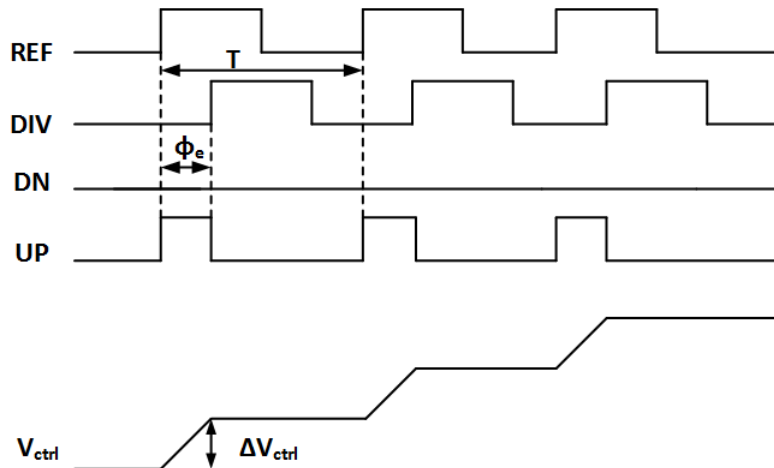


Figure 3.6: Charge Pump Operation

Figure 3.6 shows the operation of the charge pump. When the UP signal is high,  $V_{ctrl}$  will steadily increase.

The step increase in  $V_{ctrl}$  is given by:

$$\Delta V_{ctrl} = \frac{I_{CP}}{C_1} \cdot \frac{\phi_e}{2\pi} \cdot T \quad (3.1)$$

The average charge dumped per cycle is:

$$Q_{ctrl} = \frac{C_1 \cdot \Delta V_{ctrl}}{T} = \frac{I_{CP}}{2\pi} \cdot \phi_e \quad (3.2)$$

Thus the PFD and charge pump can be modeled together by the following relation in the locked state:

$$K_{PFD} = \frac{I_{CP}}{2\pi} \quad (3.3)$$

When the UP and DN pulses have a small width i.e. when the phase error is small, it is possible that the charge pump will not output any current proportional to the phase error. This condition is called the dead zone and is undesirable. This phase error appears as jitter at the output of the PLL. Avoiding this condition requires adding a minimum delay in the reset path of the PFD such that both the UP and DN pulse width is greater than the turn on time of the CP. Figure 3.7 illustrates the dead zone condition.

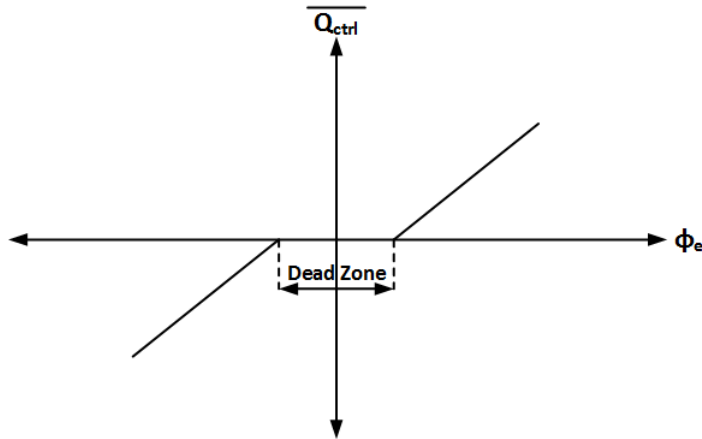


Figure 3.7: Dead Zone Condition

### 3.2.3 Loop Filter

The loop filter consists of a resistor in series with a capacitor. This combination is in parallel with another capacitor as shown in figure 3.8. Capacitor  $C_1$  adds an integrator path and introduces a pole. The resistor adds a proportional path and introduces a zero. Thus the system is stable with the addition of these two paths. Capacitor  $C_2$  is used to suppress the  $V_{ctrl}$  ripple; however the addition of an additional capacitor comprises on the stability of the PLL. The loop filter functions as a low-pass filter and rejects the input high-frequency noise of the PLL and generates the control voltage to drive the VCO.

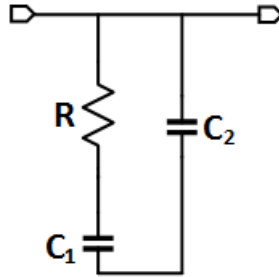


Figure 3.8: Loop Filter

The transfer function of the loop filter is given as:

$$LF(s) = \frac{s + \frac{1}{RC_1}}{C_2 s^2 (s + \frac{C_1 + C_2}{RC_1 C_2})} \quad (3.4)$$

### 3.2.4 Voltage Controlled Oscillator

The voltage controlled oscillator outputs a clock based on the control voltage supplied to it. VCOs have different architectures with two popular topologies which are the ring oscillator and the LC tank oscillator. A VCO is the biggest contributor of high-frequency noise at the output of the PLL. This translates to jitter in the time domain, and hence efforts should be made to minimize phase noise [4]. LC tank oscillators have lower phase noise; however LC tanks consume a lot of on-chip area compared to ring oscillators.

The transfer function of the VCO in the Laplace domain is derived as follows:

$$\omega_{out}(t) = K_{VCO}v_{ctrl}(t) \quad (3.5)$$

$$\mathcal{L}[\omega_{out}(t)] = K_{VCO}v_{ctrl}(s) \quad (3.6)$$

$$\phi_{out}(t) = \int_0^t \omega_{out}(\tau)d\tau = \int_0^t K_{VCO}v_{ctrl}(\tau)d\tau \quad (3.7)$$

$$\mathcal{L}[\phi_{out}(t)] = \frac{\omega_{out}(s)}{s} = \frac{K_{VCO}v_{ctrl}(s)}{s} \quad (3.8)$$

The transfer function of the VCO is given as:

$$H_{VCO}(s) = \frac{\phi_{out}(s)}{v_{ctrl}(s)} = \frac{K_{VCO}}{s} \quad (3.9)$$

where  $K_{VCO}$  is the VCO gain.

### 3.2.5 Divider

The VCO generates a high-frequency clock which when divided must be matched in frequency to the reference clock in the locked state. The divider divides the VCO clock using the same factor by which it is larger than the reference clock. The log of this factor to the base 2 indicates the number of stages to be used in the fractional-N divider circuit.



# CHAPTER 4

## PLL LOOP DYNAMICS

In this chapter we will analyze the loop dynamics of the PLL in the locked state. The PLL has a particular order and type which are determined as follows [5]:

1. The type of the PLL is determined by the number of integrators.
2. The order of the PLL is determined by the number of poles.

Every PLL must be of at least type 1 and order 1. This is because the VCO has an in-built pole and addition of other poles can be in the system. For the PLL designed, we have three poles in the system: one from the VCO and two contributed from the loop filter. Thus, we see that the order of the PLL can be determined by the number of poles in the system.

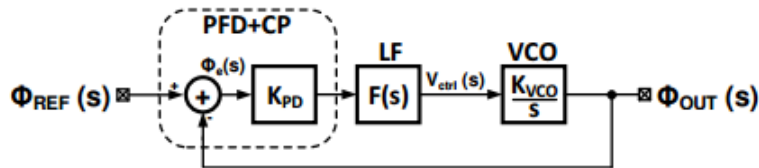


Figure 4.1: Linear Model of PLL

Figure 4.1 shows the linear model of the PLL.

The open-loop transfer function is:

$$LG(s) = K_{PD} \cdot F(s) \cdot \frac{K_{VCO}}{s} \quad (4.1)$$

$$= K_{PD} \cdot K_{VCO} \cdot \frac{s + \frac{1}{RC_1}}{C_2 s^2 (s + \frac{C_1 + C_2}{RC_1 C_2})} \quad (4.2)$$

On analyzing the transfer function, we get:

$$\omega_z = \frac{1}{RC_1}; \omega_{p1} = \omega_{p2} = 0; \omega_{p3} = \frac{C_1 + C_2}{RC_1C_2} \quad (4.3)$$

Thus the phase margin is given as:

$$\phi_M = \tan^{-1} \left( \frac{\omega_{ugb}}{\omega_z} \right) - \tan^{-1} \left( \frac{\omega_{ugb}}{\omega_{p3}} \right) \quad (4.4)$$

where  $\omega_{ugb}$  is the unity gain bandwidth and  $\omega_z < \omega_{ugb}$ .

In order to find the maximum value of the phase margin, we will differentiate Eq. 4.4 with respect to  $\omega_{ugb}$  and set the result to 0 [3]. On doing so, we will obtain the following result [6]:

$$\omega_{ugb} = \omega_z \sqrt{\frac{C_1}{C_2} + 1} \quad (4.5)$$

On plugging this result back into Eq. 4.4 we get:

$$\phi_{M_{max}} = \tan^{-1} \left( \sqrt{\frac{C_1}{C_2} + 1} \right) - \tan^{-1} \left( \frac{1}{\sqrt{\frac{C_1}{C_2} + 1}} \right) \quad (4.6)$$

Figure 4.2 shows the sub-optimal and optimal phase margin required for the stability of the PLL.

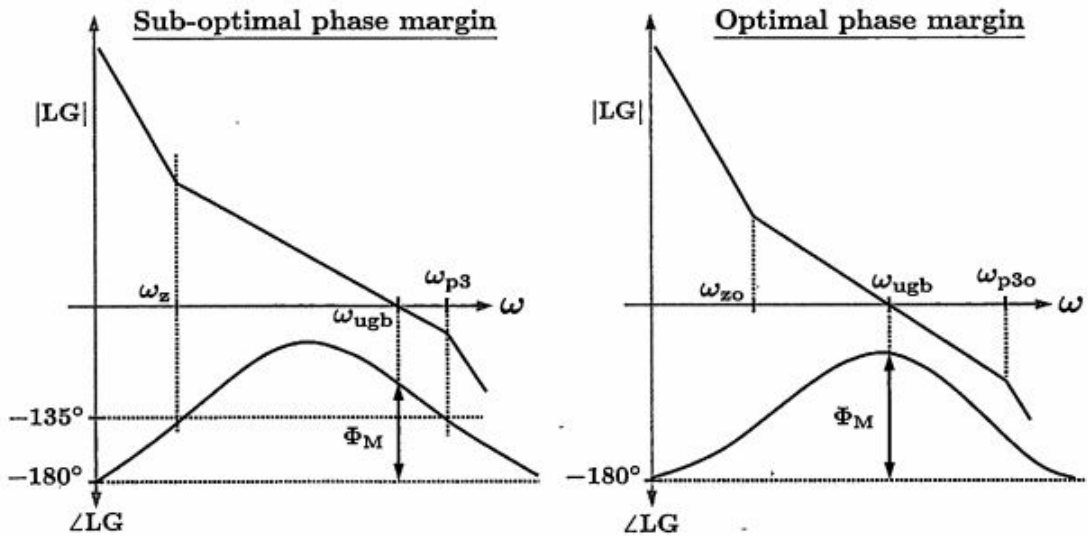


Figure 4.2: Magnitude and Phase Response

## 4.1 Design Procedure and Parameter Calculation

We need to calculate the values of the capacitors and resistor in the loop filter, the zeros and the poles in the system as well as the charge pump current before implementing our design. This is done by first choosing the unity-gain bandwidth, phase margin and resistor R.

Then  $K_c$  which is the ratio of  $C_1$  to  $C_2$  is calculated as:

$$K_c = \frac{C_1}{C_2} = 2((\tan^2(\phi_M) + \tan(\phi_M)\sqrt{\tan^2(\phi_M) + 1})) \quad (4.7)$$

From Eq. 4.5 we have:

$$\omega_z = \frac{\omega_{ugb}}{\sqrt{\frac{C_1}{C_2} + 1}} \quad (4.8)$$

Thus,

$$C_1 = \frac{1}{\omega_z R}; C_2 = \frac{C_1}{K_c} \quad (4.9)$$

Once all the above parameters are determined, the charge pump current is calculated as:

$$I_{CP} = \frac{2\pi C_2}{K_{VCO}} \cdot \omega_{ugb}^2 \cdot \sqrt{\frac{\omega_{p3}^2 + \omega_{ugb}^2}{\omega_z^2 + \omega_{ugb}^2}} \quad (4.10)$$

## 4.2 PLL Bandwidth

In the previous sections, terms such as PLL bandwidth were used. It is necessary to understand what this means. The PLL acts as a low-pass filter with respect to the reference signal and rejects high-frequency noise whereas it acts like a high-pass filter with respect to the VCO clock as seen in figure 4.3. Thus PLL bandwidth can be defined as the frequency at which the PLL begins to lose lock with the changing reference signal (-3 dB) [7].

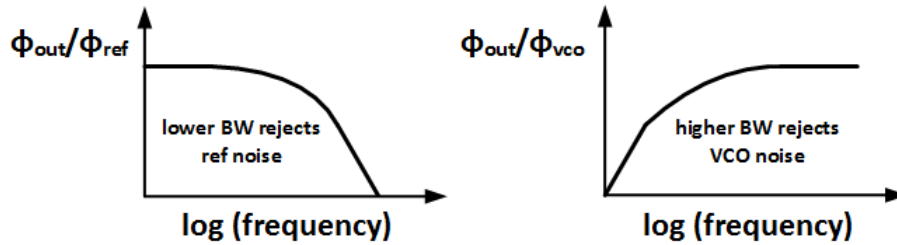


Figure 4.3: PLL Bandwidth

Bandwidth also plays an important role in determining the lock time of the PLL. The higher the BW, the lower is the lock time as a larger BW allows for faster correction of any phase and frequency errors as seen in figure 4.4.

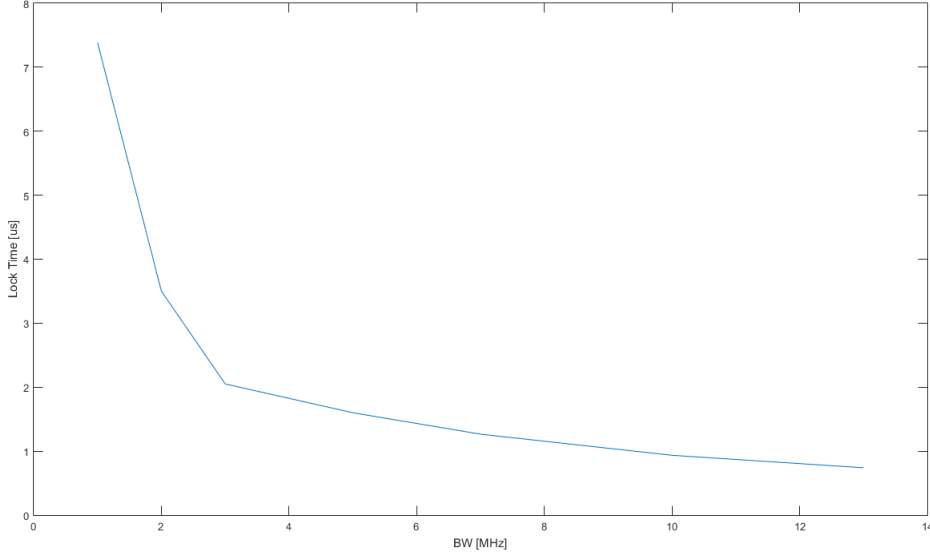


Figure 4.4: Lock Time vs. Bandwidth

### 4.3 PLL Locking Verification

It is important to check that the PLL will lock when a frequency step is applied at its input. Let the input frequency step be  $\omega_{in} = \frac{\Delta\omega}{s}$  and hence  $\Phi_{in}(s) = \frac{\Delta\omega}{s^2}$ . The closed-loop transfer function is given as:

$$H_{PLL}(s) = \frac{LG(s)}{1 + LG(s)} \quad (4.11)$$

The steady-state error transfer function is:

$$\frac{\Phi_{error}(s)}{\Phi_{in}(s)} = H_e(s) = 1 - H_{PLL}(s) = \frac{1}{1 + LG(s)} \quad (4.12)$$

Using the final value theorem,

$$\Phi_{ss\_error}^{F\_step} = \lim_{s \rightarrow 0} s \cdot H_e(s) \cdot \Phi_{in}(s) \quad (4.13a)$$

$$= \lim_{s \rightarrow 0} s \cdot \frac{1}{1 + LG(s)} \cdot \frac{\Delta\omega}{s^2} \quad (4.13b)$$

$$= \lim_{s \rightarrow 0} \frac{[RC_1C_2s^2 + (C_1 + C_2)s]\Delta\omega}{RC_1C_2s^3 + (C_1 + C_2)s^2 + K_{VCO}K_{PDS} + 1} \quad (4.13c)$$

$$= \frac{0}{1} \quad (4.13d)$$

$$= 0 \quad (4.13e)$$

Thus any steady-state phase error is eliminated and the PLL will achieve lock.

## 4.4 PLL Noise Analysis

Each component of the PLL generates noise at the output. The noise transfer functions can be derived to give overall noise of the PLL. Figure 4.5 shows the noise model of the PLL.

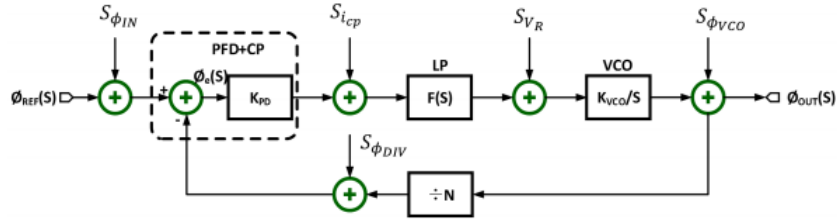


Figure 4.5: PLL Noise Model

$$S_{\Phi_{OUT}}^{\phi_{IN}} = S_{\phi_{IN}} |NTF_{IN}(s)|^2 \quad (4.14)$$

$$S_{\Phi_{OUT}}^{i_{CP}} = S_{i_{CP}} |NTF_{CP}(s)|^2 \quad (4.15)$$

$$S_{\Phi_{OUT}}^{v_R} = S_{v_R} |NTF_R(s)|^2 \quad (4.16)$$

$$S_{\Phi_{OUT}}^{\phi_{VCO}} = S_{\phi_{VCO}} |NTF_{VCO}(s)|^2 \quad (4.17)$$

where,

$$NTF_{IN}(s) = \frac{\Phi_{OUT}(s)}{\Phi_{IN}(s)} = \frac{N.LG(s)}{1 + LG(s)} \quad (4.18)$$

$$NTF_{DIV}(s) = NTF_{IN}(s) \quad (4.19)$$

$$NTF_{CP}(s) = \frac{\Phi_{OUT}(s)}{i_{CP}(s)} = \frac{2\pi}{i_{CP}}.NTF_{IN}(s) \quad (4.20)$$

$$NTF_R(s) = \frac{\Phi_{OUT}(s)}{v_R(s)} = \frac{K_{VCO}/s}{1 + LG(s)} \quad (4.21)$$

Shown in figure 4.6 are NTF plots of each of the PLL blocks.

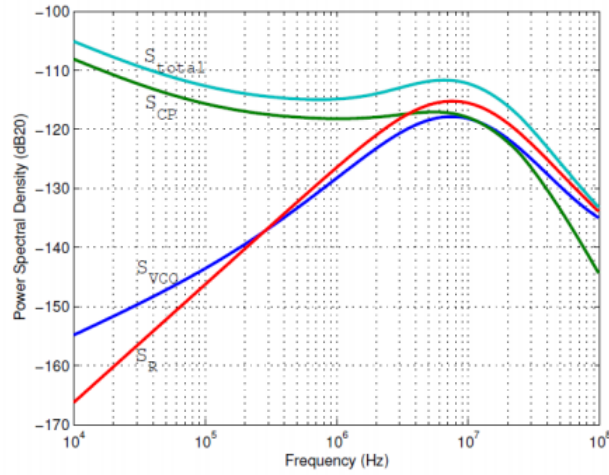


Figure 4.6: PLL Output Referred Noise

# CHAPTER 5

## PLL DESIGN CHOICES

### 5.1 PFD

A NAND PFD was chosen to implement in this thesis. Figure 5.1 shows the topology of a NAND-based PFD.

