DESIGN OF A PHASE LOCKED LOOP BASED CLOCKING CIRCUIT FOR HIGH SPEED SERIAL LINK APPLICATIONS

BY

RISHI RATAN

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, 2014

Urbana, Illinois

Adviser:

Professor José Schutt-Ainé

ABSTRACT

Technology scaling and unprecedented growth in demand for ubiquitous, fast, robust computing have been the driving forces leading the innovations in high-speed interfaces. With the rise of heavy duty data centers to handheld mobile devices, the desire for faster, low-power integrated inter-IC communication protocols is at an all-time high and has led the roadmap of the semiconductor industry, making it one of the fastest growing yet fiercely competitive industries. With the growing needs for ultra-low power yet multi-Gbps signaling in both wired as well as wireline applications, integrated systems on chip (SoCs) have become mainstream critical components in modern computing systems. The ability to process and access 'big-data' is the fundamental demand in modern society where every second saved in prompt communication as well as computation of information is critical. In order to meet these needs of fast, robust signaling over the same old "lossy" channels, the clock-frequencies need to scale accordingly and clever I/O links need to be developed. The most crucial component of any high-speed I/O link is the clocking circuitry: clock generator at the transmit (TX) end and clock-recovery unit on the receive (RX) end.

This thesis provides an in-depth tutorial on circuit design, analysis and simulation of on-chip PLL based clocking generator circuits for high-speed serial link applications. An overview of high-speed links, along with the basic building blocks that make up a serial link, is presented. The fundamentals of PLLs are introduced and a complete guide to analysis and simulation of a charge-pump phase-locked loop based clocking circuit at both behavioral as well as transistor levels is presented for use as a synthesizer in a serial link. Finally, a survey of potential future research areas to explore for both PLLs in high-speed links as well as the complete serial link is provided with an emphasis on signal integrity applications for future students pursuing graduate studies in the fields of Signal Integrity and Mixed-Signal IC Design.

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

ACKNOWLEDGMENTS

As I approach the finishing stages of my graduate career, looking back there are countless individuals who have helped make this journey special and memorable. Graduate school is full of numerous uncertainties, various road-blocks in terms of design and implementation of ideas; thus it is a journey that though embarked on by one individual, really is a culmination of the efforts of many people who have helped behind the scenes along the way. At the end of it all, it is this support system that I am deeply indebted to for helping each step of the way, motivating me in the most difficult of times where there were only questions and no answers.

First and foremost, I want to thank my wonderful advisor Professor José Schutt-Ainé for being my mentor, always motivating me to push myself to the next level and being there for me like family. Dr. Schutt-Ainé was one of the first to see potential in me back when I was just a sophomore and had no direction in what I wanted to pursue in my career. He introduced me to the field of circuits and sparked an interest in engineering. He has always been extremely patient throughout the countless occasions I have visited his office hours. Being a mentor and father-figure for me throughout my educational career at UIUC, Prof. Schutt-Ainé has been the biggest contributor to my successes throughout my collegiate career and will forever be a role-model who I look up to as I start my professional career in industry.

Secondly, I would like to convey my heartfelt thank you to Professor Pavan Kumar Hanumolu for guiding me throughout this thesis as a mentor. Prof. Hanumolu stood by me with patience throughout the course of this thesis work, and took time off from his busy schedule in providing me formal lectures on PLLs one-on-one in his office at a time when it was very hectic for him given that he had just moved to UIUC. Without Dr. Hanumolu's constant guidance and feedback this thesis would never have been completed.

Thirdly, I would like to sincerely thank my wonderful colleagues, mentors and friends in graduate school: Da Wei, Xu Chen, Tom Comberiate, Romesh Nandwana, Mrunmay Talegaonkar, Yubo Liu, Ahmed Elkholy, Saurabh Saxena, Tejasvi Anand, Guanghua Shu, Woo-Seok Choi, Drew Newell, Xinying Wang, Jerry Yang and Maryam Hajimiri for always being patient to answer all my questions, solving arcane problems during the simulation and design process. These fine graduate students have stood by me since day one, provided constructive criticism on my work, painstakingly critiqued every figure and most importantly constantly pushed me towards striving for nothing short of excellence.

Furthermore, I am extremely grateful to Professor Steven Franke, Professor Christopher Schmitz, Professor Milton Feng, Professor Elyse Rosenbaum, Professor Naresh Shanbhag, Professor Venugopal Veeravalli and Dr. Chandrashekhar Radhakrishnan for supporting me throughout my educational career at UIUC and mentoring me every step of the way. Even when the chips were down, they always believed in me and helped me gain an opportunity to pursue my graduate studies.

I thank my family for always being by my side, motivating me to move forward whenever I was faced with roadblocks. Lastly, I am eternally grateful to my friends Ian Wetherbee, Rohan Bambery, Eric Iverson, Pourya Assem, Sai Zhang, Min-Sun Keel, Anish Chivukula, Eclair Hanjing Gao, Dennis Yuan, Jerry Sun, Nishant Nookala and Eric Kim for always standing by me throughout my ECE career at UIUC. I am positive I have missed a few people from the above list but I truly am very thankful to all those who I have interacted with throughout my time at UIUC. Additionally, I am very grateful for getting an opportunity to work with wonderful undergraduates Rushabh Mehta, Ishita Bisht, Brady Salz, Ankit Jain, Haodong Guo, and Pradyut Paul who have worked very diligently in getting the High Speed-SerDes Design project off the ground from scratch.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Over the last 50 years, advances in Semiconductor Fabrication Technology (SFT) coupled with innovations in Integrated Circuit (IC) technology scaling have fueled an unparalleled growth in computing. This aggressive scaling has revolutionized every aspect of modern society and triggered an insatiable demand for faster data rates and higher processing power resulting in clock frequencies and corresponding data rates approaching multi-GHz and multi-Gbps ranges in everyday computing devices like personal computers, mobile devices, entertainment consoles and other such devices. Access to information promptly and efficiently in terms of power and portability/ease of use is the major driver pushing the limits of IC technology. Thus, the need for robust, high-speed, low-power and highly integrable compact systems-on-chip (SOCs) is paramount for inter-IC communication interfaces such as network switches, processor/memory interfaces across backplane channels. In order to meet this growing demand for wideband systems, the Input/Output (I/O) links need to scale proportionally with the increased data-rate scaling; however in reality the off-chip I/O bandwidth (BW) has not scaled appropriately and has become a major bottleneck in the overall system performance. Furthermore, along with the off-chip I/O BW limitations, the channel as well as package/connector interfaces have not scaled with SFT making the design of high-speed I/O links extremely challenging due to the increased transmission line loss, crosstalk, and signal distortion resulting in intersymbol interference. As the demand for high data-rate interfaces has skyrocketed, the clock-frequencies needed to realize such systems have correspondingly reached the multi-GHz range necessitating the use of phase-locked loops (PLLs) for on-chip clock synthesis.

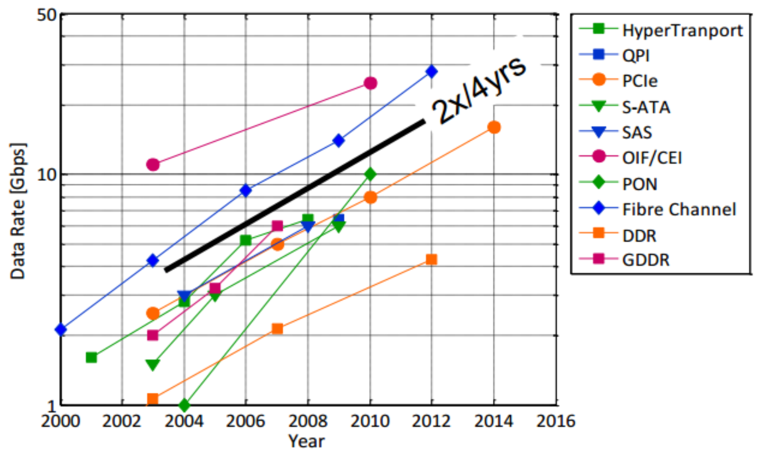


Figure 1.1: IO Link Signaling Data-Rate Trends

Figure 1.1 shows the trends in data-rate scaling of I/O high-speed signaling links as forecasted by the International Solid State Circuits Conference (ISSCC) 2011 annual semiconductor roadmapping report [5]. The key take-away from this graph is that the data-rates in inter-IC communication links are scaling by a factor of 2X every 4 calendar years while IO channel BW remains the same. The I/O BW scaling problem aside, the ability to design robust, low-jitter on-chip clock synthesizer circuits is in itself an extremely challenging task. Though research in the field of integrated high-frequency clocking circuits has been going on for the past two-decades and lots of innovative designs have come into existence, one common facet missing from the whole paradigm is complete documentation on process of simulation using the Electronic Design Automation (EDA) tools [6]. Most of the literary works in this area primarily focus on novel system level designs for PLL based clock synthesizers and some go into transistor-level details of the sub-blocks; however very rarely do any of the prominent works describe the actual simulation process. As the clock-frequencies scale and demand for robustness in the on-chip synthesizers increases, circuit designers also need to be aware of potential Signal-Integrity (SI) problems associated with their intricate designs. Since the channel BW is essentially the same at high-frequencies of operation the PCB traces act as transmission-lines (TLs) leading to severe degradation in signal quality due to reflections, ringing and cross-talk effects. Thus, every integrated circuit designer will invariably face SI problems in their design which until recently was not a concern as the frequencies were low enough that digital design did not require a formal understanding of signal integrity during the development as well as verification process. Therefore, the motivation for this thesis is to bridge the gap between circuit design and simulation for signal-integrity engineers who need the basic expertise in mixed-signal design process to be able to provide the required assistance to IC designers on designing high-speed SI aware systems.

1.2 Outline

This thesis is organized to serve as a training manual for students pursuing mixed-signal integrated circuit design as their field of study in graduate school. The goal is for this thesis to be their go-to guide to grasp a high-level understanding of high-speed links and learn the simulation setup/procedure to validate PLL based clocking circuits using the popular EDA tool Cadence Virtuoso.

1. Chapter 1 provides an introduction to the research problem describing the need for high-speed serial links and their future trends.

2. Chapter 2 provides an overview of high-speed links with an emphasis on describing each of the building blocks, figures of merit to characterize these links and lay the motivation for the industry-wide shift from parallel to serial-link design for low power, cost-effective robust I/O link design.

3. Chapter 3 describes the fundamentals of Phase-Locked Loops (PLLs) and provides a brief overview of their ubiquitous use in modern day wireline/wireless systems.

4. Chapter 4 covers a special class of PLLs, Charge-Pump PLLs, and provides a linear model for small-signal as well as noise-analysis of these PLLs.

5. Chapter 5 presents the transistor-level design of a Charge-Pump based Integer-N clock generator circuit operating at an output frequency of 1.6GHz.

6. Chapter 6 describes the procedure for behavioral modeling and simulation using Verilog-AMS for the clock-generator circuit described in Chapter 5.

7. Chapter 7 describes the procedure for transistor-level simulation for the clock-generator circuit described in Chapter 5.

8. Chapter 8 concludes the thesis with a discussion of the take-aways from the clocking circuit designed earlier and provides a brief anecdote on the signal integrity focus areas in high-speed link design and lists design improvements on the basic Integer-N analog clock generating circuit to accommodate industry trends within the field.

9. Lastly, the Appendix provides a step-by-step guide for installing and configuring the Cadence Virtuoso environment.

CHAPTER 2

HIGH SPEED SERIAL LINKS OVERVIEW

2.1 Simple Link Design

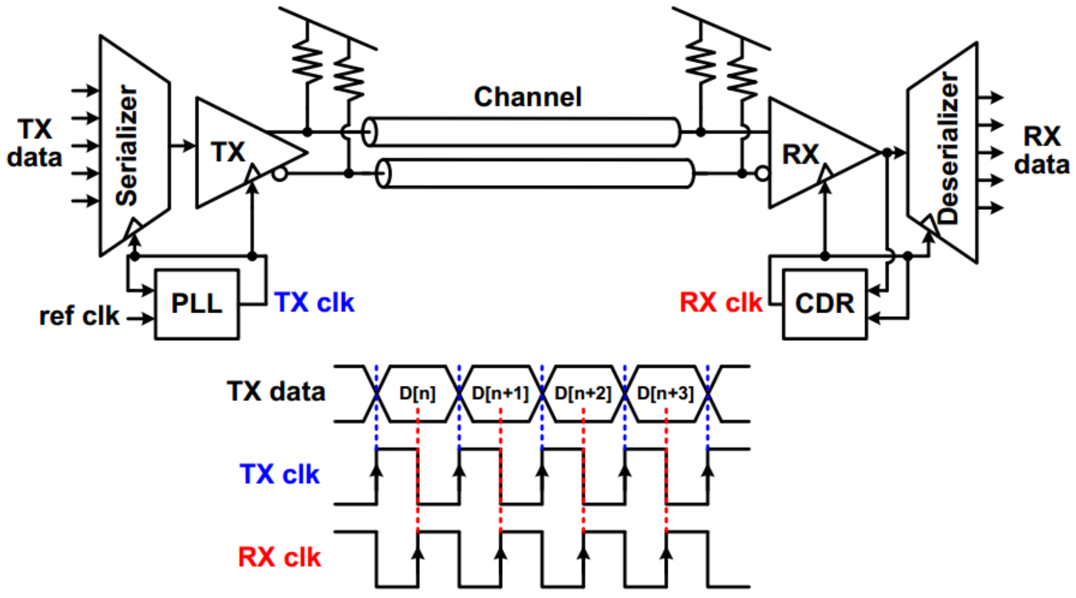


Figure 2.1: Typical High-Speed Link Block Diagram

Generalized model of a High-Speed Serial Link (HSSL), as shown in Figure 2.1 [1], consists of a serializer and transmitter (TX) driven by a PLL clock synthesizer, a channel and a receiver (RX) and Deserializer driven by a Clock-Data Recovery (CDR) unit. The serializer accepts the incoming parallel data-stream and converts it into a serial data-stream which is then sent to the transmitter. The TX generates a train of pulses depending on the data symbols to be transmitted across the channel and the pulse-width which is determined by the timing instant of the transmit clock at both begin/end/edges. The receiver basically comprises a sampler and a decision circuit whose purpose is to sample the received data-bit stream from the channel and recover, both the transmitted data as well as the clock. Once the receiver recovers the transmitted serial bit-stream it is sent to the Deserializer block whose job as the name suggests is to convert the received serial data back to its original parallel form for future interfaces.

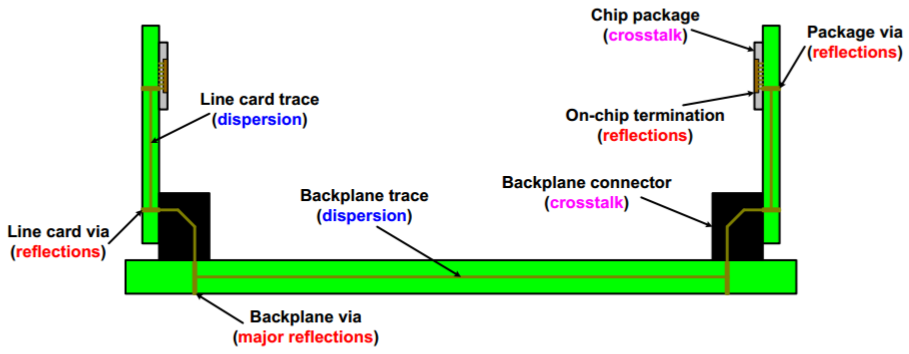


Figure 2.2: Typical Backplane Channel Interface

As discussed in the previous chapter, though the desired data-rates have scaled along with technology scaling, the off-chip channel I/O BW has remained the same. The channel is the electrical path between the TX and RX blocks and in inter-IC communication systems typically comprises printed circuit-board (PCB) traces, vias, connectors and other such I/O interface components. Generally speaking, in high-speed I/O interfaces the channel is typically a 'backplane' which essentially connects two PCBs together and typically looks like the interface shown in Figure 2.2 [1]. The channel is something that the designer has no control over and is thus just a 'known unknown' to the link designer. It is known in the sense that the channel impulse-response is known via measurement of S-parameters of the interface using a Vector-Network-Analyzer (VNA) or via computational electromagnetic modeling software such as Ansys HFSS. The manner in which channel degrades the transmitted signal stream is the unknown aspect and the designer's aim is to design a mechanism for counteracting this degradation. Thus, the whole challenge in high-speed link design is that we need to design a high signal-fidelity communication system that is fast, robust to losses incurred in the channel and on top of all this it needs to be low-power and must occupy the least possible area. The challenge in meeting all of the aforementioned requirements is that at high-speeds the channel suffers from various kinds of microwave losses due to impedance discontinuities between connectors, substrate loss, cross-talk effects, reflections and ringing, all which are difficult to predict and model [1].

Figure 2.3 [2] shows an example of the I/O link interface for a 10Gbps serial link across a backplane channel. Notice that a clean signal when transmitted across a backplane channel incurs tremendous amounts of loss from the channel at an operating speed of 10Gbps making the signal at the receiver virtually indistinguishable from noise and thus virtually garbage. In order to limit the degradation of signal quality during transmission and reception, the goal of mixed-signal designers is to design fast high-frequency clocks with minimum timing skew at the TX end and minimal sampling errors at the RX end.

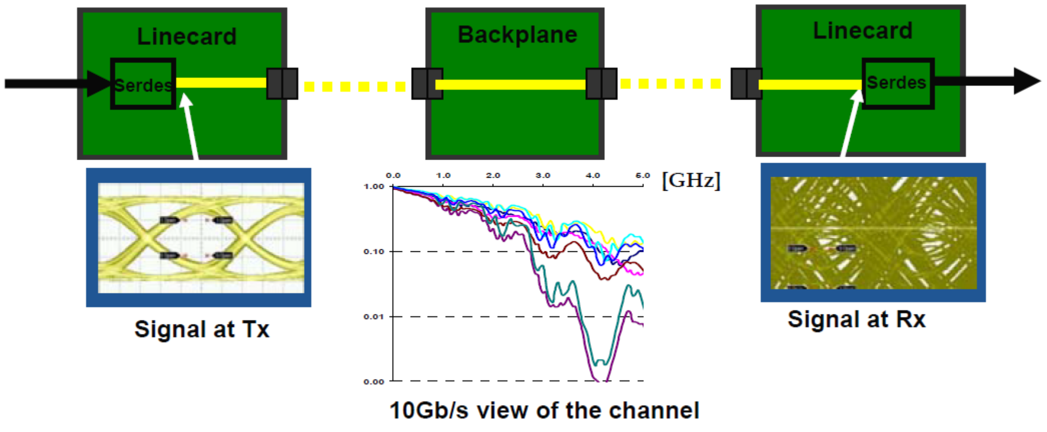


Figure 2.3: 10Gbps Backplane Serial Link Interface

2.2 Serial vs. Parallel Data Transmission

Historically, parallel links have been widely used in I/O systems that are connected to the CPU in computers via interfaces like PCI, PCI-X buses. However, as the data-rates have scaled into multi-gigabit ranges, the parallel link performance has not scaled accordingly with high signal fidelity. The tolerance level in timing skew between parallel signaling links has reached the practical limit achievable using traditional printed-circuit-boards (PCBs) that typically use FR-4 substrates. Additionally, as the supply voltage levels in modern CMOS process technologies have scaled down tremendously, the legacy parallel bus voltage levels have not scaled proportionally, making them incompatible with modern processes [6]. Thus, in order to mitigate this performance limitation and supply voltage scaling problems posed by conventional parallel-link design the industry has shifted to electrical point-to-point serial link interfaces.

Serial links occupy small area on chip and require very few I/O pins as compared to case of parallel links because the number of pins is not directly proportional to the number of data input/output signals. In serial communication links clock-skew is not a problem at the receiver since TX clock is typically not forwarded to the RX. In parallel links, on the other hand clock-skew is the major source of signal degradation at the RX side since the TX clock and data are transmitted separately. Furthermore, cross-talk effects are minimized in serial links due to the absence of multiple conducting channels in parallel that each have varying signals transmitted, whereas in parallel links this is a major problem due to the presence of capacitive/inductive coupling between multiple conducting parallel interconnect channels.

In the consumer electronics industry, serial links have found widespread acceptance in the form of USB (Universal Serial Bus) that connects peripheral electronic systems to computer, and SATA (Serial Advanced Technology Attachment) which connects the computer motherboard with mass storage devices (e.g. hard disk) and PCI-Express (Peripheral Component Interconnect) that is used to connect cards (sound, video or other) to the motherboard. Therefore serial communication has become the solution to higher and more efficient data transmission in order to meet the demands and trends of the higher capacity of communication technology [7].

2.3 SerDes Building Blocks

2.3.1 Serializer

The serializer circuit, as the name suggests, converts the input parallel-bus data into a serial bit-stream form. It is a completely digital block and it precedes the TX driver circuit. At a fundamental level, a serializer is essentially a Multiplexer circuit whose driving clock for the serialization process is the TX_CLK signal generated by the TX PLL.

2.3.2 Driver Amplifier

Driver amplifiers are found both at the TX as well as RX ends. The DA (Driver Amplifier) is used to amplify the input serial bit-stream before it is sent to the receiver through the channel. Another important task accomplished by the DA is that it provides impedance terminations that terminate the channel input/output with 50Ω impedance.

2.3.3 Phase-Locked Loop (PLL) Clock Generator

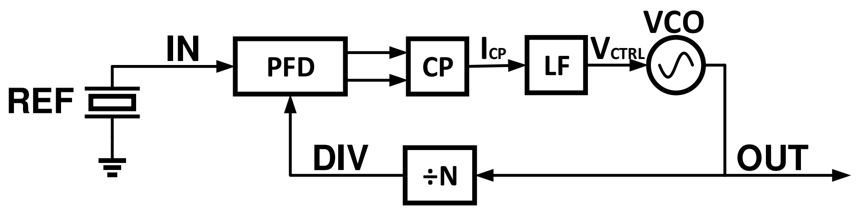


Figure 2.4: Typical PLL Based Clock-Generator Block Diagram

A PLL is a negative feedback system whose sole purpose is to use a reference input clock of frequency f_{REF} and generate a local clock on-chip at a desired frequency, f_{OUT}, such that the output clock is matched in phase to the input clock and $f_{OUT} = \alpha f_{REF}$, where α is a multiplying factor. PLLs are used in every modern day high-speed system whether it be wired or wireless because generating a high spectral purity clock at microwave frequencies/data-rates is practically not feasible yet. Piezo-electric crystals are used exclusively as the reference clocks for almost every on-chip interface system because they have the highest-spectral purity and can output truly periodic, jitter-free clock signals typically up to 200MHz. Due to the insatiable demand for robust, high-speed signaling, a mere crystal oscillator is not enough to meet the necessary requirements, so a PLL is essential. The most important task of a PLL is therefore to produce clock signals with minimal timing noise, i.e. the lowest possible jitter (in time domain) and phase-noise (in frequency-domain). At a block-level, a typical PLL clock-generator circuit (as shown in Figure 2.4) used to generate the high-speed on-chip clocks consists of a phase-frequency detector (PFD), charge-pump (CP), loop-filter (LF), voltage-controlled oscillator (VCO) and clock-divider (DIV). PFD tracks the phase and frequency difference between the reference signal and the divider output signal outputs digital pulse-width modulated (PWM) signals to the CP which essentially converts these digital pulses into an analog current signal. The LF then takes the CP output current signal, low-pass filters the high-frequency noise components and outputs a control voltage that drives the VCO. The VCO is the most-critical component within the PLL as it is the circuit-block that generates the final output clock that is used to drive the digital circuits of the link. Thus, a low phase-noise VCO is of paramount importance in the PLL as the VCO phase-noise is the dominant noise-form in the PLL. Finally, the divider is used in the feedback loop back to the PFD as the VCO output needs to be brought back down to the same frequency level as the reference clock so that the loop can dynamically drive all static-phase errors between reference clock and divider clock to zero such that $f_{OUT} = \alpha f_{REF}$, where α is the multiplying factor and the loop is "locked" to output a stable clock at the desired frequency of operation. The various intricacies involved in PLL design are covered in the remainder of the thesis.

2.3.4 Channel

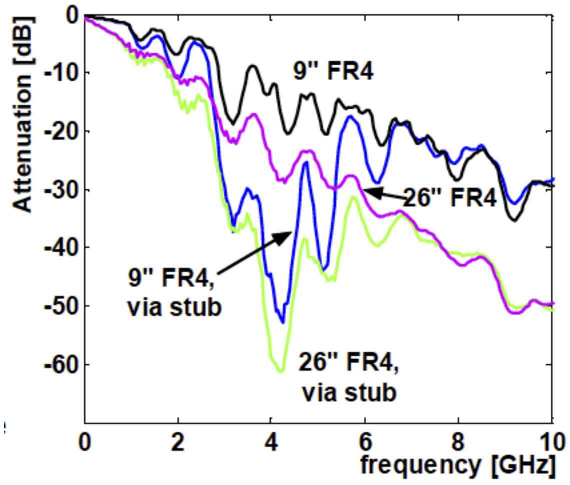


Figure 2.5: Typical Serial Link Channel Responses

Figure 2.5 [3] shows the attenuation levels a typical serial link channel incurs as a function of operating frequency. Figure 2.6 and 2.7 show the eye diagram outputs from a backplane channel interface at 1Gbps and 10Gbps data-rates respectively. Notice that the eye is fully open at 1Gbps but at 10Gbps the signal is almost indistinguishable from the noise at the receiver side due to the tremendous loss and distortion incurred along the channel. The HSSL designers need to be able to account for such losses when designing the blocks of the HSSL at both a system as well as circuit level. As stated earlier, the channel induced degradation is the primary limiting factor during the entire link-design process.

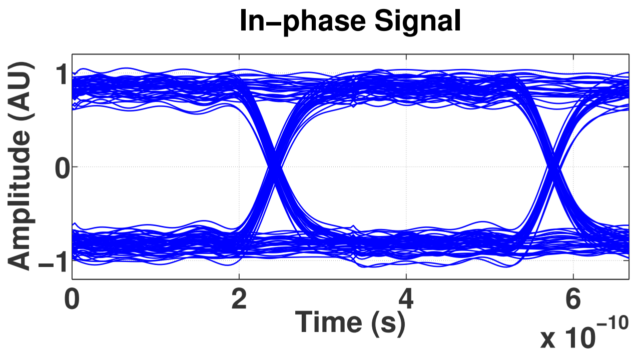


Figure 2.6: 1Gbps Backplane Link Eye Diagram

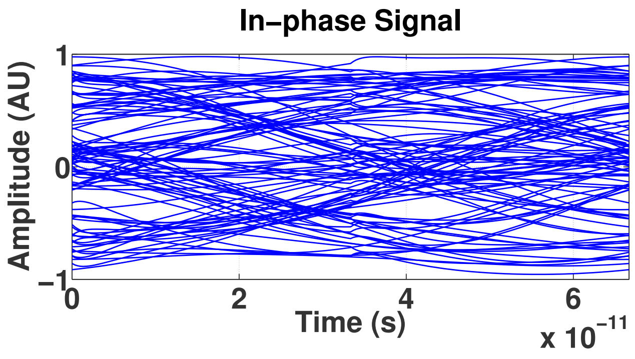


Figure 2.7: 10Gbps Backplane Link Eye Diagram

2.3.5 Equalization

Equalization is a method of combatting the detrimental effects of intersymbol interference (ISI) caused by the bandlimited channel. Equalizers are typically implemented as linear or non-linear adaptive filters. Equalization performed before the channel is referred to as pre-emphasis and basically involves passing the TX signal through a filter whose transfer function is the inverse of the channel transfer function. Conversely, equalization at the RX end is used to undo the distortion incurred in the received signal due to the channel loss and dispersion. Most RX equalization schemes are adaptive and are implemented using DSP techniques to cancel out the channel loss from the received data-bits.

2.3.6 Clock and Data Recovery (CDR)

A CDR as the name suggests is responsible for extracting the clock information of the transmitter from the received signal. In modern HSSLs, a clock-recovery mechanism is essential at the receiver end because TX clock information is typically embedded inside the incoming data pulse-stream at the receiver input. At heart, a CDR is essentially a modified PLL circuit wherein the phase-detector now has to sample the incoming data-stream and extract both data and phase information from it [3]. The PD of the CDR detects the transitions in the received data-stream and the VCO generates a periodic clock that drives the decision circuit within the PD to retime the distorted received data and then regenerates the system clock with lower skew and jitter. CDR design is a lot more intricate than PLL design because in the case of the CDR, loop bandwidth is often very small and governed by the jitter tolerance specifications of the system, meaning there is not much room for VCO phase-noise reduction. The most common implementation of a CDR (as shown in Figure 2.8) includes a regular PLL loop to track the exact-frequency of the TX clock, a phase-tracking loop with special phase-detector to produce the retimed data and a common VCO to output a low-jitter, phase-noise replica of the TX clock.

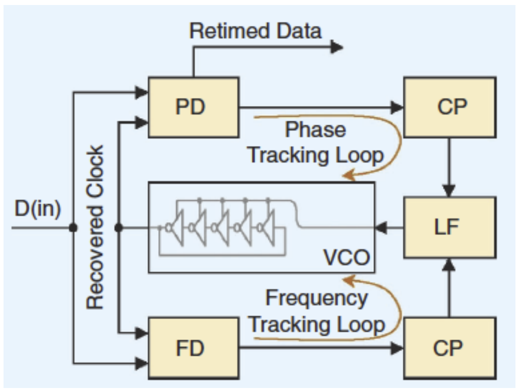


Figure 2.8: Typical Clock and Data Recovery Unit Implementation

2.3.7 Deserializer

The deserializer circuit, as the name suggests, converts the input serial bit-stream data back into its original parallel bus form. It is also a completely digital block and it succeeds the RX driver circuit. Basically, the deserializer is just a demultiplexer circuit that is driven by the clock that is recovered by the CDR.

2.3.8 Coding Schemes

Nonreturn-to-zero (NRZ) pulses are commonly used as the basis function for discrete data transmission. The response of the channel to the NRZ pulse is defined as the pulse response and is traditionally used to analyze and model the effects of a channel on data transmission and also in the design of equalizers in the case of channels with large attenuation at the frequency of interest. Apart from NRZ signaling, designers can also implement advanced modulation techniques for faster, robust signaling. For example, multilevel PAM like PAM-4 has much higher spectral efficiencies and can transmit two bits per symbol. This enables the transmission of an equivalent amount of data in half the channel bandwidth. In modern serial links along with the signaling schemes, some amount of encoding is also present in the data-stream. Most commonly used encoding schemes are 8B/10B and 16B/20B wherein 10bits or 20bits are sent but only 8bits/16bits are actual meaningful data bits, and this is powerful as it improves the BER. The only drawback of encoding is that it further adds complexity to the transceiver design as a encoder/decoder circuit needs to be designed and more bits need to be sent through the same bandwidth-limited channel.

2.3.9 HSSL Figures of Merit

HSSL performance is limited by the channel as well as the process technology used during circuit design. Since data-rates are scaling faster than the available channel bandwidths, the major constraint in realizing robust, high-speed interfaces is improving the maximum available clock frequency for on-chip synthesis. The channel bandwidth is the major constraint in overall system performance; thus, dealing with channel loss and designing clever equalization techniques are the biggest design challenges in HSSLs today. Robustness therefore is the most important metric of performance for link designers. The primary figures-of-merit (FOM) for HSSLs are bit-error-rate (BER), jitter, crosstalk analysis and timing/noise analysis [1].

BER in modern HSSLs is typically between 10^{-12} and 10^{-15} and it is the main metric used to signify the integrity of the received data-bits. A BER of 10^{-12} means that 1 bit will incur an error along the link when we transmit a total of 10^{12} bits. Measurement/Estimation of BER is one of the fundamental challenges faced by link designers because in order to accurately conclude that the link actually has a BER of the order 10^{-12}, one needs to simulate a random sequence of at least 10^{12} bits which even in current state-of-the art simulators is next to impossible. Therefore, most simulators use statistical means to collectively analyze the effects of deterministic noise sources such as Intersymbol Interference (ISI), supply-noise, timing-jitter as well as random noise sources like white-thermal noise and random jitter when estimating the system BER.

A common method to measure timing jitter is to use eye-diagrams. Eye diagrams are constructed by slicing the time-domain signal waveform into small sections and overlaying them on top of each other such that the resulting shape resembles an 'eye'. The horizontal axis of the eye diagram represents time and is typically one or two symbols wide, and the vertical axis represents the amplitude of the signal. Ideally, we want the eye to be as "open" as possible, since a larger eye opening signifies that there is a large enough margin to meet any voltage and timing requirements needed by the system. Quantitatively speaking, the minimum height and width of the data at the receiver are key metrics for evaluating link performance. As link designers, we want the receiver eye to be wide enough to provide adequate time to satisfy the setup and hold requirement of the flip-flops used, and have sufficient height to ensure that the voltage levels meet v_{il} and v_{ih} requirements of the system in the presence of multiple noise sources. Figure 2.9 [6] shows an example of what sampling an eye with and without jitter means.

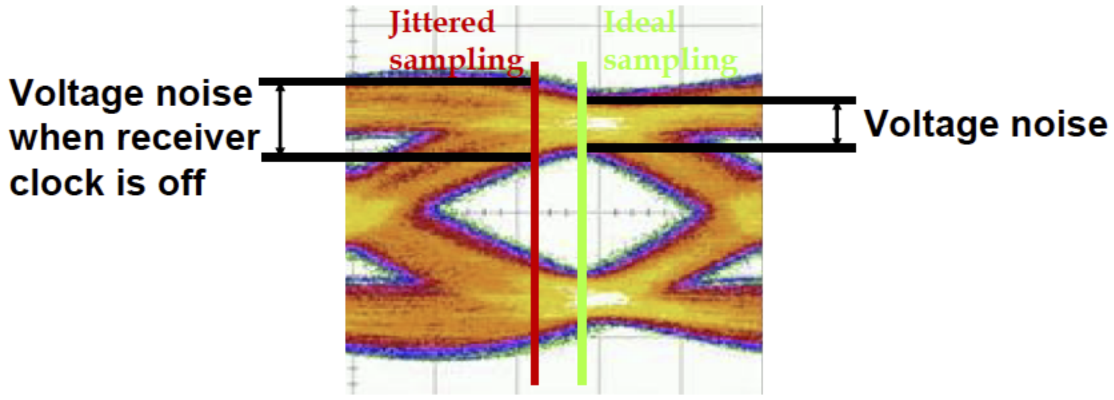


Figure 2.9: Eye Diagram Terminology and FOMs

The most prominent source of signal degradation in HSSLs over a bandlimited channel is Intersymbol Interference (ISI). ISI results when a sequence of signals are passed through a channel whose bandwidth is insufficient to allow passage of all the spectral components of the signal. It is a form of signal distortion caused due to reflections, channel resonances, and channel loss/dispersion. Simplest way to understand ISI is to view it as interference between symbols wherein current bit/symbol causes distortion in subsequent/preceding bit/symbol. ISI degrades as data-rates increase and channel bandwidth remains the same, and the only way to combat it is through clever equalization techniques on either the TX, RX ends or both. Another form of interference which is slowly becoming a major hindrance for link designers as data-rates scale is crosstalk (XT), which is a phenomenon occurring due to presence of capacitive as well as inductive coupling between neighboring signal lines in a transceiver. Typically, most of the XT effects are felt at the connector/package levels of a channel where the signal spacings are small compared to the distance between shields [6]. Near-End XT (NEXT) and Far-End XT (FEXT) are the two classes of XT wherein NEXT is defined as the XT due to energy dissipated from coupling between transmitted signal and the reflected signal on the same chip, while FEXT is defined as the XT due to energy dissipated from coupling between transmitted signals of two different chips. NEXT is by far the most deleterious type of XT and the most commonly observed kind because the TX energy levels are typically very high compared to the RX signal levels so the received signal can really be submerged inside it if proper care is not taken.

Finally, the last major metric in calculating the timing margin of a HSSL is the jitter. Characterization of deterministic as well as random timing jitter in a clock output is very important to a link designer. Essentially, jitter is the time-domain variation in the clock-signal as shown in Figure 2.10 [10]. A commonly used method for jitter calculation is to close either side of the eye horizontally by the amount of peak clock jitter. While this method can be helpful in evaluating the effects of jitter at the receiver end, we will show in this paper that this is an overly optimistic approximation of noise margin degradation for transmitter jitter. Due to the need for integration of clock generators such as PLLs in large digital chips, clock jitter is dominated by power-supply and substrate noise, both of which do not scale with technology. Therefore, as data rates increase, bit-periods become shorter and the performance of multi-gigabit links will be limited by the clock jitter, thereby initiating the importance of accurately analyzing the effects of clock jitter on high-speed serial links. Figure 2.11 [5] provides a summary of common jitter profiles in a typical serial link.

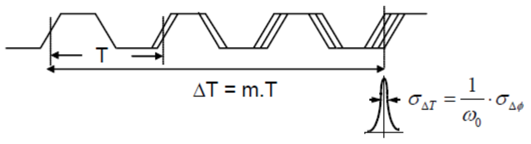

T
ΔT = m.T
$\sigma_{\Delta T} = \frac{1}{\omega_0} \cdot \sigma_{\Delta\phi}$

Figure 2.10: Timing Jitter Example

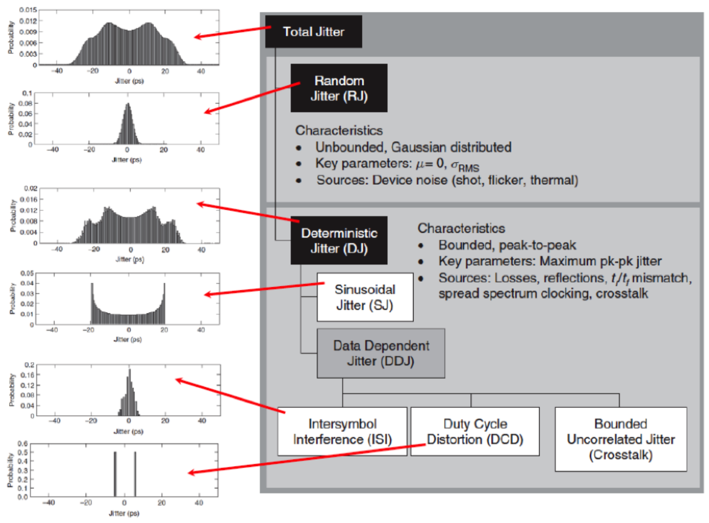

Total Jitter
Random Jitter (RJ)
Characteristics
Unbounded, Gaussian distributed
Key parameters: $\mu = 0$, σ_{RMS}
Sources: Device noise (shot, flicker, thermal)
Deterministic Jitter (DJ)
Characteristics
Bounded, peak-to-peak
Key parameters: Maximum pk-pk jitter
Sources: Losses, reflections, t_r/t_f mismatch, spread spectrum clocking, crosstalk
Sinusoidal Jitter (SJ)
Data Dependent Jitter (DDJ)
Intersymbol Interference (ISI)
Duty Cycle Distortion (DCD)
Bounded Uncorrelated Jitter (Crosstalk)
Probability
Jitter (ps)

Figure 2.11: Summary of Common Jitter Profiles

CHAPTER 3

PLL THEORY AND BACKGROUND

3.1 PLL Applications

Phase-Locked Loops (PLLs) are one of the most fundamental and ubiquitous circuits found in any communications (wireless, wireline) and high-speed digital systems. Monolithic CMOS implementation of PLLs has gained lots of popularity over the last few decades due to an insatiable demand for high performance digital systems. Most common uses of a PLL are in the form of frequency synthesizers and carrier/clock recovery circuits both in the RF domain as well as the high-speed digital domain.

3.2 Basic PLL Building Blocks

3.2.1 Phase/Phase-Frequency Detector (PD or PFD)

In a PLL, unlike many other feedback systems, the variable of interest changes dimension around the loop: it is converted from phase to voltage (or current) by the phase detector, processed by the LPF as such, and converted back to phase by the VCO. In the lock condition, the input and output frequencies are exactly equal, regardless of the magnitude of the loop gain (although the phase error may not be zero). This is an extremely important property because many applications are intolerant or even small (systematic) differences between the input and output frequencies [15].

The PD compares the phase of the output signal with the phase of the reference signal and develops an output signal that is approximately proportional to the phase-error Φ_e. The output voltage of the PD is proportional to the phase-difference between the reference signal and the output signal. The

PD serves as an "error'-amplifier" in the feedback loop, thereby minimizing the phase-difference, $\Delta\phi$, between the reference signal, $V_{ref}(t)$ and the oscillator output signal, V_{out}. The loop is considered to be "locked" if $\Delta\phi$ is constant with time, a result of which is that the input and output frequencies are equal. In locked condition, all the signals in the loop have reached steady state and the PLL operates as follows. The phase detector produces an output whose DC value is proportional to $\Delta\phi$. The low-pass filter suppresses high-frequency components in the PD output, allowing the DC value to control the VCO frequency. The VCO then oscillates at a frequency equal to the input frequency and with a phase-difference equal to $\Delta\phi$. Thus, the LPF filter generates the proper control voltage for the VCO. The VCO phase can be seen to be an initial condition of the system, as it is independent of the initial conditions in the LPF. Whenever two frequencies become equal at a point in time and $\Delta\phi$ has not established the required control voltage for the VCO, the loop will continue the transient, temporarily making the frequencies unequal again. In other words, both "frequency-acquisition" and "phase-acquisition" must be completed. This behavior is to be expected because for lock to occur again, all the initial conditions of the system, including the VCO output phase, must be updated [15].

The biggest pitfall of using just a PD is that it does not capture any step changes in frequency; thus, in order to be able to track both phase and frequency we need to use a phase-frequency detector (PFD). The purpose of a PFD is to compare the reference clock signal and the VCO output clock after division in both phase and frequency. These frequencies are generally denoted by F_{REF} and F_{VCO} respectively. The basic structure can be divided into logic control part and a charge pump. The charge pump is a current source in series with a current sink and the output node is like a switch that resides in between the source and sink. The logic part consists of two D-Flip-Flops (DFFs) and the outputs of these DFFs control the switch of the charge pump. Conceptually the PFD can be viewed as a state machine with three states. The initial state is 0 and both DFFs will be reset if VCO and reference signal are both high. In state -1 only current sink is turned on and sinks charge out of the load, thereby decreasing the output voltage; in state 0 current source and current sink are turned off so no charge is injected or extracted out of the output node, thereby keeping the output voltage unchanged. In state 1 only current source is turned on so charge can be injected into the output node, thereby increasing the output voltage. The state transitions are controlled by the edges of VCOs output and reference signal; thus it is clear that the PFD is a purely digital circuit.

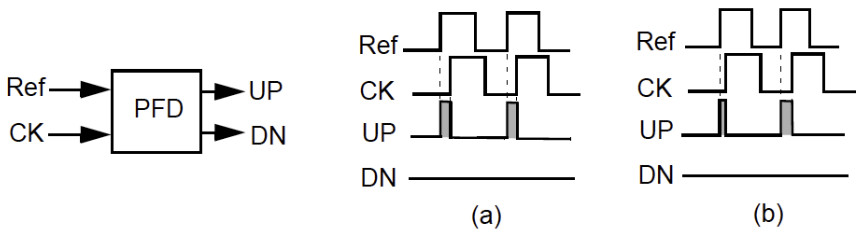


Figure 3.1: PFD Operation

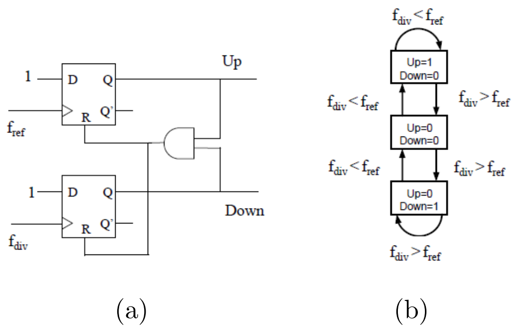


Figure 3.2: PFD Functionality

Figures 3.1 and 3.2 show the block diagram of a PFD and demonstrate its functionality. Essentially, when the reference clock is faster than the divider clock, UP signal is High, DN signal is Low and vice versa. Note that when both the reference and divider clocks are synchronized both UP and DN signals are set to be High. The phase frequency detector (PFD) is a circuit that linearly translates the phase difference into voltage signals. The ideal average input/output relationship should be:

$$V_e = K_{PD} \times \phi_e \tag{3.1}$$

$$where\ |\phi_e| < 2\pi$$

K_{PFD} is defined as the PFD gain.

3.2.2 Charge-Pump (CP)

The charge pump is the device that translates the digital voltage signals generated from PFD into a current signal. Since the voltage controlled oscillator needs a stable voltage to control the oscillating frequency, a charge storage capacitor is needed. In order dump enough charge into the capacitor, a charge pump is needed here. Together with the PFD the s-domain transform becomes the following:

$$K_{PFD} = \frac{i_{Charge\,Pump}}{2\pi} \tag{3.2}$$

3.2.3 Loop-Filter (LF)

PLLs act as high-pass filters so the purpose of the loop filter is to filter out the high-frequency components from the output of the PFD. Typically, loop-filters are just simple passive RC networks whose main objective is to filter out the high-frequency noise data from the PFD output.

3.2.4 Voltage Controlled Oscillator (VCO)

VCOs are the most important and complex component of the overall PLL design. The essential idea behind a VCO design is to generate a clock signal based on the Barkhausen criteria for oscillation which states that the magnitude of the VCO transfer function at the oscillation-frequency is 1 while the phase is -180 degrees. Two most popular VCO topologies whose sample architectures are ring-based and LC-tank based. Due to the superior noise performance we chose to design a LC-Tank based VCO. VCO is the device that generates the target clock. Ideally, its output frequency should be linearly related to the input control voltage. The Laplace transform function of the VCO is derived as follows:

$$\omega_{out}(t) = K_{VCO} v_{ctrl}(t) \tag{3.3}$$

$$\mathcal{L}[\omega_{out}(t)] = \omega_{out}(s) = K_{VCO} v_{ctrl}(s) \tag{3.4}$$

$$\phi_{out}(t) = \int_0^t \omega_{out}(\tau)d\tau = \int_0^t K_{VCO}v_{ctrl}(\tau)d\tau \tag{3.5}$$

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

Thus, the Laplace transform function for the VCO is:

$$H_{VCO}(s) = \frac{\phi_{out}(s)}{v_{ctrl}(s)} = \frac{K_{VCO}}{s} \tag{3.7}$$

The K_{VCO} is defined as the VCO gain.

3.2.5 Divider

A frequency divider is needed to produce a clock signal that runs many times faster than the reference clock. The PFD input clock and reference clock have to be synchronized for PLL to be in locked condition. In order to perform this task we use a fractional-N divider circuit, which divides the VCO clock by the highest power of 2 factor to synchronize reference clock signal and the divider output clock.

3.2.6 Analysis of a PLL in Locked-State

The open-loop transfer function of the PLL is equal to $H_O = K_{PD}G_{LPF}(s)\frac{K_{VCO}}{s}$, yielding the closed-loop transfer function $H(s) = \frac{\Phi_{out}(s)}{\Phi_{in}(s)} = \frac{K_{PD}K_{VCO}G_{LPF}(s)}{s+K_{PD}K_{VCO}G_{LPF}(s)}$. In its simplest form, a low pass filter is implemented to have the transfer function $G_{LPF}(s) = \frac{1}{1+\frac{s}{\omega_{LPF}}}$,where $\omega_{LPF} = \frac{1}{RC}$. Thus, for a PLL containing a first-order LPF the closed-loop response is represented as $H(s) = \frac{K_{PD}K_{VCO}}{\frac{s}{\omega_{LPF}}+s+K_{PD}K_{VCO}}$ indicating that the system is of second-order, where one pole is contributed by the VCO and the other by the LPF. Here, $K = K_{PD}K_{VCO}$ is called the loop-gain and expressed in rad/s. In order to understand the dynamic behavior of the PLL, the denominator of the second-order closed-loop response is converted to a form commonly used in control theory: $s^2 + 2\zeta\omega_n s + \omega_n^2$, where ζ is the damping factor and ω_n is the natural frequency of the system. Therefore, the closed-loop response can now be expressed as $H(s) = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$, where $\omega_n = \sqrt{\omega_{LPF}K}$ and $\zeta = \frac{1}{2}\sqrt{\frac{\omega_{LPF}}{K}}$. Note that ω_n is the geometric mean of the -3dB bandwidth of the LPF and the loop-gain, and thus an indicator of the gain-bandwidth product of the loop. The damping factor is inversely proportional to the loop gain. Typically, in a well designed second order system, ζ is usually greater than 0.5 and preferably equal to $\frac{\sqrt{2}}{2}$ so as to provide an optimally flat response. Thus, K and ω_{LPF} cannot be chosen independently; for example if $\zeta = \frac{\sqrt{2}}{2}$, then K $= \frac{\omega_{LPF}}{2}$. If $s \to 0$, we note that $H(s) \to 1$; i.e. a static phase shift at the input is transferred to the output unchanged. We can examine the "phase error transfer function" defined as $H_e(s) = 1 - H(s) = \frac{\Phi_e(s)}{\Phi_{in}(s)} = \frac{s^2+2\zeta\omega_n s}{s^2+2\zeta\omega_n s+\omega_n^2}$ which drops to 0 as $s \to 0$ [15].

3.2.7 PLL Characteristics and Figures of Merit

The most important metrics of a PLL are Order, Type, Hold-In range, Lock-in range, and Pull-in range. The order of a PLL is determined by the number of poles in the loop while the type is determined by the number of integrators. The VCO always has a pre-existing pole because it generates frequency from phase via an integration; thus every PLL is at least of order 1 and type 1. As the loop-filter poles increases and the PLL order and type increases as well and higher the type, the better the PLL is at tracking both frequency and phase. For instance, a type 2 PLL is capable of tracking both a step change in phase as well as frequency with zero steady-state phase error while a type 1 PLL can only track a step change in phase.

Hold-In range of a PLL is a measure of the DC loop gain and the range of frequencies under which the PLL can maintain a lock. Lock-In range is a measure of the range of frequencies under which a PLL can acquire lock without slipping any clock-cycles. Finally, the Pull-In range is the measure of the range of frequencies for which the PLL can acquire lock by missing a few clock cycles. It is important to note that the hold-in range is the largest of the three and lock-in range is the smallest of the three metrics.

CHAPTER 4

PLLs IN CLOCKING CIRCUITS

4.1 Charge-Pump (CP) PLLs Overview

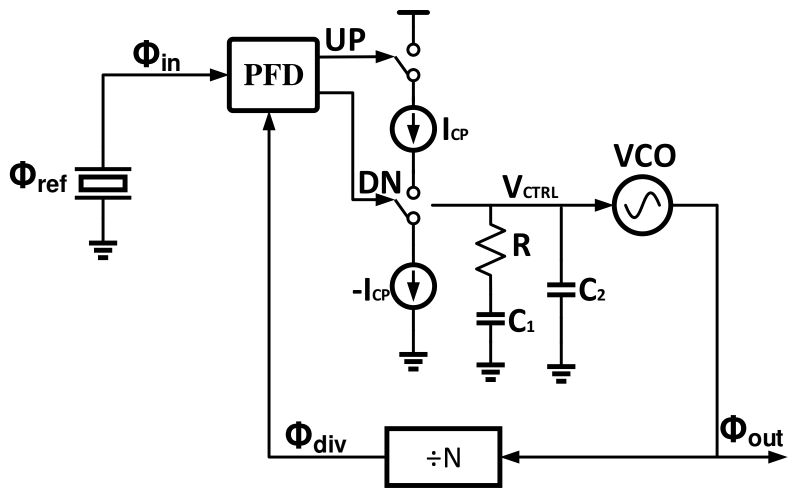


Figure 4.1: Charge-Pump PLL Block Diagram

Figure 4.1 shows the basic building blocks of a CPLL. Charge-Pump PLLs offer many advantages over the classical voltage phase-detector PLL including an infinite pull-in range and zero steady-state phase error. CPLLs also allow one to use a passive filter and still have many of the benefits of using an active filter with the voltage phase detector. The exception to this case is when the VCO tuning voltage needs to be higher than the PLL can supply; in this case, an active filter is necessary [15].

Phase-frequency detectors with charge-pump combination offer several advantages over the voltage charge pump and have all but replaced it. The PFD and CP blocks are universally present in every PLL based synthesizer chip. Using this approach completely bypasses issues of steady-state phase error and hold-in range [10].

4.2 CPLL Linear Model and Analysis

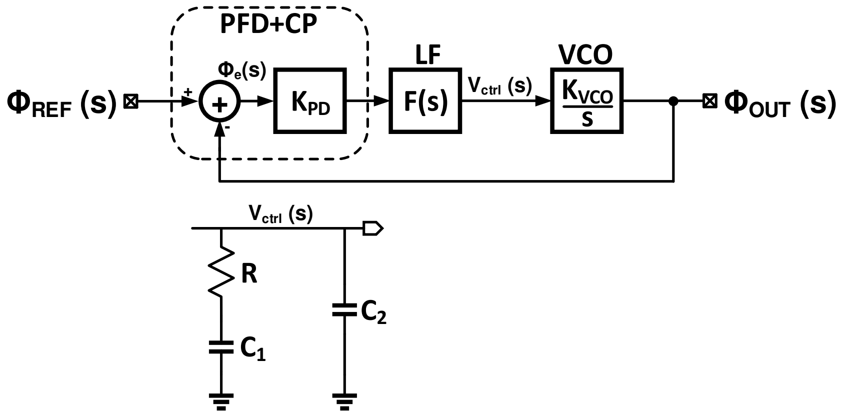


Figure 4.2: Linear s-Domain Model for Charge-Pump PLL

Figure 4.2 shows the linear s-domain model for CPLLs. From the previous section we can now define the open loop transfer function as follows:

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

$$= K_{PD} \cdot K_{VCO} \cdot \frac{s + \frac{1}{RC_1}}{C_2 s^2 \left(s + \frac{C_1+C_2}{RC_1C_2}\right)} \tag{4.1b}$$

From the open loop gain we notice that

$$\omega_z = \frac{1}{RC_1};\ \omega_{p1} = \omega_{p2} = 0; \omega_{p3} = \frac{C_1 + C_2}{RC_1C_2} \tag{4.2}$$

The phase margin will be:

$$\phi_M = \arctan\left(\frac{\omega_{ugb}}{\omega_z}\right) - \arctan\left(\frac{\omega_{ugb}}{\omega_{p3}}\right) \tag{4.3}$$

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

In order to achieve maximum phase margin, the value of C_1 and C_2 have to be chosen carefully. To calculate the expression of ϕ_{M_max} we take the first order derivative of Eq. 4.3 with respect to ω_{ugb} and equate the result to zero, such that:

$$\omega_{ugb} = \omega_z\sqrt{\frac{C_1}{C_2} + 1} \tag{4.4}$$

Subsequently,

$$\phi_{M\_max} = \arctan(\sqrt{\frac{C_1}{C_2}+1}) - \arctan(\frac{1}{\sqrt{\frac{C_1}{C_2}+1}}) \tag{4.5}$$

The design procedure of the loop filter is as follows:

1. Choose desired bandwidth ω_{ugb}, phase margin ϕ_M and resistor R according to specification. Then calculate the K_c from Eq. 4.6:

$$K_c = \frac{C_1}{C_2} = 2(\tan^2(\phi_M) + \tan(\phi_M\sqrt{\tan^2(\phi_M)+1})) \tag{4.6}$$

2. From Eq. 4.4 we have:

$$\omega_z = \frac{\omega_{ubg}}{\sqrt{\frac{C_1}{C_2}+1}} \tag{4.7}$$

$$C_1 = \frac{1}{\omega_z R}; C_2 = \frac{C_1}{K_c}; \tag{4.8}$$

3. From aforementioned equations, we can determine the value for I_{CP}:

$$I_{CP} = \frac{2\pi C_2}{K_{VCO}} \cdot \omega_{ugb}^2 \cdot \sqrt{\frac{\omega_{p3}^2 + \omega_{ugb}^2}{\omega_z^2 + \omega_{ugb}^2}} \tag{4.9}$$

It is vital to analytically confirm that the PLL will indeed lock when there is a frequency step applied at the input. Without loss of generality assume there is input frequency step $\omega_{in} = \frac{\Delta\omega}{s}$, then $\Phi_{in}(s) = \frac{\Delta\omega}{s^2}$. First, obtain the closed loop transfer function:

$$H_{PLL}(s) = \frac{LG(s)}{1+LG(s)} \tag{4.10}$$

Lastly, define steady state error transfer function:

$$\frac{\Phi_{error}(s)}{\Phi_{in}(s)} = H_e(s) = 1 - H_{PLL}(s) = \frac{1}{1+LG(s)} \tag{4.11}$$

Applying the final value theorem, we get the steady state error to be:

$$\begin{aligned}
\Phi_{ss\_error}^{F_{step}} &= \lim_{s\to 0} s \cdot H_e(s) \cdot \Phi_{in}(s) && (4.12a)\\
&= \lim_{s\to 0} s \cdot \frac{1}{1+LG(s)} \cdot \frac{\Delta\omega}{s^2} && (4.12b)\\
&= \lim_{s\to 0} \frac{[RC_1C_2s^2 + (C_1+C_2)s]\Delta\omega}{RC_1C_2s^3 + (C_1+C_2)s^2 + K_{VCO}K_{PD}s + 1} && (4.12c)\\
&= \frac{0}{1} && (4.12d)\\
&= 0 && (4.12e)
\end{aligned}$$

Eq. 4.12(a) to 4.12(e) indicate that the PLL we have designed can eliminate any steady state phase error and relock when a frequency step is applied at the input [8].