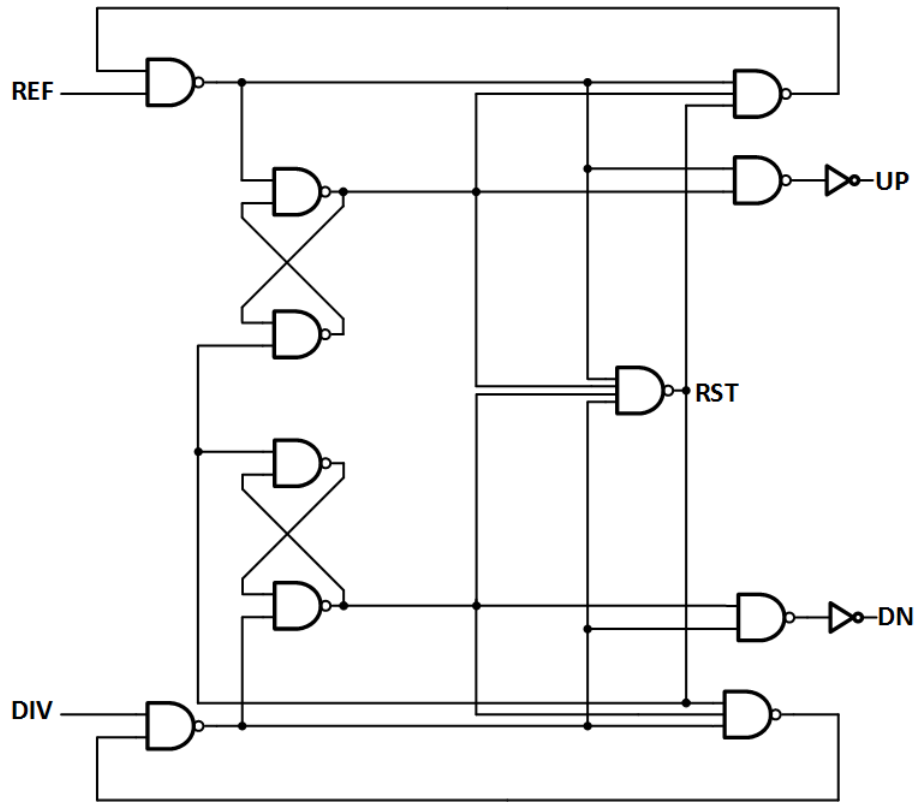


Figure 5.1: NAND PFD

A NAND PFD is a three-state PFD with an UP, DN and RST state. The NAND PFD must be carefully designed to avoid the dead zone problem. The reset delay must be greater than the switch on-time of the transistors in the charge pump. The delay is given by  $T_{RST} = 2T_{NAND2} + T_{NAND4}$  and the

maximum frequency of operation of the PFD is controlled by the reset delay and is  $F_{max} < \frac{1}{2T_{RST}}$ .

## 5.2 Charge Pump

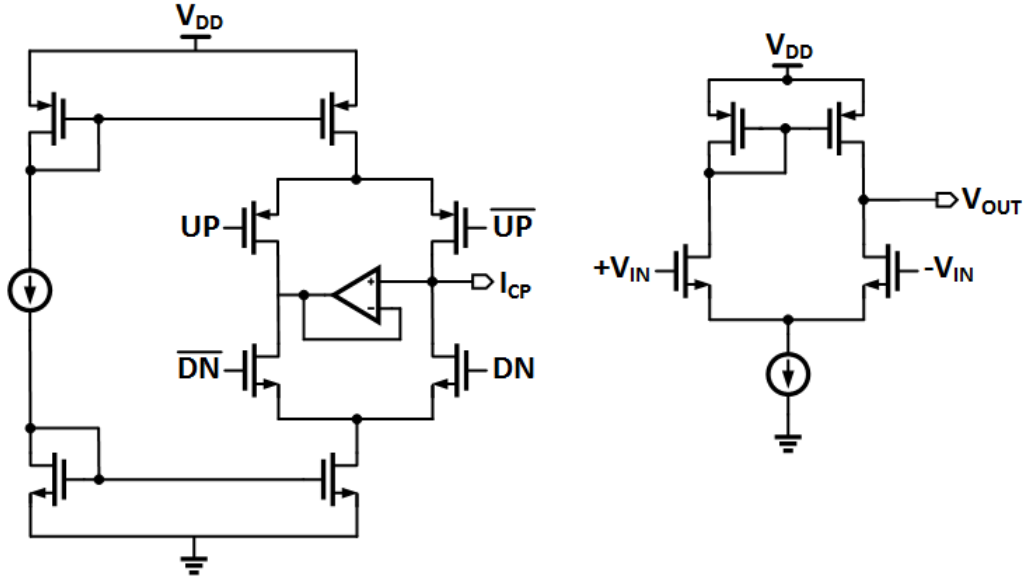


Figure 5.2: Bootstrapped Charge Pump

Figure 5.2 shows the topology of a bootstrapped charge pump. Matching the up and down currents is required for the  $V_{ctrl}$  range of operation. A mismatch in these currents will contribute to jitter at the output. Taking this into account, a bootstrapped charge pump was implemented. This charge pump implements a unity gain amplifier which ensures that the same voltage is maintained at the input and the output of the amplifier which allows the up current to equal the down current. The bootstrap CP allows for differential current steering and can operate with low swing UP/DN signals.

## 5.3 Loop Filter

The loop filter consists of a single resistor and two capacitors as shown in figure 5.3. It effectively functions as a low-pass filter and filters out the high-frequency noise at the input of the PLL.



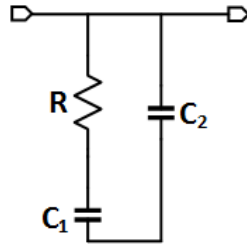


Figure 5.3: Loop Filter

## 5.4 Voltage Controlled Oscillator

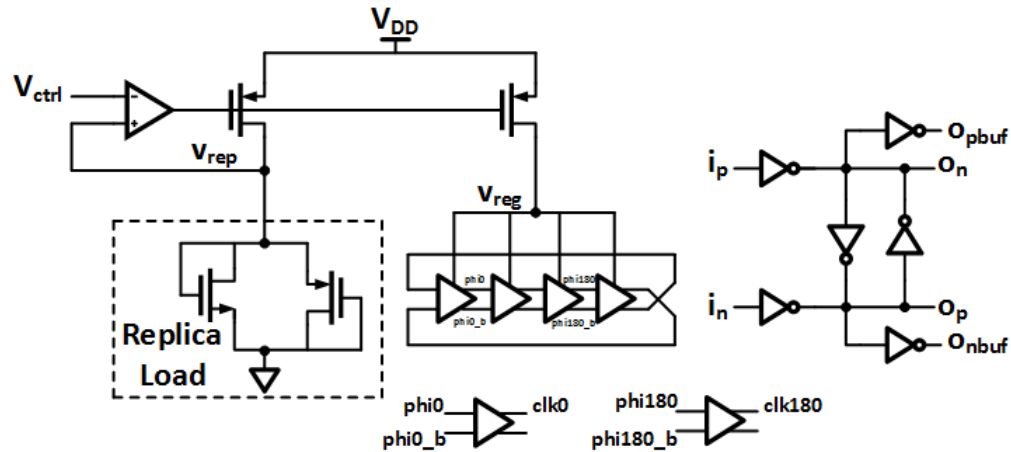


Figure 5.4: Differential VCO

A differential VCO is implemented using the replica load technique as shown in figure 5.4. The control voltage is replicated and fed to the differential chain which consists of inverters and cross coupled latches. The output of the chain is fed to a buffer which generates the differential output clocks. The advantage of using a replica bias regulator is that it provides high bandwidth (good high-frequency PSRR) and good stability. The differential VCO also has lower phase noise compared to a single-ended VCO.

## 5.5 Divider

The divider consists of five D Flip-Flops (DFFs) connected in the manner shown in figure 5.5. Each DFF functions as a divide-by-2 divider. Hence the entire chain performs a divide-by-32 operation. A factor of 32 is chosen because the VCO clock is operating at 6.4 GHz and a reference clock of 200 MHz is being used. Each DFF employs a True-Single Phase Clock (TSPC) architecture which is fast and has low skew.

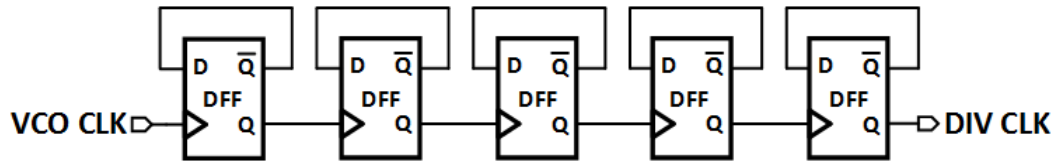


Figure 5.5: Divider

# CHAPTER 6

## BEHAVIORAL LEVEL SIMULATION

### 6.1 Behavioral Modeling

SPICE is the most popular simulation engine to simulate analog/mixed-signal circuits. However, while simulating a large circuit the simulation time can be extremely long. Also, it is difficult to predict the behavior of the circuit until the transistor level structure is known. SPICE needs to be regularly updated with technology scaling and becomes obsolete very quickly. For this reason, behavioral simulations of analog/mixed-signal circuits are performed. Behavioral modeling allows a designer to predict the nature of the circuit without the transistor level design, i.e. it is process independent. Besides allowing the designer to accurately debug any bugs, it also cuts down on simulation time. VerilogAMS (Verilog Analog-MixedSignal) is the most popular Hardware Description Language (HDL) used to perform such simulations. This chapter presents a detailed tutorial on simulating the different building blocks of the PLL using VerilogAMS.

#### 6.1.1 PFD+CP

1. Once virtuoso is loaded, click on *Tools* → *Library Manager* and the Library Manager window will pop up as shown in figure 6.1.

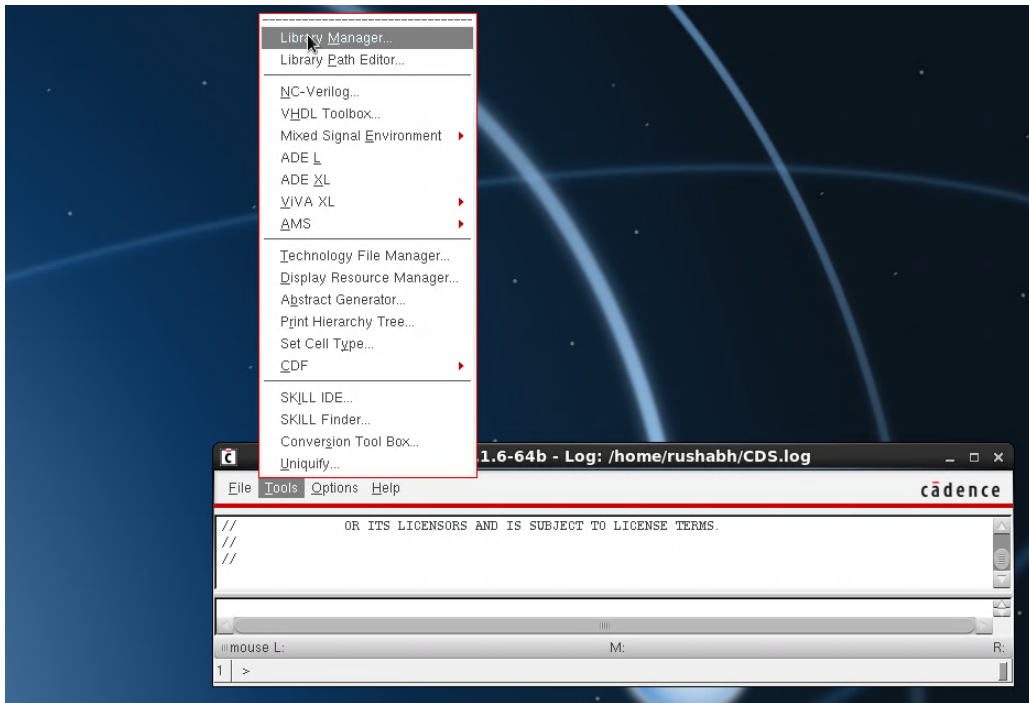


Figure 6.1: Library Manager Window

2. Create a new library by clicking on *File* → *New* → *Library*. Name the library PLLBehavioral.
3. The next task is to create a cell view under the PLLBehavioral library. Click on *File* → *New* → *Cell View*. Select the PLLBehavioral library. Name the cell view *pdf*. Change the type to *VerilogAMSText* as shown in figure 6.2.

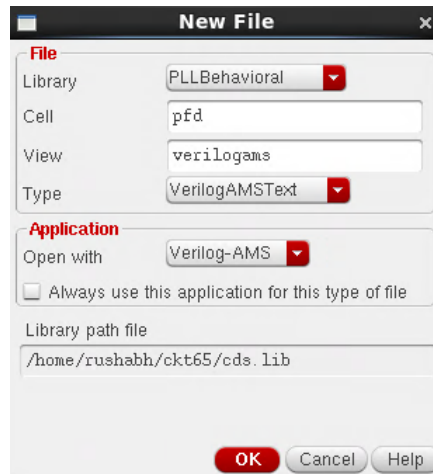


Figure 6.2: Opening a VerilogAMS Text Editor

4. Click on OK. The cell will show up in the corresponding library. A text editor will open. Type the code for the PFD as shown in figure 6.3. The PFD has two inputs: `fref` and `fdiv` and four outputs: `up`, `dn`, `upb`, `dnb`. The `upb` and `dnb` outputs are inverted versions of `up` and `dn`. An additional test output is created which is the reset signal. To account for non-idealities in this design, a delay of 80 ps is added to the output denoted by `#8`.

```

//Verilog-AMS HDL for "PLLBehav", "pfd" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 10ps / 1ps

module pfd (up, dn, upb, dnb, fref, fdiv, test);

input fref;
input fdiv;
output up, upb, dn, dnb;

wire fv_rst, fr_rst;
wire reset;
reg q0, q1;

assign fr_rst = reset | (q0 & q1);
assign fv_rst = reset | (q0 & q1);
assign reset = fref & fdiv;

always @ (posedge fdiv or posedge fv_rst) begin
if (fv_rst) q0 <= #8 0; else q0 <= #8 1;
end
always @ (posedge fref or posedge fr_rst) begin
if (fr_rst) q1 <= #8 0; else q1 <= #8 1;
end
assign up = q1;
assign dn = q0;
assign upb = ~q0;
assign dnb = ~q1;
assign test = reset;
endmodule

```

Figure 6.3: PFD VerilogAMS Code

- Once the code is written, save and exit the text editor. On exiting, a pop-up window will appear. Click on yes to generate the symbol for the pfd. Figure 6.4 illustrates this process.

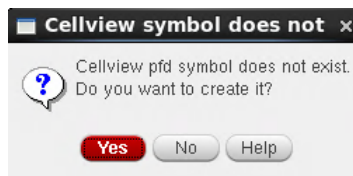


Figure 6.4: PFD Symbol

- Go to the PLLBehavioral library and create a new cell view called *pfd.tb*. Change the type to *schematic* as shown in figure 6.5.

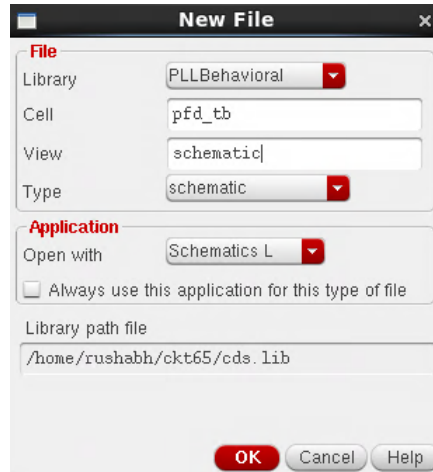


Figure 6.5: PFD Schematic

- To set up the test bench for the PFD, we need the PFD, wires and voltage sources. This can be done by creating an instance or pressing *I*. Import the symbol for the PFD from the *pfd* cell view and vpulse sources from the analogLib library. These sources will be used as inputs to the ref and div signals. Make sure that the two sources are slightly off in frequency from one another and have a delay of 500 ps between them. This can be done by selecting the sources, pressing *Q* and inputting the necessary values. Wires can be obtained by pressing *W* and connecting the necessary nodes. Figure 6.6 shows the PFD test bench.

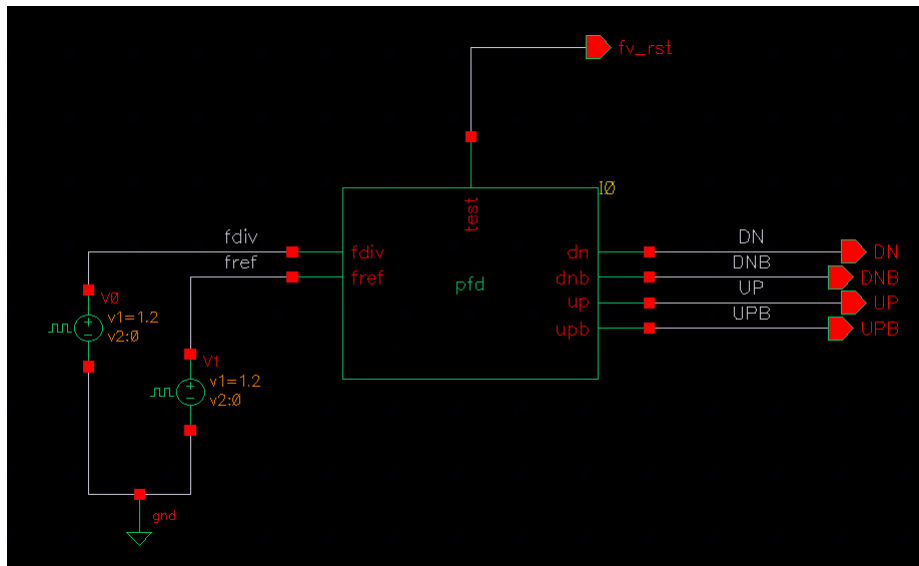


Figure 6.6: PFD Test Bench

- The next step is to create a *config* file whose job is to link the verilog simulation engine and analog test bench sources together. Go to the same cell view in which the pfd schematic is created i.e. *pfd\_tb*. Change the type to config. The pop-up window should look like the one in figure 6.7. Click OK and a new window will pop-up.



Figure 6.7: PFD Config View

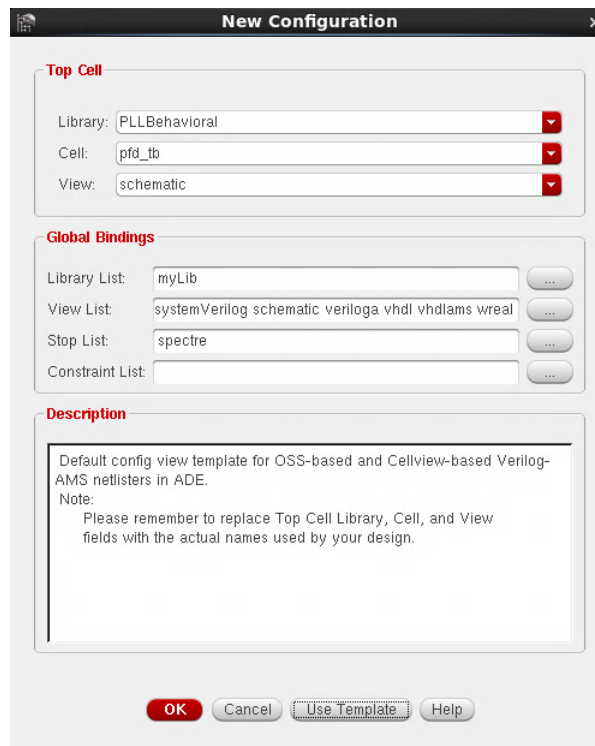


Figure 6.8: Config Setup



9. Change the view to *schematic*. Click on Use Template and select AMS. Once everything is done, your window should look similar to the one in figure 6.8.
10. Click OK and the following window will appear as shown in figure 6.9.