4.3 CPLL Noise-Analysis

The following equations describe the noise-transfer functions of the CPLL and are used in determining the optimal PLL BW.

$$NTF_{IN}(s) = \frac{\Phi_{OUT}(s)}{\Phi_{IN}(s)} = \frac{N \cdot LG(s)}{1+LG(s)} \tag{4.13}$$

$$NTF_{DIV}(s) = NTF_{IN}(s) \tag{4.14}$$

$$NTF_{CP}(s) = \frac{\Phi_{OUT}(s)}{i_{CP}(s)} = \frac{2\pi}{I_{CP}} \cdot NTF_{IN}(s) \tag{4.15}$$

$$NTF_R(s) = \frac{\Phi_{OUT}(s)}{v_R(s)} = \frac{\frac{K_{VCO}}{s}}{1+LG(s)} \tag{4.16}$$

$$S_{\Phi_{OUT}}^{\Phi_{IN}} = S_{\Phi_{IN}}|NTF_{IN}(s)|^2 \tag{4.17}$$

$$S_{\Phi_{OUT}}^{i_{CP}} = S_{i_{CP}}|NTF_{CP}(s)|^2 \tag{4.18}$$

$$S_{\Phi_{OUT}}^{v_R} = S_{v_R}|NTF_R(s)|^2 \tag{4.19}$$

$$S_{\Phi_{OUT}}^{\Phi_{VCO}} = S_{\Phi_{VCO}}|NTF_{VCO}(s)|^2 \tag{4.20}$$

$$G_{LPF}(s) = \frac{1}{1+\frac{s}{\omega_{LPF}}} \tag{4.21}$$

$$H(s) = \frac{K_{PD}K_{VCO}}{\frac{s}{\omega_{LPF}} + s + K_{PD}K_{VCO}} \tag{4.22}$$

$$s^2 + 2\zeta\omega_n s + \omega_n^2 \tag{4.23}$$

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{4.24}$$

$$\omega_n = \sqrt{\omega_{LPF}K}, \zeta = \frac{1}{2}\sqrt{\frac{\omega_{LPF}}{K}} \tag{4.25}$$

$$\zeta = \frac{\sqrt{2}}{2}, \implies K = \frac{\omega_{LPF}}{2} \tag{4.26}$$

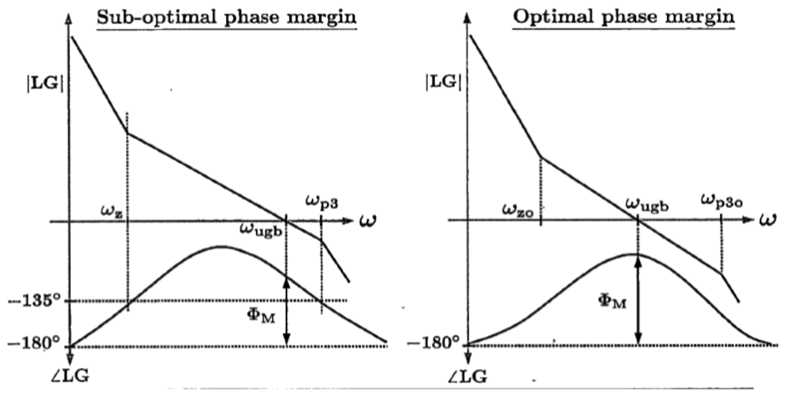


Figure 4.3: Loop-Gain and Phase-Margin Response

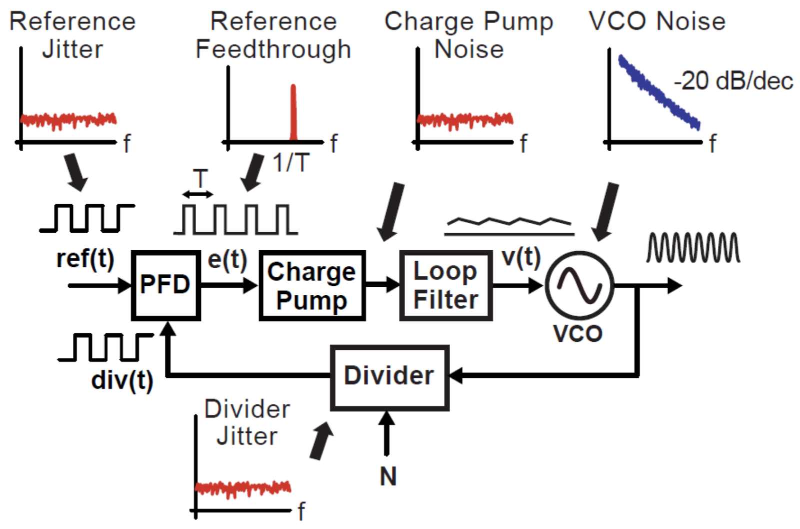


Figure 4.4: CPLL Output Noise Model

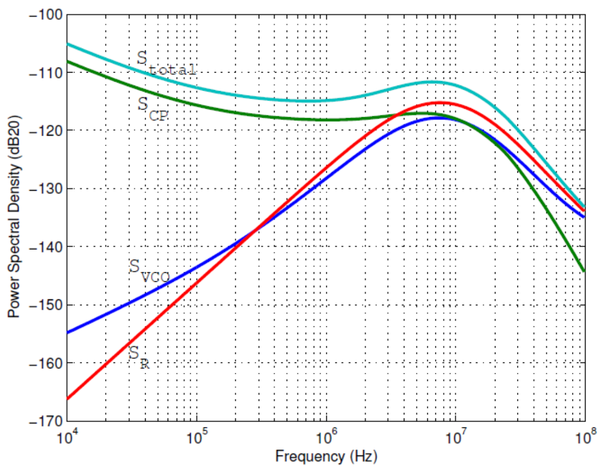


Figure 4.5: CPLL Output Noise Simulation

Figure 4.3 [8] shows the typical loop-gain and phase-margin plot for a CPLL. Recall that phase-margin is the difference in phase between $-180\,^{\circ}\mathrm{C}$ and the phase value corresponding to ω_{ugb}. Figure 4.4 shows the typical noise-profile for each component in a CPLL based Integer-N synthesizer [12]. Figure 4.5 shows the noise-transfer function characterization for the PLL using the above equations implemented in MATLAB. The beauty of this analysis is that it accurately predicts what the noise-profile for the PLL will look like so that the designer can determine the BW specifications for the PLL from which the rest of the Loop-Filter and VCO specifications can be determined.

CHAPTER 5

PLL BASED CLOCK GENERATOR

5.1 PFD

Figure 5.1 shows the NAND PFD implementation used in the design of the PLL used in this thesis. In Figure 5.2 the transistor-level implementation for each of the circuits shown in Figure 5.1 are displayed with the appropriate sizing.

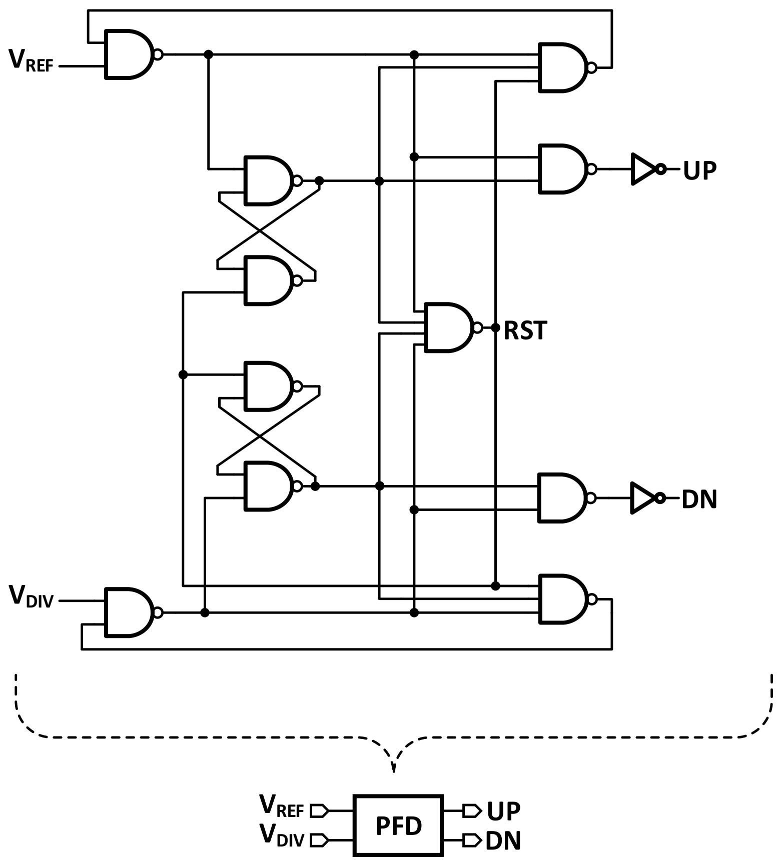


Figure 5.1: NAND PFD Implementation

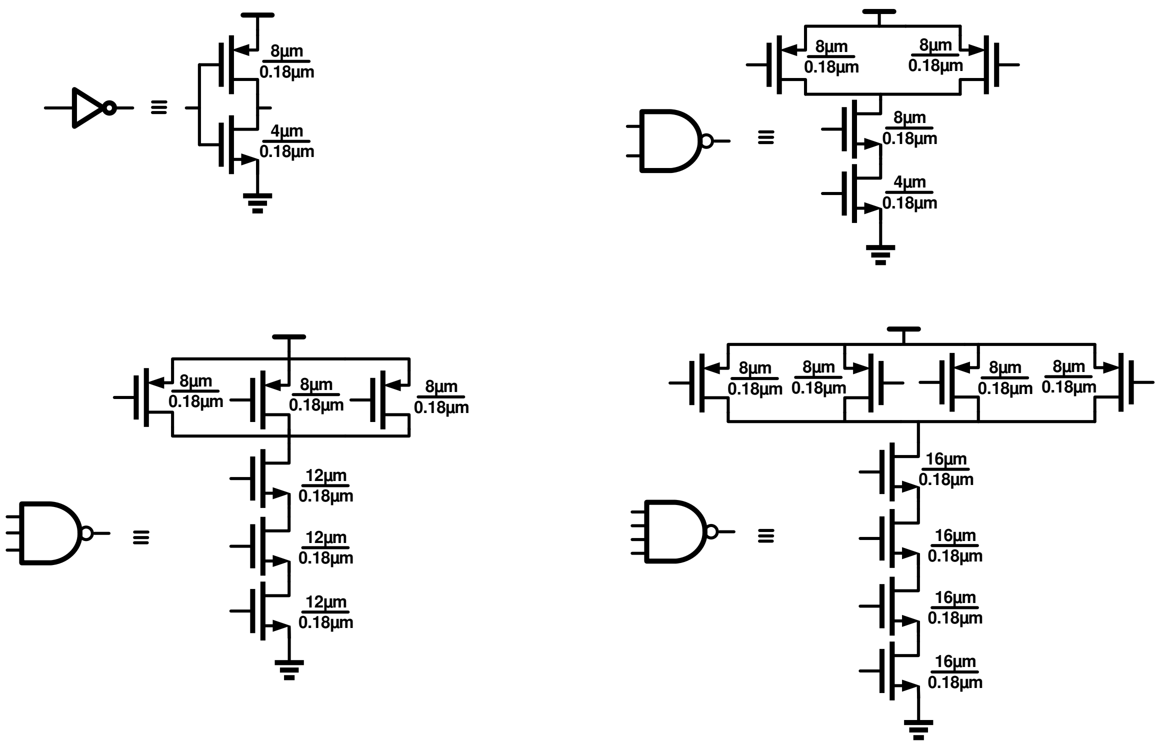


Figure 5.2: PFD Transistor Level Circuit Blocks

One of the major challenges during the design of an efficient PFD circuit is the “dead-zone” problem. “Dead-zone” is refers to the region wherein there is no output for inputs. It is equal to the sum of the on-times of the pull-up/pull-down switches in the charge-pump and is typically a problem because the presence of dead-zone causes the PLL to operate in “open-loop” when the phase-error is zero. One method to overcome the dead-zone issue is to ensure that the PFD generates equal UP/DN pulses whose width is larger than the switch-on time of the CP switches. The NAND-PFD implementation is one example of a PFD circuit where the dead-zone issue is avoided. The D-Flip-Flops are designed using cross-coupled NAND-latches and even though this uses up a lot of on-chip area and burns a lot of power in the PFD circuit, the large delay in UP/DN feedback paths allows the pulse-widths to be just larger than the switch-on time of the PMOS/NMOS transistors that act as the pull-up/pull-down switches in the CP. The maximum operating frequency of a PFD circuit is determined by the reset path delay such that $F_{max} < \frac{1}{2T_{RST}}$. In the case of a NAND PFD circuit, $T_{RST} = 2T_{NAND2} + T_{NAND4}$, where we intentionally design the NAND4 circuit to have a high delay to minimize the reset period.

5.2 CPs

Figure 5.3 shows the CP implementation used in the design of the PLL used in this thesis. The transistor-level implementation is also displayed with the appropriate sizing.

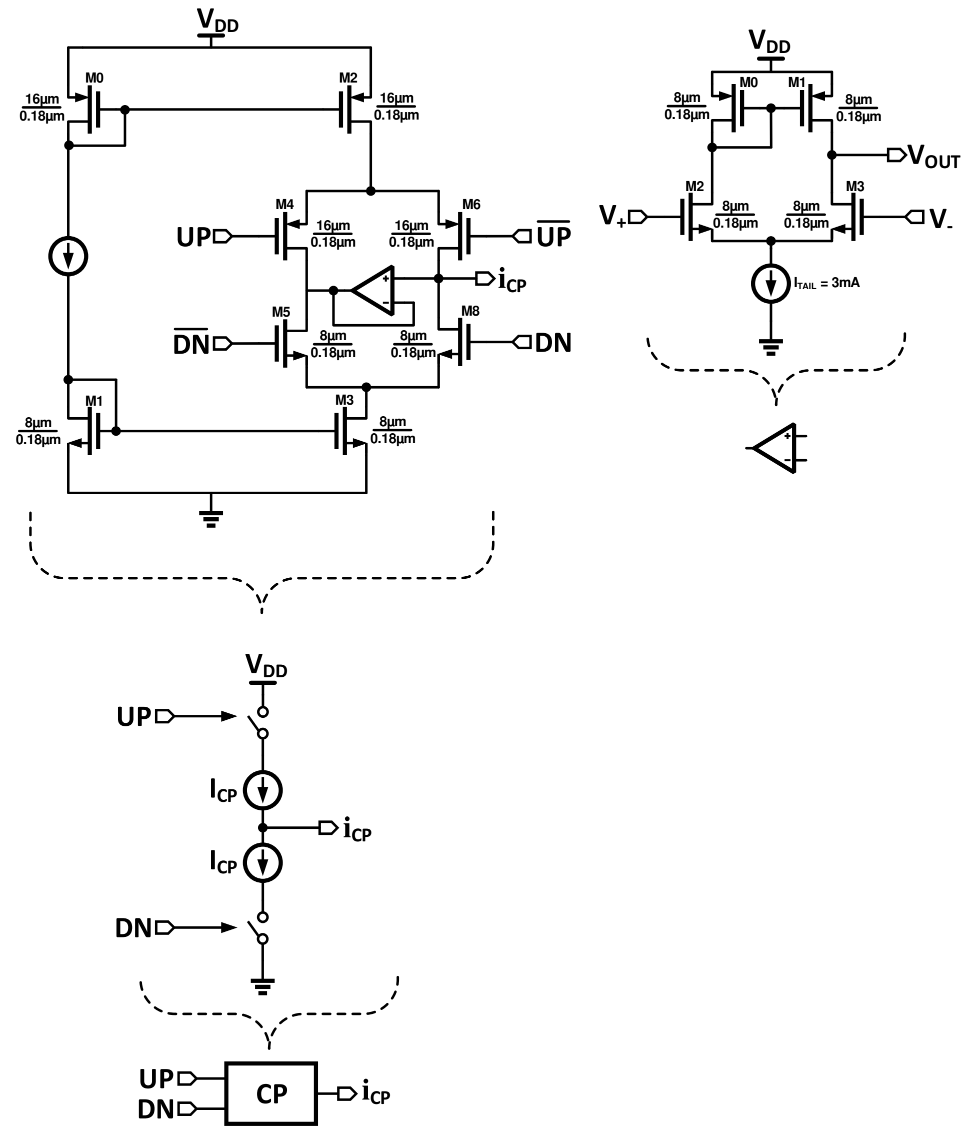


Figure 5.3: Bootstrapped Charge-Pump Implementation

The charge-pump circuit needs to be carefully designed in because it is the main contributor to low-frequency PLL noise. The current mismatch in the charge-pump leads to static-phase offset and causes ripples in the control voltage, thereby creating a 'jittery' VCO output clock. Thus, it is important to design a CP circuit that has equivalent pull-up and pull down currents and equal on-time for the PMOS/NMOS switches. Though several CP architectures exist, a ‘Bootstrapped’ CP design is used in the clock-generating PLL studied in this thesis. The advantage of the Bootstrapped architecture is that it allows differential current steering, it can operate with low-swing UP, DN signals. It is thus very prominent in PLLs that use high-speed reference clock signals. The term ‘bootstrapped’ are appropriate because the voltage following op-amp between the pull-up and pull-down current networks ensures that an equal voltage level is maintained on either ends such that the pull-up current is equal to the pull-down current.

5.3 LF

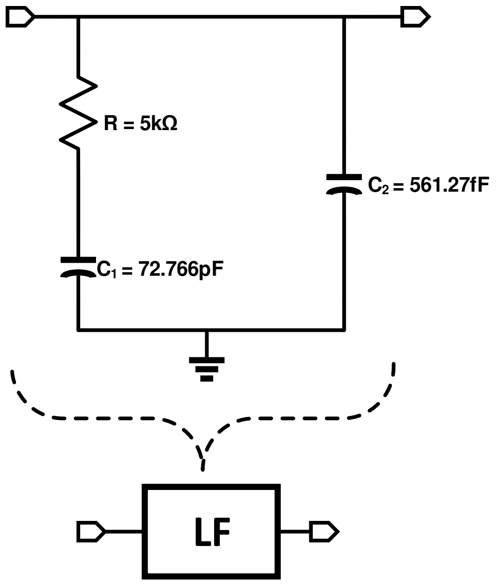


Figure 5.4: Loop-Filter Implementation

Loop-Filter is designed using the design-procedure described in Chapter 4 in the CPLL design procedure algorithm. The algorithm was implemented in MATLAB to choose the values shown above in Figure 5.4.

5.4 VCOs

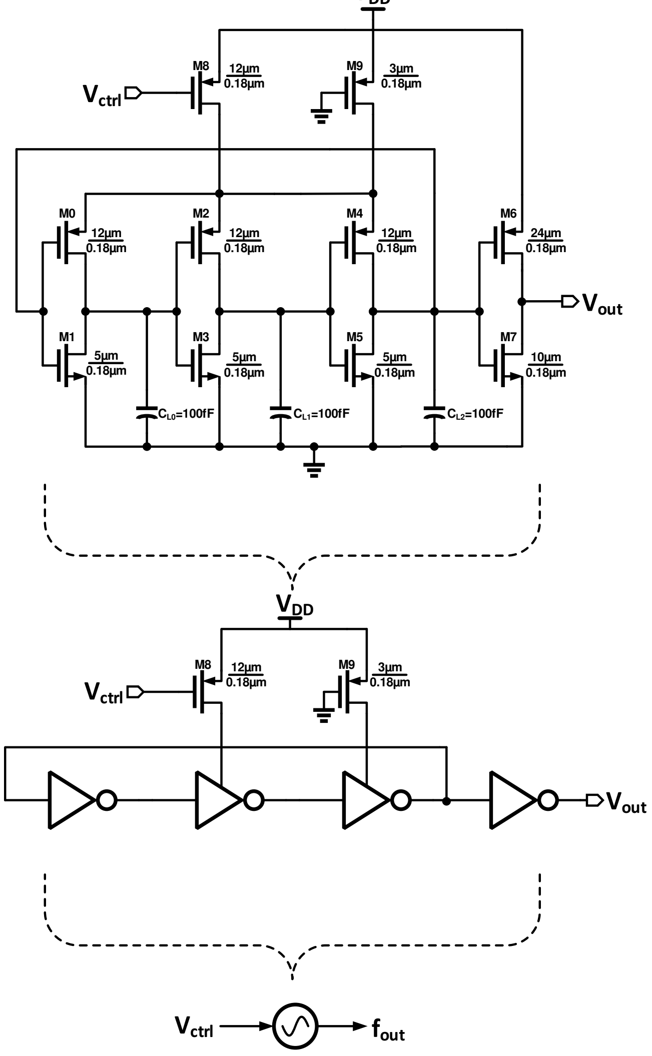


Figure 5.5: Single-Ended 3-Stage Ring Oscillator

A 3-Stage ring-oscillator is implemented with a driver inverter (as shown in Figure 5.5 with full transistor-level sizing) whose size is fixed such that the input capacitance seen by the divider remains constant while the VCO frequency is changing. M8 and M9 act as the pull-up resistors, i.e. they are PMOS transistors that are biased to be in triode/resistive region. In order to ensure the oscillation starts, the gate of M9 is driven to ground while gate of M8 is driven by the control voltage which alters the phase-delay between the ring to vary to the oscillation frequency.

5.5 Divider

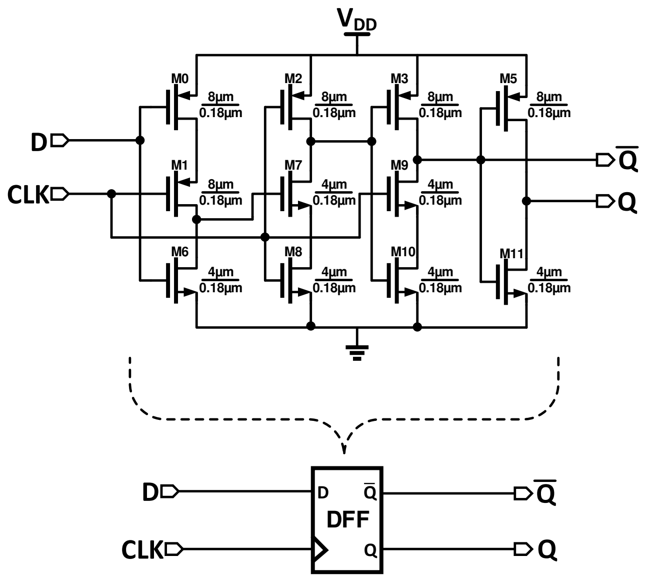


Figure 5.6: TSPC Based D Flip-Flop Architecture

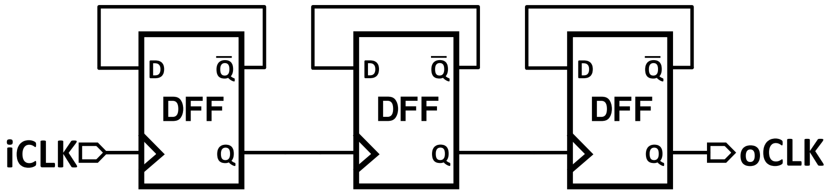


Figure 5.7: Divider Architecture

The divider circuit consists of 3-DFFs that are connected together in the manner shown in Figure 5.7 to realize a divide-by-8 operation. Division in binary is essentially a left-shift operation; thus, tying the outputs of each DFF to clock input of the next while connecting the input to the inverse of output in a feedback ensures a left-shift operation. Since the PLL output frequency is in the GHz range, the DFF design is very critical. To realize a fast DFF with low clock-skew and delay a TSPC (True-Single Phase Clock) architecture (as shown in Figure 5.6) is employed. The basic idea is that when CLK is high transistors M1 and M9 are ON/OFF respectively and vice versa, thereby preserving the state except when CLK goes from low to high, in which case the output follows the data-input signal denoted by D.

CHAPTER 6

BEHAVIORAL LEVEL SIMULATION

6.1 Why Behavioral Modeling?

Traditionally SPICE is used as a common simulation engine to simulate analog/mixed-signal circuit. However, when simulating large networks the simulation times can become extremely long, thereby limiting the allowed design revisions to the circuit designer. It is very tedious to describe the behavior of a circuit using SPICE unless the complete physical transistor-level structure of the circuit is known to the designer. Furthermore, the SPICE simulation process is very technology dependent in that with technology scaling the SPICE models need to be updated as the older models become obsolete and invalid for accurate simulation. The aforementioned design process has remained virtually the same over the past few decades and even though the digital design synthesis process has progressed significantly by incorporating electronic system-level (ESL) design automation techniques, the mixed signal design process is very slow, laborious and therefore error-prone. Digital design engineers, though working with millions of transistors, have been able to automate the design flow, but analog designers have been unable to do so even though most analog circuits only consist of tens of thousands of devices.

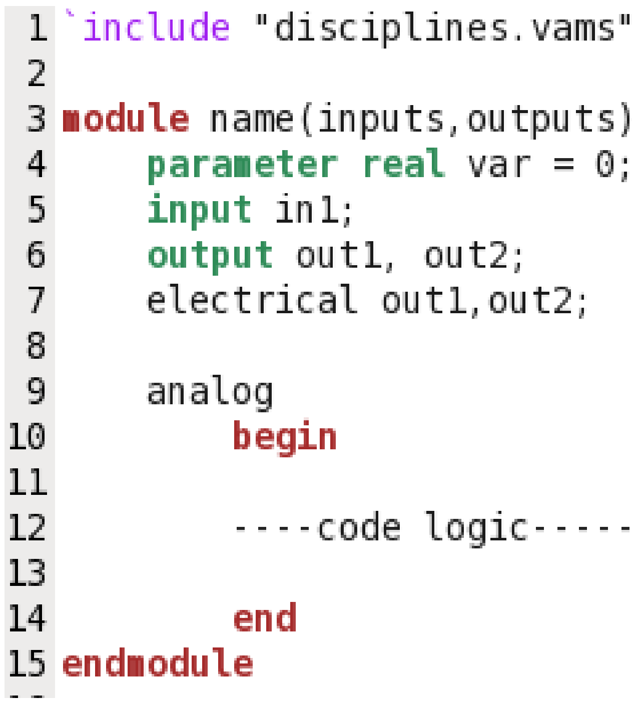
6.2 Why Verilog-AMS?

Verilog-AMS is a high-level Hardware Description Language (HDL) used to describe the structure and behavior of analog and mixed-signal systems. It is an extension to the IEEE 1364 Verilog HDL standard and is very powerful in providing fast prototyping capabilities for mixed-signal systems. The key advantage of circuit modeling using Verilog-AMS is that it provides a single language and simulator ecosystem that can be shared between analog, digital and system-level designers. Verilog-AMS leverages the superior speed and capacity offered by traditional Verilog and allows event-driven capabilities within analog model simulation, making it an attractive choice when simulating highly complex mixed-signal circuits such as PLLs, CDRs, ADCs, and DACs. The only pitfall of using Verilog-AMS is that it cannot replace traditional transistor level SPICE simulation completely as it does not have synthesis capabilities like its digital counterpart Verilog. However, at the onset of the design phase, using Verilog-AMS for circuit modeling is very powerful for a mixed-signal circuit/system design engineer as it offers fast prototyping/verification for behavioral level simulation, thereby expediting the time-to-market for the system.

Verilog-AMS combines both Verilog-D and Verilog-A including a few additional mixed-signal constructs to provide a HDL language capable of performing truly mixed-signal simulation. Cadence has been the front-runner in promoting the language making it an industry standard, and has led the majority of the advancement efforts ever since its release in 2003. The power of Verilog-AMS simulator in Cadence Virtuoso is that it can perform co-simulation among behavioral analog/digital blocks described by corresponding Verilog-A and Verilog-D models respectively as well as transistor-level circuit blocks by running the Spectre simulation. When a circuit consisting of transistor-level circuit elements, analog behavioral modules written in Verilog-A and digital behavioral modules written in Verilog-D is simulated, the AMS simulator in Cadence partitions the testbench into analog and digital components. The simulator then merges the analog simulation results from Spectre with the digital simulation results from NC-SIM and the resulting output is plotted just like that in the case of traditional Spectre simulation [4].

6.3 Basic Verilog-A/AMS Syntax

A typical skeleton of a Verilog-AMS code is shown in Figure 6.1 where the main components of a Verilog-A/AMS code are listed.



```
`include "disciplines.vams"

module name(inputs,outputs)
    parameter real var = 0;
    input in1;
    output out1, out2;
    electrical out1,out2;

    analog
        begin

        ----code logic-----

        end
endmodule
```

Figure 6.1: Verilog-AMS Sample Code

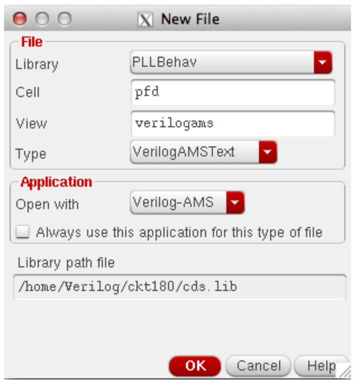
In the first line of the sample code shown in Figure 6.1 [4], we include the ‘disciplines.vams’ header file. This file is a collection of physical signal types that are commonly used in Verilog-AMS and are thus referred to as ‘natures’. Electrical disciplines consist of ‘voltages’ and ‘currents’ and are used most commonly during mixed-signal system modeling where ‘voltage’ and ‘current’ are ‘natures’. Every Verilog-AMS component is defined as a ‘module’ and modules are the basic building blocks of any given Verilog-AMS files as they describe the component being modeled. Ports are the points where connections are made to the given component. Every port is required to have a direction associated with it, and by default in Verilog-AMS language there are three types of ports: **input, output** and **inout**. The keyword **electrical** signifies that the signals associated with the ports described as electrical are of ‘voltage’ and ‘current’ natures. Additionally, **analog** is the keyword after which point the Verilog-AMS compiler starts actual modeling as the logic/process starts after the ‘analog begin’. Finally, every Verilog-AMS component code should end with the word **endmodule** as it signifies the point at which the compiler stops parsing of the code [4].

6.4 PLL Simulation in AMS Using Cadence Virtuoso

6.4.1 PFD+CP

1. Create a new library and name it 'PLLBehav'. Now within the Library Manager window, click on $File \rightarrow New \rightarrow Cell\ View$ and call the new VerilogAMS file pfd. Choose the 'VerilogAMSText' option from the drop-down Menu as shown in Figure 6.2. Click 'OK' and a text editor window will open up.

2. Figure 6.3 shows the 'PFD' code used in the design. The PFD is a completely digital circuit; thus, this code is essentially in Verilog-D syntax where the UP, DN signals are generated by comparing the rising edges of the flip-flops that have a CLK signal of F_{REF} and F_{DIV} respectively.


New File

File

Library: PLLBehav

Cell: pfd

View: verilogams

Type: VerilogAMSText

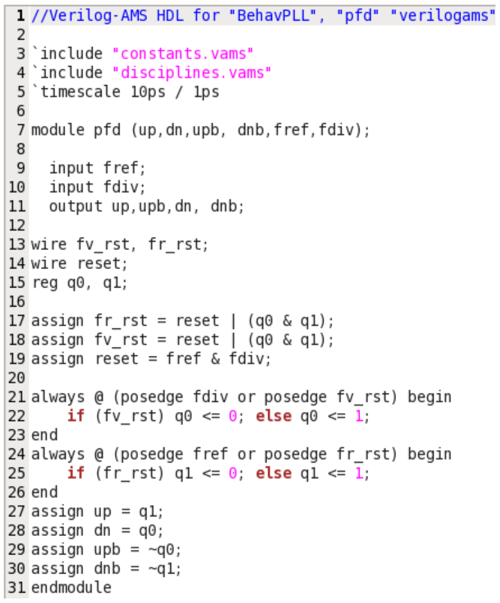
Application

Open with: Verilog-AMS

☐ Always use this application for this type of file

Library path file: /home/Verilog/ckt180/cds.lib

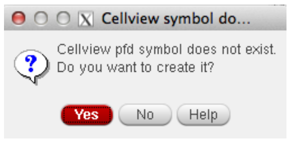
Figure 6.2: PFD Verilog-AMS Code Setup



```
//Verilog-AMS HDL for "BehavPLL", "pfd" "verilogams"

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

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

  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 <= 0; else q0 <= 1;
end
always @ (posedge fref or posedge fr_rst) begin
    if (fr_rst) q1 <= 0; else q1 <= 1;
end
assign up = q1;
assign dn = q0;
assign upb = ~q0;
assign dnb = ~q1;
endmodule
```

Figure 6.3: PFD Verilog-AMS Code

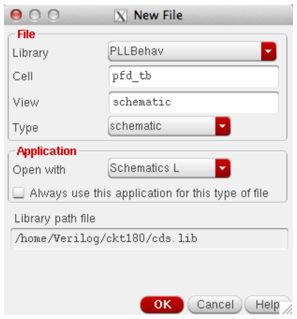
3. Once you have written the code as shown in Figure 6.4, save and exit the text editor. A pop-up window like Figure 6.4 will open up. Click 'Yes' to generate the symbol for the 'pfd'.



Cellview pfd symbol does not exist.
Do you want to create it?

Figure 6.4: PFD Verilog-AMS Symbol

4. Within the 'PLLBehav' library you created above, click on *File* → *New* → *Cell View* and call the new schematic *pfd_tb* as shown in Figure 6.5. Double-click on this cell-view and a schematic window will open up.


New File

File

Library: PLLBehav

Cell: pfd_tb

View: schematic

Type: schematic

Application

Open with: Schematics L

☐ Always use this application for this type of file

Library path file: /home/Verilog/ckt180/cds.lib

Figure 6.5: PFD Verilog-AMS Schematic

5. In order to create a circuit in the schematic editor, we need to add 'instances' or circuit-components like transistors, supply nets and wires. To add an instance press **I** from your keyboard. This will open up a 'Component Browser'. Choose the 'PLLBehav' library and within it select the symbol for 'pfd'. Repeat the same process to add the 'vpulse' components found in the 'analogLib' library. Make sure the 'fref' and 'fdiv' sources have a 100ps delay between each other. Figure 6.6 shows what your test-bench schematic should look like.

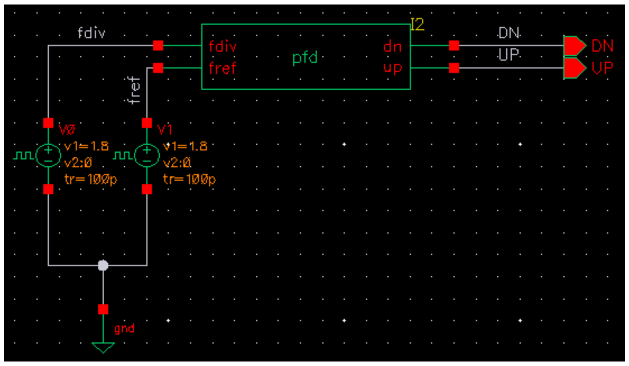
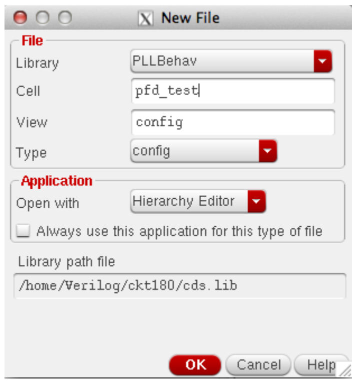


Figure 6.6: PFD Verilog-AMS Testbench

6. When simulating Verilog-AMS files in Cadence Virtuoso, we need to create a 'config' file whose job is to link the analog test-bench sources and the verilog simulation engines together. In order to do so, within the 'pfd_test' cell-view click on *File* → *New* → *Cell View* and call the new config *pfd_test* as shown in Figure 6.7. Double-click on this cell-view and a New Configuration window will open up. Click on 'Use Template', choose the AMS template and configure the setup as shown in Figure 6.8(a). Finally the configuration setup will look like that

shown in Figure 6.8(b), so click on ‘Save’ and press ‘Open’. Now a window like the schematic view will open up but this time it will have config in the title.


New File

File

Library: PLLBehav

Cell: pfd_test

View: config

Type: config

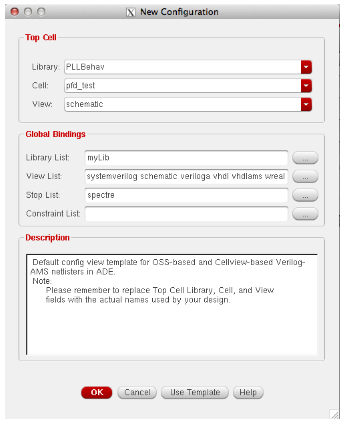
Application

Open with: Hierarchy Editor

Always use this application for this type of file

Library path file: /home/Verilog/ckt180/cds.lib

Figure 6.7: Config File Creation


New Configuration

Top Cell

Library: PLLBehav

Cell: pfd_test

View: schematic

Global Bindings

Library List: myLib

View List: systemverilog schematic veriloga vhdl vhdlams wreal

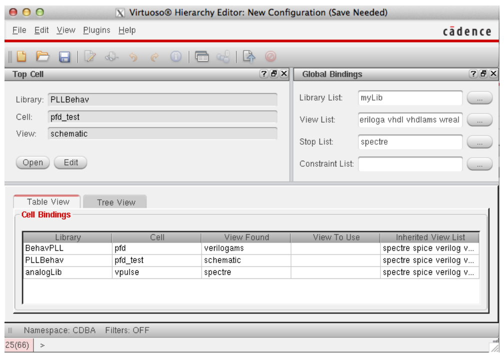
Stop List: spectre

Constraint List:

Description

Default config view template for OSS-based and Cellview-based Verilog-AMS netlisters in ADE.
Note:
Please remember to replace Top Cell Library, Cell, and View fields with the actual names used by your design.

(a)


Virtuoso® Hierarchy Editor: New Configuration (Save Needed)

Top Cell

Library: PLLBehav

Cell: pfd_test

View: schematic

Global Bindings

Library List: myLib

View List: eriloga vhdl vhdlams wreal

Stop List: spectre

Constraint List:

Cell Bindings

Library	Cell	View Found	View To Use	Inherited View List
BehavPLL	pfd	verilogams		spectre spice verilog v...
PLLBehav	pfd_test	schematic		spectre spice verilog v...
analogLib	vpulse	spectre		spectre spice verilog v...

Namespace: CDBA Filters: OFF

25(66) >

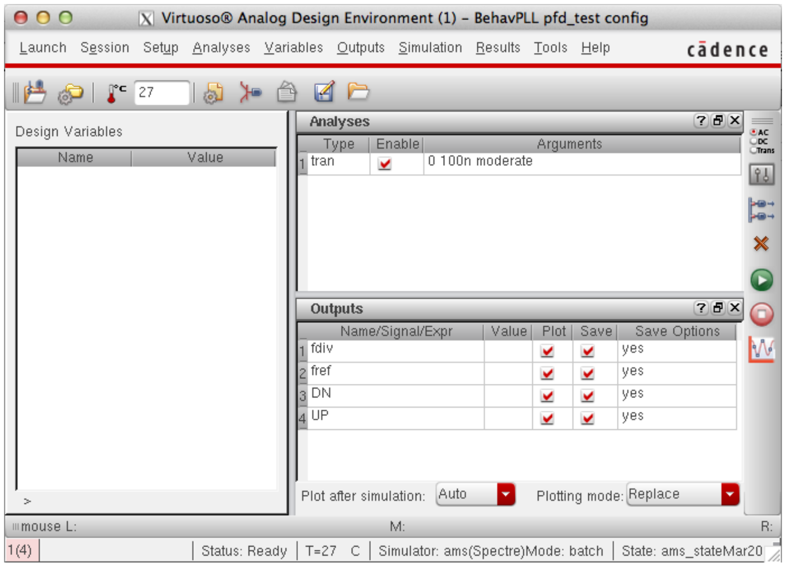
(b)

Figure 6.8: Config File Setup

7. We will simulate our circuits using Cadence AMS Simulation engine. AMS is capable of simulating Verilog-AMS as well as Spectre components. Spectre is a variant of HSPICE developed by Cadence and provides greater accuracy, speed and flexibility especially when dealing with mixed signal circuits.

8. Make sure you first ‘Check and Save’ your config file and click on *Launch* → *ADE* to open up the ADE window.

9. Click on *Setup* → *Simulator* to make sure the Simulator is set to AMS. Select the output nodes and choose a transient simulation for 100ns as shown in Figure 6.9.


Virtuoso® Analog Design Environment (1) – BehavPLL pfd_test config
Launch Session Setup Analyses Variables Outputs Simulation Results Tools Help
cādence
27
Design Variables
Name
Value
AC
DC
Trans
mouse L:
M:
R:
1(4)

Analyses

	Type	Enable	Arguments
1	tran	✓	0 100n moderate

Outputs

	Name/Signal/Expr	Value	Plot	Save	Save Options
1	fdiv		✓	✓	yes
2	fref		✓	✓	yes
3	DN		✓	✓	yes
4	UP		✓	✓	yes

Plot after simulation: Auto Plotting mode: Replace

Status: Ready | T=27 C | Simulator: ams(Spectre)Mode: batch | State: ams_stateMar20

Figure 6.9: PFD Verilog-AMS ADE Outp Window

10. In the final output waveform shown in Figure 6.10 it is clear that the PFD is functioning correctly. Notice that the UP,DN pulses are appropriately modulated as 'REF' and 'DIV' signals diverge from one another.

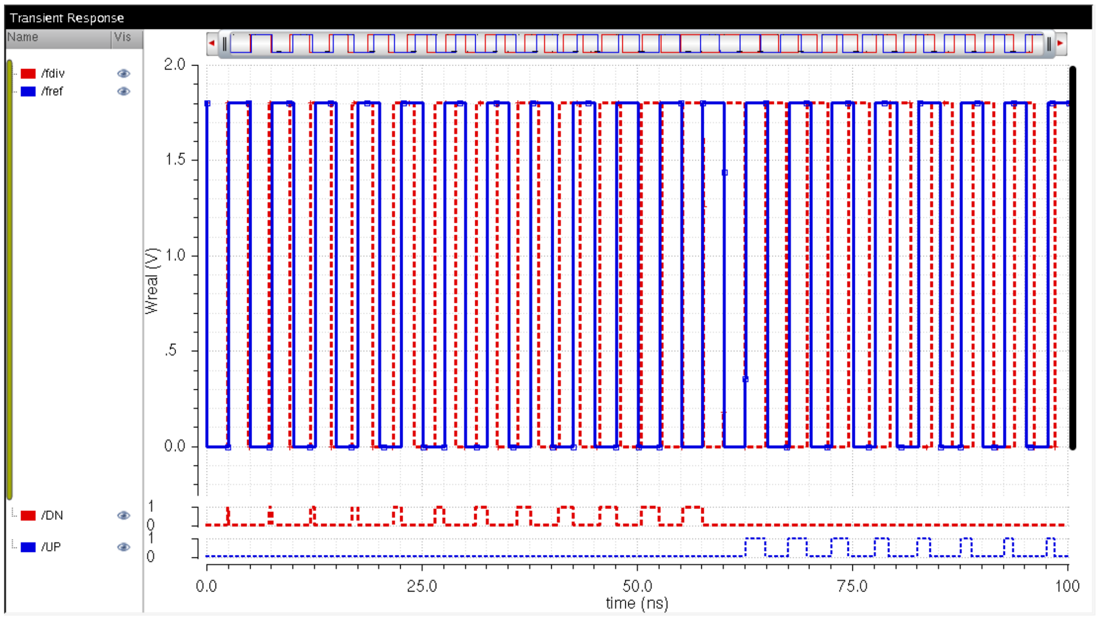
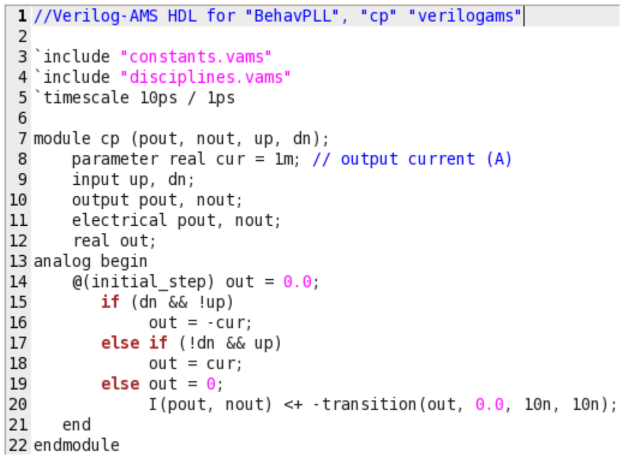


Figure 6.10: PFD Verilog-AMS Simulation Output

11. Within the ‘PLLBehav’ library follow the steps described earlier to create a model for the CP as shown in Figure 6.11 and save the file as ‘cp’.



```
//Verilog-AMS HDL for "BehavPLL", "cp" "verilogams"

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

module cp (pout, nout, up, dn);
    parameter real cur = 1m; // 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, 10n, 10n);
   end
endmodule
```

Figure 6.11: CP Verilog-AMS Code

12. Figure 6.12 shows the ‘PFD+CP’ testbench schematic. Create a new schematic named ‘cp_test’ as well as a config file following the same procedure as the PFD. When simulating using the ADE AMS simulator follow the procedure similar to that shown in Figure 6.9.

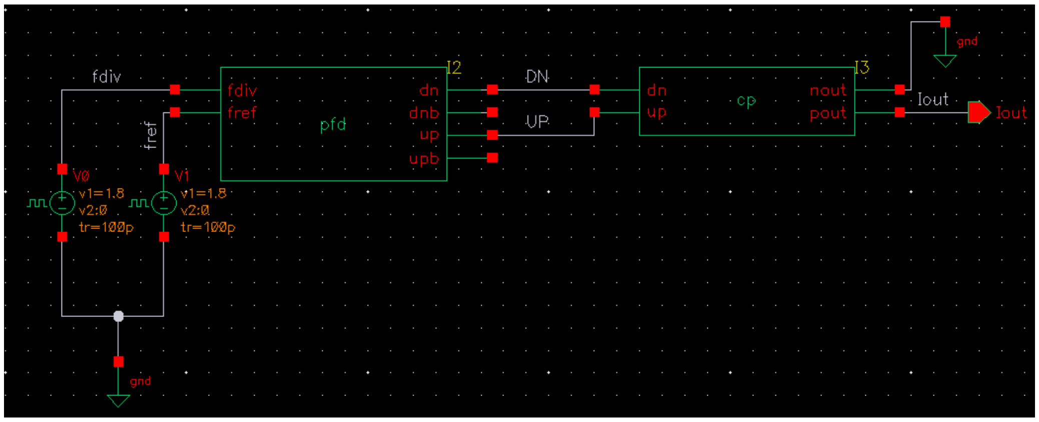
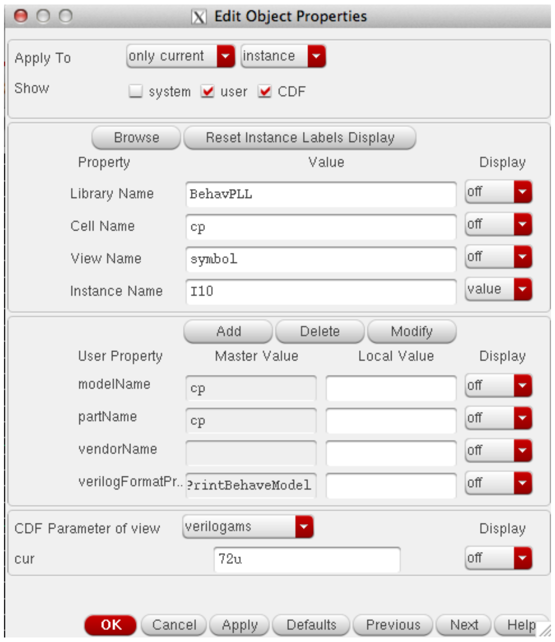


Figure 6.12: PFD+CP Verilog-AMS Testbench

13. One of the powerful advantages of behavioral modeling is that we can easily alter values of design variables to modify the functionality of a block. In the 'config' testbench file if you click on the 'CP' block and press **q**, a window as shown in Figure 6.13 will appear. Enter the appropriate value of charge-pump current as per the design objectives.


Edit Object Properties

Apply To: only current, instance

Show: ☐ system ✓ user ✓ CDF

Browse | Reset Instance Labels Display

Property	Value	Display
Library Name	BehavPLL	off
Cell Name	cp	off
View Name	symbol	off
Instance Name	I10	value

Add | Delete | Modify

User Property	Master Value	Local Value	Display
modelName	cp		off
partName	cp		off
vendorName			off
verilogFormatPr..	?rintBehaveModel		off

CDF Parameter of view: verilogams — Display

cur	72u	off

OK | Cancel | Apply | Defaults | Previous | Next | Help

Figure 6.13: CP Verilog-AMS Testbench Variable Setup

14. The purpose of the charge-pump is to convert the digital PWM signal outputs from the PFD into a current. As seen in the code and from the final output waveform shown in Figure 6.14, it is clear that the

‘PFD+CP’ is functioning correctly. When UP is high the current the pull-up current source is on and when DN is high the pull-down current source is on.

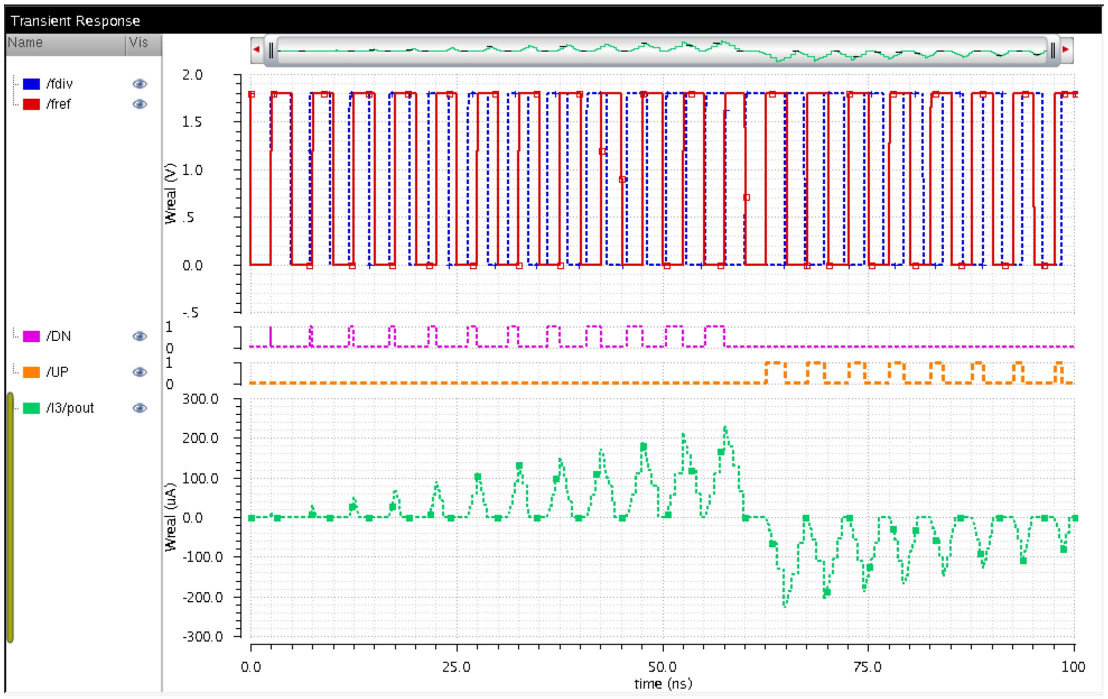


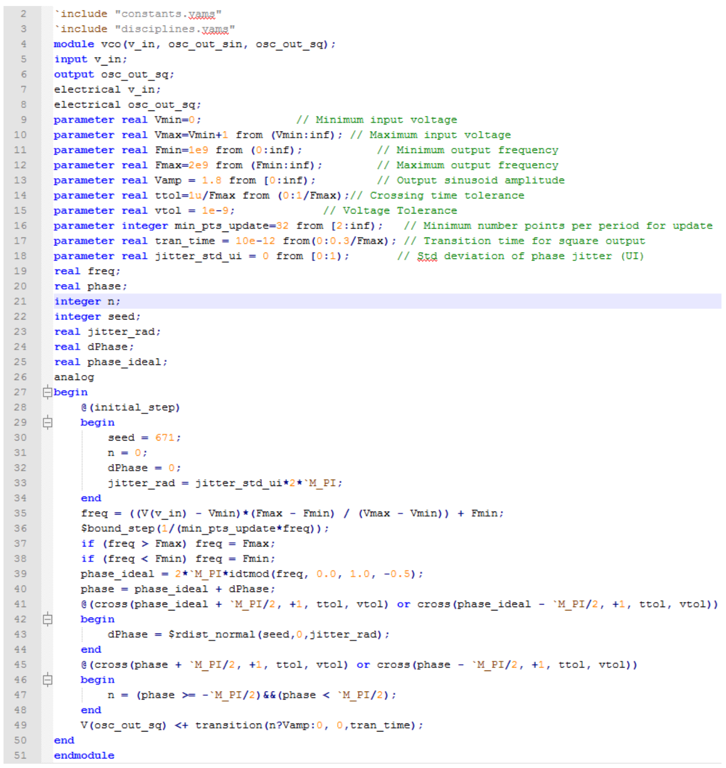
Figure 6.14: PFD+CP Verilog-AMS Simulation Output

6.4.2 LF

We use the analog loop-filter as shown in Figure 5.4.

6.4.3 VCO

1. The VCO is the most critical component of the PLL we try to model using Verilog-AMS because it allows us to behaviorally estimate the jitter specifications. Within the ‘PLLBehav’ library follow the steps described earlier to create a model for the VCO as shown in Figure 6.15 and save the file as ‘vco’. Only the white-noise jitter is considered in this design and it is modeled by a Gaussian white-noise probability distribution function.



```
`include "constants.vams"
`include "disciplines.vams"
module vco(v_in, osc_out_sin, osc_out_sq);
input v_in;
output osc_out_sq;
electrical v_in;
electrical osc_out_sq;
parameter real Vmin=0;                  // Minimum input voltage
parameter real Vmax=Vmin+1 from (Vmin:inf); // Maximum input voltage
parameter real Fmin=1e9 from (0:inf);            // Minimum output frequency
parameter real Fmax=2e9 from (Fmin:inf);         // Maximum output frequency
parameter real Vamp = 1.8 from [0:inf);          // Output sinusoid amplitude
parameter real ttol=1u/Fmax from (0:1/Fmax);// Crossing time tolerance
parameter real vtol = 1e-9;              // Voltage Tolerance
parameter integer min_pts_update=32 from [2:inf);   // Minimum number points per period for update
parameter real tran_time = 10e-12 from(0:0.3/Fmax); // Transition time for square output
parameter real jitter_std_ui = 0 from [0:1);       // Std deviation of phase jitter (UI)
real freq;
real phase;
integer n;
integer seed;
real jitter_rad;
real dPhase;
real phase_ideal;
analog
begin
    @(initial_step)
    begin
        seed = 671;
        n = 0;
        dPhase = 0;
        jitter_rad = jitter_std_ui*2*`M_PI;
    end
    freq = ((V(v_in) - Vmin)*(Fmax - Fmin) / (Vmax - Vmin)) + Fmin;
    $bound_step(1/(min_pts_update*freq));
    if (freq > Fmax) freq = Fmax;
    if (freq < Fmin) freq = Fmin;
    phase_ideal = 2*`M_PI*idtmod(freq, 0.0, 1.0, -0.5);
    phase = phase_ideal + dPhase;
    @(cross(phase_ideal + `M_PI/2, +1, ttol, vtol) or cross(phase_ideal - `M_PI/2, +1, ttol, vtol))
    begin
        dPhase = $rdist_normal(seed,0,jitter_rad);
    end
    @(cross(phase + `M_PI/2, +1, ttol, vtol) or cross(phase - `M_PI/2, +1, ttol, vtol))
    begin
        n = (phase >= -`M_PI/2)&&(phase < `M_PI/2);
    end
    V(osc_out_sq) <+ transition(n?Vamp:0, 0,tran_time);
end
endmodule
```

Figure 6.15: VCO Verilog-AMS Code

2. Figure 6.16 shows the 'VCO' testbench schematic. Create a new schematic named 'vco_test' as well as a config file following the same procedure as the PFD. When simulating using the ADE AMS simulator follow the procedure similar to that shown in Figure 6.9.

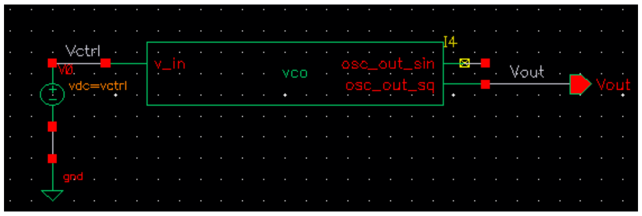
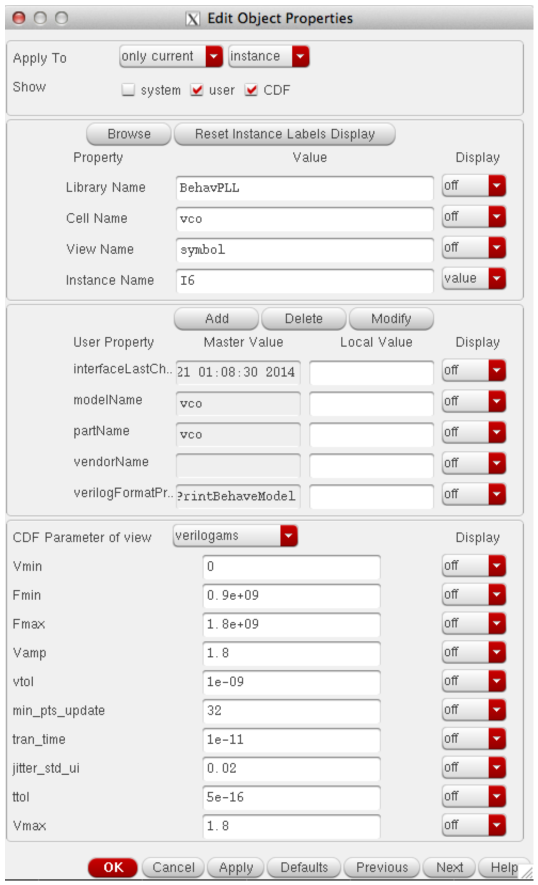


Figure 6.16: VCO Verilog-AMS Testbench

3. Just like in the case of the 'CP' in the 'config' testbench file, if you click on the 'VCO' block and press **q**, a window as shown in Figure 6.17 will appear. Enter the appropriate value VCO design parameters as per the design objectives.



Apply To: only current, instance
Show: [ ] system [✓] user [✓] CDF

Property	Value	Display
Library Name	BehavPLL	off
Cell Name	vco	off
View Name	symbol	off
Instance Name	I6	value

User Property	Master Value	Local Value	Display
interfaceLastCh..	21 01:08:30 2014		off
modelName	vco		off
partName	vco		off
vendorName			off
verilogFormatPr..	?rintBehaveModel		off

CDF Parameter of view: verilogams

		Display
Vmin	0	off
Fmin	0.9e+09	off
Fmax	1.8e+09	off
Vamp	1.8	off
vtol	1e-09	off
min_pts_update	32	off
tran_time	1e-11	off
jitter_std_ui	0.02	off
ttol	5e-16	off
Vmax	1.8	off

Figure 6.17: VCO Verilog-AMS Testbench Variable Setup

4. The VCO circuit is supposed to generate a periodic square-wave output at the desired frequency of interest (as a function of the control voltage) with a certain jitter level which in our case is chosen to be 2% Unit-Interval (UI) of period. From the final output waveform shown in Figure 6.18 it is clear that the 'VCO' is functioning correctly.