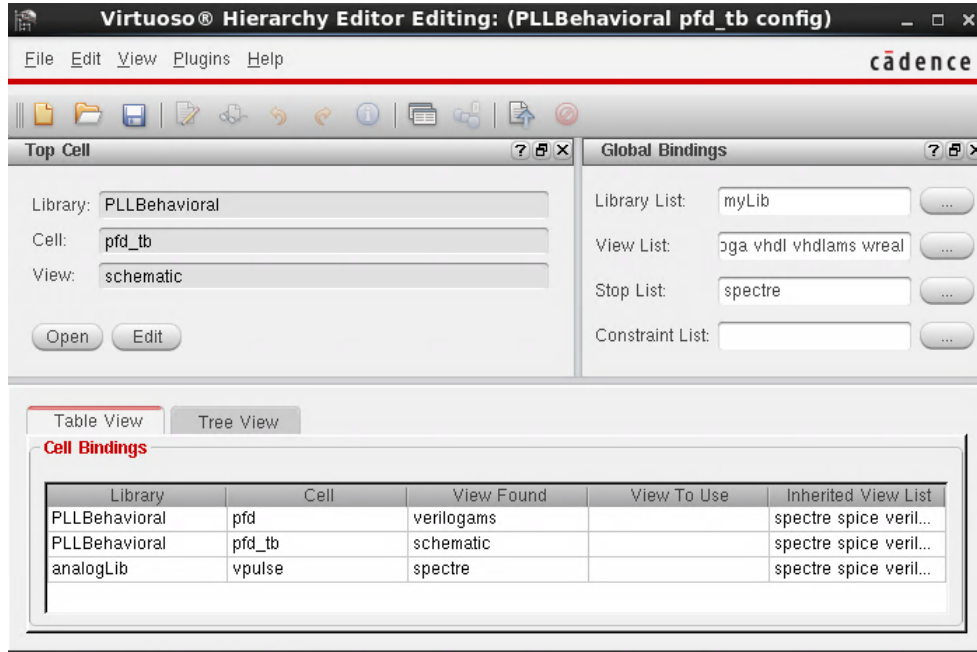


Figure 6.9: Config Setup

11. Click on Save and Open the schematic. The schematic will now look similar to the one previously created but now it has config in the title.
12. Check and Save the schematic. Once that is done, go to *Launch* → *ADE L* to open the ADE L window.
13. The ADE L window is the one where we will set up our analysis to be performed on the PFD. Since we selected the template to be AMS, we need to ensure that AMS is our simulator. To verify it, go to *Setup* → *Simulator* and change the simulator to AMS. Once that is done, click on *Outputs* → *To Be Plotted* → *Select On Schematic* and select the input and output signals. The final setup is the type of simulation to be performed. Click on *Analysis* → *Choose* and select transient for a stop time of 100 ns. The ADE L window should now look like the one in figure 6.10.

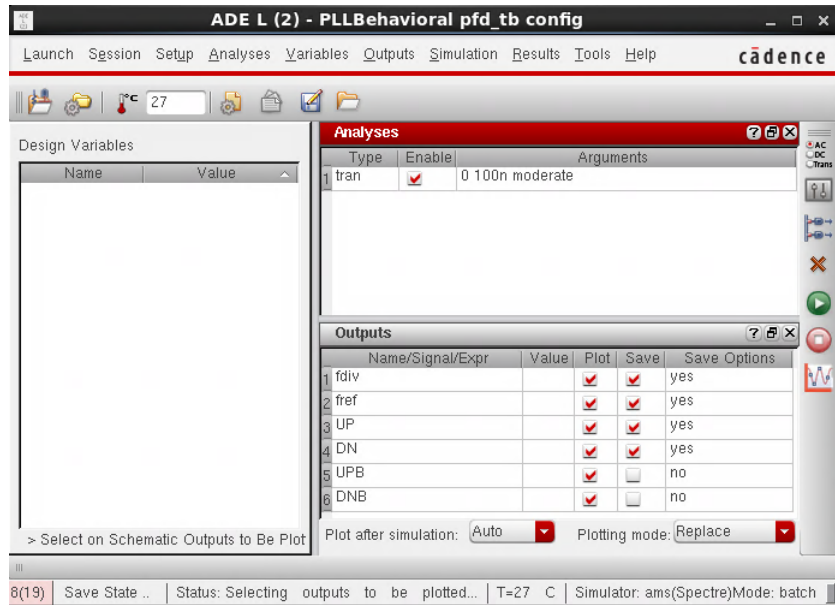


Figure 6.10: ADE L Window

- The PFD waveforms should now look like the one shown in figure 6.11. Notice that the PFD is functioning correctly. When the ref signal is leading the div signal, UP goes high. When div goes high, DN goes high and the reset path is activated which pulls UP and DN to 0.

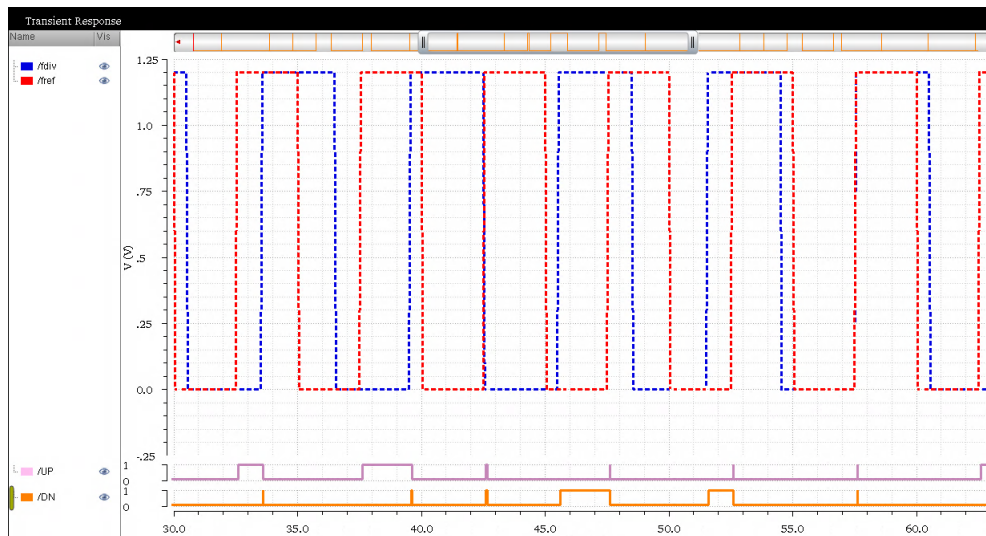


Figure 6.11: PFD Waveform

- In a similar fashion, set up the test bench for the charge pump as shown in figure 6.13. The verilog code for the CP is shown in figure 6.12. The

current is set to 70 uA since that is the charge pump current required in the PLL design.

```

//Verilog-AMS HDL for "PLLBehav", "cp" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 10ps / 1ps

cp (pout, nout, up, dn);
parameter real cur = 70u; // output current (A)
input up, dn;
output pout, nout;
electrical pout, nout;
real out;
analog begin
@(initial_step) out = 0.0;
if (dn && !up)
    out = -cur;
else if (!dn && up)
    out = cur;
else out = 0;
I(pout, nout) <+ -transition(out, 0.0, 10p, 10p);
end
endmodule

```

Figure 6.12: CP VerilogAMS Code

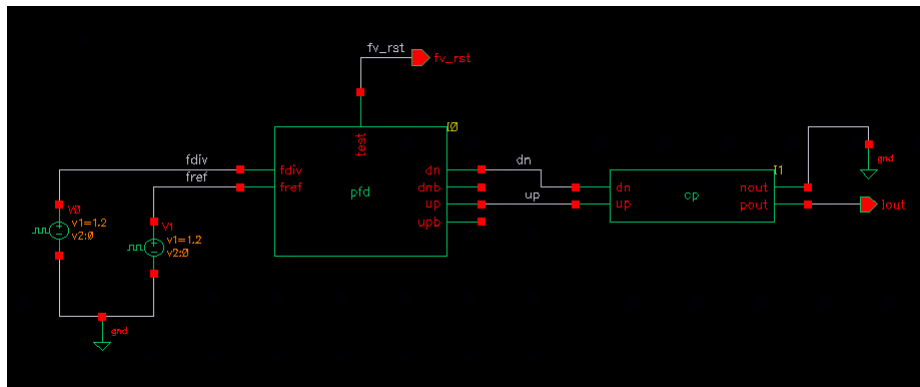


Figure 6.13: CP Test Bench

- Although the charge pump current is set to 70 uA, it can always be changed. This is done by selecting the CP symbol, pressing  $Q$ , and changing the current to the user's choice as shown in figure 6.14.

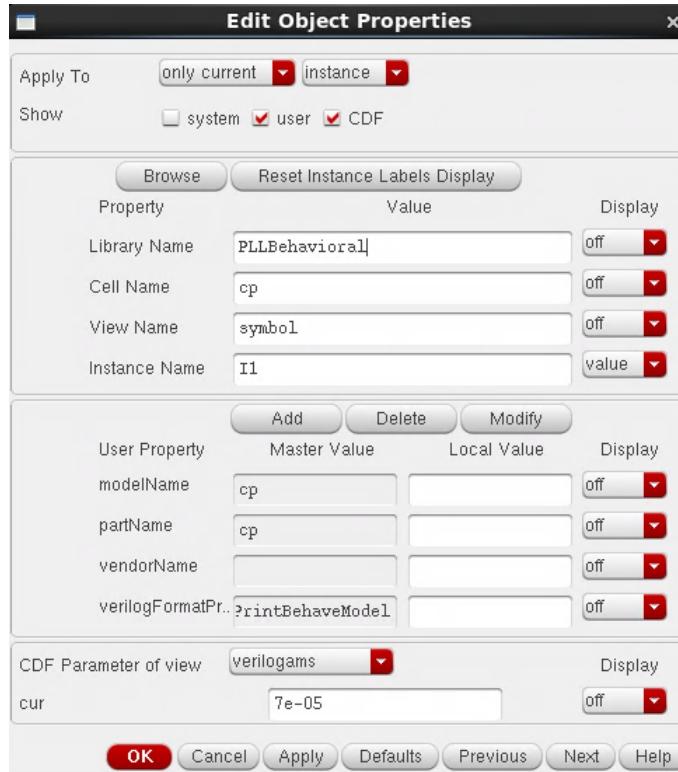


Figure 6.14: CP Current Setup

17. The waveform for the CP is shown in figure 6.15. Note that current is defined negative in AMS. The CP is functioning correctly. We get +70 uA when UP is high and -70 uA when DN is high. Thus in an ideal setting, the UP and DN currents are matched.

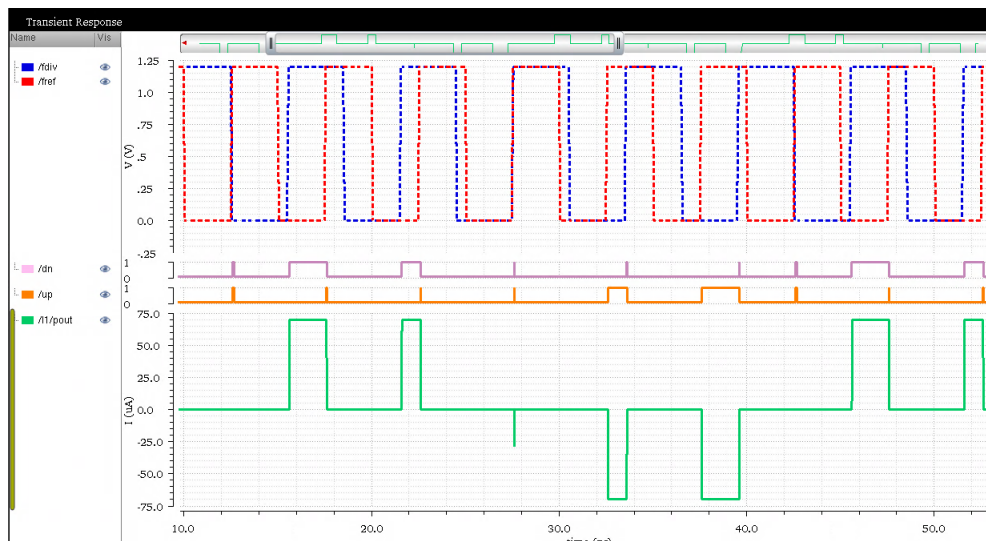


Figure 6.15: CP Waveform

## 6.1.2 LF

A MATLAB script was written to generate the loop parameters. There is no VerilogAMS code involved here.

## 6.1.3 VCO

1. The VCO is set up in a fashion similar to the other blocks of the PLL. The verilog code for the differential VCO is shown in figure 6.16 and figure 6.17. As the name suggests, we have two outputs. The phase noise is manually entered according to the jitter specifications of the PLL. A control voltage of 0.4 V is targeted.

```
//Verilog-AMS HDL for "Vmodels", "vco" "verilogams"
// VCO Behavioral Model (w/ phase noise)
// by rmehta (03/11/16)

`define PI 3.14159265358979323846264338327950288419716939937511
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1fs

module dco(vco_out1, vco_out2, vctrl);
    input vctrl;
    electrical vctrl;

    parameter real noise_acc_dbc = -100; // VCO FM noise @ fos (single-sideband)
    parameter real fos = 1M;
    parameter real noise_white_dbc = -125; // VCO PM noise (single-sideband)
    parameter pn_en = 1; // phase noise enable(1), disable(0)
    parameter Vctrl_target = 0.4;

    output vco_out1; // DCO output clk
    output vco_out2;
    reg vco_out1;
    reg vco_out2;
    reg vco_out_ideal;
    reg vco_out_jitt;

    parameter real freq = 6400M; // DCO output frequency
    parameter real Kvco = 500e6; // DCO gain (ppm/LSB)

    real nom_delay;
    real ideal_delay;
    real jitt_delay;

    real fm_jitt_std;
    real pm_jitt_std;

    real fm_del;
    real pm_del_2;
    real pm_del;
    integer seed1 = -311;
    integer seed2 = -561;
    integer file;
    integer i;
    integer count;

    initial begin
        // vco_out = 1'b0;
        vco_out_ideal = 1'b0;
        vco_out_jitt = 1'b0;
        nom_delay = 1/freq*1e9/2;
        ideal_delay = nom_delay;
        jitt_delay = nom_delay;
        if (noise_acc_dbc != 0)
            fm_jitt_std = fos*sqrt(10**(noise_acc_dbc/10)/(freq**3))*1e9; // FM jitter (period jitter std)
    end
endmodule
```

Figure 6.16: VCO Verilog Code

```

else
    pm_jitt_std = 0;
    fm_del = 0;
    pm_del = 0;
    pm_del_2 = 0;
    //file = $fopen("jitter",file_count+0*,".txt"),"w");
    file = $fopen("jitter.txt","w");
    count = 0;
end
always @(vco_out_ideal) begin
    //ideal_delay = nom_delay * (1 + V(vctrl)*Kvco/freq);
    ideal_delay = 1e9/(2*(freq+(V(vctrl)-Vctrl_target)*Kvco));
end

always @(negedge vco_out_jitt) begin // accumulating jitter modeling
    pm_del_2 = pm_del;
    pm_del = pm_jitt_std/2*$dist_normal(seed1,0,1000000)*1e-6;
    fm_del = fm_jitt_std/2*$dist_normal(seed2,0,1000000)*1e-6; // divided by 2 for half period
    jitt_delay = ideal_delay + pm_del - pm_del_2 + fm_del;
end

always #(ideal_delay) vco_out_ideal <= ~vco_out_ideal;
always #(jitt_delay) vco_out_jitt <= ~vco_out_jitt;

//always @(posedge dco_out) begin
//    count = count + 1;
//    if (count > 1250)
//        $fwrite(file,"%f\n",$realtime);
//end

always @(pn_en,vco_out_ideal, vco_out_jitt) begin
    case (pn_en)
        0 : begin
            vco_out1 <= vco_out_ideal;
            vco_out2 <= ~vco_out_ideal;
        end
        1 : begin
            vco_out1 <= vco_out_jitt;
            vco_out2 <= ~vco_out_jitt;
        end
    endcase
end
endmodule

```

Figure 6.17: VCO Verilog Code

2. Set up the VCO schematic and config file as described earlier. The VCO test bench should like the one is figure 6.18.

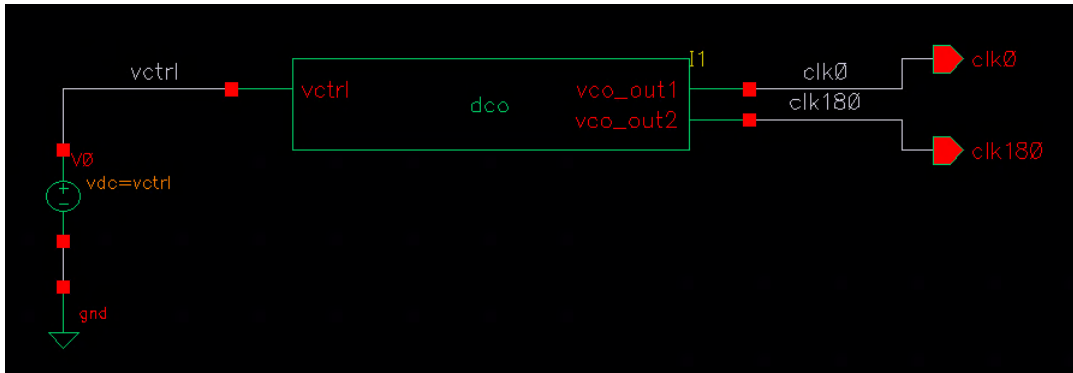


Figure 6.18: VCO Test Bench

3. Once again, we can alter the VCO parameters to the design requirements. Here, the targeted VCO frequency is 6.4 GHz and a  $K_{VCO}$  is 500 MHz/V. A white noise of -125 dBc/Hz is used along with a targeted control voltage of 0.4 V. Figure 6.19 shows the setup of these parameters.

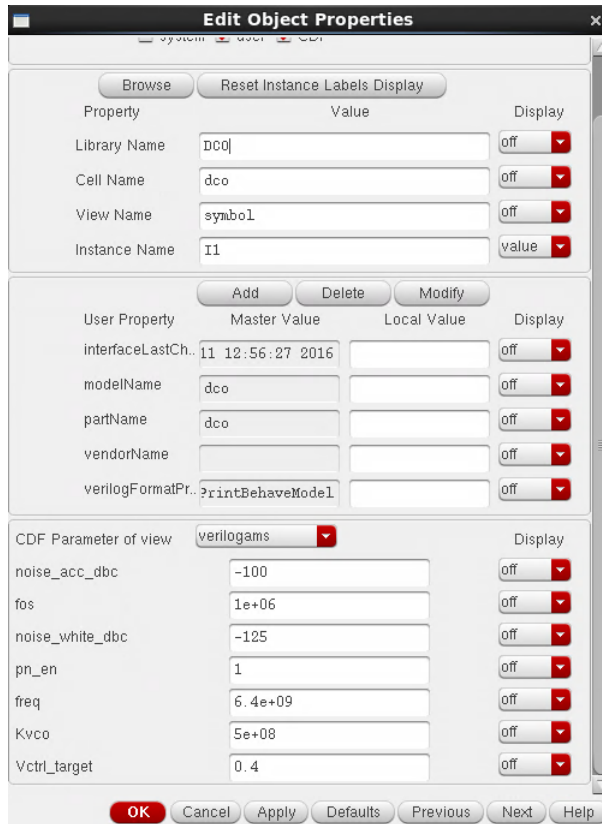


Figure 6.19: VCO Parameters Setup

4. We see the two differential output clocks both operating at 6.4 GHz in figure 6.20. These clocks will be used toward the serializer and the driver.