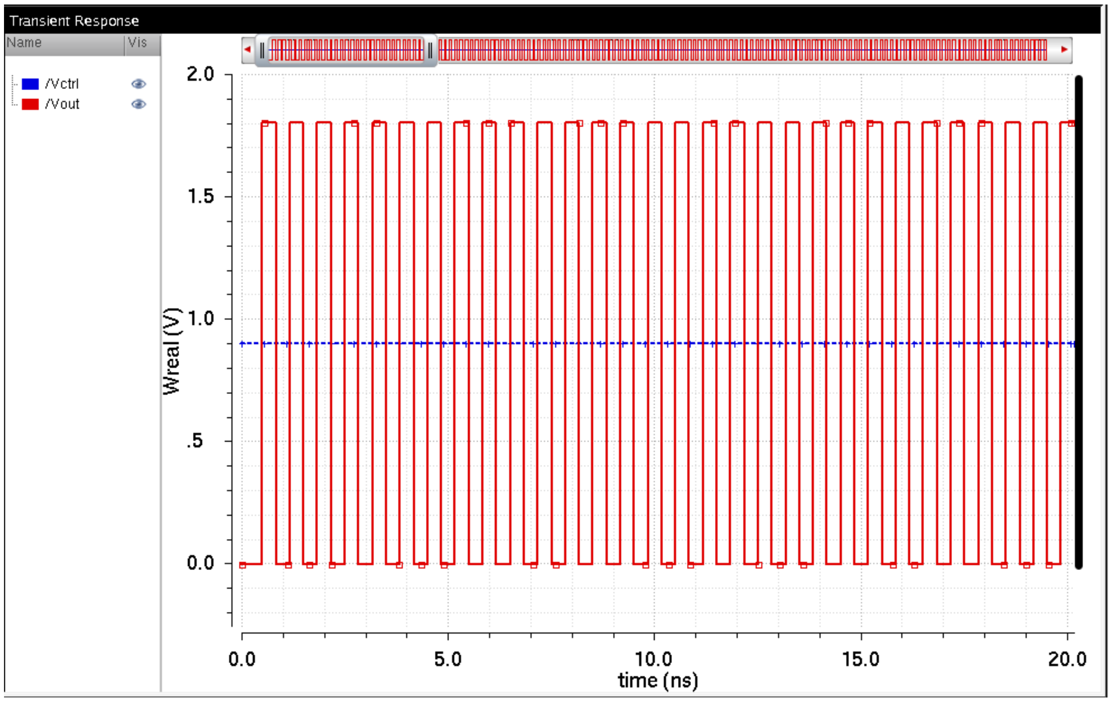
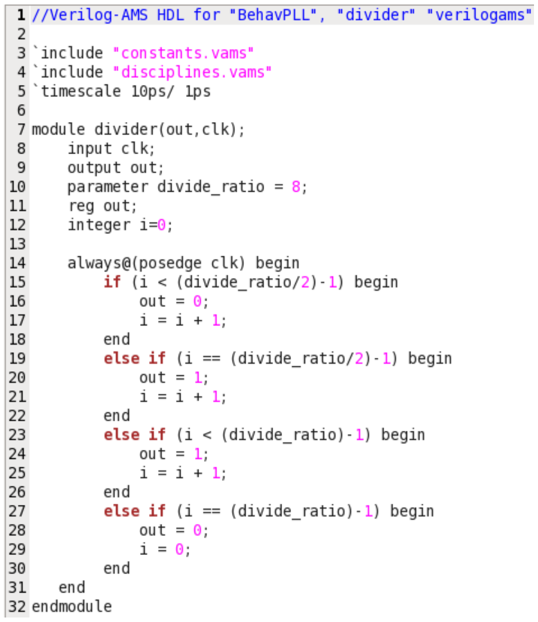


Figure 6.18: VCO Verilog-AMS Simulation Output

6.4.4 Divider

1. Divider is essential when designing a clock-generating circuit as we need to scale down the VCO output clock to the reference frequency level such that the two signals can be compared. Within the 'PLLBehav' library follow the steps described earlier to create a model for the Divider as shown in Figure 6.15 and save the file as 'div'. Figure 6.19 shows the code to implement the divider in Verilog.



```
//Verilog-AMS HDL for "BehavPLL", "divider" "verilogams"

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

module divider(out,clk);
    input clk;
    output out;
    parameter divide_ratio = 8;
    reg out;
    integer i=0;

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

Figure 6.19: Divider Verilog-AMS Code

2. Figure 6.20 shows the 'Divider' testbench schematic. Create a new schematic named 'div_test' as well as a config file following the same procedure as the PFD. When simulating using the ADE AMS simulator follow the procedure similar to that shown in Figure 6.9.

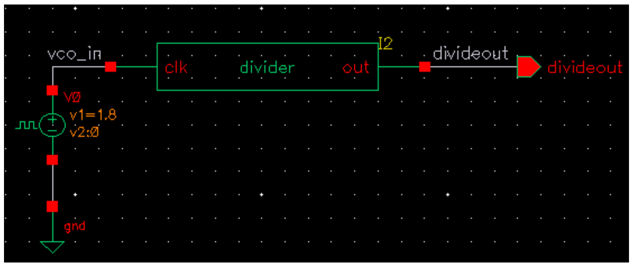
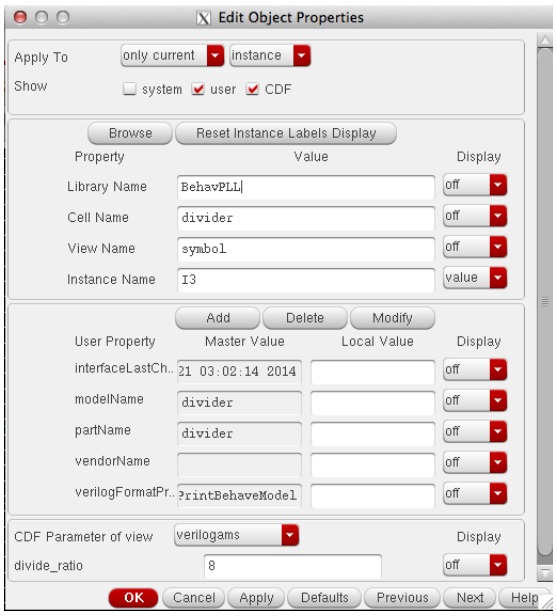


Figure 6.20: Divider Verilog-AMS Testbench

3. Just like in the case of the 'CP and VCO', in the 'config' testbench file if you click on the 'Divider' block and press **q**, a window as shown in Figure 6.21 will appear. Enter the appropriate value of divide ratio as per the design objectives.



Edit Object Properties

Apply To	only current	instance
Show	☐ system ☑ user ☑ CDF	

Browse | Reset Instance Labels Display

Property	Value	Display
Library Name	BehavPLL	off
Cell Name	divider	off
View Name	symbol	off
Instance Name	I3	value

Add | Delete | Modify

User Property	Master Value	Local Value	Display
interfaceLastCh..	21 03:02:14 2014		off
modelName	divider		off
partName	divider		off
vendorName			off
verilogFormatPr..	PrintBehaveModel		off

CDF Parameter of view	verilogams	Display
divide_ratio	8	off

OK | Cancel | Apply | Defaults | Previous | Next | Help

Figure 6.21: VCO Verilog-AMS Testbench Variable Setup

4. The divider circuit is supposed to generate a periodic square-wave output that is fraction of the VCO output frequency. From the final output waveform shown in Figure 6.22 it is clear that the 'Divider' is functioning correctly in that it divides the VCO output signal by a factor of 8.

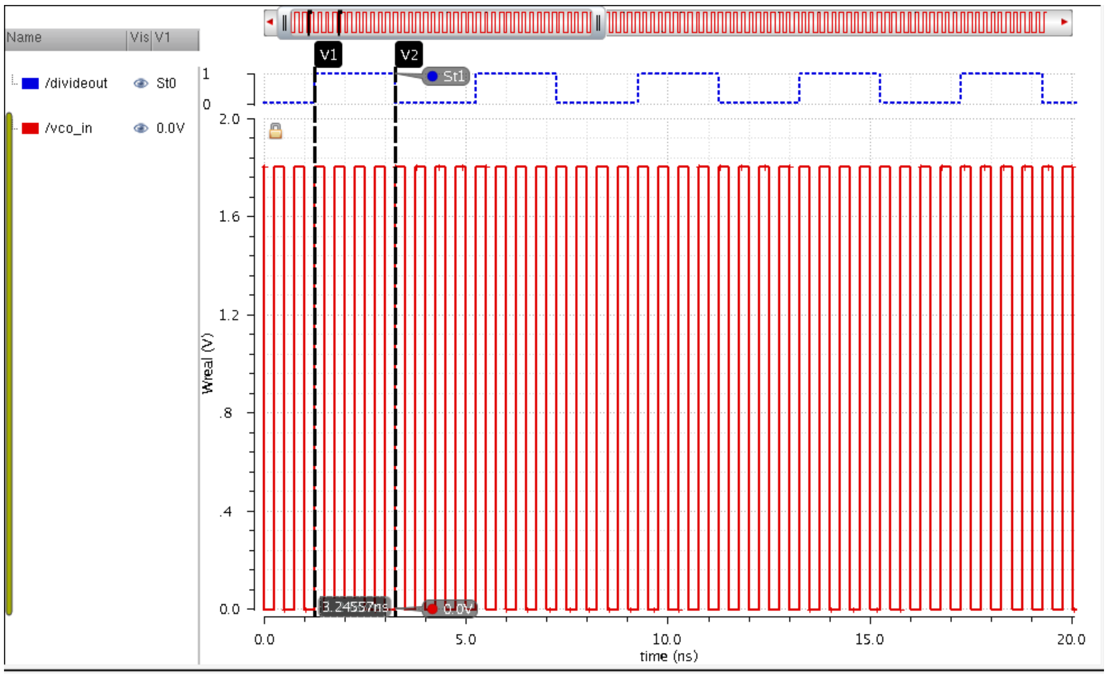


Figure 6.22: Divider Verilog-AMS Simulation Output

6.4.5 Complete PLL Analysis with Jitter

1. Create a new schematic within the ‘PLLBehav’ library and name it ‘PLL’. Your schematic should look that shown in Figure 6.23. Now create a config file for this setup and at the end your configuration window should look like Figure 6.24.

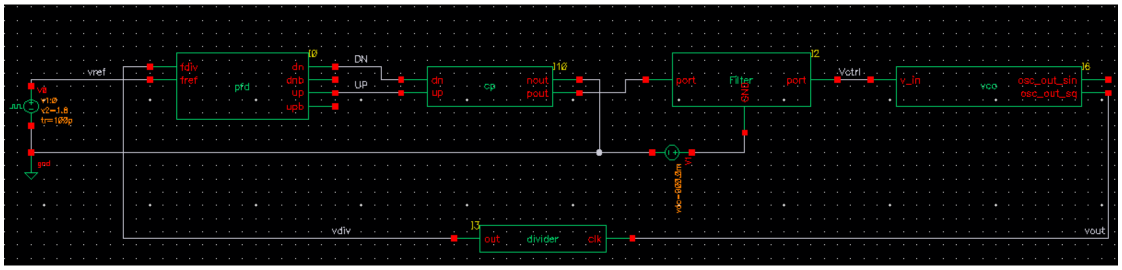
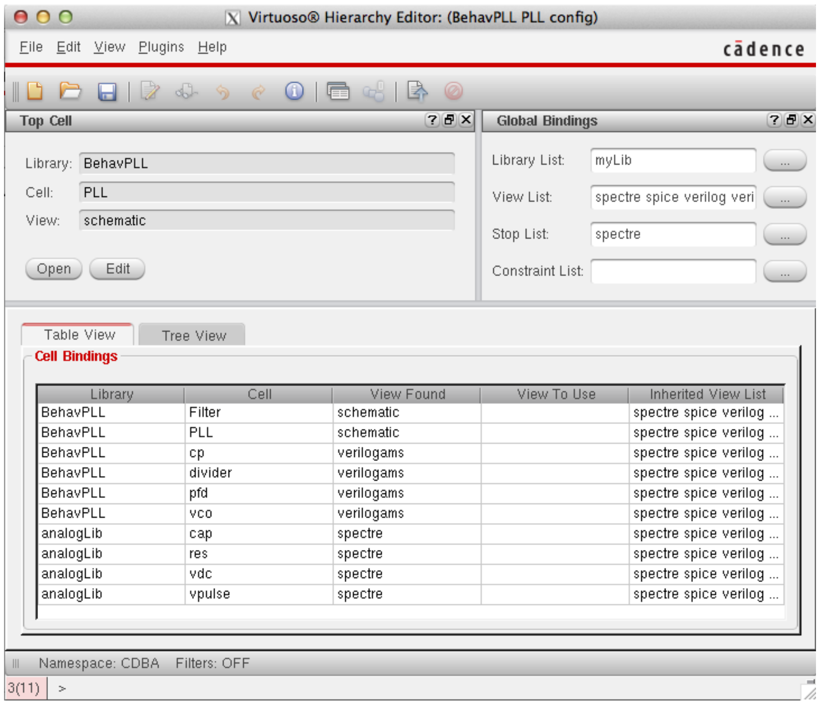


Figure 6.23: PLL Verilog-AMS Testbench


Virtuoso® Hierarchy Editor: (BehavPLL PLL config)
File Edit View Plugins Help
cadence
Top Cell
Global Bindings
Open
Edit
Table View
Tree View
Cell Bindings
Namespace: CDBA Filters: OFF
3(11) >

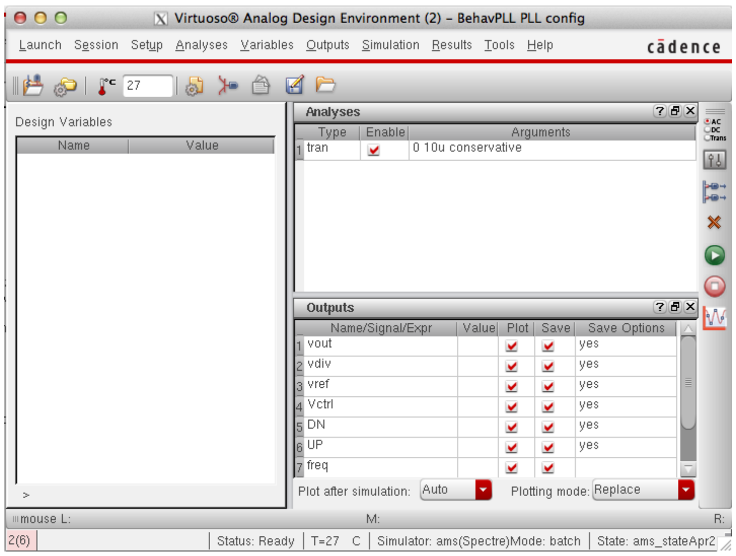
Library:	BehavPLL
Cell:	PLL
View:	schematic

Library List:	myLib
View List:	spectre spice verilog veri
Stop List:	spectre
Constraint List:	

Library	Cell	View Found	View To Use	Inherited View List
BehavPLL	Filter	schematic		spectre spice verilog ...
BehavPLL	PLL	schematic		spectre spice verilog ...
BehavPLL	cp	verilogams		spectre spice verilog ...
BehavPLL	divider	verilogams		spectre spice verilog ...
BehavPLL	pfd	verilogams		spectre spice verilog ...
BehavPLL	vco	verilogams		spectre spice verilog ...
analogLib	cap	spectre		spectre spice verilog ...
analogLib	res	spectre		spectre spice verilog ...
analogLib	vdc	spectre		spectre spice verilog ...
analogLib	vpulse	spectre		spectre spice verilog ...

Figure 6.24: PLL Verilog-AMS Config Setup

2. Using the steps described earlier in this chapter, configure your ADE window as shown in Figure 6.25 and simulate the circuit.


Virtuoso® Analog Design Environment (2) – BehavPLL PLL config
Launch Session Setup Analyses Variables Outputs Simulation Results Tools Help
cādence

Design Variables

Name	Value

Analyses

	Type	Enable	Arguments
1	tran	✓	0 10u conservative

Outputs

	Name/Signal/Expr	Value	Plot	Save	Save Options
1	vout		✓	✓	yes
2	vdiv		✓	✓	yes
3	vref		✓	✓	yes
4	Vctrl		✓	✓	yes
5	DN		✓	✓	yes
6	UP		✓	✓	yes
7	freq		✓	✓	

Plot after simulation: Auto   Plotting mode: Replace

mouse L: M: R:

2(6) | Status: Ready | T=27 C | Simulator: ams(Spectre)Mode: batch | State: ams_stateApr2

Figure 6.25: PLL Verilog-AMS ADE Setup

3. The PLL circuit outputs are shown in Figure 6.26. It is clear that the PLL achieves lock within the first 100ns because in the testbench we provide an initial condition of $V_{ctrl} = 0.9V$ and keep the currents at the loop-filter capacitors at an initial condition of 0A. These initial conditions are provided to ensure that the simulation time is small. From the final output waveforms it is clear that the 'PLL' is indeed functioning correctly.

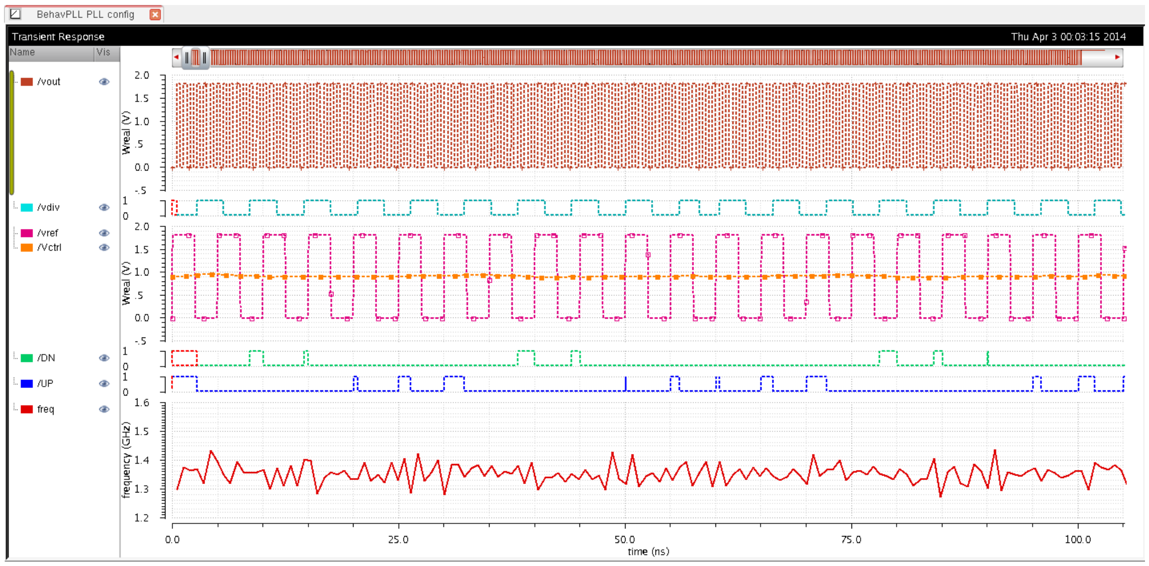
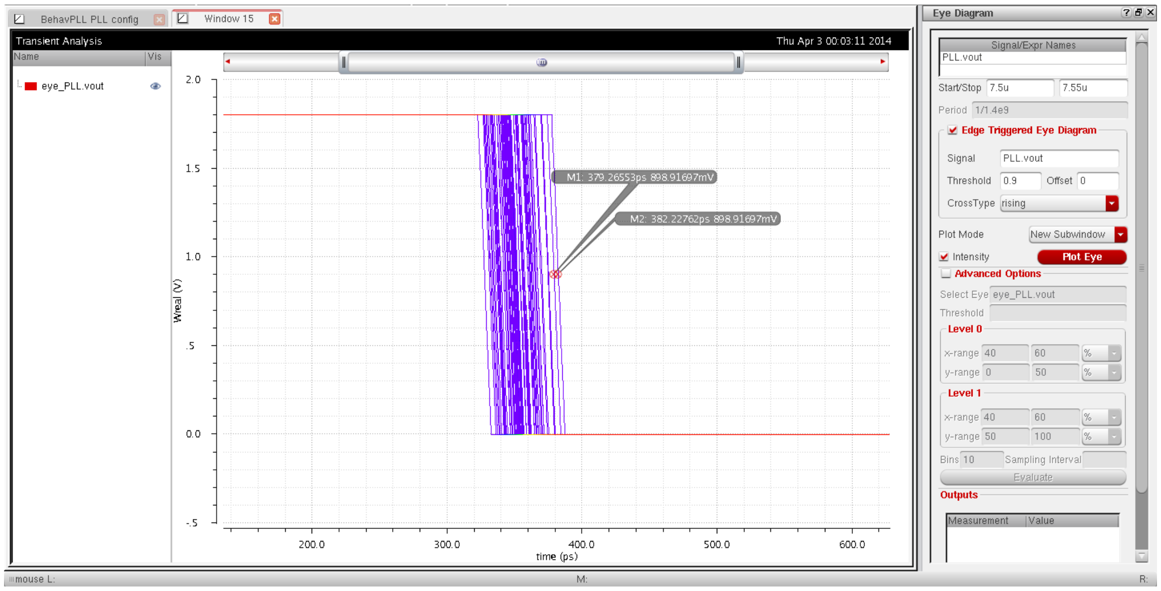


Figure 6.26: PLL Verilog-AMS Simulation Output

4. To simulate the jitter at the VCO output during lock-condition, select the *vout* waveform, click on *Measurements* → *EyeDiagram* and configure the setup as shown in Figure 6.27. Your final output should look like that shown in Figure 6.27 once you click on ‘Plot Eye’. The simulated edge-to-edge jitter is 0.96ps which is extremely good. However, it is important to note that this number is not realistic as we have only accounted for random jitter caused by white-noise and the model is only behavioral so any transistor-level non-idealities are not captured. Nevertheless, behavioral modeling is very powerful in performing rapid prototyping of the PLL circuit elements and performs a system level noise/timing budget for the design before delving straight into transistor level design.



Signal/Expr Names: PLL.vout
Start/Stop: 7.5u 7.55u
Period: 1/1.4e9
[✓] Edge Triggered Eye Diagram
Signal: PLL.vout
Threshold: 0.9 Offset: 0
CrossType: rising
Plot Mode: New Subwindow
[✓] Intensity
Plot Eye
[ ] Advanced Options
Select Eye: eye_PLL.vout
Threshold:
Level 0
x-range 40 60 %
y-range 0 50 %
Level 1
x-range 40 60 %
y-range 50 100 %
Bins 10 Sampling Interval
Evaluate
Outputs

Measurement	Value

Figure 6.27: PLL Verilog-AMS Jitter

CHAPTER 7

TRANSISTOR LEVEL SIMULATION

7.1 What is SPICE?

Simulation Program with Integrated Circuit Emphasis (SPICE) is a general-purpose circuit simulation program that was originally developed at the University of California-Berkeley to serve as a numerical circuit solver that is capable of performing DC, Transient, as well as AC analyses for electronic circuits. The simulator in general is capable of performing the aforementioned analyses on circuits containing resistors, capacitors, inductors, independent voltage and current sources, dependent sources, lossless and lossy transmission lines, switches, uniform distributed RC lines, and the five most common semiconductor devices: diodes, BJTs, JFETs, MESFETs, and MOSFETs. Many variants of SPICE have been developed since with the most popular ones being HSPICE and Spectre.

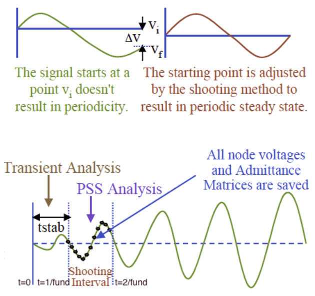
7.2 SPICE vs. Spectre

The Spectre circuit simulator is a variant of SPICE that was developed by Cadence to simulate analog and digital circuits at the differential equation level. Although at a high-level the Spectre and SPICE circuit simulators are quite similar in terms of functionality, Spectre directly is not dependent on SPICE and the two simulators also have differing syntax. The parent algorithms for both are primarily the same in that both use the Modified Nodal Analysis (MNA) method involving implicit integration methods, Newton-Raphson, and direct matrix solution, but the source codes are not borrowed from original open-source SPICE. Spectre is optimized for faster speed as well accuracy compared to SPICE and is thus much more reliable and accurate.

7.3 Transient, PSS and PNoise Simulation Overview

Transient response is the time-domain simulation response for a given circuit and is used to study the time-domain behavior of voltages and currents at any given node in a network. It is a powerful analysis method to study amplifier circuits; however, in the case of oscillators it falls short in being able to accurately characterize the harmonic behavior of the outputs. Thus, to study oscillator, mixer circuits or for that matter any circuit that has a time-varying or periodic nature, the Periodic Steady State (PSS) analysis is the preferred method of simulation.

PSS is a large-signal analysis tool and is powerful in accurately determining the approximate small-signal period of the circuit being analyzed. It uses the Iterative Shooting Newton method to algorithmically determine the fundamental frequency of the circuit/system based on the input-source frequency excitation. In PSS, a circuit is evaluated for one period of the target frequency and this period is dynamically adjusted until all node voltages and branch currents fall within a specified tolerance level. Thus, when simulating large networks the PSS simulation often fails to converge and the time-step needs to be manually adjusted. It is also possible that the simulator is just not robust enough for PSS to converge if the time-step is made too small. The first step in a PSS simulation is to perform a transient simulation on the network from time $t = 0$ to $t = \frac{1}{f_{fund}}$. The next step is then to adjust the time-step adaptively such that the voltage and currents at stabilize within the threshold levels set for the start and stop times of the shooting interval. Figure 7.1 further describes this phenomenon graphically. Note: It is critical to remember that PSS simulation is only valid, and thus will only work, if the circuit/system being analyzed is periodic as the fundamental assumption of PSS analysis is periodicity.



v_i
ΔV
v_f
The signal starts at a point v_i doesn't result in periodicity.
The starting point is adjusted by the shooting method to result in periodic steady state.
Transient Analysis
PSS Analysis
All node voltages and Admittance Matrices are saved
tstab
Shooting Interval
t=0
t=1/fund
t=2/fund

Figure 7.1: PSS Simulation Algorithm

As discussed in earlier chapters, when studying oscillators and PLL circuits the phase-noise is a very important parameter to calculate/simulate. Phase noise is the most significant source of noise in oscillators, and since it is spectrally centered around the fundamental oscillation frequency, methods like filtering cannot eliminate it. PNoise analysis engine within Spectre is equipped to predict the phase-noise, as well as the total-noise profile which includes thermal, flicker and shot noise. Once the PSS simulation for the circuit being analyzed has been completed, the PNoise analysis can be started and it computes the frequency convention, noise-folding and aliasing effects for the circuit/system.

7.4 PLL Simulation in Spectre Using Cadence Virtuoso

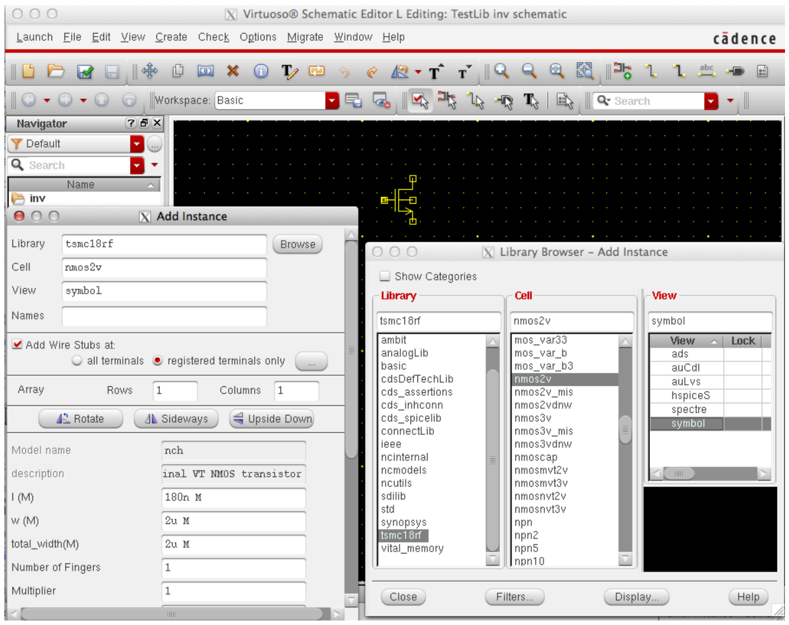
This section presents a detailed step-by-step tutorial on conducting transistor level simulations using Cadence Virtuoso's Spectre circuit simulator engine. First, an inverter circuit used inside the PFD is described to illustrate the basic steps required in creating a new schematic and testbench. Second, an in-depth VCO simulation guide illustrates the steps involved in performing PSS and PNoise simulations. Lastly, the full PLL consisting of the PFD, CP, Filter, VCO and Divider blocks is simulated at the transistor level along with the various steps involved in validating lock condition, noise profile and transient response.

7.4.1 Creating a New Schematic

1. Create a new library and name it PLL. Now within the Library Manager window click on *File* $\rightarrow$ *New* $\rightarrow$ *Cell View* and call the new schematic *inv*. Double-click on this cell-view and a schematic window will open up.

2. In order to create a circuit in the schematic editor we need to add 'instances' or circuit-components like transistors, supply nets and wires. Since we use an inverter as an example, recall that we need one NMOS and one PMOS transistor; thus, to add an instance press **I** from your keyboard. This will open up a 'Component Browser'. Choose the

‘analogLib’ library. You will notice all the components housed within the ‘tsmc18rf’ library listed. The key trick to know is that you can search for a specific component from the ‘Filter’. Search for ‘nmos2v’ and follow the steps outline in Figure 7.2.


Virtuoso® Schematic Editor L Editing: TestLib inv schematic
Launch File Edit View Create Check Options Migrate Window Help
cādence
Workspace: Basic
Navigator
Default
Search
Name
inv
Add Instance
Browse
Rotate
Sideways
Upside Down
Library Browser – Add Instance
Show Categories
Close
Filters...
Display...
Help

Add Instance

Library: tsmc18rf
Cell: nmos2v
View: symbol
Names:

[✓] Add Wire Stubs at: all terminals [✓ registered terminals only]

Array Rows 1 Columns 1

Model name: nch
description: inal VT NMOS transistor
l (M): 180n M
w (M): 2u M
total_width(M): 2u M
Number of Fingers: 1
Multiplier: 1

Library Browser – Add Instance

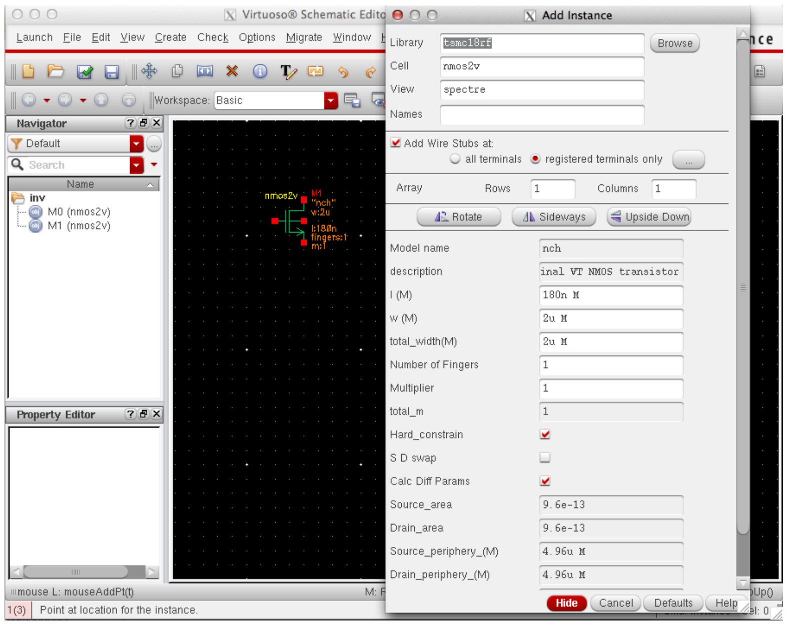
Library: tsmc18rf
ambit, analogLib, basic, cdsDefTechLib, cds_assertions, cds_inhconn, cds_spicelib, connectLib, ieee, ncinternal, ncmodels, ncutils, sdilib, std, synopsys, tsmc18rf, vital_memory

Cell: nmos2v
mos_var33, mos_var_b, mos_var_b3, nmos2v, nmos2v_mis, nmos2vdnw, nmos3v, nmos3v_mis, nmos3vdnw, nmoscap, nmosmvt2v, nmosmvt3v, nmosnvt2v, nmosnvt3v, npn, npn2, npn5, npn10

View: symbol

View	Lock
ads	
auCdl	
auLvs	
hspiceS	
spectre	
symbol	

(a)



inv
M0 (nmos2v)
M1 (nmos2v)

mouse L: mouseAddPt()

1(3) Point at location for the instance.

Add Instance

Library: tsmc18rf
Cell: nmos2v
View: spectre
Names:

[✓] Add Wire Stubs at: all terminals [✓ registered terminals only]

Array Rows 1 Columns 1

Model name: nch
description: inal VT NMOS transistor
l (M): 180n M
w (M): 2u M
total_width(M): 2u M
Number of Fingers: 1
Multiplier: 1
total_m: 1
Hard_constrain: ✓
S D swap: ☐
Calc Diff Params: ✓
Source_area: 9.6e-13
Drain_area: 9.6e-13
Source_periphery_(M): 4.96u M
Drain_periphery_(M): 4.96u M

(b)

Figure 7.2: Inserting NMOS Transistor on Schematic

3. Similarly, following the same steps as (2), add a PMOS transistor to your schematic by choosing the ‘pmos2v’ transistor from the ‘tsmc18rf’ library. Your schematic should now look like Figure 7.3.

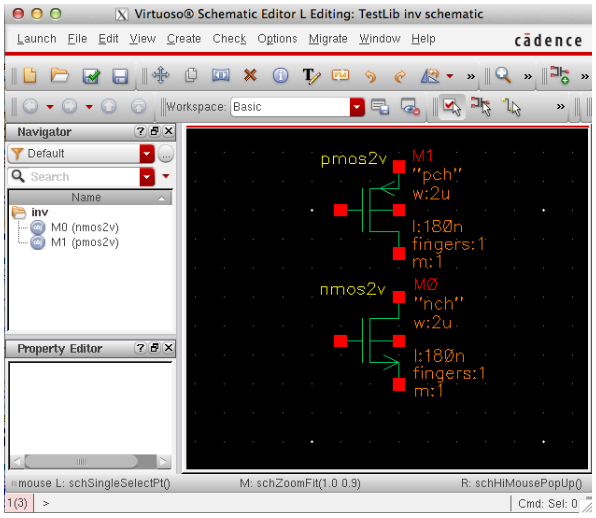
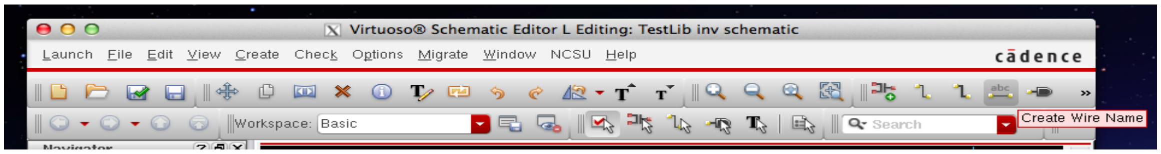
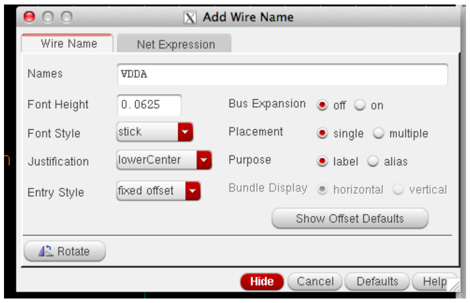


Figure 7.3: PMOS Transistor

4. In order to add wires to your schematic, press **W** from your keyboard and make appropriate connections across all transistor elements. Figure 7.4 demonstrates the steps involved in labeling wires with a circuit schematic. This will come in very handy during simulation, especially when dealing with circuits with several components.


Virtuoso® Schematic Editor L Editing: TestLib inv schematic
Launch File Edit View Create Check Options Migrate Window NCSU Help
cādence
Workspace: Basic
Search
Create Wire Name

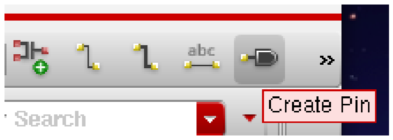
(a)



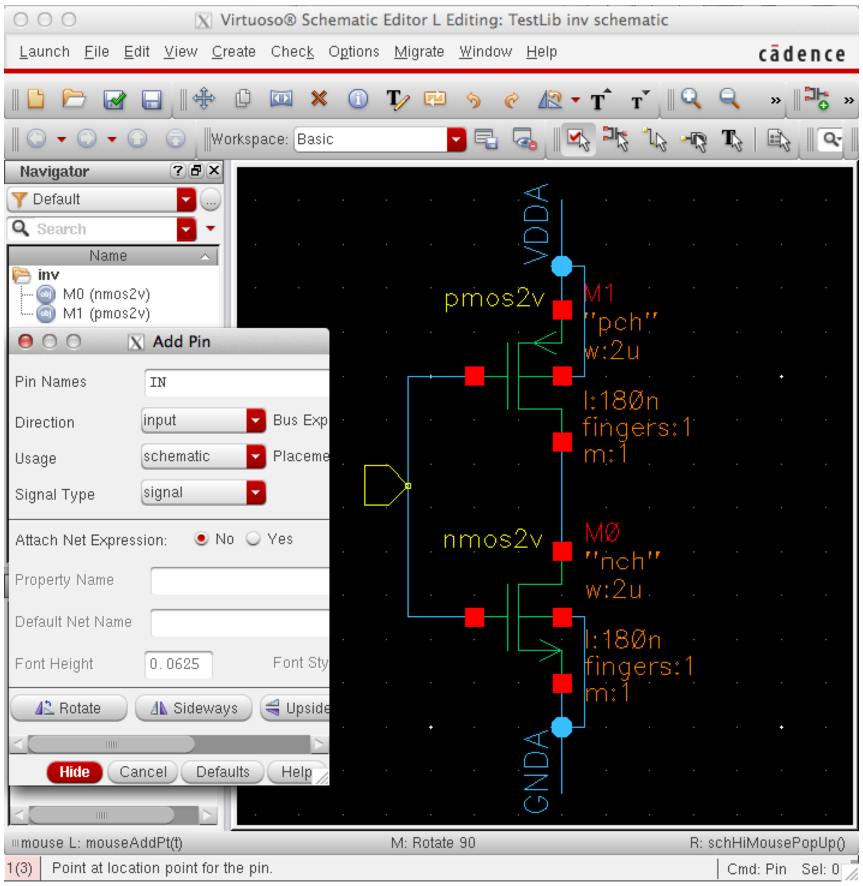
Names: VDDA
Font Height: 0.0625
Font Style: stick
Justification: lowerCenter
Entry Style: fixed offset
Bus Expansion: [✓ off] on
Placement: [✓ single] multiple
Purpose: [✓ label] alias
Bundle Display: [✓ horizontal] vertical

(b)

Figure 7.4: Inserting Wire Names on Circuit



(a)



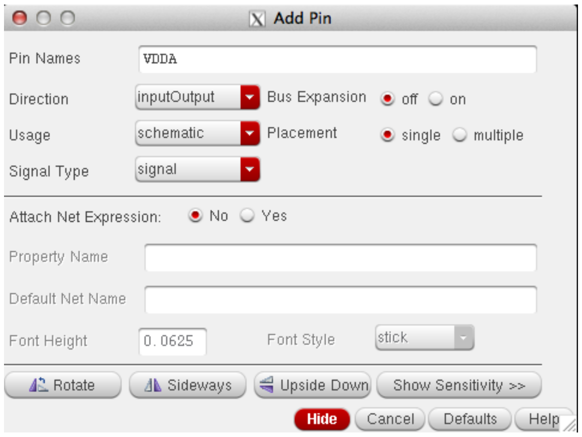
Add Pin

Pin Names: IN
Direction: input
Usage: schematic
Signal Type: signal
Attach Net Expression: [✓ No] Yes
Property Name
Default Net Name
Font Height: 0.0625

mouse L: mouseAddPt(t) M: Rotate 90 R: schHiMousePopUp()

1(3) Point at location point for the pin. Cmd: Pin Sel: 0

(b)



Add Pin

Pin Names: VDDA
Direction: inputOutput
Bus Expansion: [✓ off] on
Usage: schematic
Placement: [✓ single] multiple
Signal Type: signal
Attach Net Expression: [✓ No] Yes
Property Name
Default Net Name
Font Height: 0.0625
Font Style: stick

(c)

Figure 7.5: Creating Pin Names

5. It is often advisable to add 'Pin' names to each of the IO terminals in a circuit. Thus, to add pins to your schematic press **P** from your keyboard or click on the pin symbol as shown in Figure 7.5 and make appropriate connections across all IO ports. Figure 7.5 demonstrates the steps involved in labeling wires with a circuit schematic.
   Note: The 'VDDA' and 'GNDA' pins should be chosen to be 'InputOutput' when selecting the 'Direction' during pin creation.