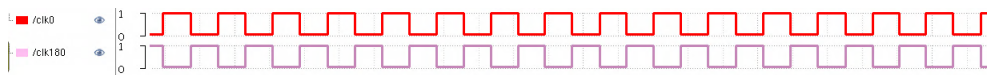


Figure 6.20: VCO Waveform

### 6.1.4 Divider

1. The divider is setup in a similar way to the other PLL blocks. It consists of five DFFs put together to divide the 6.4 GHz clock by a factor of 32 to scale it down to 200 MHz to match it to the reference clock. Figure 6.21 shows the VAMS code for the divider. A delay of 50 ps is added to the output.

```

//Verilog-AMS HDL for "PLLBehav", "div" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 10ps / 1ps

module div(out,clk);
  input clk;
  output out;
  parameter divide_ratio = 32;
  reg out;
  integer i=0;

  always@(posedge clk) begin
    if (i < (divide_ratio/2)-1) begin
      #5 out = 0;
      i = i + 1;
    end

    else if (i == (divide_ratio/2)-1) begin
      #5 out = 1;
      i = i + 1;
    end

    else if (i < (divide_ratio)-1) begin
      #5 out = 1;
      i = i + 1;
    end

    else if (i == (divide_ratio)-1) begin
      #5 out = 0;
      i = 0;
    end
  end
end
endmodule

```

Figure 6.21: Divider Code

2. Set up the divider schematic and config file as described earlier. The divider test bench should look like the one in figure 6.22.

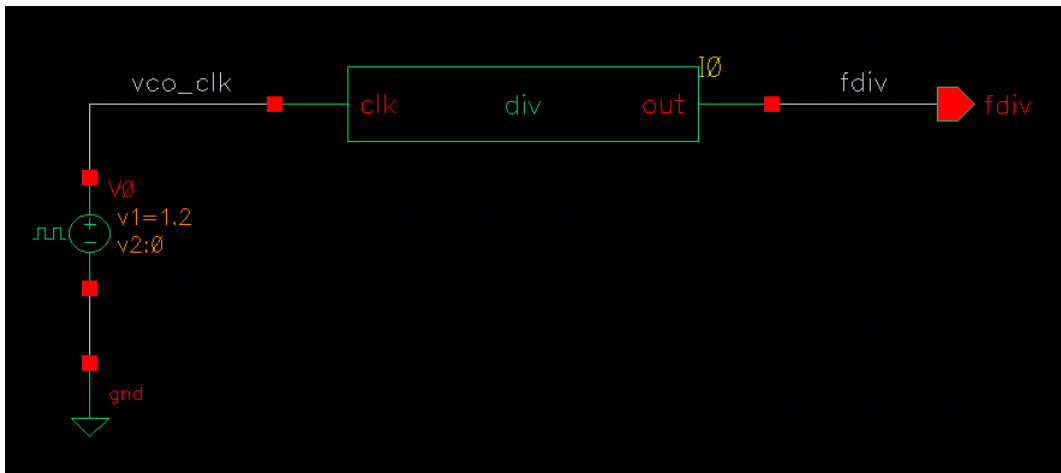


Figure 6.22: Divider Test Bench

3. The required division factor is 32 in this case. However, it can be changed by pressing  $Q$  and setting it according to the design requirement as shown in figure 6.23.



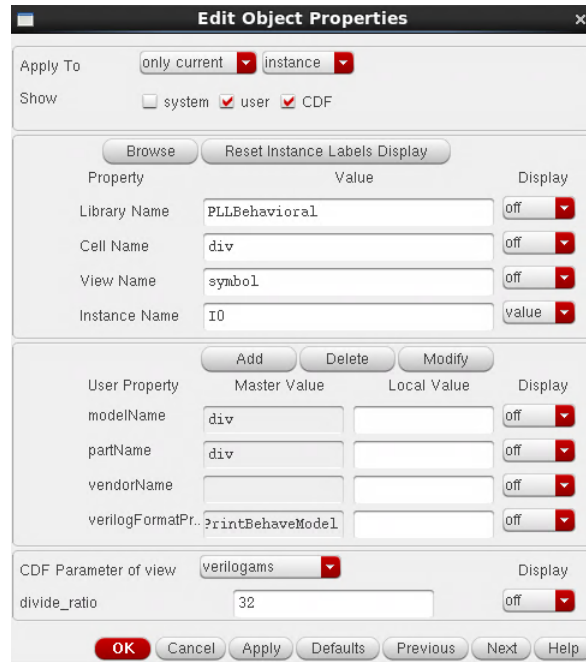


Figure 6.23: Divider Parameters Setup

4. We see the high-frequency VCO clock and the low-frequency divided clock. In figure 6.24, we notice 16 pulses of the VCO clock in half-a-cycle of the divided clock which implies 32 pulses of the VCO clock per cycle of the divided clock hence verifying the functionality of the divider.

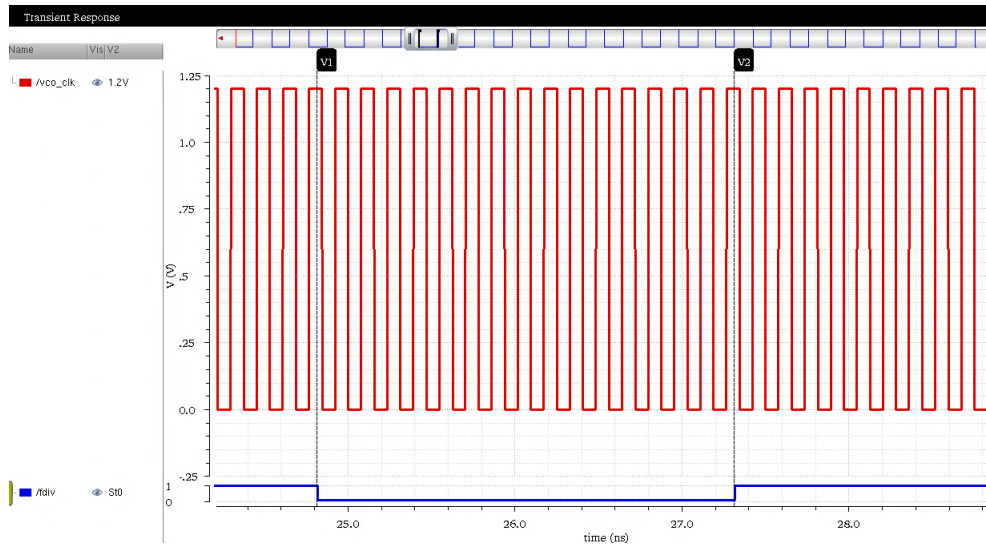


Figure 6.24: Divider Waveform

## 6.2 PLL Analysis

1. Now that all the blocks of the PLL are set up and working, they will be integrated together to test the working of the PLL. The config file and schematic for the PLL are created in a manner just like the other blocks. The test bench of the PLL should resemble the one shown in figure 6.25.

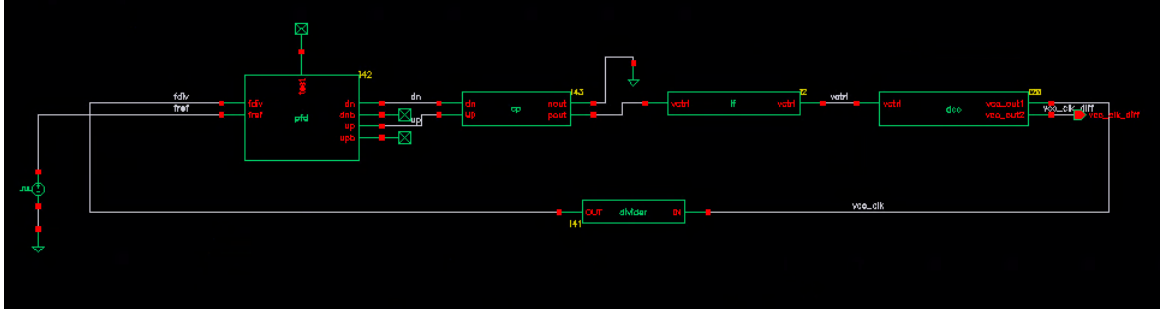


Figure 6.25: PLL Test Bench

2. The waveforms shown in figure 6.26 are captured at an instant where the PLL is trying to acquire lock. We observe that initially the reference clock is faster than the divided clock and hence we have UP pulses. This causes current to be dumped into the loop filter and the control voltage to increase in steps. The VCO output clock has varying frequency with time and the PLL is not locked.

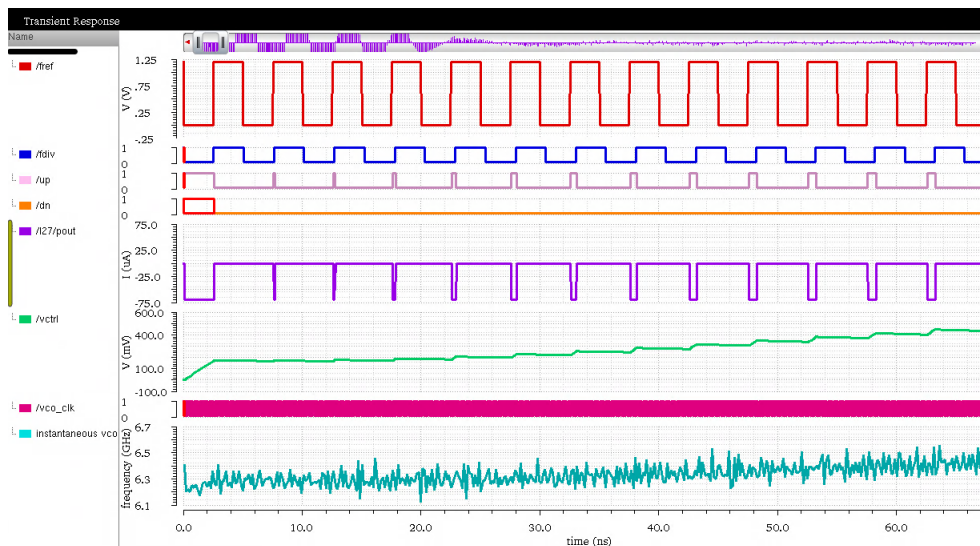


Figure 6.26: PLL Trying to Acquire Lock

- In figure 6.27, the PLL is completely locked. The reference and divided signal have zero phase and frequency offset. The UP and DN pulses go momentarily high but are brought down by the reset signal. The charge pump does not pump in or draw current from the loop filter and the control voltage settles to a constant value. The VCO clock is oscillating at 6.4 GHz. On observing the VCO frequency vs. time, we notice fluctuations in frequency. This arises due to jitter in the VCO. However, the PLL is said to be locked at this point of time.

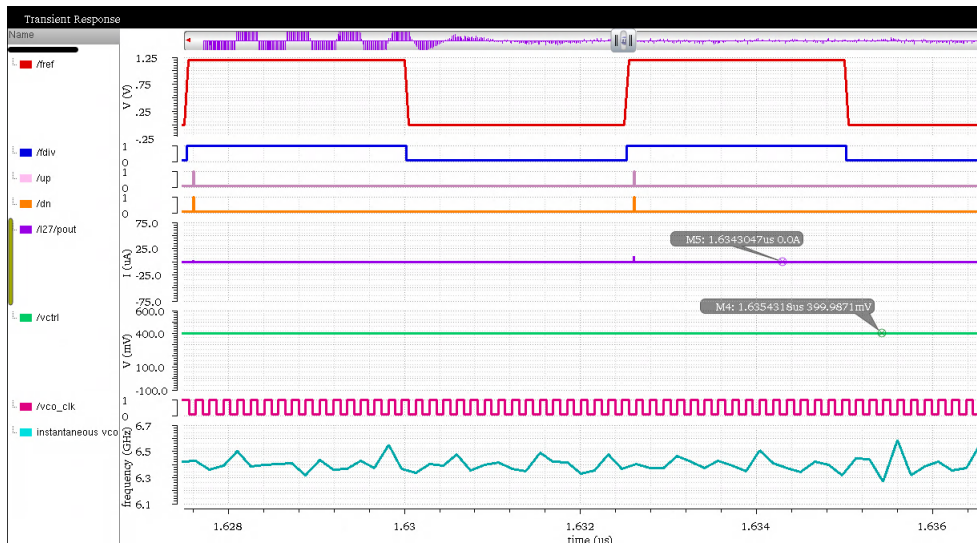


Figure 6.27: PLL in Locked State

- A plot of the control voltage is shown in figure 6.28. We can see how nicely the control voltage settles to a constant value of 400 mV once lock is achieved.

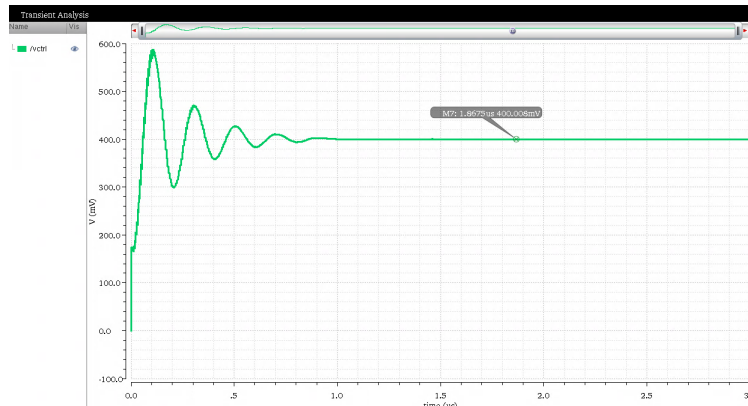


Figure 6.28: Control Voltage

# CHAPTER 7

## TRANSISTOR LEVEL SIMULATION

### 7.1 What Is Cadence Virtuoso?

Cadence Virtuoso is an Electronic Design Automation (EDA) tool used for analog, mixed-signal and RF IC design. It allows for construction of transistor-level schematics, layout, simulation and behavioral modeling.

### 7.2 Simulator

Cadence Spectre circuit simulator provides fast, accurate SPICE-level simulation for complex analog, RF and mixed-signal circuits. It is tightly integrated with the Virtuoso custom design platform and provides detailed transistor-level analysis in multiple domains.

### 7.3 Transient Simulations

The transient response is the response of a system to a change from a steady-state condition. It is used to study the time-varying behavior of a system and is used to analyze the design of the PLL.

### 7.4 Cadence Virtuoso: Getting Started

This section presents a detailed tutorial on constructing and simulating a D flip-flop. This includes transistor-level design, symbol generation and circuit simulation.

## 7.4.1 Creating a Schematic

1. Once Virtuoso is loaded, click on *Tools* → *Library Manager* and the Library Manager window will pop up as shown in figure 7.1.

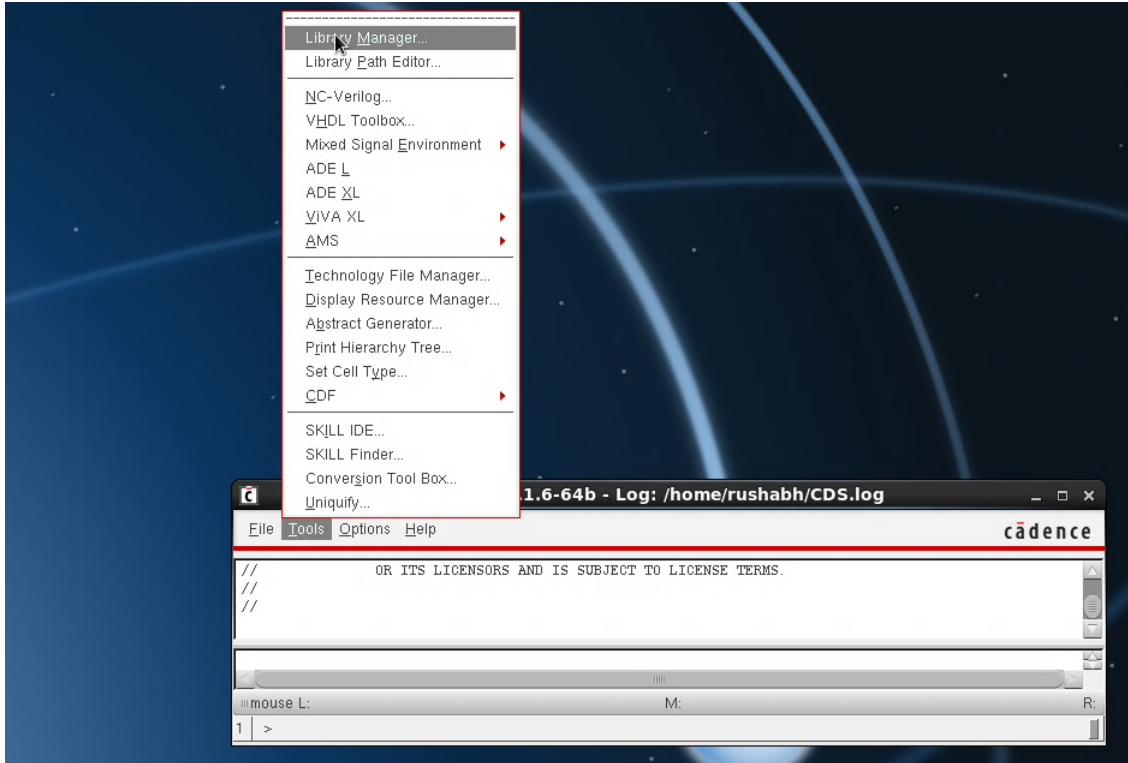


Figure 7.1: Library Manager Window

2. Create a new library by clicking on *File* → *New* → *Library*. Name the library PLL.
3. The next task is to create a cell view under the PLL library. Click on *File* → *New* → *Cell View*. Select the PLL library. Name the cell view DFF. The default setting for View will be schematic. If not, change the setting to View as shown in figure 7.2.

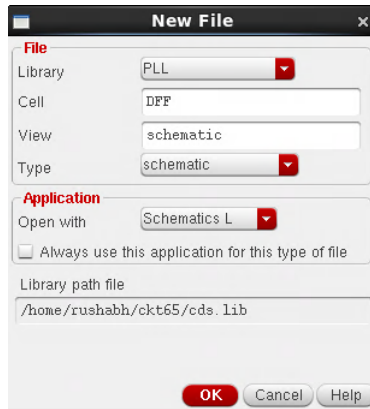


Figure 7.2: Creating a New Schematic Window

4. Click on OK. The cell will show up in the corresponding library. A schematic for the cell will also show up. Double-click on the schematic to open the schematic window.
5. A positive edge triggered DFF is created using NMOS and PMOS. These components are found in a different library. To access these components, use the create instance icon or alternatively press the key *I*. The Add Instance window will pop up. Select the *tsmcN65* library and the cell as *nch*. Select the view as symbol as shown in figure 7.3. Similarly, a PMOS can be selected by naming the cell as *pch*.
6. Press Enter. The transistor will now be placed on the schematic window and should look like the one displayed in figure 7.4.
7. These transistors can be replicated by pressing the key *C*, selecting the transistor and placing the replicated version at a different location. This is time efficient since we do not need to access the *Add Instance* window every time for instantiating the same component.
8. By selecting the component and pressing the *Q* key, different properties of the transistor, such as length, width and multiplier can be added and edited.