6. Finally your schematic should look like Figure 7.6. Now click on 'Check and Save' icon (as shown in Figure 7.7) in the toolbar so that you can move onto the next step of creating a symbol for the inverter schematic.

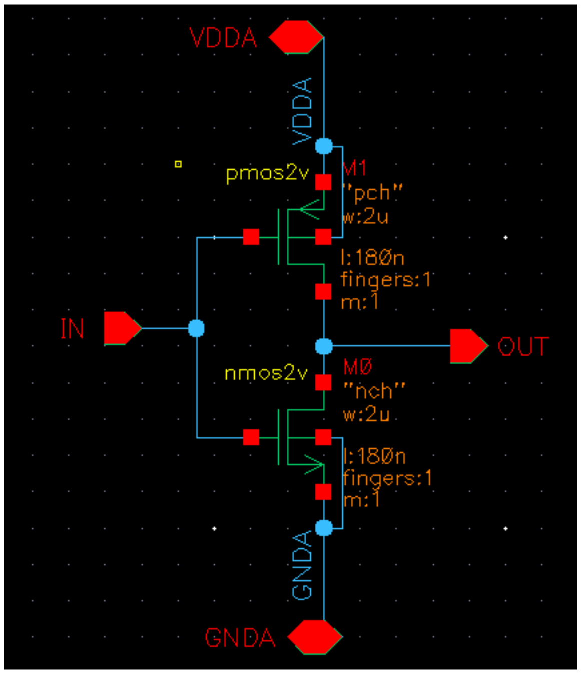


Figure 7.6: Inverter Schematic

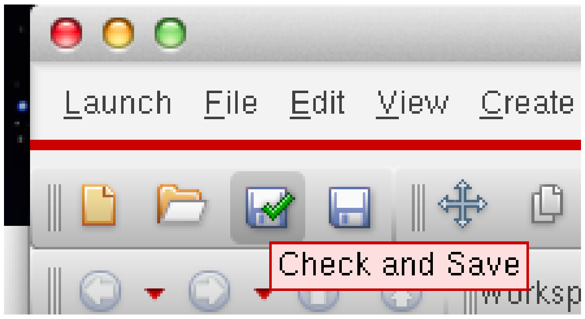
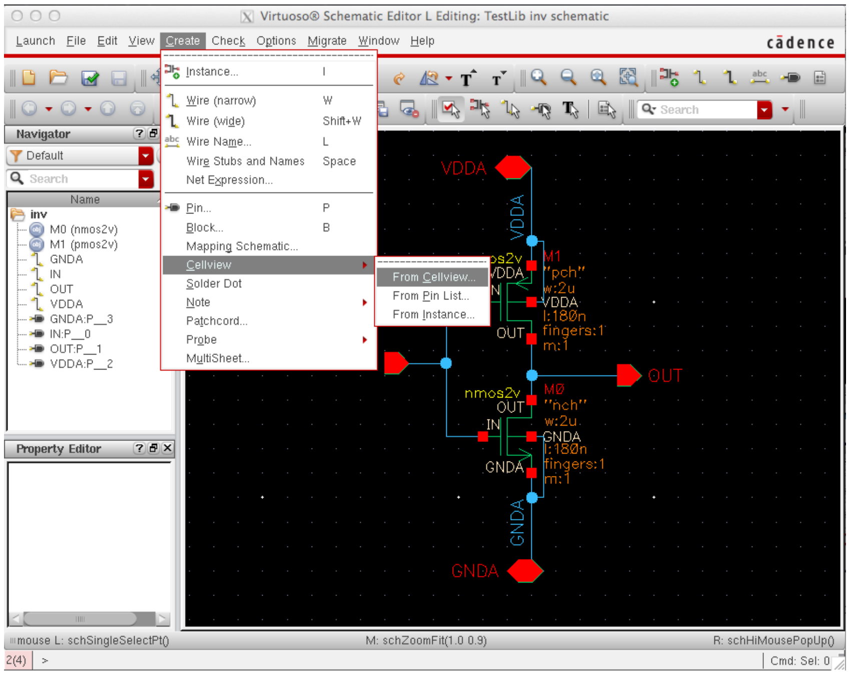


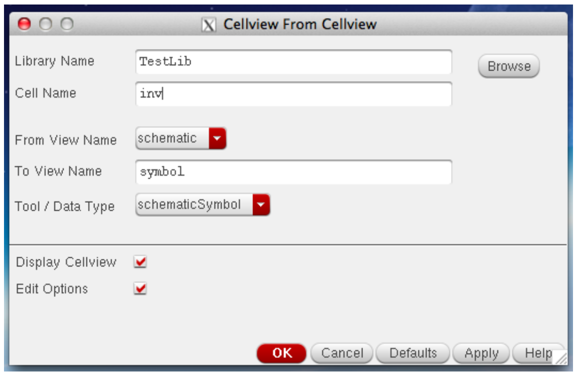
Figure 7.7: Check and Save

7.4.2 Creating a Symbol

1. When dealing with large circuits its often advisable to generate symbols for each sub-circuit in the design and perform all simulations by placing the corresponding symbols in a testbench. Figure 7.8 summarizes the steps involved in generating a symbol from the inverter schematic designed in the previous section.



(a)



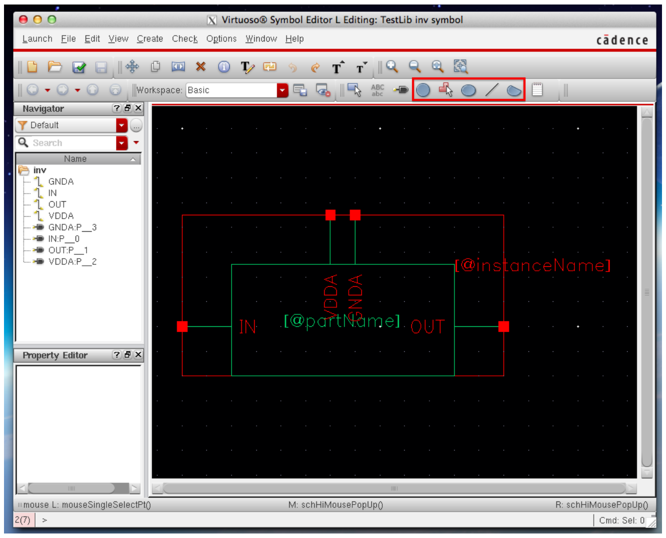
Library Name	TestLib
Cell Name	inv
From View Name	schematic
To View Name	symbol
Tool / Data Type	schematicSymbol
Display Cellview	✓
Edit Options	✓

(b)

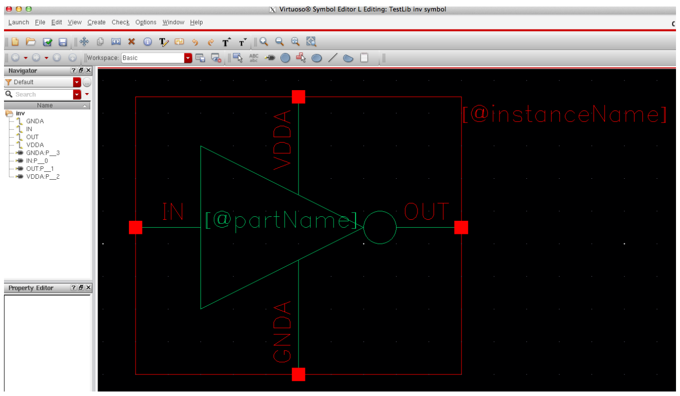
Figure 7.8: Generating Symbol from Schematic

2. Once you create the symbol it will pop up in a new window. By default Cadence will generate a rectangular symbol, but you can edit the

generated symbol as per your needs. In our case we will edit the symbol shape to make it resemble the traditional inverter symbol used in conventional system design (as shown in Figure 7.9). To edit the shape use the 'Edit Pallete' as shown in Figure 7.9(a) via a red highlighted box.



(a)



(b)

Figure 7.9: Designing Schematic Symbol

7.4.3 PFD, CP, Filter, VCO and Divider Schematics

1. Using the steps mentioned in the subsections above, create a new schematic as well as symbols for 'nand2', 'nand3' and 'nand4' circuits respectively. Make sure you change the symbol shapes for the *nand* circuits to the conventional symbols shapes.

2. Create a new schematic and save it as ‘pfd’. Place the ‘nand2’,‘nand3’,‘nand4’ and ‘inv’ symbols in the schematic and connect the four components in the NAND-PFD form as shown in Figure pfd.

3. Create a new schematic and save it as ‘bias_amp’. Now recreate the biasing op-amp that is part of the charge-pump schematic shown in Figure 5.3.

4. Create new schematic and save it as ‘cp’. Place the biasing amplifier created in the previous step and recreate the charge-pump circuit shown in Figure 5.3 from section 5.2.

5. Create a new schematic and save it as ‘filter’. Recreate the loop-filter circuit shown in Figure 5.4 from section 5.3.

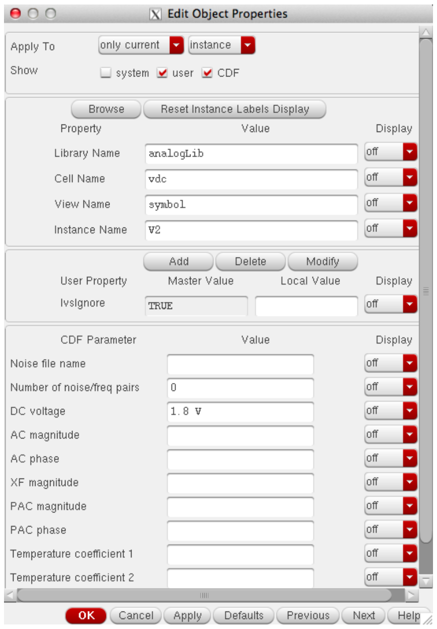
6. Create a new schematic and save it as ‘vco’. Recreate the VCO circuit shown in Figure 5.5 from section 5.4.

7. Create a new schematic and save it as ‘dff’. Now recreate the biasing D Flip-Flop that is part of the divider schematic shown in Figure 5.6. from section 5.5.

8. Create a new schematic and save it as ‘div’. Recreate the Divider circuit shown in Figure 5.7 from section 5.5.

7.4.4 Creating a Testbench

Create a new schematic following the steps outlined in the earlier sections and name it ‘Tb_pfd_cp_filter’. This will be the testbench schematic from which we will run all our simulations to test that the PFD, CP and the Filter are functioning as expected. Insert ‘vdc’, ‘gnd’ and two ‘vpulse’ from the Component Library by navigating to the ‘Analog Parts’ library. Figure 7.10 shows the initial conditions to be set for the voltage sources ‘vdc’, ‘vpulse for Vref’ and ‘vpulse for Vdiv’, respectively and Figure 7.11 shows what your testbench schematic should look like at the end of this step.



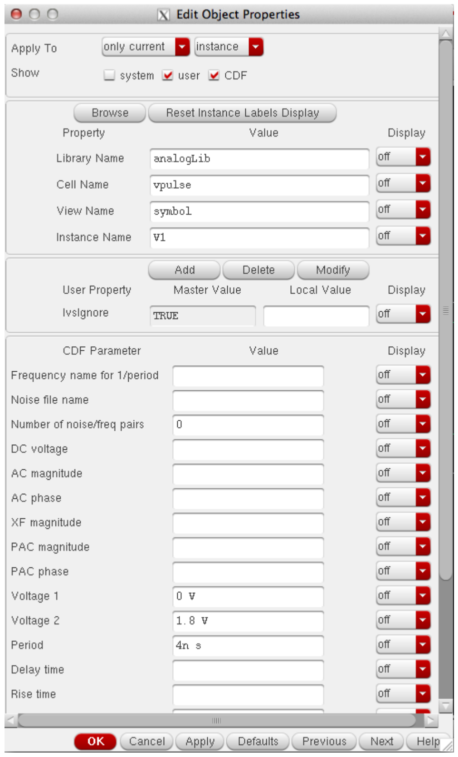
Apply To: only current, instance
Show: system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	analogLib	off
Cell Name	vdc	off
View Name	symbol	off
Instance Name	V2	off

User Property	Master Value	Local Value	Display
lvsIgnore	TRUE		off

CDF Parameter	Value	Display
Noise file name		off
Number of noise/freq pairs	0	off
DC voltage	1.8 V	off
AC magnitude		off
AC phase		off
XF magnitude		off
PAC magnitude		off
PAC phase		off
Temperature coefficient 1		off
Temperature coefficient 2		off

(a)



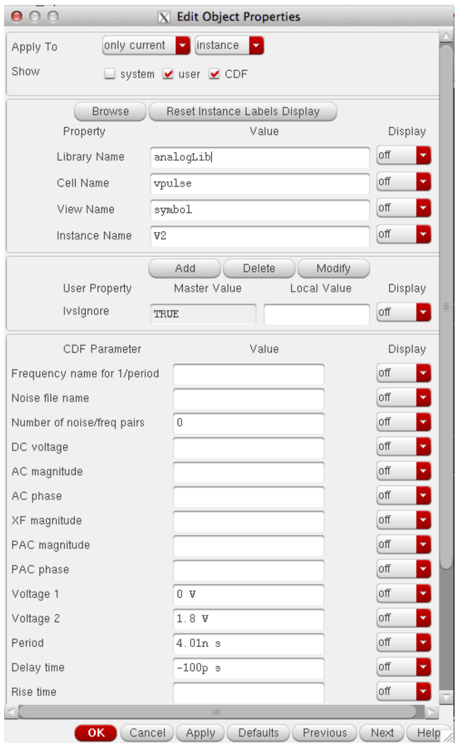
Apply To: only current, instance
Show: system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	analogLib	off
Cell Name	vpulse	off
View Name	symbol	off
Instance Name	V1	off

User Property	Master Value	Local Value	Display
lvsIgnore	TRUE		off

CDF Parameter	Value	Display
Frequency name for 1/period		off
Noise file name		off
Number of noise/freq pairs	0	off
DC voltage		off
AC magnitude		off
AC phase		off
XF magnitude		off
PAC magnitude		off
PAC phase		off
Voltage 1	0 V	off
Voltage 2	1.8 V	off
Period	4n s	off
Delay time		off
Rise time		off

(b)



Apply To: only current, instance
Show: system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	analogLib	off
Cell Name	vpulse	off
View Name	symbol	off
Instance Name	V2	off

User Property	Master Value	Local Value	Display
lvsIgnore	TRUE		off

CDF Parameter	Value	Display
Frequency name for 1/period		off
Noise file name		off
Number of noise/freq pairs	0	off
DC voltage		off
AC magnitude		off
AC phase		off
XF magnitude		off
PAC magnitude		off
PAC phase		off
Voltage 1	0 V	off
Voltage 2	1.8 V	off
Period	4.01n s	off
Delay time	-100p s	off
Rise time		off

(c)

Figure 7.10: Sources in PFD+CP+Filter Testbench

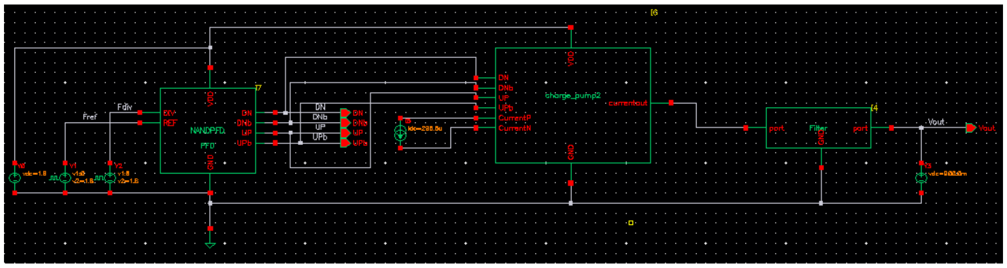


Figure 7.11: PFD+CP+Filter Testbench

Now create a new schematic and name it 'Tb_vco'. This will be the testbench schematic from which we will run all our simulations to test that the VCO is functioning as expected. Insert 'vdc', 'gnd' and two 'vdc' from the Component Library by navigating to the 'Analog Parts' library. The initial conditions to be set for VDD are same as shown in Figure 7.10(a), while for the second 'vdc' source the DC voltage should be set to a parametric variable 'vctrl'. Figure 7.12 shows what your testbench schematic should look like at the end of this step.

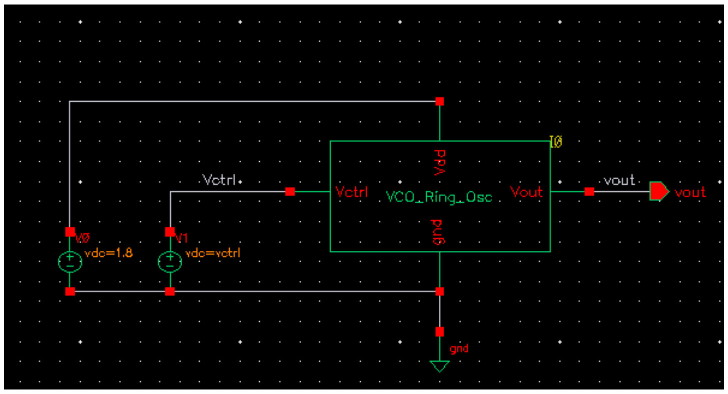
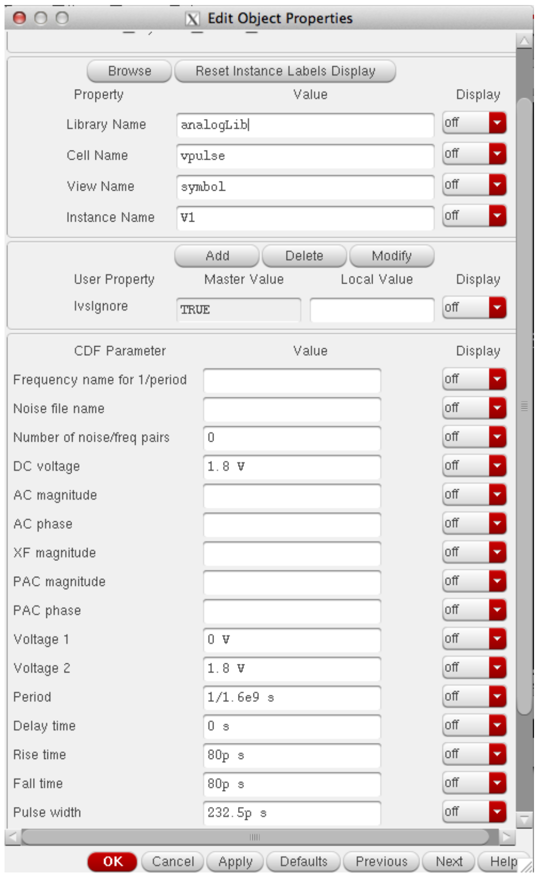


Figure 7.12: VCO Testbench

Finally, create a new schematic and name it 'Tb_div'. This will be the testbench schematic from which we will run all our simulations to test that the VCO is functioning as expected. Insert 'vdc', 'gnd' and a 'vpulse' from the Component Library by navigating to the 'Analog Parts' library. The initial conditions to be set for VDD are same as shown in Figure 7.10(a), while for the second 'vpulse' source they are shown in Figure 7.13. Figure 7.14 shows what your testbench schematic should look like at the end of this step.



Property	Value	Display
Library Name	analogLib	off
Cell Name	vpulse	off
View Name	symbol	off
Instance Name	V1	off

User Property	Master Value	Local Value	Display
lvsIgnore	TRUE		off

CDF Parameter	Value	Display
Frequency name for 1/period		off
Noise file name		off
Number of noise/freq pairs	0	off
DC voltage	1.8 V	off
AC magnitude		off
AC phase		off
XF magnitude		off
PAC magnitude		off
PAC phase		off
Voltage 1	0 V	off
Voltage 2	1.8 V	off
Period	1/1.6e9 s	off
Delay time	0 s	off
Rise time	80p s	off
Fall time	80p s	off
Pulse width	232.5p s	off

Figure 7.13: Vpulse configuration for Divider

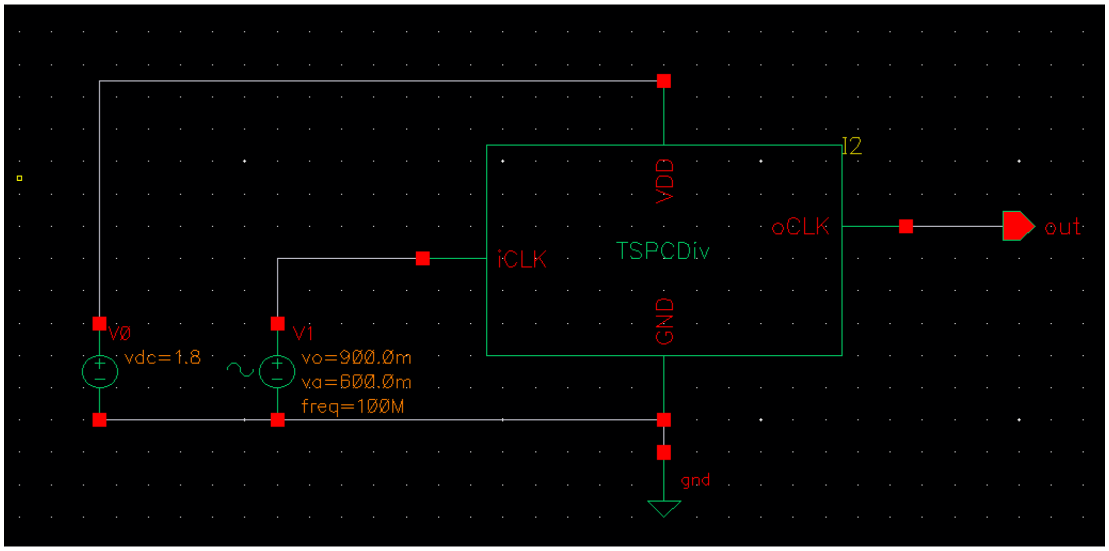


Figure 7.14: Divider Testbench

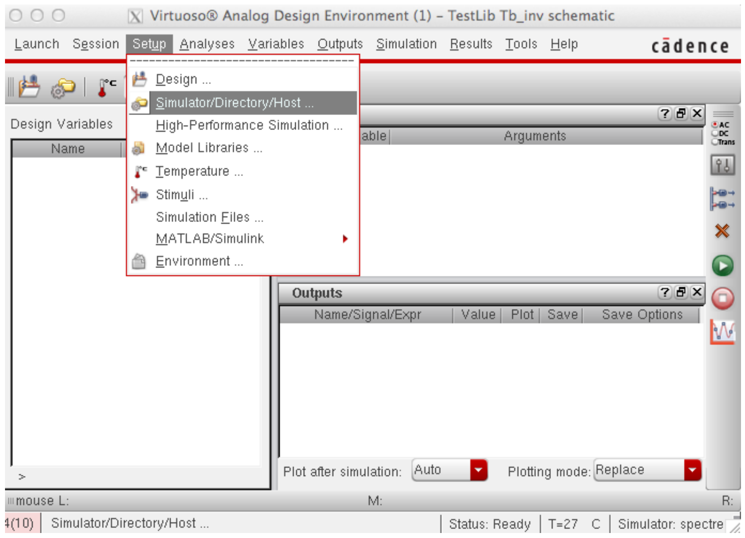
7.4.5 Circuit Simulation Using Spectre

7.4.6 Launching ADE

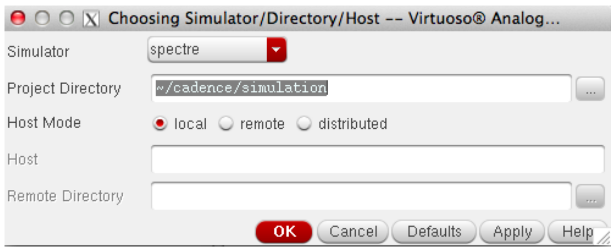
1. We will simulate our circuits using Cadence Spectre Simulation engine. Spectre is a variant of HSPICE developed by Cadence and provides greater accuracy, speed and flexibility especially when dealing with mixed signal circuits.

2. Make sure you first 'Check and Save' your testbench schematic and click on *Launch* → *ADE* to open up the ADE window as shown in Figure 7.15.

3. Click on *Setup* → *Simulator* to make sure the Simulator is set to Spectre as shown in Figure 7.15.



(a)


Choosing Simulator/Directory/Host -- Virtuoso® Analog...

Simulator: spectre

Project Directory: ~/cadence/simulation

Host Mode: [✓ local] remote distributed

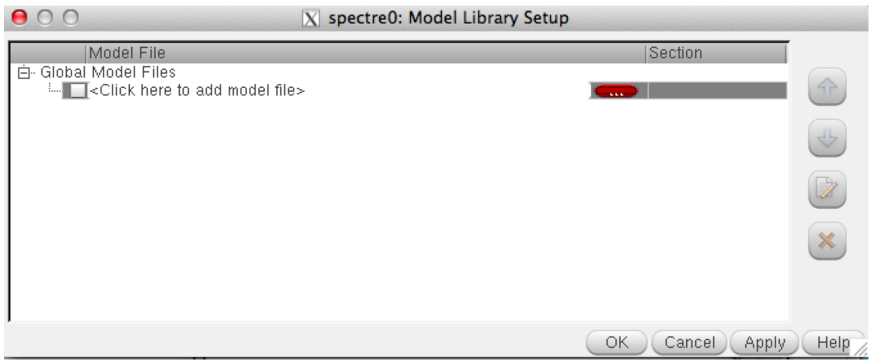
Host:

Remote Directory:

(b)

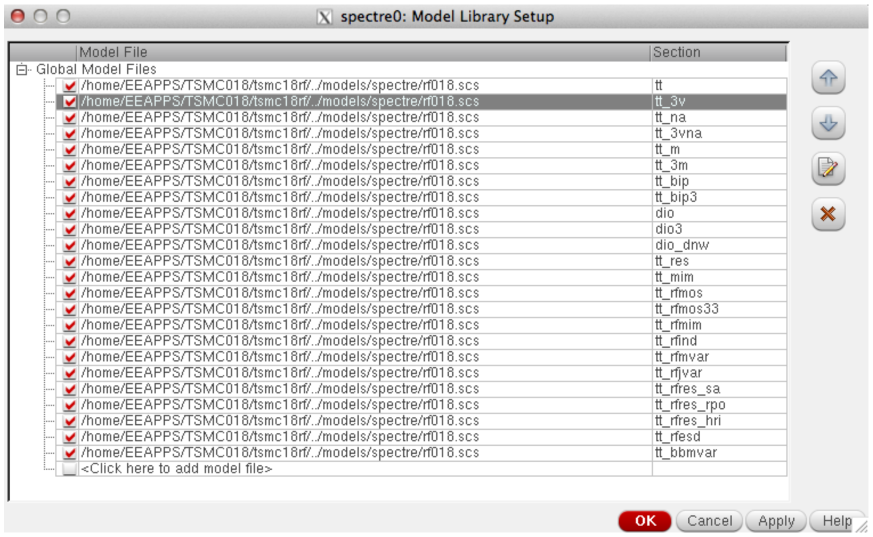
Figure 7.15: Simulating Circuit with ADE

4. Now click on *Setup* → *Model Libraries* to configure the Spectre model files. Figure 7.16 shows the path you need to browse to in order to get the correct model files for the PDK. You most likely would not need to manually type the model file paths as Virtuoso should take care of it, but in case you do the path is listed.



Model File	Section
Global Model Files	
<Click here to add model file>	

(a)



Model File	Section
Global Model Files	
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_3v
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_na
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_3vna
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_m
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_3m
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_bip
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_bip3
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	dio
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	dio3
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	dio_dnw
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_res
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_mim
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfmos
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfmos33
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfmim
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfind
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfmvar
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfjvar
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfres_sa
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfres_rpo
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfres_hri
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_rfesd
/home/EEAPPS/TSMC018/tsmc18rf/../models/spectre/rf018.scs	tt_bbmvar
<Click here to add model file>	

(b)

Figure 7.16: Configuring Model Files

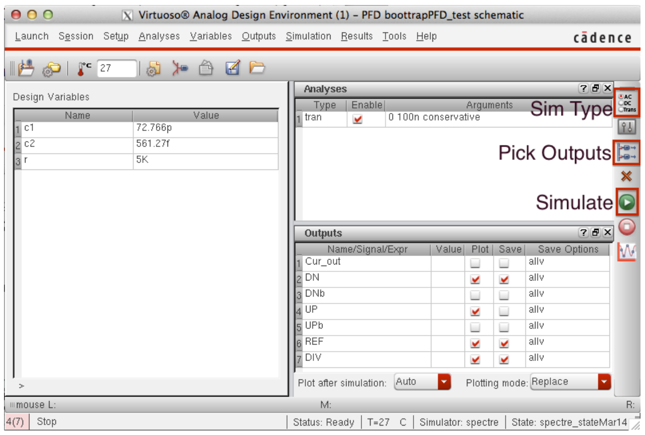
7.4.7 PFD+CP+Filter ADE Setup

1. Transient analysis of any circuit is key to study the time domain behavior. We will simulate the PFD+CP+Filter testbench and observe the resulting plots for 'UP', 'DN', signals to ensure that the PFD is functioning properly.

2. In ADE window click on the **AC,DC,Tran** icon on the right pane. Choose the 'tran' simulation type, pick the stop time to be 100ns and choose 'moderate' in the 'Accuracy details'.

3. Click on *Variables* → *Copy From Cellview* and insert the filter parameters as shown in Figure 7.17.

4. Click on the green 'Play' button to run the simulation and the plots should automatically pop up in a new output window. If you right click on the name of the signal listed in the left panel, you can navigate to options that change the thickness and color of the output waveform. Additionally, right-clicking anywhere on the output window and navigating to 'Graph Properties' allows you to alter the background color as well.


Virtuoso® Analog Design Environment (1) – PFD boottrapPFD_test schematic
Launch Session Setup Analyses Variables Outputs Simulation Results Tools Help
cādence
Sim Type
Pick Outputs
Simulate
mouse L:
M:
R:
4(7) Stop
Status: Ready | T=27 C | Simulator: spectre | State: spectre_stateMar14

Design Variables

	Name	Value
1	c1	72.766p
2	c2	561.27f
3	r	5K

Analyses

	Type	Enable	Arguments
1	tran	✓	0 100n conservative

Outputs

	Name/Signal/Expr	Value	Plot	Save	Save Options
1	Cur_out				allv
2	DN		✓	✓	allv
3	DNb				allv
4	UP		✓		allv
5	UPb				allv
6	REF		✓	✓	allv
7	DIV		✓	✓	allv

Plot after simulation: Auto   Plotting mode: Replace

Figure 7.17: PFD+CP+Loop-Filter Testbench ADE Window

5. Your final output waveform should look like that shown in Figure 7.18. Notice that the UP,DN pulses are appropriately modulated as 'REF' and 'DIV' signals diverge from one another; thus, the PFD is indeed functioning correctly.

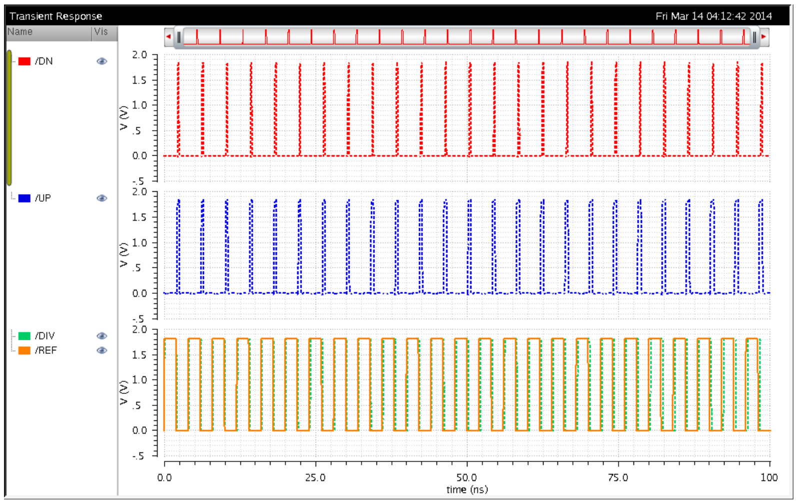


Figure 7.18: PFD Spectre Simulation Output

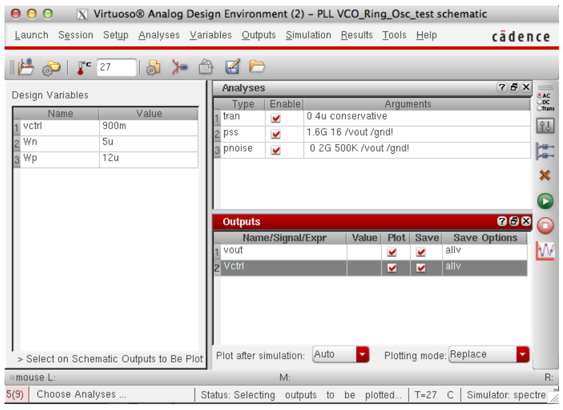
7.4.8 VCO ADE Setup

Recall from the earlier sections that for oscillators it is critical to perform Transient as well as PSS simulations. We will therefore simulate the VCO testbench and observe the resulting transient and PSS simulation outputs for 'Vout' signals to ensure that the VCO is functioning properly. Additionally, since the VCO is the major noise contributor to the PLL, we will also characterize the VCO Phase-Noise by performing PNoise simulation on the testbench.

1. In ADE window click on the **AC,DC,Tran** icon on the right pane. Choose the 'tran' simulation type, pick the stop time to be $4\mu s$ and choose 'moderate' in the 'Accuracy details'.

2. Click on $Variables \rightarrow Copy\ From\ Cellview$ and insert the PMOS, NMOS width and 'vctrl' parameters as shown in Figure 7.19.

3. Click on the green 'Play' button to run the simulation, and the plots should automatically pop up in a new output window. If you right-click on the name of the signal listed in the left panel, you can navigate to options that change the thickness and color of the output waveform. Additionally, right-clicking anywhere on the output window and navigating to 'Graph Properties' allows you to alter the background color as well.


Virtuoso® Analog Design Environment (2) – PLL VCO_Ring_Osc_test schematic
Launch Session Setup Analyses Variables Outputs Simulation Results Tools Help
cādence
27

Design Variables

	Name	Value
1	vctrl	900m
2	Wn	5u
3	Wp	12u

> Select on Schematic Outputs to Be Plot

Analyses

	Type	Enable	Arguments
1	tran	✓	0 4u conservative
2	pss	✓	1.6G 16 /vout /gnd!
3	pnoise	✓	0 2G 500K /vout /gnd!

Outputs

	Name/Signal/Expr	Value	Plot	Save	Save Options
1	vout		✓	✓	allv
2	Vctrl		✓	✓	allv

Plot after simulation: Auto   Plotting mode: Replace

mouse L: M: R:

5(9)   Choose Analyses ...   Status: Selecting outputs to be plotted...   T=27 C   Simulator: spectre

Figure 7.19: VCO Testbench ADE Window

4. Your final output waveform should look like that shown in Figure 7.20. Notice that the output node voltage 'vout' is oscillating thus the VCO is indeed functioning correctly.

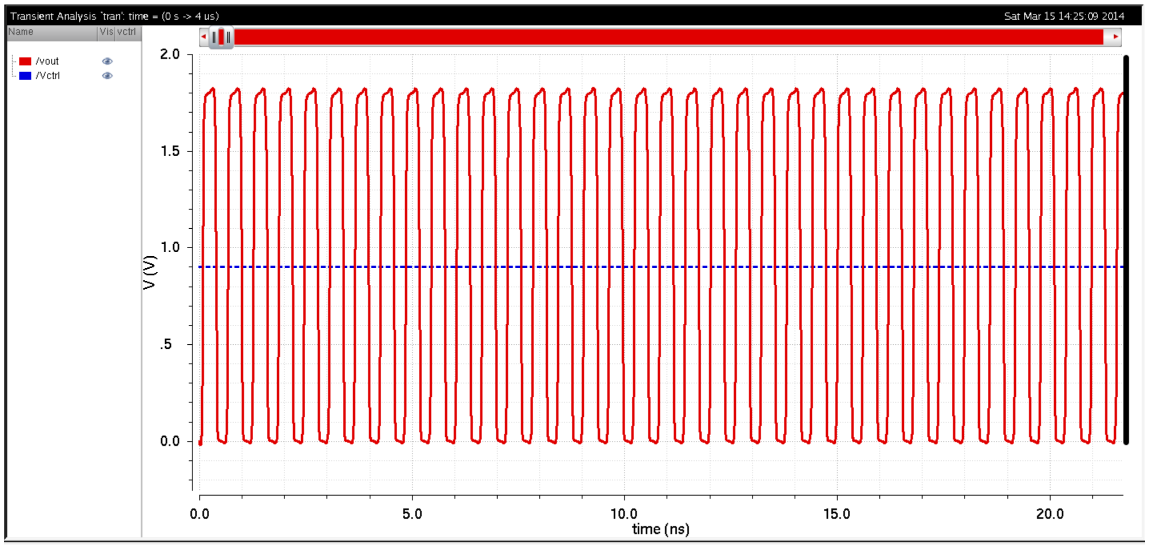
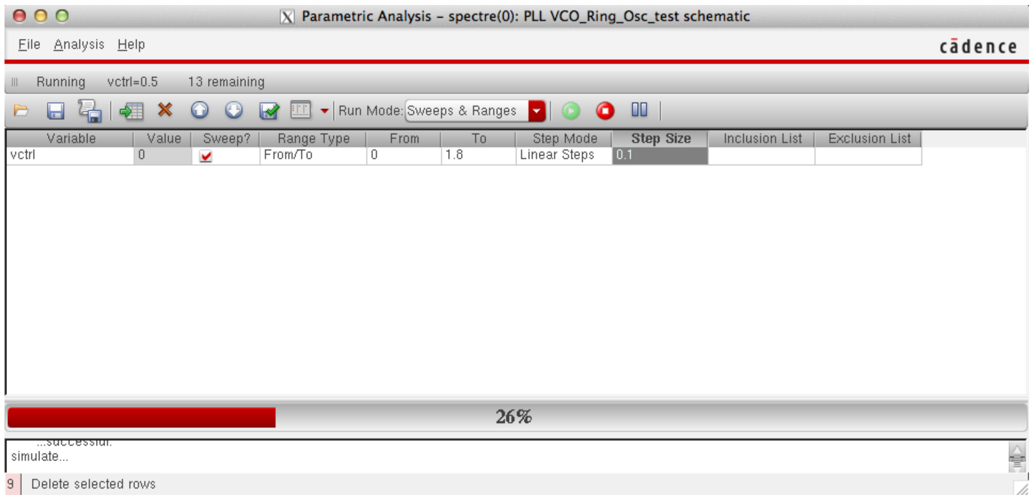


Figure 7.20: VCO Transient Simulation Output

5. For a VCO, a key figure-of-merit is the control voltage tuning range. Thus, we have to perform a parametric analysis in order to observe the change in 'frequency' as well as K_{VCO} as a function of 'Vctrl'. In order to do so in the ADE setup window click on *Tools* → *Parametric Analysis* and a window like Figure 7.21 should pop-up. Within the parametric analysis window, when you double-click on the variable box, a drop-down list will show up from which you should pick 'vctrl'.



Parametric Analysis – spectre(0): PLL VCO_Ring_Osc_test schematic

Running vctrl=0.5 13 remaining

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Step Size	Inclusion List	Exclusion List
vctrl	0	✓	From/To	0	1.8	Linear Steps	0.1		

26%

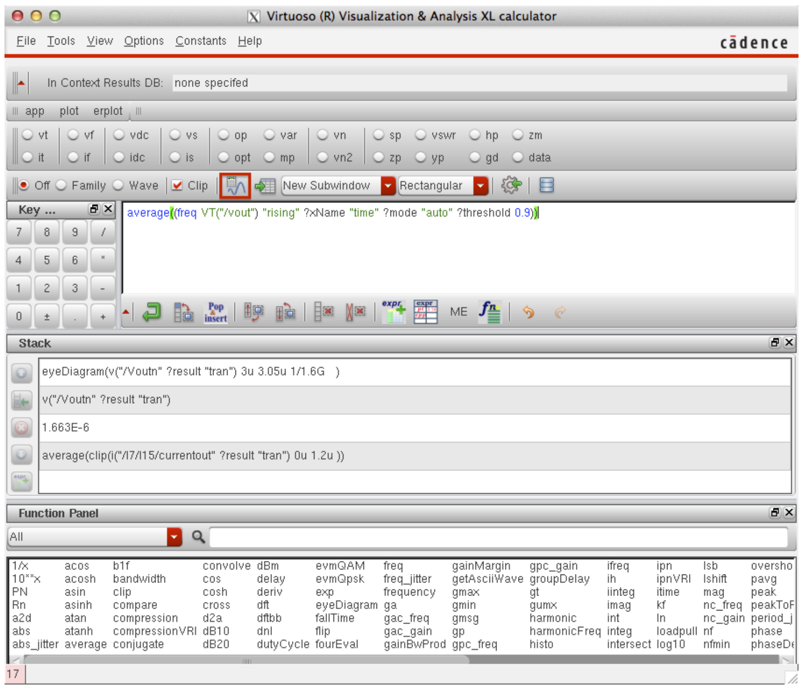
```
...successful.
simulate...
```

9 Delete selected rows

Figure 7.21: Vctrl Parametric Analysis Setup

To run the parametric analysis, click on the 'Play' within the Parametric-Analysis window. This setup is basically going to run the transient simulation $\frac{To-From}{StepSize}$ times by varying the control-voltage input to the VCO.

6. To plot frequency vs. Vctrl and K_{VCO} vs. Vctrl we need to use the 'Calculator' tool in-built within ADE. Click on *Tools* $\rightarrow$ *Calculator*. The Calculator window as shown in Figure 7.22 will open up and within it now you should select 'Vt' from the toolbar. The schematic will open up, so within the schematic select the 'vout' node. From the 'Function-Panel' within the Calculator window choose the 'frequency' and 'average' functions to make up the function shown in Figure 7.22. Now go back to the ADE window, click on the right-pane and select the 'Pick-Outputs' button. A window will pop up so within it select 'Get-Expression' and name it 'freq'. This will bring the expression you just created in the Calculator so that you can plot it. Conversely, you can also click on the 'plot' button shown in the red-box in Figure 7.22 to plot the expression; however, doing so makes the title of plot look a little too crammed.


Virtuoso (R) Visualization & Analysis XL calculator
File Tools View Options Constants Help
cādence
In Context Results DB: none specifed
app plot erplot
vt vf vdc vs op var vn sp vswr hp zm
it if idc is opt mp vn2 zp yp gd data
Off Family Wave Clip
New Subwindow
Rectangular
Key ...
Stack
Function Panel
All
17