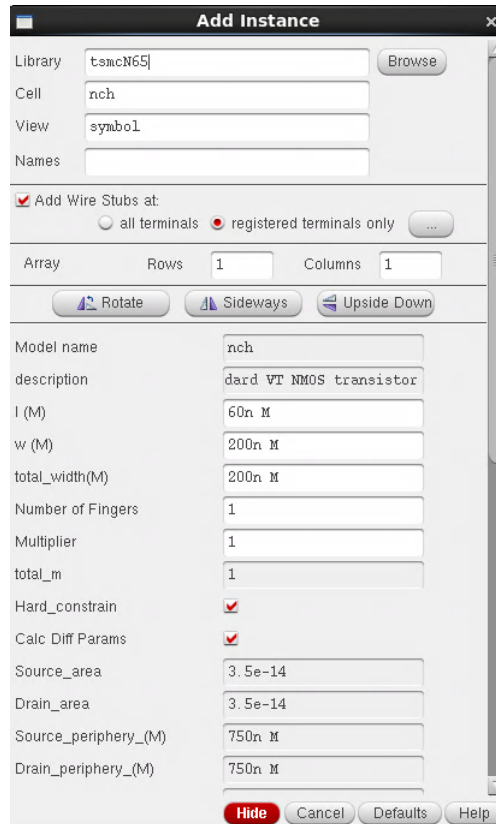


Figure 7.3: Instantiation

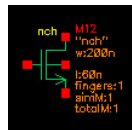


Figure 7.4: NMOS Transistor

9. Once the components are laid out, they can be wired by pressing the *W* key and selecting the nodes to be wired.
10. Once the TSPC positive edge triggered DFF is constructed, the schematic looks like the one in figure 7.5.

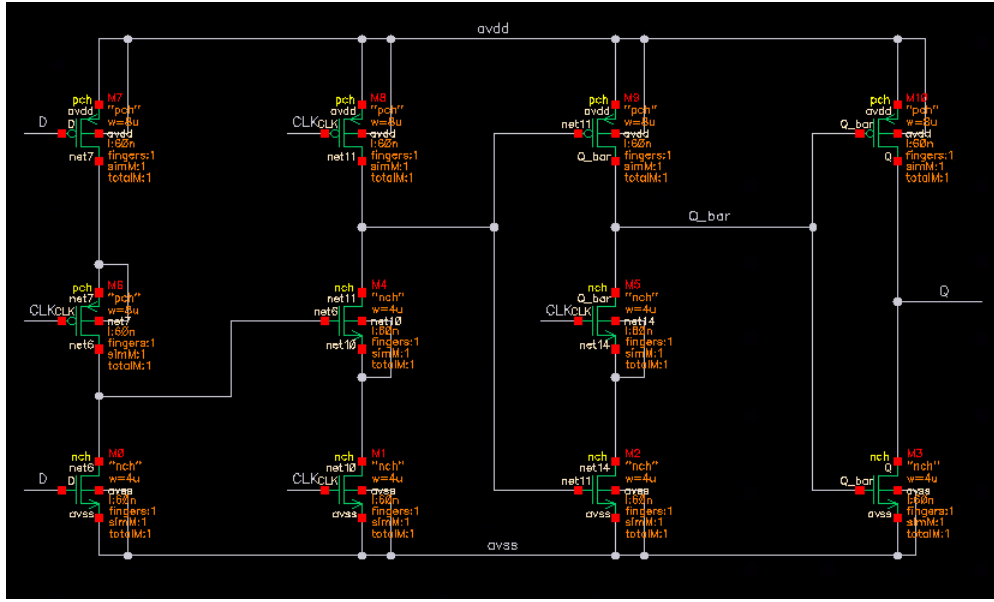


Figure 7.5: Positive Edge Triggered DFF Schematic

- The input and output pins are added by clicking on the *P* key. Once clicked, the window shown in figure 7.6 will pop up.

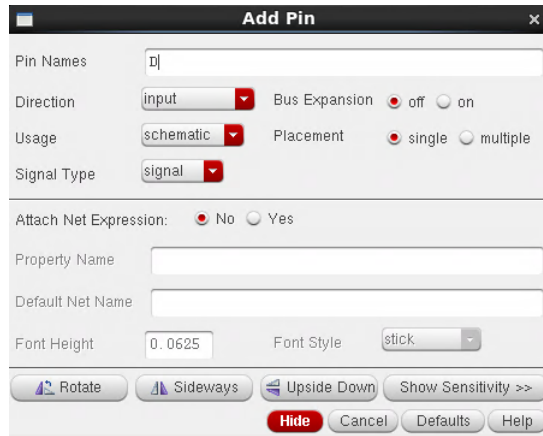



Figure 7.6: Pin Assignment

Type in the pin name and give it the appropriate direction. Use *inputOutput* for *VDD* and *GND*.

- The schematic is now ready. One of the most important steps is to click on the *Check and Save*  icon before closing the schematic window.



## 7.4.2 Creating a Symbol

1. Open the schematic of the component whose symbol is to be created.
2. Go to *Create* → *Cellview* → *From Cellview*. The window shown in figure 7.7 will pop up.

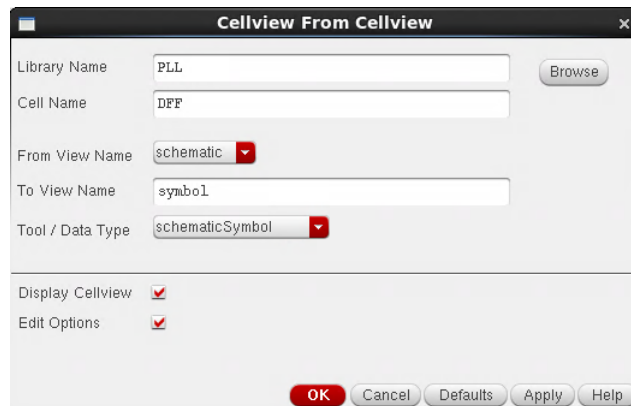


Figure 7.7: Creating a Symbol

3. Click OK. Once that is done a window similar to the one in figure 7.8 will pop up.

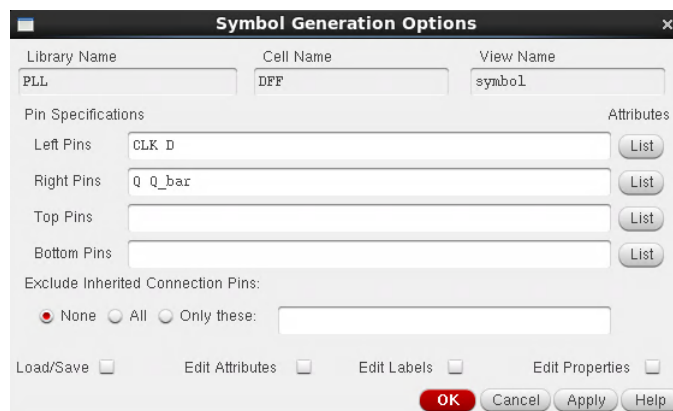


Figure 7.8: Pin Direction Allocation

Here you can allocate which pins will appear on which side of the symbol block.

4. On clicking OK, a symbol will be created and will look like figure 7.9.

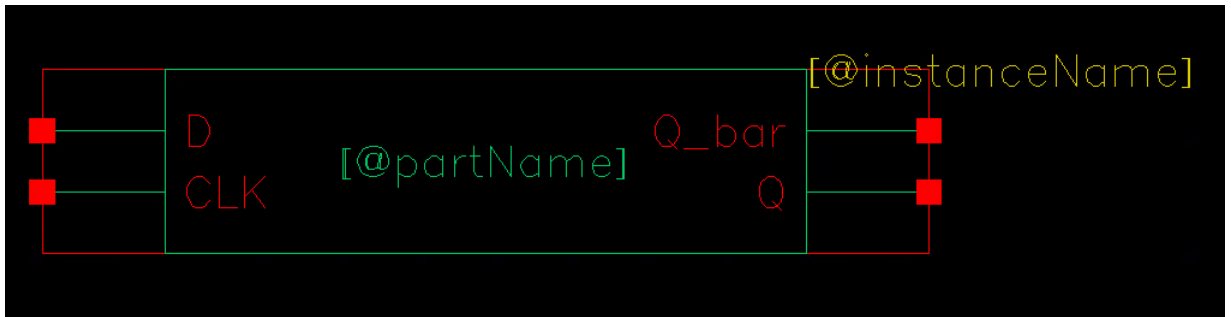


Figure 7.9: Symbol of a Positive Edge Triggered DFF

### 7.4.3 Creating a Testbench

Once all the components and symbols are assembled, simulations will be performed on them. A testbench is an environment where all the components are compressed into one symbol and the external sources, power supply and ground are connected to that symbol to perform the simulations. This reduces the complexity of the overall work needed to extract the simulations.

1. Go to *File* → *New* → *Cell View* under the PLL library and call this cell *DFF.tb*.
2. A schematic window will pop up. Press *I* to look for the symbols of the DFF, vpulse sources and ground. The DFF will be under the PLL library since it was created at that location. The vpulse and ground can be found under the analogLib library.
3. Once the symbols are found, connect them to set up the testbench for the DFF as shown in figure 7.10.

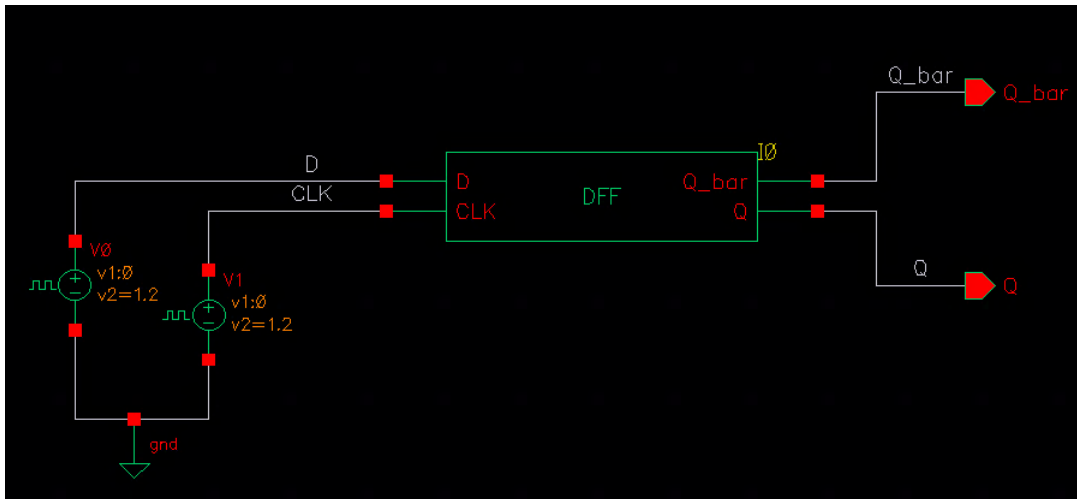


Figure 7.10: DFF Test Bench

- The specifications for the source need to be input in order to obtain the desired result. This can be done by selecting the source and then pressing  $Q$ . The window in figure 7.11 will pop up.

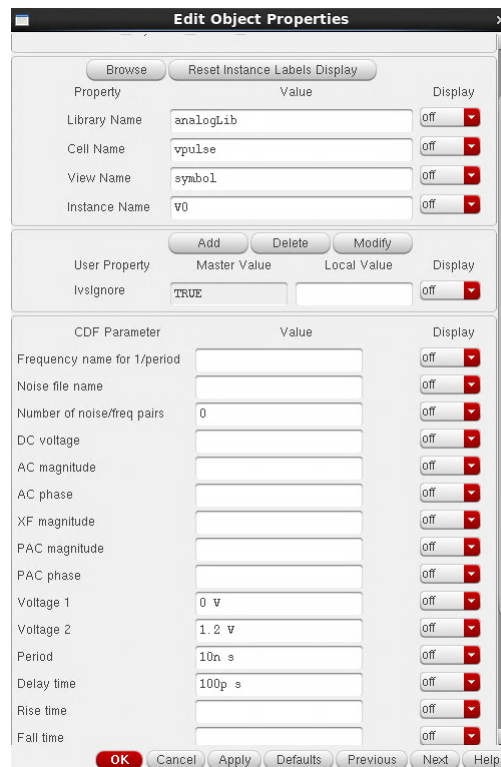


Figure 7.11:  $V_{source}$  Specifications

- Enter the required values for the sources. Shown are the specifications

for the D input of the DFF. The pulse is contained between 0 and 1.2 V with a start value of 0 V and a delay of 100 ps. The period of the pulse is 10 ns.

6. Finally, the test bench of the DFF is ready and can now proceed with the simulations.

#### 7.4.4 Performing Simulations Using Spectre

1. Once everything is set up, click on *Check and Save*. Without this step, we cannot proceed. Next, click on *Launch* → *ADE L* to launch the Analog Design Environment (ADE) window as shown in figure 7.12.

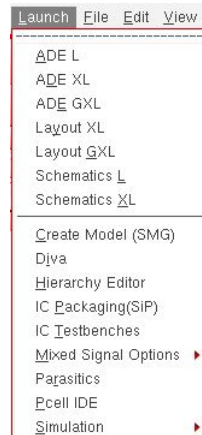


Figure 7.12: Launching ADE L

2. The window shown in figure 7.13 will pop up.

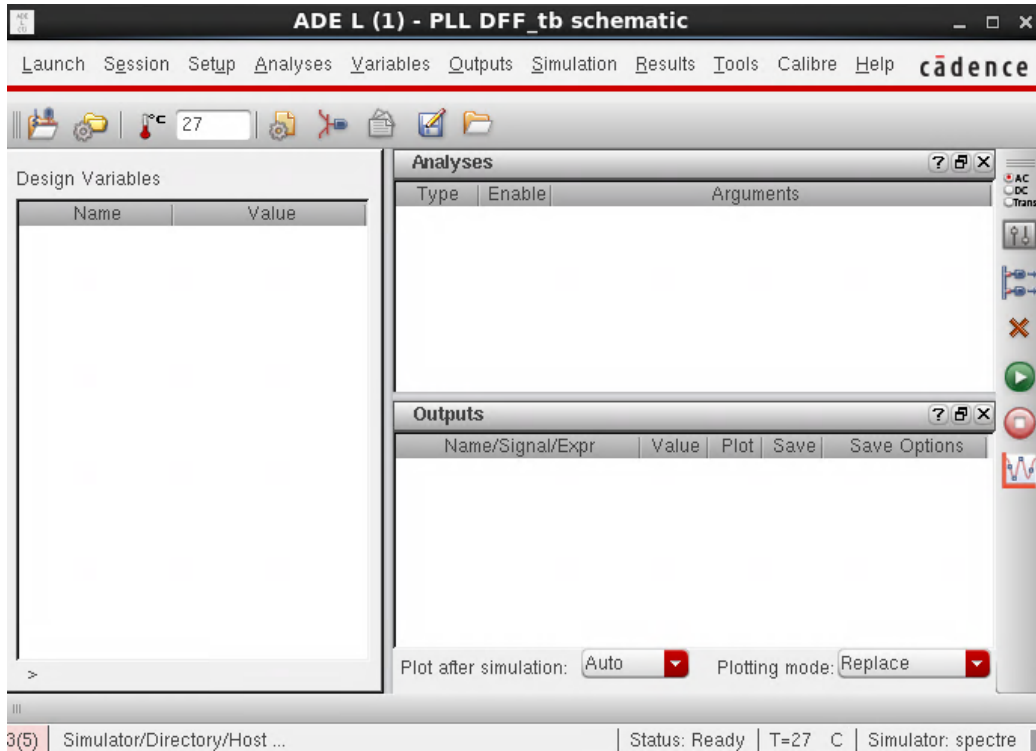


Figure 7.13: ADE Window

- It is imperative to ensure that the model libraries are correctly set and the default simulator is spectre. To check whether it is correctly set, go to *Setup* → *Model Libraries* and it should look something like the window shown in figure 7.14.

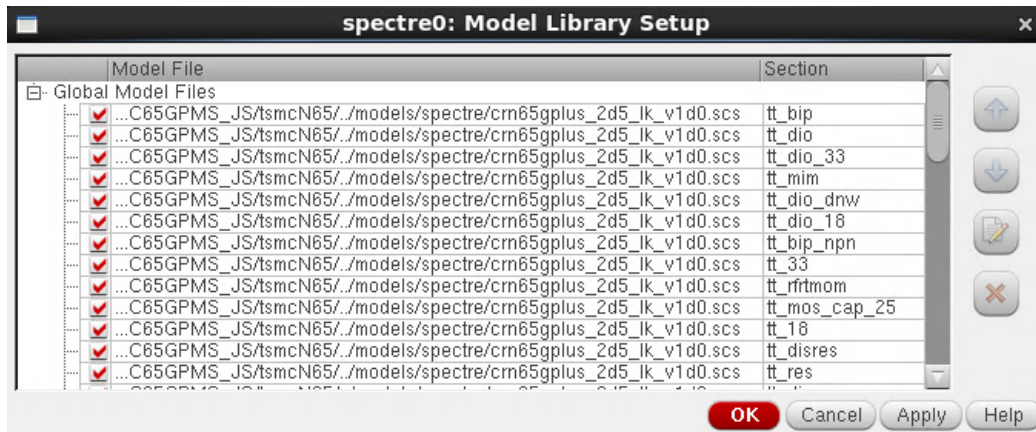


Figure 7.14: Model Library Setup

- Also change the simulator to spectre if it is not the default simulator. Figure 7.15 illustrates how to do it.



Figure 7.15: Simulator Setup

- Now the simulation environment is ready. Since we are analyzing the time-varying response of the DFF, we will perform transient analysis. To perform transient analysis, in the ADE window click on *Analyses* → *Choose* → *tran*. Choose the stop time to be 100 ns. This means that the simulation will run for 100 ns. This is shown in figure 7.16.

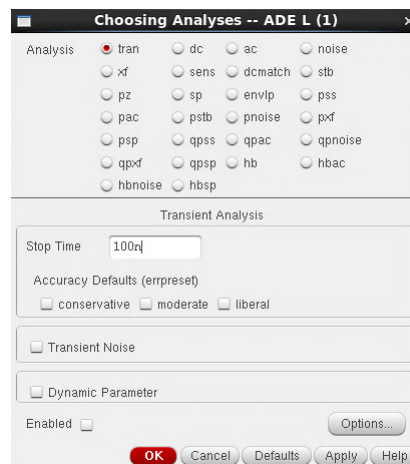


Figure 7.16: Choosing Analyses for Simulation

- Select the signals that need to be plotted by clicking *Outputs* → *To Be Plotted* → *Select On Schematic* as shown in figure 7.17. This will direct the user to the schematic and the required signals can be selected to be plotted.

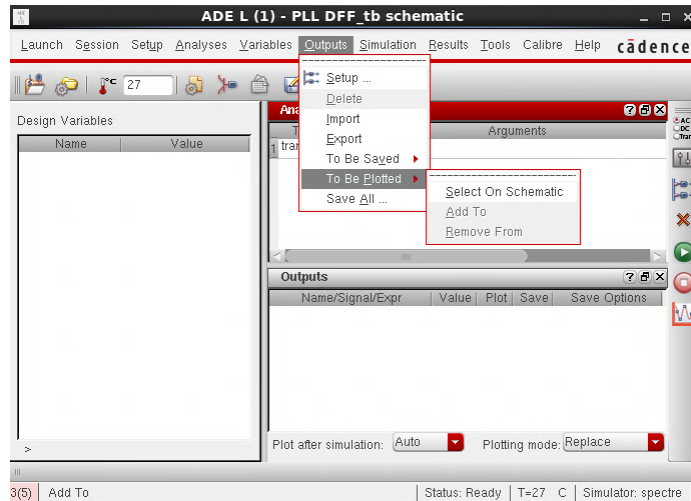


Figure 7.17: Selecting Signals to be Plotted

- Once everything is set up, click on the green arrow shown in figure 7.18. This will start the simulation. Once the simulation is completed and successful, the main virtuoso window will display the simulation successful. If not, error messages will appear.



Figure 7.18: Starting Simulation

#### 7.4.5 Transient Response

Based on the signals selected to be plotted, a new Visualization window will pop up displaying all the aforementioned signals.

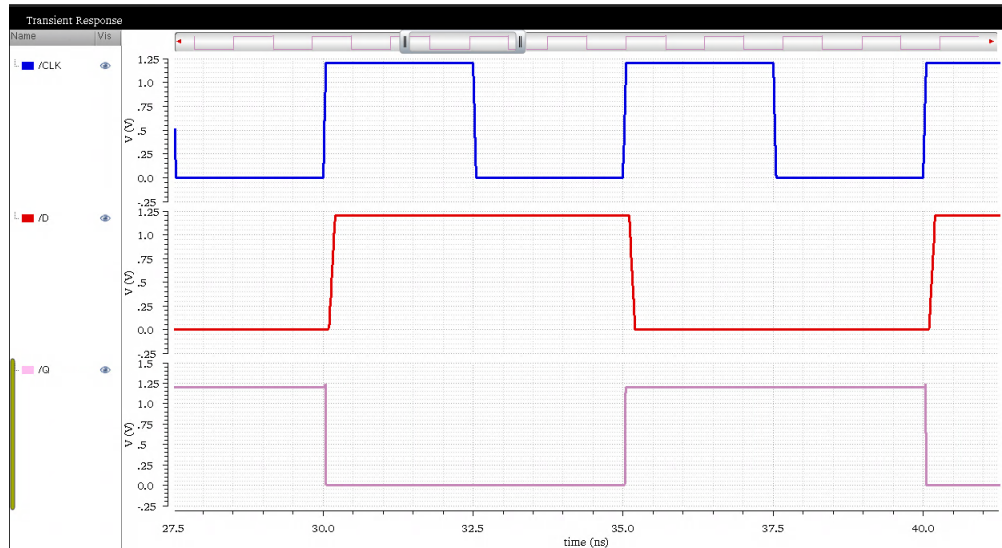


Figure 7.19: DFF Waveform

From the waveforms in figure 7.19, we notice that Q follows D on the rising edge of the clock verifying the working of the DFF.

## 7.5 PLL Simulation Using Cadence Virtuoso

Now that we have experience and practice using Virtuoso, we will move on to building the blocks of the PLL which are listed in section 7.5.1.

### 7.5.1 PFD+CP+LF Setup

1. The PFD used in this design is a NAND PFD which uses two-input, three-input and four-input NAND gates along with inverters. The CP is a bootstrapped charge pump and the loop filter is a low-pass filter. Construct the schematic and symbol of these three blocks and set up the testbench. The result will look similar to figure 7.20.



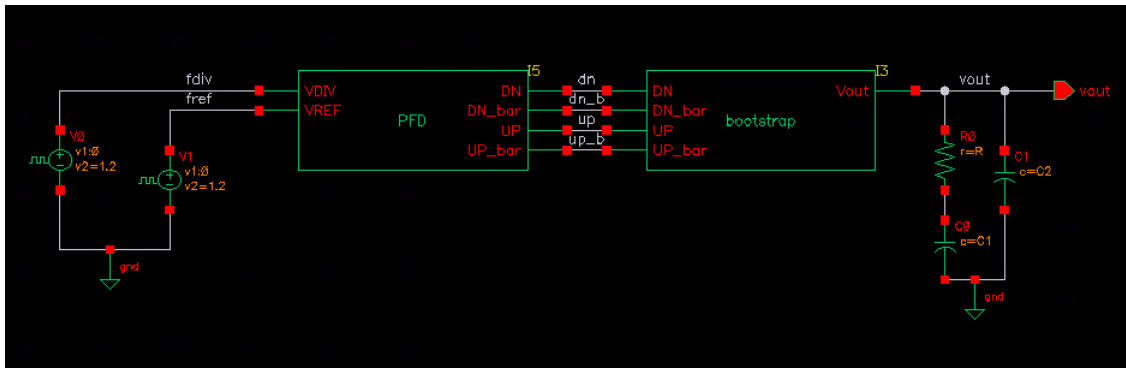


Figure 7.20: PFD+CP+LF Test Bench

- The fref and fdiv input are set up to have slightly varying frequencies and a phase offset in order to test the working of the PFD. The images in figure 7.21 and figure 7.22 illustrate their setup.

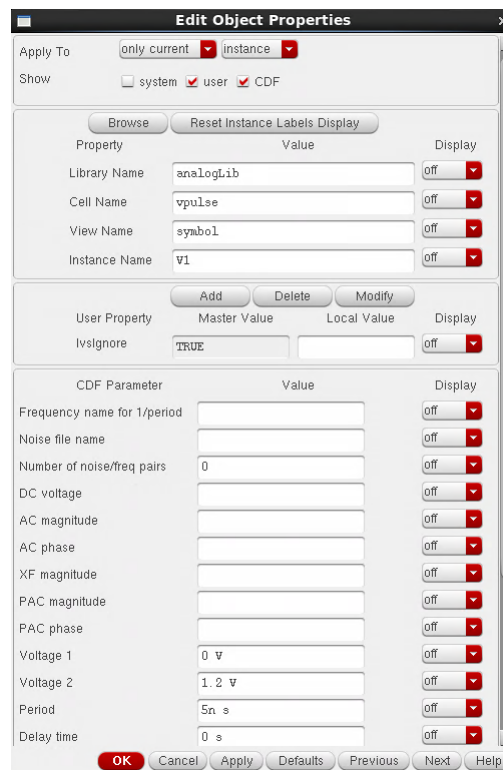


Figure 7.21: fdiv Setup

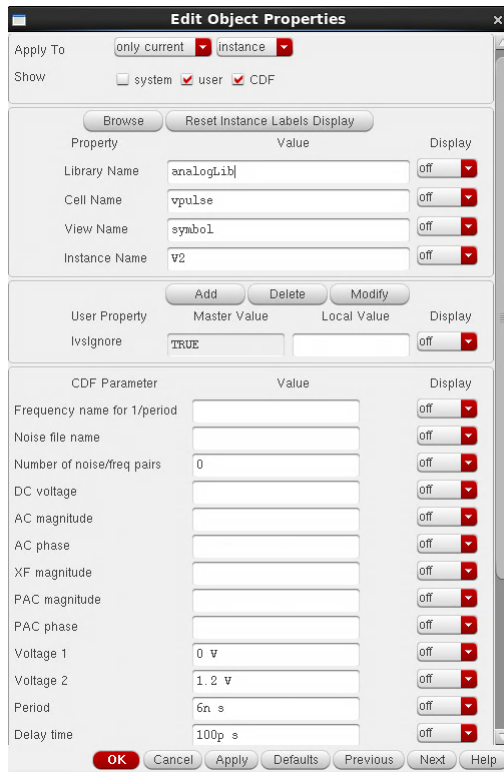


Figure 7.22: freq Setup

3. Launch the ADE L window and set up the transient analysis for a period of 100 ns. If the loop filter values are parametrized in the test-bench, we can set them manually in the ADE window. To do that click on *Variables* → *Copy From Cellview*. Once the parametrized values show up, set them as per the design requirements. The ADE window should now look like the one in figure 7.23.

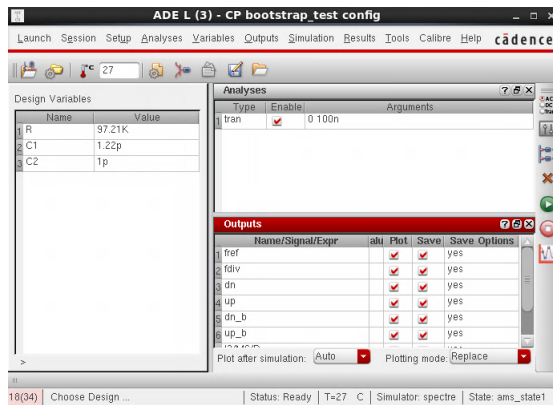


Figure 7.23: PFD+CP+LF ADE Window

- On simulating, we observe the waveforms in figure 7.24. We notice the current functionality of the circuit with 70 uA of charge pump current. The delay in the NAND PFD is approximately 32 ps.

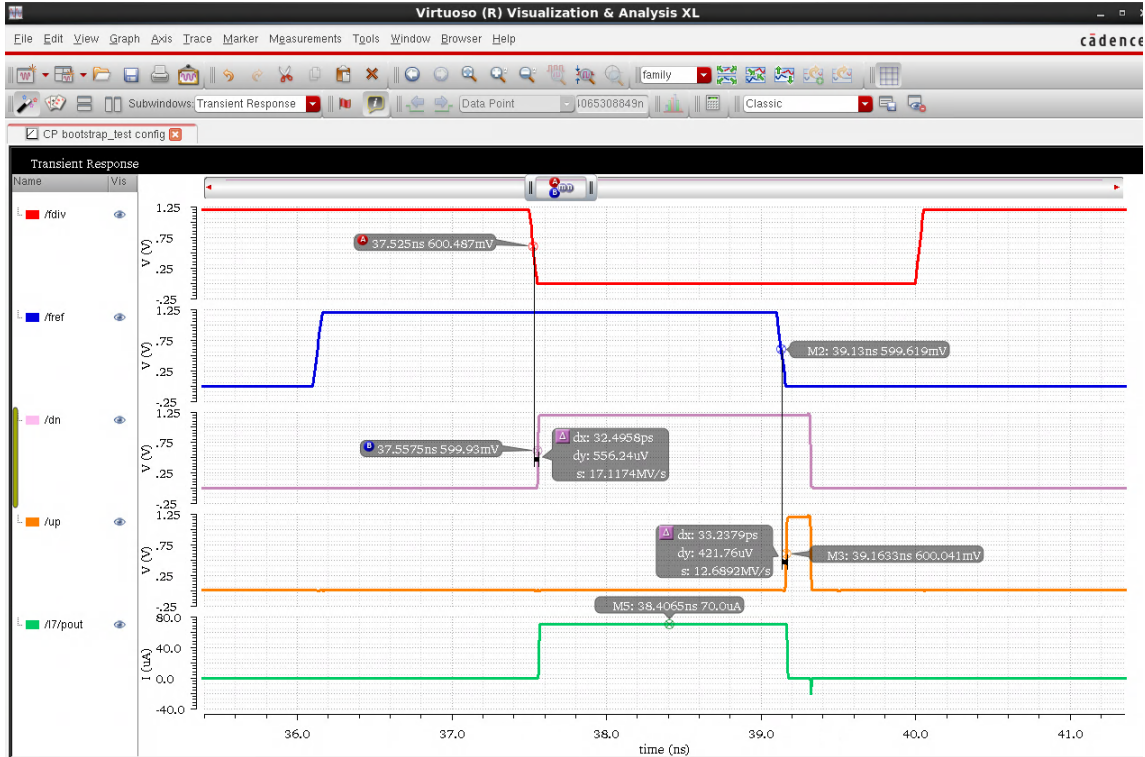


Figure 7.24: PFD+CP+LF Waveforms

## 7.5.2 VCO Setup

- The VCO used is a differential VCO using inverter chains with cross-coupled latches. However, for this tutorial, we will be using a single-ended VCO with a PMOS input. It consists of three inverter chains and an additional inverter for getting full output swing. The schematic and the corresponding test bench is shown in figure 7.25 and figure 7.26 respectively.

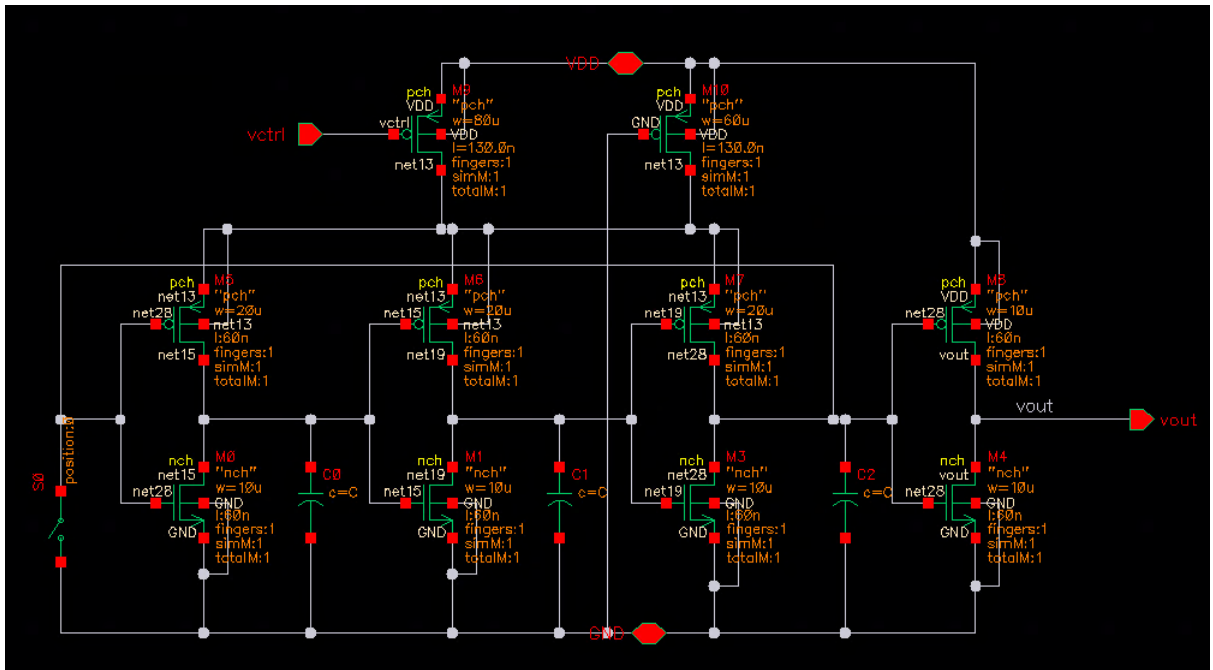


Figure 7.25: Single-Ended VCO Schematic

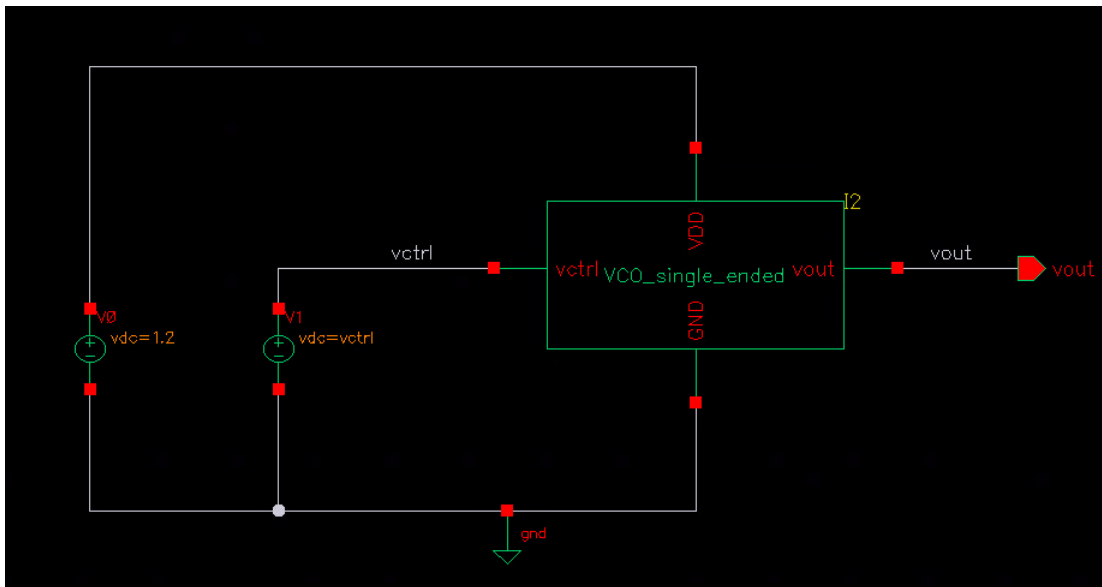


Figure 7.26: Single-Ended VCO Test Bench

2. Launch the ADE L window and set up the transient analysis for a period of 100 ns. Set the capacitance to 197.5 fF and  $V_{ctrl}$  value to 400 mV since that is the value we will be using in our design. The ADE window should now look like the one in figure 7.27.

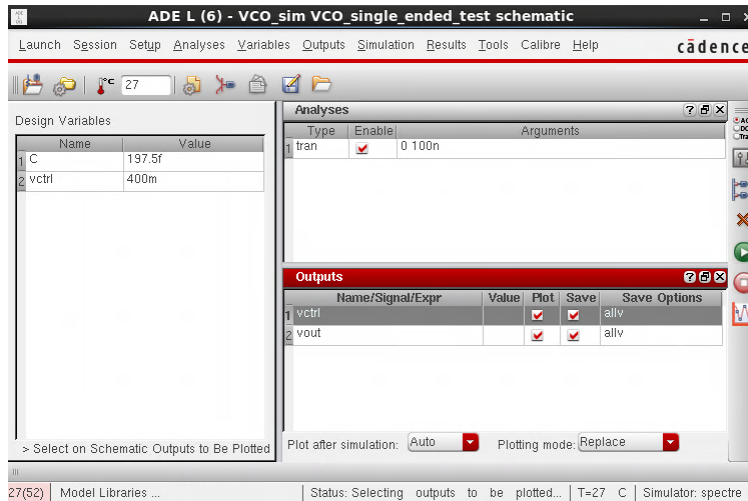


Figure 7.27: VCO ADE Window

- On simulating, in figure 7.28, we observe that the period is 156.25 ps which means that the VCO is operating at 6.4 GHz.

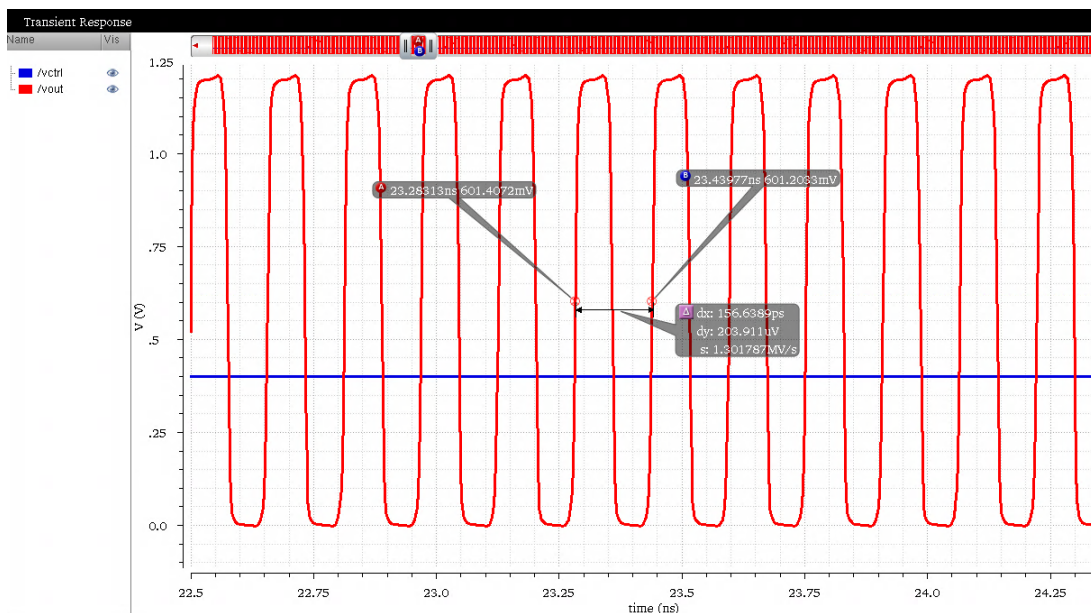


Figure 7.28: VCO Waveform

### 7.5.3 Plotting VCO Output Frequency vs. Control Voltage and VCO Gain

After simulating the VCO, it is important to observe and understand the figures of merit. Figures of merit include the VCO gain  $K_{VCO}$ ,  $V_{ctrl}$  tuning

range and phase noise. VCO phase noise is the major noise contributor to the PLL. This section presents a step-by-step tutorial on simulating and observing these parameters.

1. To plot the VCO frequency vs. the control voltage we will make use of the Calculator tool. Click on *Tools* → *Calculator* in the ADE window as shown in figure 7.29.

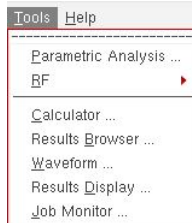


Figure 7.29: Calculator Tool

2. Click on vt. This will open the schematic of the test bench of the VCO. Select the output of the VCO.
3. Click on the functions panel and select freq. Then select average. The calculator expression will look like the one in figure 7.30.

```
average(freq(VT("/vout") "rising" ?xName "time" ?mode "auto" ?threshold 0.0))
```

Figure 7.30: Calculator Expression for Average of VCO Frequency

4. Go back to the ADE window. Click on *Outputs* → *Setup* → *Get Expression*. The expression in the Calculator window will now be displayed here. Give the output a suitable name. Figure 7.31 explains this step.

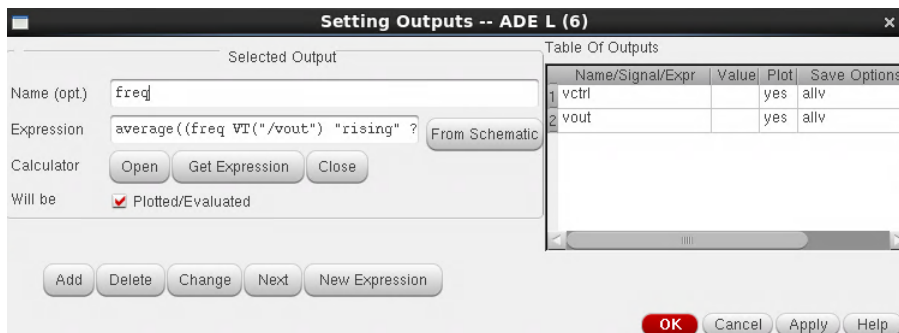


Figure 7.31: Extracting the Expression for the VCO Frequency

- Go to *Tools* → *Parametric Analysis*. Select the variable as *vctrl* and sweep it from 0 to 1.2 V in step sizes of 0.1. Click on the green button to initiate the parametric analysis as shown in figure 7.32.



Figure 7.32: Parametric Analysis

- Once the simulation is done, the output will be displayed as shown in figure 7.33. On taking the derivative of this plot, we can calculate the VCO gain. We notice that at 400 mV, the VCO frequency is 6.397 GHz and  $K_{VCO} = 521.6$  MHz/V.

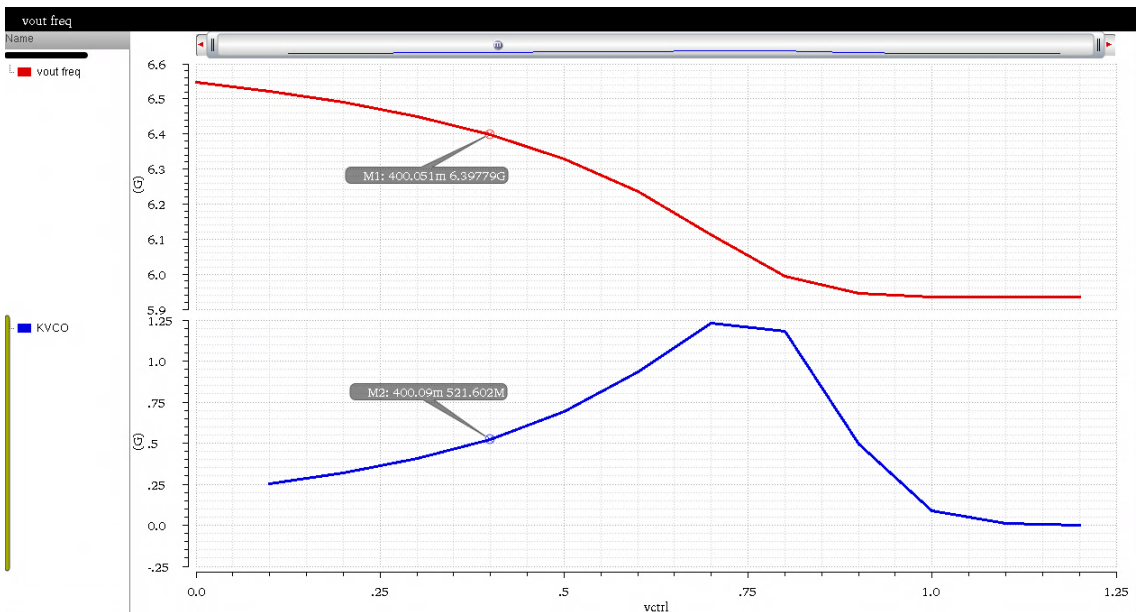


Figure 7.33: Frequency vs.  $V_{ctrl}$

#### 7.5.4 VCO Phase Noise

- In order to simulate VCO phase noise, it is necessary to run a Periodic Steady State (PSS) simulation first. To do that, go to *Analyses* → *Choose* → *pss* and set the parameters as shown in figure 7.34. The beat frequency is the VCO target frequency and hence we need to check



“oscillator” and select “vout” and “gnd” as the positive and negative nodes respectively.

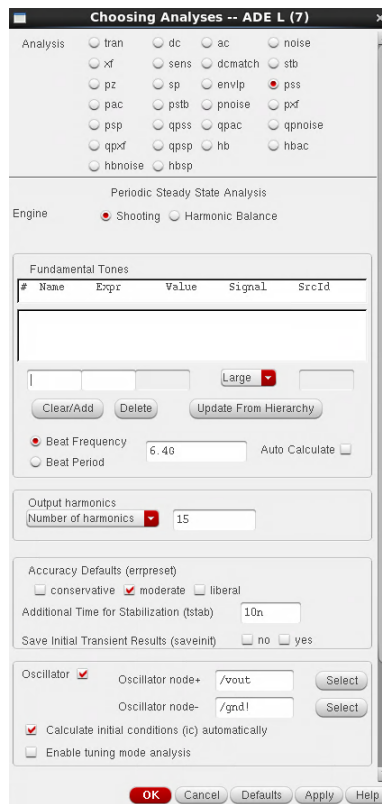


Figure 7.34: VCO PSS Setup

- For the phase noise simulation, go to *Analyses* → *Choose* → *pnoise* and set the parameters as shown in figure 7.35. We set the start and stop frequency from 1 K to 1 G. Once that is done, go back to the ADE window and start the simulations. Once the simulations are complete, we need to view the phase noise simulation. To do so, in the ADE window, click on *Results* → *Direct Plot* → *Main Form* and a window will appear as shown in figure 7.36. Choose “Phase Noise” and click the “Plot” button.



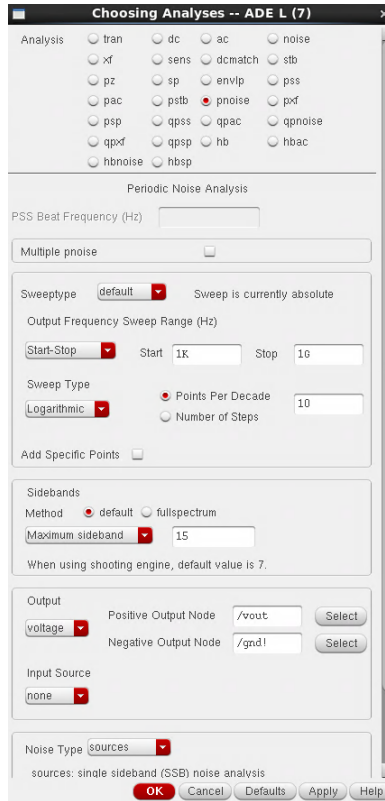


Figure 7.35: VCO Pnoise Setup

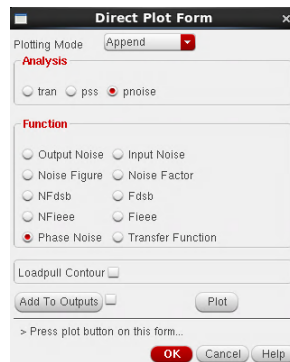


Figure 7.36: VCO Pnoise Plot Setup

3. The phase noise of the VCO is plotted vs. frequency offset as shown in figure 7.37. We notice that the phase noise for the VCO at a 1 MHz offset is -85.37 dBc/Hz which is reasonable considering our high-frequency of oscillation.

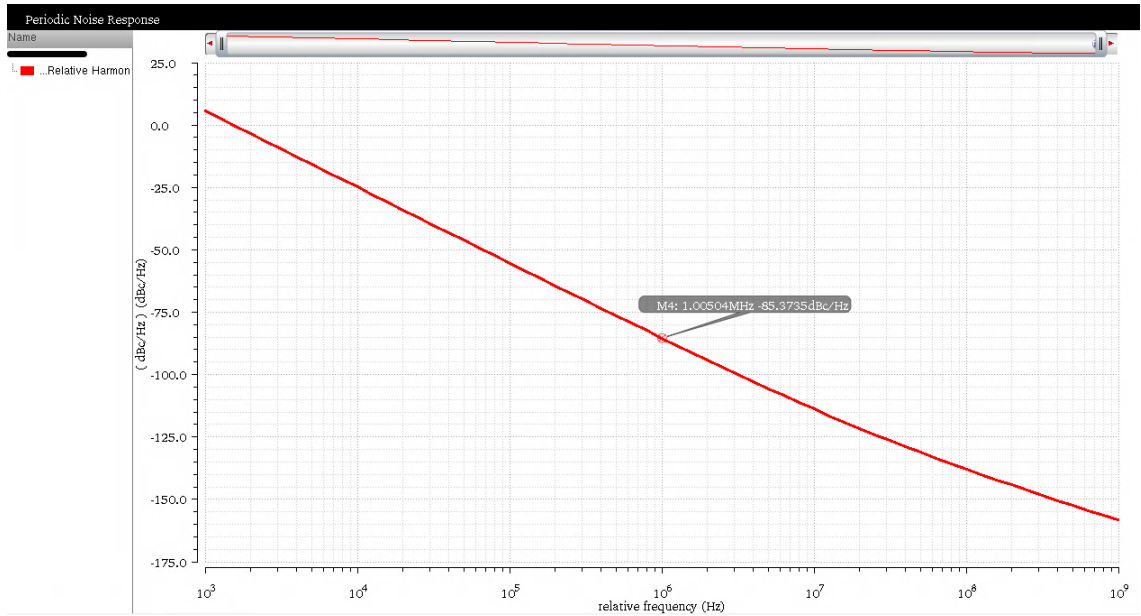


Figure 7.37: VCO Phase Noise

### 7.5.5 Divider Setup

1. Setup the divider testbench in a similar fashion to the other blocks. It should resemble the image shown in figure 7.38.

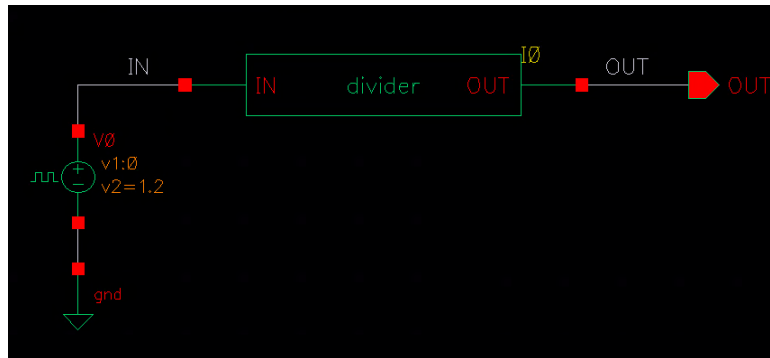


Figure 7.38: Divider Test Bench

2. In the ADE window as shown in figure 7.39, select the input and output signals and setup a transient analysis for 10 ns. We can also use the calculator to setup and calculate the frequencies of the input and output signals. On simulation, from figure 7.40, we can see that the input frequency is 6.4 GHz and is divided by a factor of 32 to output 200 MHz.

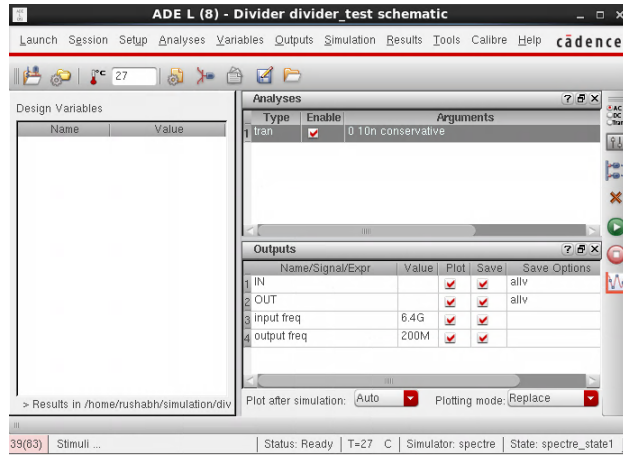


Figure 7.39: Divider ADE window

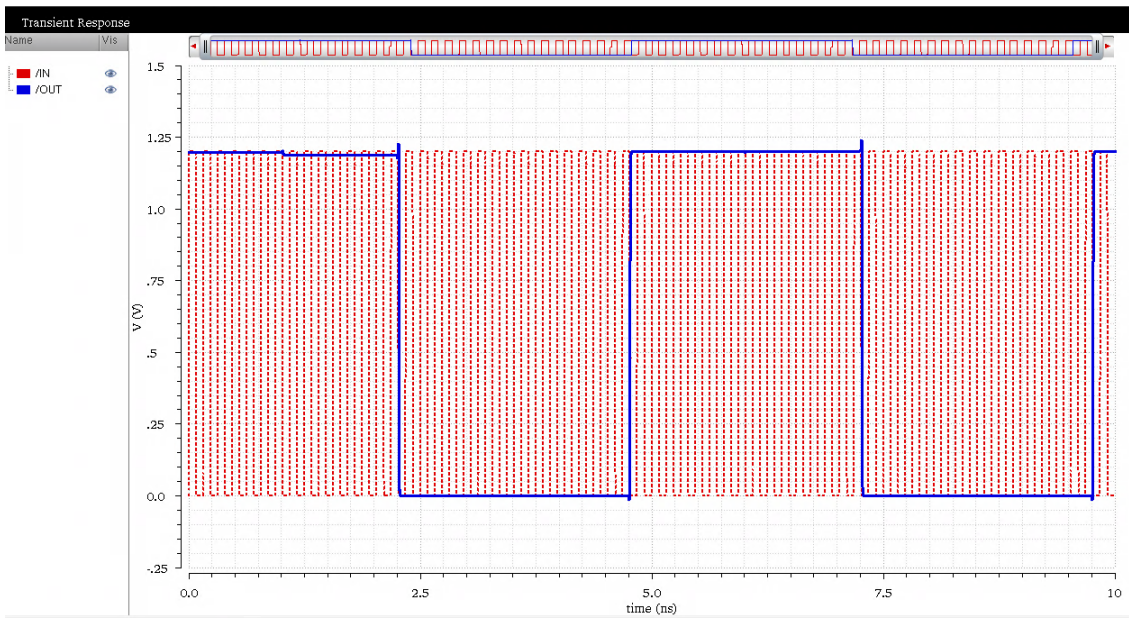


Figure 7.40: Divider Waveform

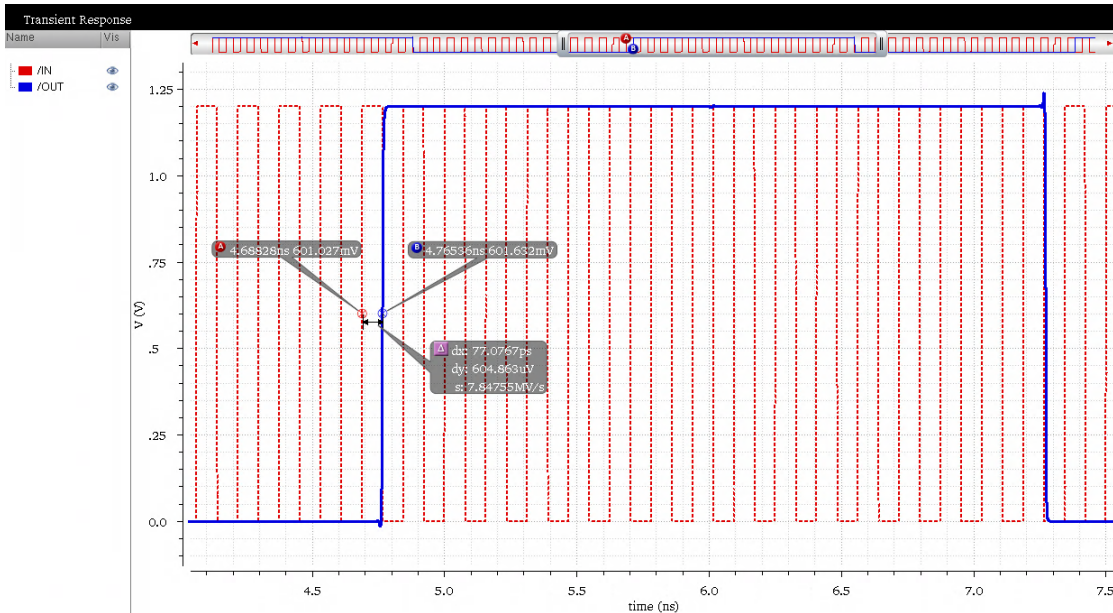


Figure 7.41: Divider Waveform (zoomed in)

3. The 16 pulses of the input correspond to half a pulse of the output which implies that there are 32 input pulses for 1 output pulse. Thus the divider is dividing by a factor of 32. Also, there is a delay of 77.07 ps between the input and the output as seen in figure 7.41.

### 7.5.6 PLL Integration and Simulation

1. Now that all our blocks are built, it is time to integrate them and simulate the entire PLL. To do this, we will create a new test bench where we will import symbols of the different blocks. The test bench should look like the one in figure 7.42.

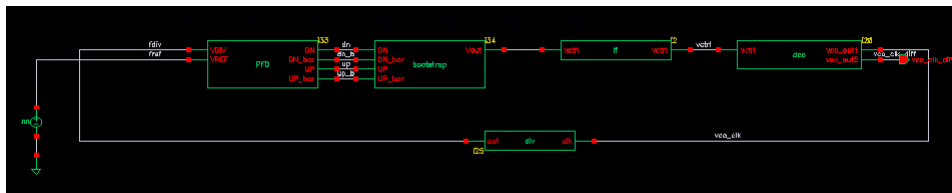


Figure 7.42: PLL Test Bench

2. There is just one input to the PLL which is the reference clock. Set up the reference clock as described earlier having a frequency of 200

MHz. Additionally, we should add a certain rise and fall time, delay time and pulse width to account for non-idealities of an actual crystal generating the clock.

3. With all transistor-level designs, the PLL will take longer to simulate and lock. Hence, in the ADE window, we need to run the transient analysis for 3 us. Select all the signals of interest to be plotted.
4. On simulation we observe the PLL characteristics both before lock and after lock. Figure 7.43 shows a picture of the PLL waveforms before lock is acquired. Here, the div signal leads the ref signal which causes the dn pulses. Due to the dn pulses, current is drawn out of the loop filter and vctrl is decreasing in steps. The frequency of the VCO clock is changing.

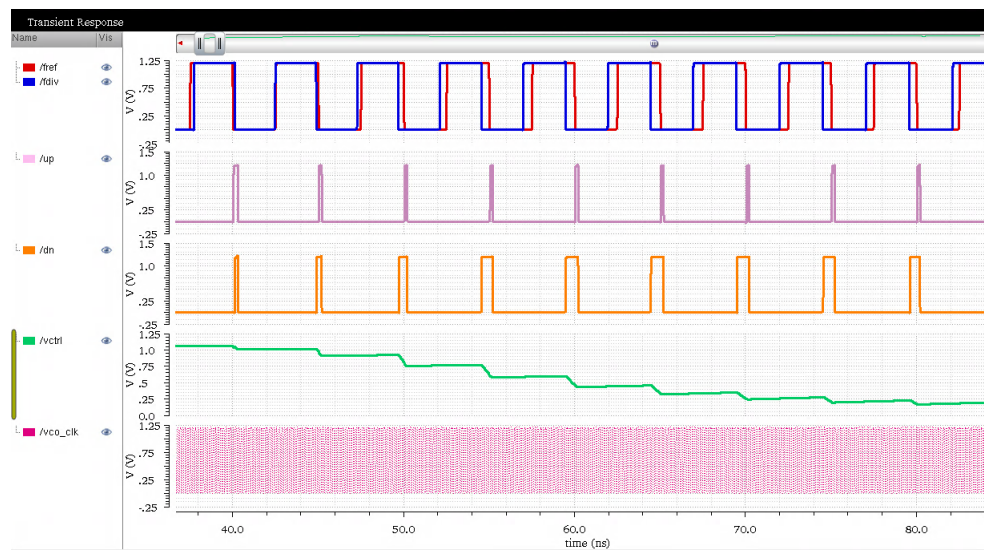


Figure 7.43: PLL Waveforms before Lock: Decreasing Control Voltage

5. Similarly, when the ref signal leads the div signal, we have up pulses which causes charge to be dumped into the loop filter and vctrl to increase in steps as seen in figure 7.44. However, since the control voltage is changing, the frequency of the VCO clock is not constant.



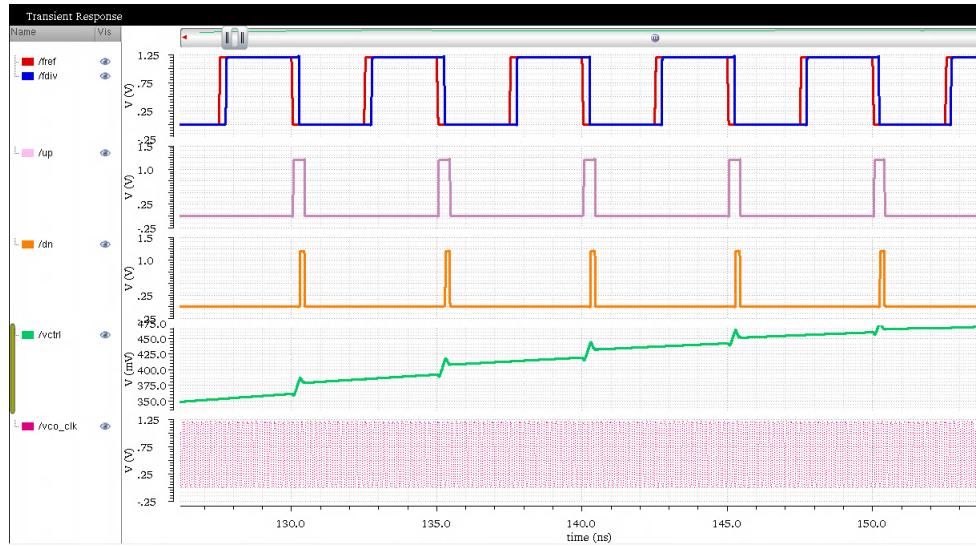


Figure 7.44: PLL Waveforms before Lock: Increasing Control Voltage

- When the PLL is locked, the control voltage settles to a constant value and the reference and divided signal match each other in phase and frequency as seen in figures 7.45, 7.46 and 7.47.

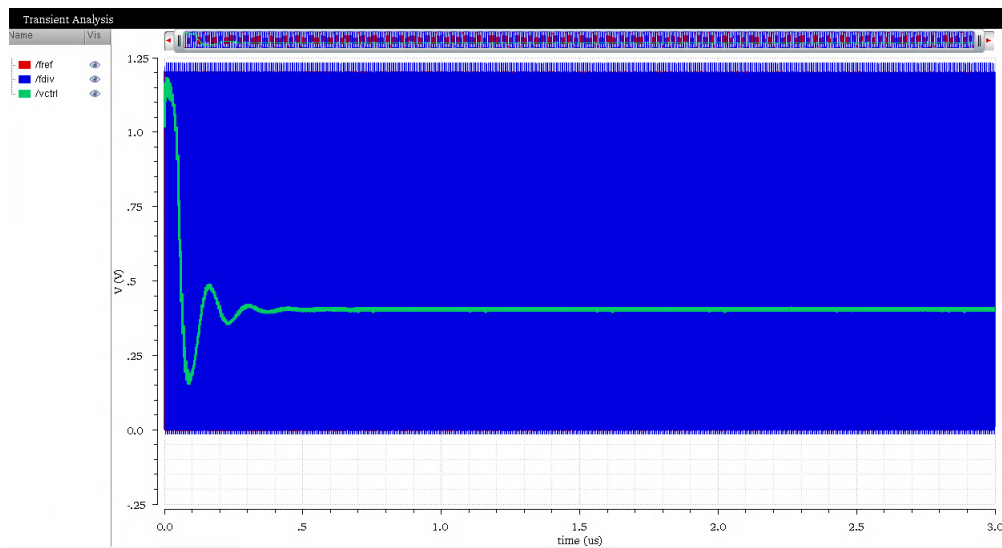


Figure 7.45: PLL Settled Transient Waveforms

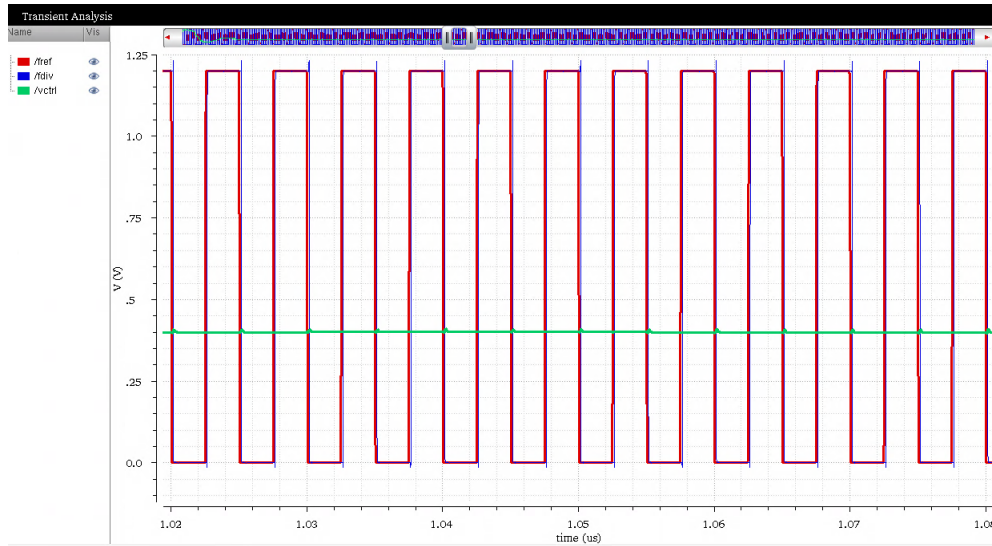


Figure 7.46: PLL Locked State: Reference and Divided Clock

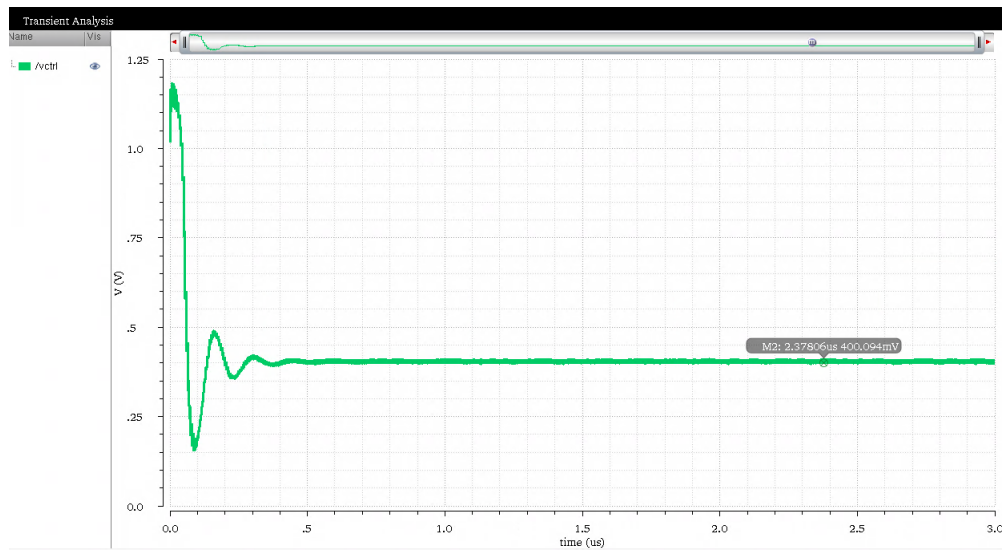


Figure 7.47: PLL Locked State: Control Voltage

- Ideally, we would expect the VCO frequency to be constant over time in the locked state. However, due to phase noise in the VCO, the clock frequency will slightly vary over time as seen in figure 7.48. In the time domain this corresponds to jitter. This is undesirable and unavoidable and efforts are needed to minimize this as much as possible.

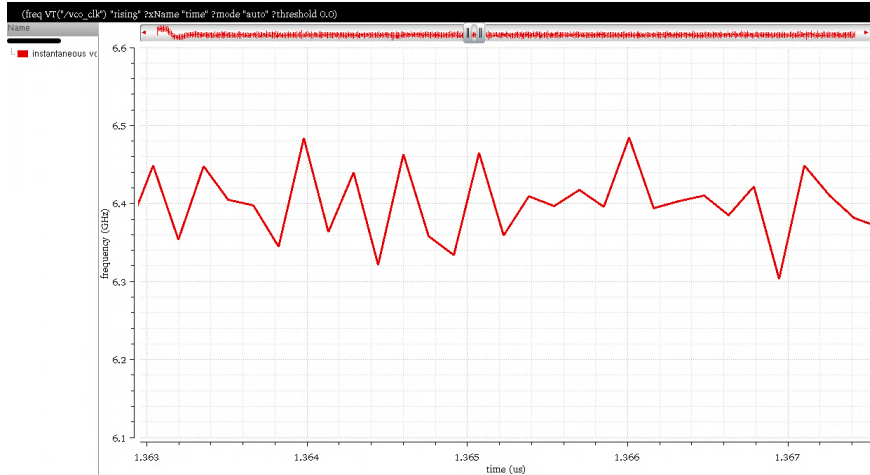


Figure 7.48: PLL Output Clock Frequency vs. Time

- The PLL designed will be used to drive the serializer and the driver. Hence the random and deterministic jitter are important figures of merit and need to be minimized to reduce timing errors. These errors could lead to a higher bit error rate (BER). In order to measure the deterministic jitter, we need to select the VCO clock and then click *Measurements* → *Eye Diagram*. A window like the one in figure 7.49 will pop up on the side. Enter the start time as 1.5 us, stop time as 1.55 us and period as 1/6.4 G (the start and stop time should be selected at a time interval where the PLL is locked). Once these values are entered, hit “Plot Eye”.

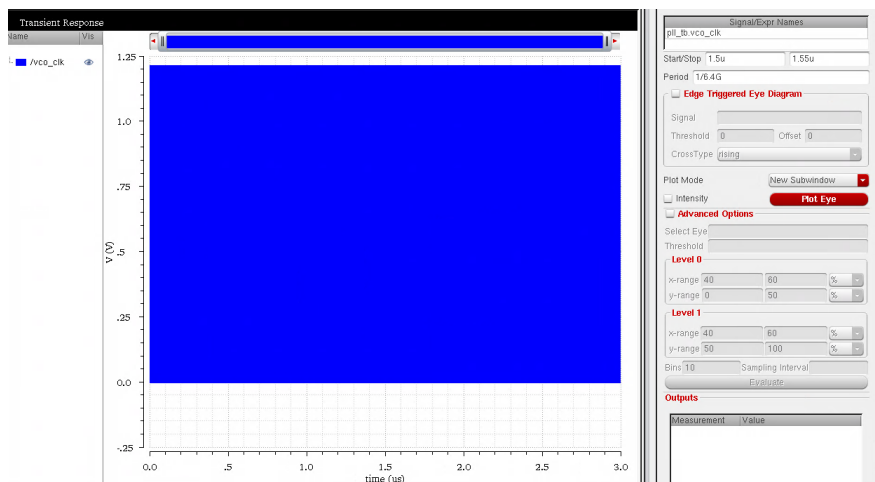


Figure 7.49: PLL Output Eye Diagram Setup



We notice that the output clock has some deterministic jitter in figure 7.50.

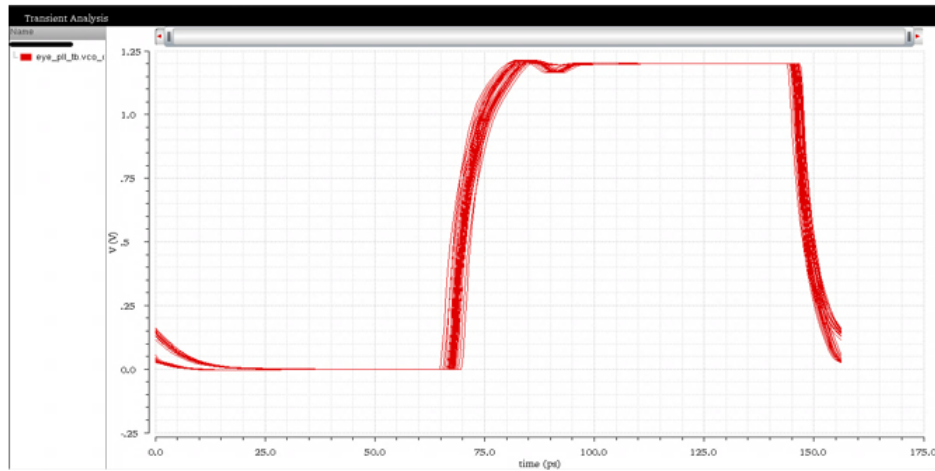


Figure 7.50: PLL Output Eye Diagram: Deterministic Jitter

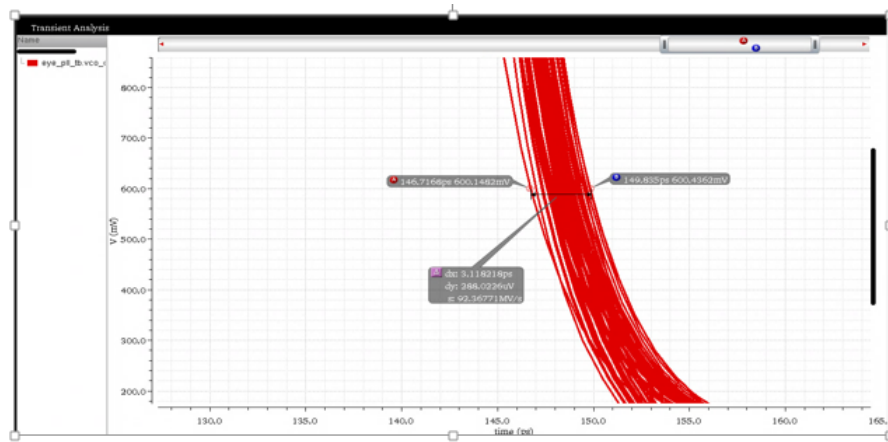


Figure 7.51: PLL Output Eye Diagram: Deterministic Jitter (zoomed in)

By putting markers at the 0.6 V crossings, we can calculate the deterministic jitter of our PLL to be 3.11 ps as seen in figure 7.51.

# CHAPTER 8

## DISCUSSION

### 8.1 Conclusion

This thesis has summarized the design and implementation of a PLL circuit and has presented a detailed tutorial on the design and simulation of the same at the behavioral as well as transistor level using Cadence Virtuoso. The PLL outputted a clock at 6.4 GHz with -85 dBc/Hz phase noise and 3.11 ps deterministic jitter. There is tremendous scope for improvement in this design with focus on phase noise reduction, random and deterministic jitter reduction and exploring different architectural designs. With increasing data rates, PLLs need to operate at a higher clock frequency pushing the limits for improving the figures of merit. The purpose of this thesis was to gain an in-depth understanding of high-speed serial links at the behavioral as well as transistor level and also to provide a blueprint for future students seeking to pursue a career in the field of analog and mixed-signal IC design.

### 8.2 Future Work

Designing a PLL at 6.4 GHz required the designer to use the building blocks that improved the performance metrics. However, with trends of higher data rates, the need for high-frequency PLLs is paramount. Besides implementing analog PLLs, there is a need for All Digital PLLs (ADPLLs). Section 8.2.1 presents a discussion on ADPLLs.

## 8.2.1 ADPLL

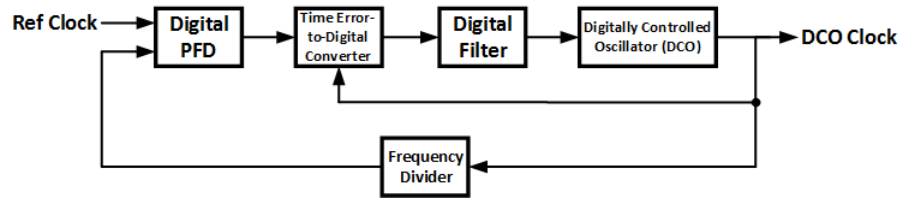


Figure 8.1: ADPLL Block Diagram

An ADPLL replaces the charge pump with a time error-to-charge digital converter, the loop filter with a discrete-time digital filter (generally second- or third-order sigma-delta) and the Voltage Controlled Oscillator (VCO) with a Digitally Controlled Oscillator (DCO) as shown in figure 8.1 [8].

The PFD detects the frequency and phase mismatch between the reference and divided clocks. The error between the two signals is detected and a control word is generated which is proportional to this error. Based on the frequencies of the two clocks, the control word either increases or decreases. This is then fed to the DCO which outputs the high-frequency clock.

ADPLLs are less susceptible to process and noise variations than analog PLLs. The use of ADPLLs increases design portability and testability. There is a greater flexibility in loop bandwidth, i.e. huge capacitors are not needed for lower BW thus reducing on-chip area and making them attractive for use in high-performance microprocessors [7].

## REFERENCES

- [1] S. Palermo, *CMOS Nanoelectronics Analog and RF VLSI Circuits, Chapter 9*. New York City, N.Y.: McGraw-Hill, 2011.
- [2] M. Assaad, “Design and modelling of clock and data recovery integrated circuit in 130 nm cmos technology for 10 gb/s serial data communications,” Ph.D. dissertation, Univ. of Glasgow, Glasgow, 2009. [Online]. Available: [theses.gla.ac.uk/707/1/2009assaadphd.pdf](http://theses.gla.ac.uk/707/1/2009assaadphd.pdf)
- [3] B. Razavi, “Design of monolithic phase-locked loops and clock recovery circuits.” [Online]. Available: [http://www.ee.ryerson.ca/fyuan/Razavi\\_PLL\\_Tutorial.pdf](http://www.ee.ryerson.ca/fyuan/Razavi_PLL_Tutorial.pdf)
- [4] A. A. Abidi, “Phase noise and jitter in cmos ring oscillators,” *IEEE Journal of Solid-State Circuits*, 2004.
- [5] M. Mansuri, “Low-power low-jitter on-chip clock generation,” Ph.D. dissertation, UCLA, 2003.
- [6] P. Hanumolu et al., “Analysis of charge-pump phase-locked loops,” *IEEE Transactions on Circuits and Systems-I*, vol. 51, no. 9, pp. 1665–1674, 2004.
- [7] D. Fischette, “First time, every time - practical tips for phase locked loop design.” [Online]. Available: [http://www.ee.ryerson.ca/fyuan/Razavi\\_PLL\\_Tutorial.pdf](http://www.ee.ryerson.ca/fyuan/Razavi_PLL_Tutorial.pdf)
- [8] A. Babu and others., “All digital phase locked loop design and implementation.” [Online]. Available: [http://www.mit.edu/bdaya/www/All\\_Digital\\_Phase\\_Locked\\_Loop\\_Design\\_and\\_Implementation.pdf](http://www.mit.edu/bdaya/www/All_Digital_Phase_Locked_Loop_Design_and_Implementation.pdf)