```
average((freq VT("/vout") "rising" ?xName "time" ?mode "auto" ?threshold 0.9))
```

```
eyeDiagram(v("/Voutn" ?result "tran") 3u 3.05u 1/1.6G  )
v("/Voutn" ?result "tran")
1.663E-6
average(clip(i("/I7/I15/currentout" ?result "tran") 0u 1.2u ))
```

Figure 7.22: ADE Calculator

7. Repeat the same steps as above to create an expression within the calculator to compute the K_{VCO}. Use the 'deriv' function within the Calculator Function Panel to do so. Finally, click on the 'Play' button within the ADE window to plot frequency vs. Vctrl and K_{VCO} vs. Vctrl curves. Your output should look like Figure 7.23.

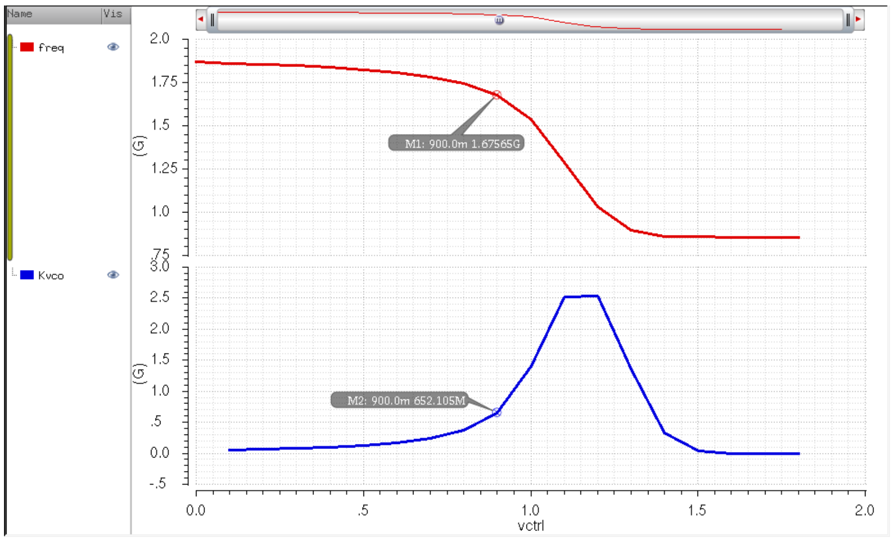
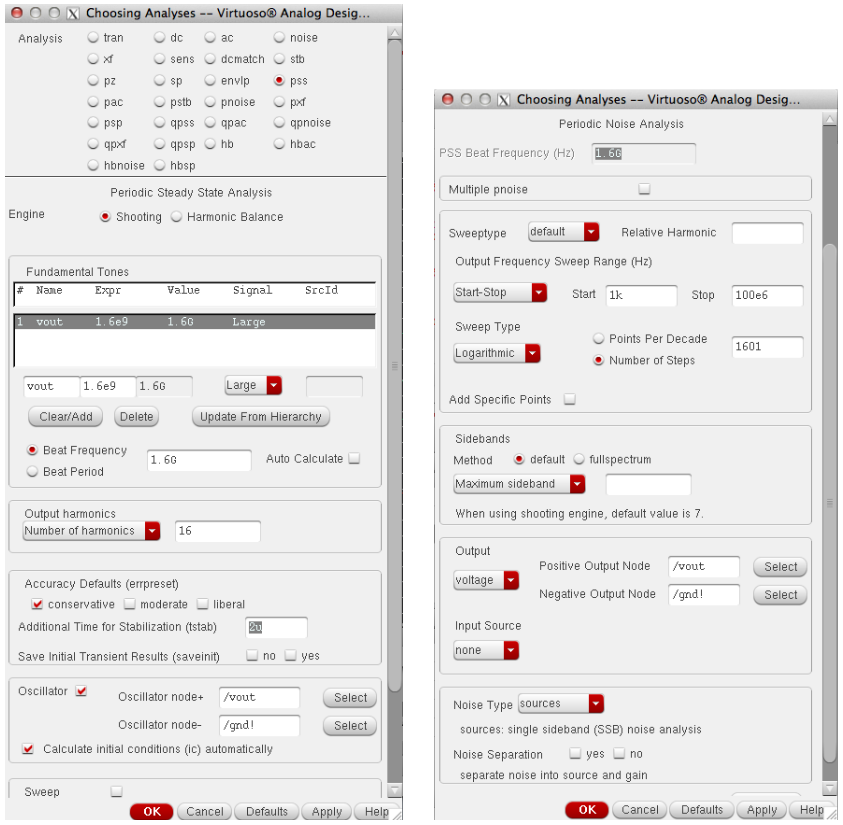


Figure 7.23: Frequency vs. Vctrl and K_{VCO} vs. Vctrl Simulation Plots

The K_{VCO} calculated in Figure 7.23 is the value used to calculate the required charge-pump current as well as C_1 and C_2 values from the loop-filter. In our case, the VCO output frequency is 1.675GHz with a Vctrl=0.9V and a K_{VCO} = 652.106MHz/V.

8. Now, in order to simulate the VCO Phase-Noise we need to perform the PSS, PNoise simulations. For the PSS simulation, in the ADE window click on the **AC,DC,Tran** icon on the right pane. Choose the 'pss' simulation type, pick the parameters using those shown in Figure 7.24(a). It is critical to note that the beat frequency here is the target frequency of the VCO and the reason we have to check the 'oscillator' option and select 'vout', 'gnd' terminals from the schematic is because by default PSS simulation expects a differential output.

9. To run the PNoise simulation, in the ADE window click on the **AC,DC**, **Tran** icon on the right pane. Choose the 'pss' simulation type, pick the parameters using those shown in Figure 7.24(b). Its critical to note that the phase-noise in the VCO is only dominant in the low-pass thus we limit our simulation frequency range to be from 1kHz to 10MHz as after the 10MHz the phase-noise will not cause any significant degradation to oscillator output performance. Note: To view the PNoise simulation results in the main ADE window click on *Results* → *Direct Plot* → *Main Form* at which a window like Figure 7.25. Choose 'Phase-Noise' and click on the 'Plot' button.



Choosing Analyses -- Virtuoso® Analog Desig...

Analysis: tran, dc, ac, noise, xf, sens, dcmatch, stb, pz, sp, envlp, [✓ pss], pac, pstb, pnoise, pxf, psp, qpss, qpac, qpnoise, qpxf, qpsp, hb, hbac, hbnoise, hbsp

Periodic Steady State Analysis

Engine: [✓ Shooting] Harmonic Balance

Fundamental Tones

#	Name	Expr	Value	Signal	SrcId
1	vout	1.6e9	1.6G	Large	

vout 1.6e9 1.6G Large

Clear/Add Delete Update From Hierarchy

[✓ Beat Frequency] Beat Period: 1.6G Auto Calculate ☐

Output harmonics: Number of harmonics 16

Accuracy Defaults (errpreset): [✓ conservative] moderate liberal

Additional Time for Stabilization (tstab): 2u

Save Initial Transient Results (saveinit): ☐ no ☐ yes

Oscillator ✓ Oscillator node+ /vout Select Oscillator node- /gnd! Select

✓ Calculate initial conditions (ic) automatically

Sweep ☐

OK Cancel Defaults Apply Help

Choosing Analyses -- Virtuoso® Analog Desig...

Periodic Noise Analysis

PSS Beat Frequency (Hz): 1.6G

Multiple pnoise ☐

Sweeptype: default Relative Harmonic

Output Frequency Sweep Range (Hz): Start-Stop Start 1k Stop 100e6

Sweep Type: Logarithmic Points Per Decade [✓ Number of Steps] 1601

Add Specific Points ☐

Sidebands Method: [✓ default] fullspectrum

Maximum sideband

When using shooting engine, default value is 7.

Output: voltage Positive Output Node /vout Select Negative Output Node /gnd! Select

Input Source: none

Noise Type: sources

sources: single sideband (SSB) noise analysis

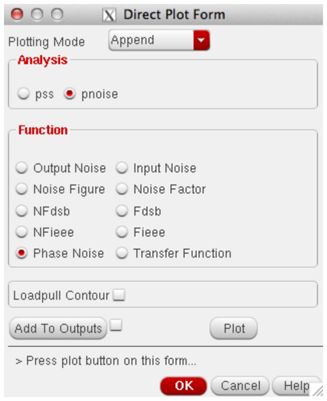
Noise Separation ☐ yes ☐ no

separate noise into source and gain

OK Cancel Defaults Apply Help

(a) (b)

Figure 7.24: VCO PSS & Pnoise Simulation Setup



Direct Plot Form

Plotting Mode: Append

Analysis: pss [✓ pnoise]

Function: Output Noise, Input Noise, Noise Figure, Noise Factor, NFdsb, Fdsb, NFieee, Fieee, [✓ Phase Noise], Transfer Function

Loadpull Contour ☐

Add To Outputs ☐ Plot

> Press plot button on this form...

OK Cancel Help

Figure 7.25: VCO Phase-Noise Simulation Plot Step

The output waveform for the simulated phase-noise will look like Figure 7.26. In our case, we find that the Phase-Noise at a 1MHz offset is equal to -94.32dBc/Hz, which is very reasonable for a ring-oscillator type single-ended VCO topology.

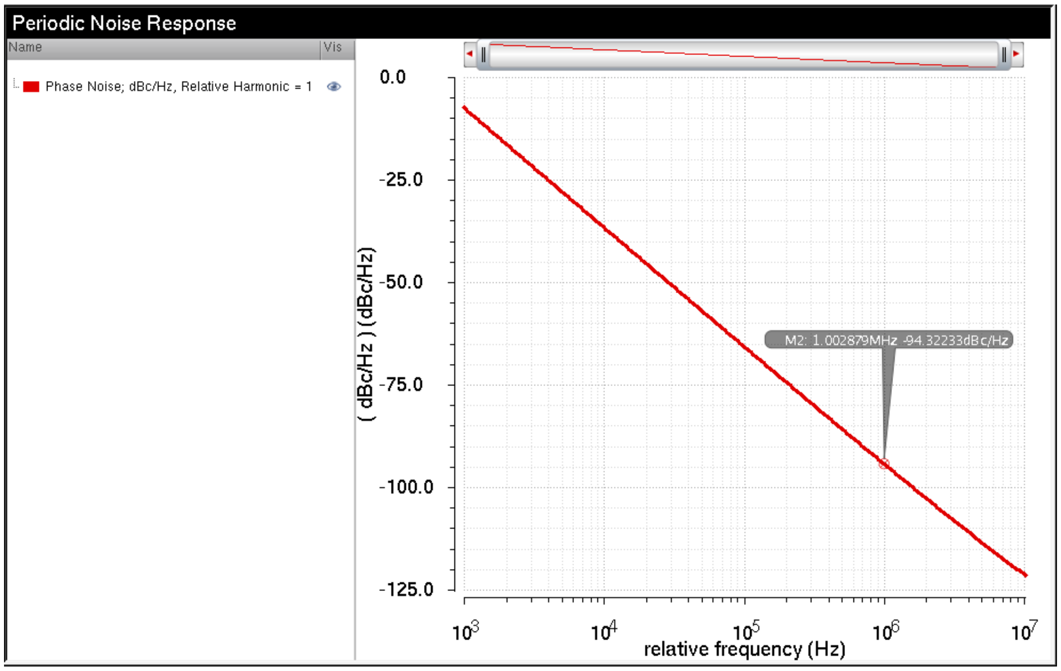


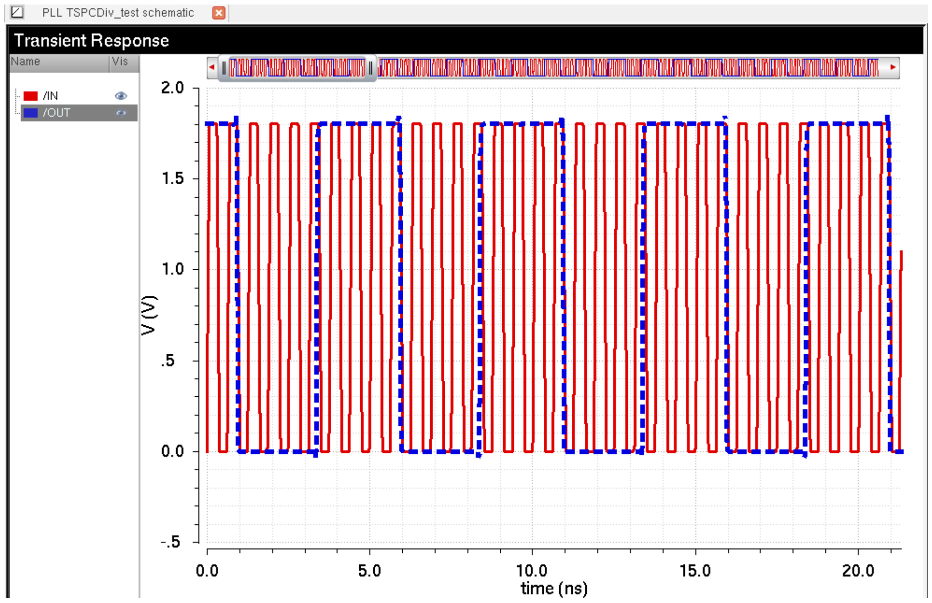
Figure 7.26: VCO Phase-Noise Simulation Plot

7.4.9 Divider ADE Setup

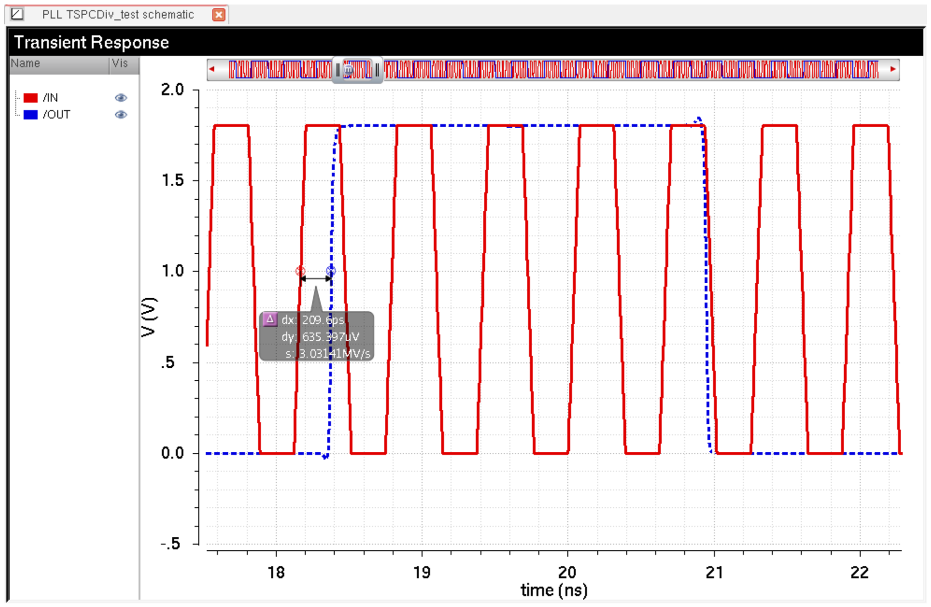
1. We now simulate the divider testbench and observe the resulting plots for 'IN', 'OUT', signals of the divider circuit to ensure its proper functionality.

2. In ADE window click on the **AC,DC,Tran** icon on the right pane. Choose the 'tran' simulation type, pick the stop time to be 100ns and choose 'moderate' in the 'Accuracy details'.

3. Click on the green 'Play' button to run the simulation and the plots should automatically pop up in a new output window.



(a)



(b)

Figure 7.27: Divider Spectre Simulation Output

4. Your final output waveform should resemble Figure 7.27. Notice that each half-wave of the output pulse comprises of four half-pulses of the input, meaning the period of the output pulse is one-eighth of the input pulse period. Thus, our divider is functioning properly in that it divides the input pulse frequency by 8 with a small setup-time delay of 209ps.

7.4.10 Complete PLL Schematic and Testbench

1. Using the steps mentioned in the subsections above create new schematic and save it as 'PLL'.

2. Place the 'pfd', 'vco' and 'div' symbols in the schematic, connect the components in together and generate a symbol for the full PLL schematic as shown in Figure 7.28.

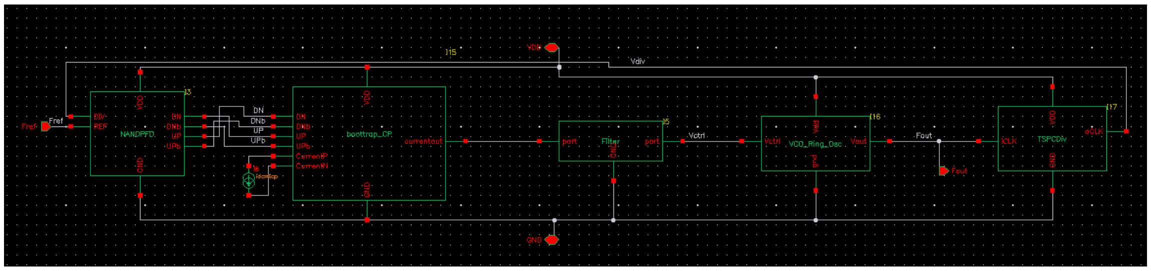


Figure 7.28: PLL Schematic

3. Create a new schematic and save it as 'Tb_PLL'. Design the testbench schematic as shown in Figure 7.29.

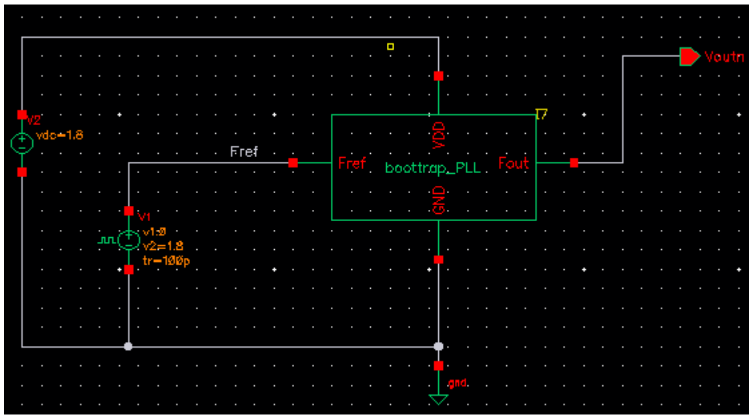
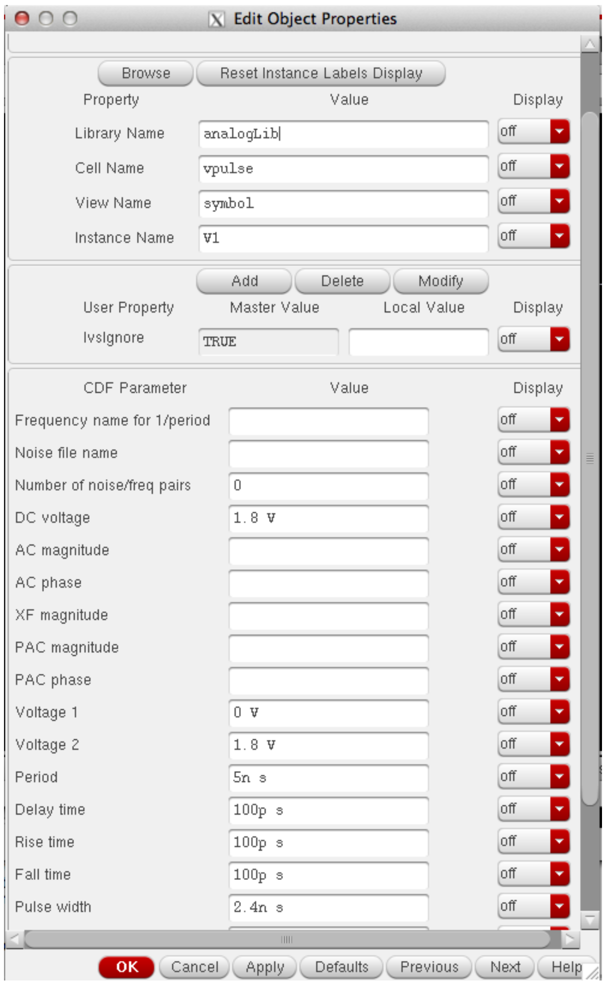


Figure 7.29: PLL Testbench

4. In the PLL testbench choose the 'Vref' using the 'Vpulse' source within analogLib and configure it as shown below in Figure 7.30.



Property	Value	Display
Library Name	analogLib	off
Cell Name	vpulse	off
View Name	symbol	off
Instance Name	V1	off

User Property	Master Value	Local Value	Display
lvsIgnore	TRUE		off

CDF Parameter	Value	Display
Frequency name for 1/period		off
Noise file name		off
Number of noise/freq pairs	0	off
DC voltage	1.8 V	off
AC magnitude		off
AC phase		off
XF magnitude		off
PAC magnitude		off
PAC phase		off
Voltage 1	0 V	off
Voltage 2	1.8 V	off
Period	5n s	off
Delay time	100p s	off
Rise time	100p s	off
Fall time	100p s	off
Pulse width	2.4n s	off

Figure 7.30: Vpulse Configuration for PLL Testbench

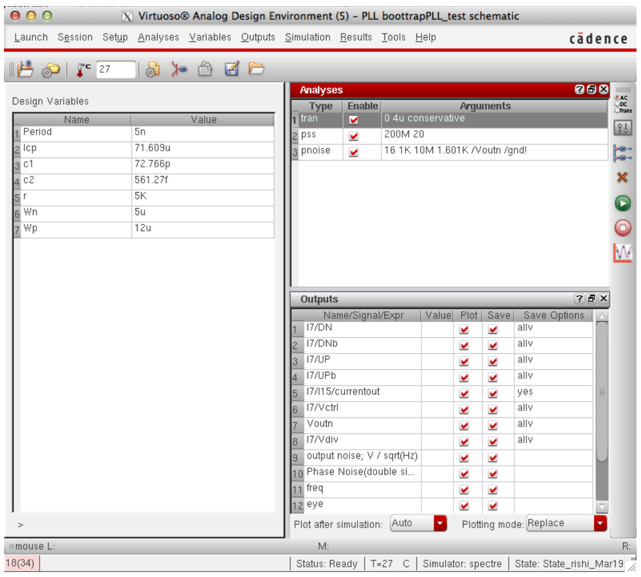
7.4.11 PLL ADE Setup

Now that we have verified the functionality of each of the components of the PLL at a transistor level, the final task is to characterize the PLL locking behavior, overall phase-noise profile and jitter profile.

1. In ADE window click on the **AC,DC,Tran** icon on the right pane. Choose the 'tran' simulation type, pick the stop time to be $4\mu s$ and choose 'moderate' in the 'Accuracy details'.

2. Click on *Variables* $\rightarrow$ *Copy From Cellview* and insert the PMOS, NMOS widths of VCO inverters, charge-pump output current, I_{CP}, reference signal period, and the loop-filter parameters as shown in Figure 7.31.

3. Click on the green 'Play' button to run the simulation and the plots should automatically pop-up in a new output window.


Virtuoso® Analog Design Environment (5) – PLL boottrapPLL_test schematic
Launch Session Setup Analyses Variables Outputs Simulation Results Tools Help
cādence
27
AC
DC
Trans

Design Variables

	Name	Value
1	Period	5n
2	Icp	71.609u
3	c1	72.766p
4	c2	561.27f
5	r	5K
6	Wn	5u
7	Wp	12u

Analyses

	Type	Enable	Arguments
1	tran	✓	0 4u conservative
2	pss	✓	200M 20
3	pnoise	✓	16 1K 10M 1.601K /Voutn /gnd!

Outputs

	Name/Signal/Expr	Value	Plot	Save	Save Options
1	I7/DN		✓	✓	allv
2	I7/DNb		✓	✓	allv
3	I7/UP		✓	✓	allv
4	I7/UPb		✓	✓	allv
5	I7/I15/currentout		✓	✓	yes
6	I7/Vctrl		✓	✓	allv
7	Voutn		✓	✓	allv
8	I7/Vdiv		✓	✓	allv
9	output noise; V / sqrt(Hz)		✓	✓	
10	Phase Noise(double si...		✓	✓	
11	freq		✓	✓	
12	eye		✓	✓	

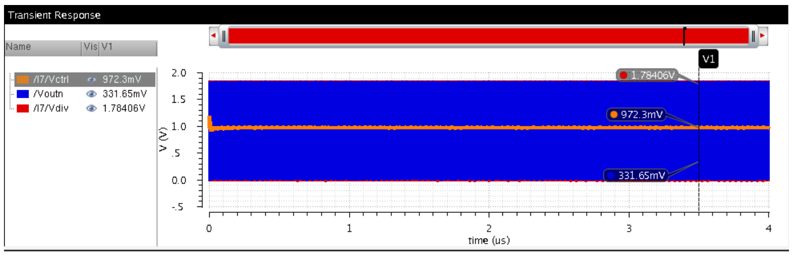
Plot after simulation: Auto Plotting mode: Replace

mouse L: M: R:

18(34) | Status: Ready | T=27 C | Simulator: spectre | State: State_rishi_Mar19

Figure 7.31: PLL Testbench ADE Window

4. Your final output waveform should look like that shown in Figure 7.32. Notice that the VCO input control voltage 'vctrl' is essentially flat and settled thus the PLL is in steady-state lock state.



Name	Vis	V1
🟧 /I7/Vctrl		972.3mV
🟦 /Voutn		331.65mV
🟥 /I7/Vdiv		1.78406V

Figure 7.32: PLL Settled Transient Simulation Output

If we zoom into a 50ns window we notice that there is a slight control voltage ripple, but the loop is approaching steady state lock point. From Figure 7.33 the rippling behavior of 'vctrl' can be seen to be prominent for the first 10ns and then slowly decaying away as we approach 50ns time-frame.

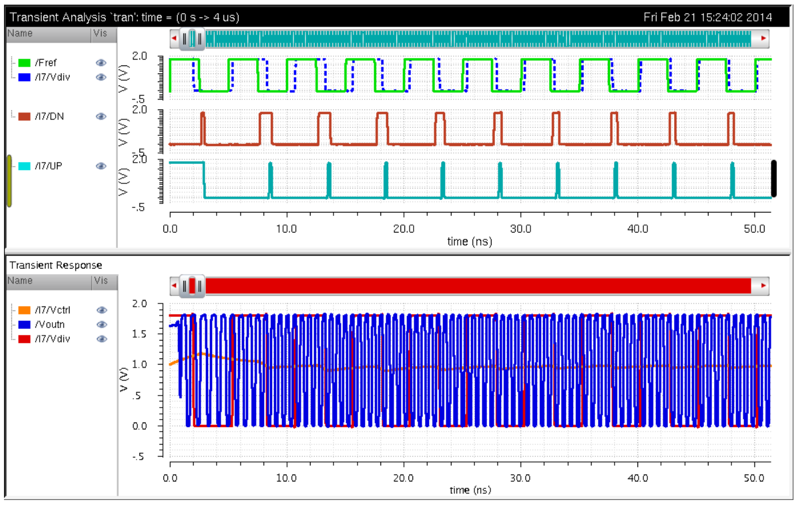


Figure 7.33: PLL Vctrl Voltage before Lock

5. Another method to verify that PLL is in steady-state locked condition is to plot the output frequency versus time. Figure 7.34 shows that the PLL achieves lock around $1\mu s$ and remains locked to 1.6GHz output frequency after that.

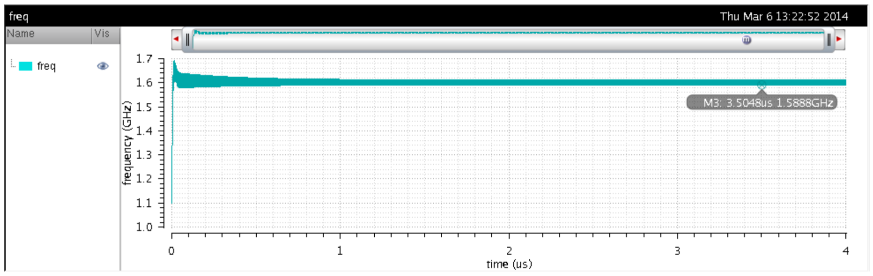
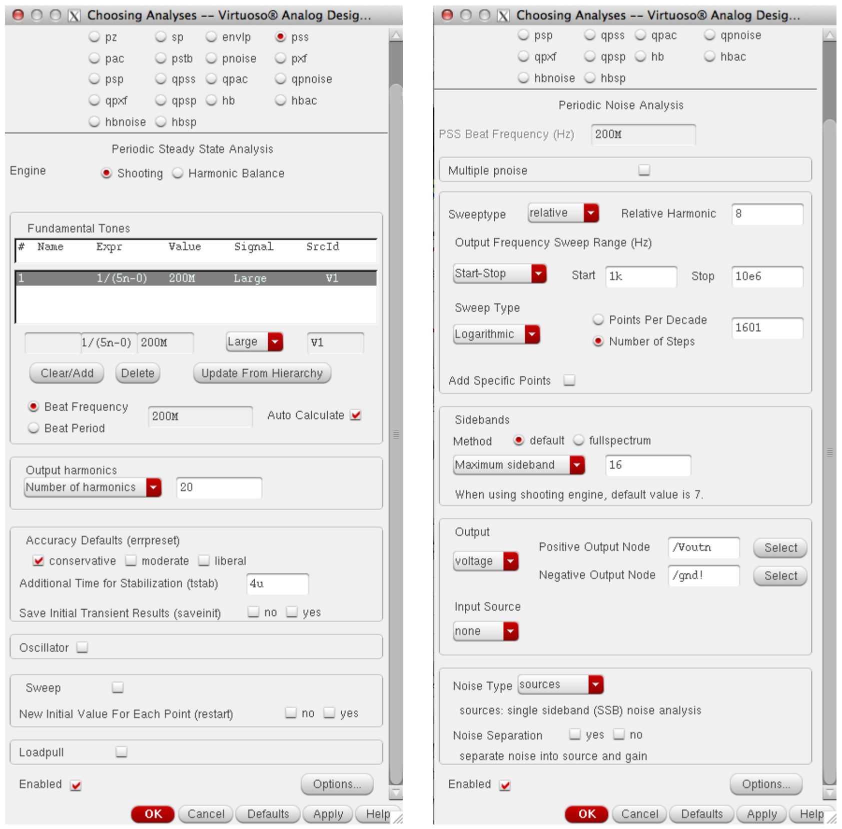


Figure 7.34: PLL Frequency Locking in Steady-State

To plot frequency we need to use the 'Calculator' tool in-built within ADE. Click on $Tools \rightarrow Calculator$. The Calculator window as shown in Figure 7.22 will open up and within it now you should select 'Vt' from the toolbar. The schematic will open up, so within the schematic select the 'vout' node. From the 'Function-Panel' within the Calculator window choose the 'freq' function to plot the PLL output frequency with respect to time. Once again, like in the case of the VCO frequency vs. vctrl plot, go back to the ADE window, click on the right-pane and select the 'Pick-Outputs' button. A window will pop-up so within it select 'Get-Expression' and name it 'freq'. This will bring the expression you just created in the Calculator so that you can plot it.

6. Now, in order to simulate the PLL Phase-Noise we perform the PSS, PNoise simulations. For the PSS simulation, in the ADE window click on the **AC,DC,Tran** icon on the right pane. Choose the ‘pss’ simulation type and pick the parameters using those shown in Figure 7.35(a). Note that in this case the beat frequency will be the reference frequency as that is the only fundamental input frequency to the PLL. Additionally, the reason we do not have to check the ‘oscillator’ option and select ‘vout’, ’gnd’ terminals from the schematic is that PLL is, as the name suggests, not an oscillator.

7. To run the PNoise simulation, in the ADE window click on the **AC,DC**, **Tran** icon on the right pane. Choose the ‘pss’ simulation type, pick the parameters using those shown in Figure 7.35(b). It is critical to note that the phase-noise in the VCO is the dominant source of phase-noise in the complete PLL, and since VCO noise is typically most prominent at a 1MHz offset, we limit our simulation frequency range to be from 1kHz to 10MHz as after the 10MHz the phase-noise will not cause any significant degradation to oscillator output performance. One key difference between the PNoise setup and VCO is that now the phase-noise of interest is of the 8^{th} relative harmonic to the fundamental reference frequency because we have a divider ratio of 8 in our PLL.



Choosing Analyses -- Virtuoso® Analog Desig...

pz sp envlp [✓ pss]
pac pstb pnoise pxf
psp qpss qpac qpnoise
qpxf qpsp hb hbac
hbnoise hbsp

Periodic Steady State Analysis

Engine [✓ Shooting] Harmonic Balance

Fundamental Tones

#	Name	Expr	Value	Signal	SrcId
1		1/(5n-0)	200M	Large	V1

1/(5n-0) 200M Large V1

Clear/Add Delete Update From Hierarchy

[✓ Beat Frequency] Beat Period 200M Auto Calculate ✓

Output harmonics
Number of harmonics 20

Accuracy Defaults (errpreset)
✓ conservative moderate liberal
Additional Time for Stabilization (tstab) 4u
Save Initial Transient Results (saveinit) no yes

Oscillator

Sweep
New Initial Value For Each Point (restart) no yes

Loadpull

Enabled ✓ Options...

OK Cancel Defaults Apply Help

(a)

Choosing Analyses -- Virtuoso® Analog Desig...

psp qpss qpac qpnoise
qpxf qpsp hb hbac
hbnoise hbsp

Periodic Noise Analysis

PSS Beat Frequency (Hz) 200M

Multiple pnoise

Sweeptype relative Relative Harmonic 8
Output Frequency Sweep Range (Hz)
Start-Stop Start 1k Stop 10e6
Sweep Type
Logarithmic Points Per Decade [✓ Number of Steps] 1601

Add Specific Points

Sidebands
Method [✓ default] fullspectrum
Maximum sideband 16
When using shooting engine, default value is 7.

Output
voltage Positive Output Node /Voutn Select
Negative Output Node /gnd! Select
Input Source
none

Noise Type sources
sources: single sideband (SSB) noise analysis
Noise Separation yes no
separate noise into source and gain

Enabled ✓ Options...

OK Cancel Defaults Apply Help

(b)

Figure 7.35: PLL PSS & Pnoise Simulation Setup

The output waveform for the simulated phase-noise will look like Figure 7.36. In our case, we find that the Phase-Noise at a 1MHz offset is equal to -113.31dBc/Hz, which is very reasonable for an Integer-N clock synthesizer PLL with an output frequency of 1.6GHz in steady-state.

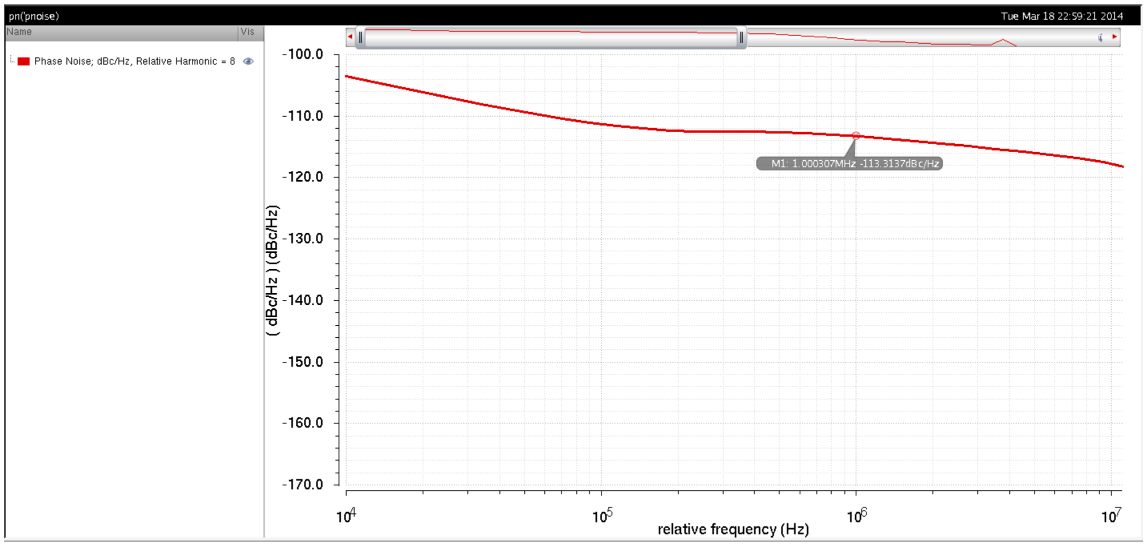
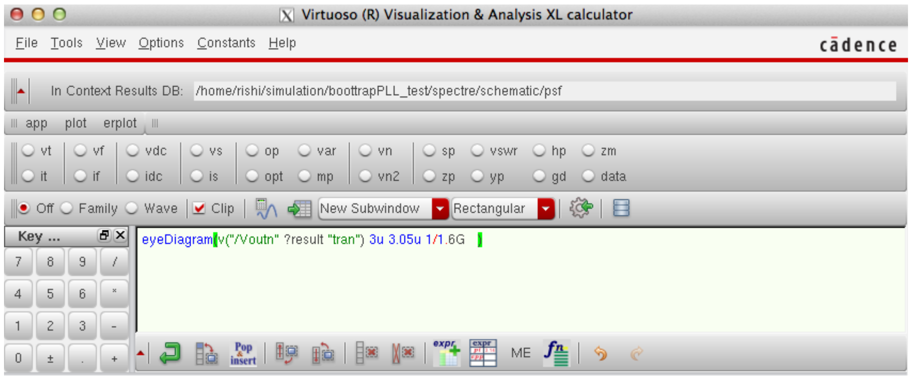


Figure 7.36: PLL Phase-Noise Simulation Plot

8. Since clock generator circuits are responsible for generating the system master-clock, the timing (deterministic) jitter as well as random jitter are key figures-of-merit to minimize clock-induced timing errors during transmission as well as reception of digital data bits. In order to characterize the deterministic timing jitter we plot the PLL eye diagram for a small time-interval once the PLL is in lock condition. Similar to the frequency measurement, we plot the eye diagram by exporting the 'vout' curve into Calculator and using the 'eyeDiagram' function as shown in Figure 7.37.


Virtuoso (R) Visualization & Analysis XL calculator
File Tools View Options Constants Help
cādence
app plot erplot
vt vf vdc vs op var vn sp vswr hp zm
it if idc is opt mp vn2 zp yp gd data
Off Family Wave Clip
New Subwindow
Rectangular
Key ...
7 8 9 /
4 5 6 *
1 2 3 -
0 ± . +
ME

In Context Results DB: /home/rishi/simulation/boottrapPLL_test/spectre/schematic/psf

```
eyeDiagram(v("/Voutn" ?result "tran") 3u 3.05u 1/1.6G
```

Figure 7.37: Eye Diagram Setup for PLL Output

In Figure 7.38 we see that the output voltage eye for the PLL has some deterministic timing jitter associated with it.

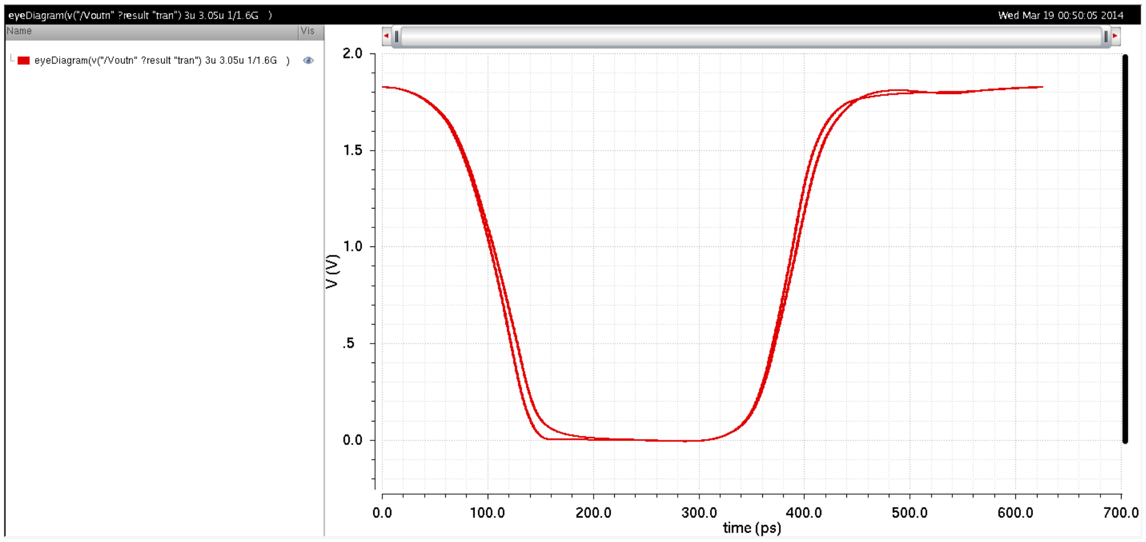


Figure 7.38: PLL Output Eye Diagram Plot

If we zoom-in to the plot, in the area of the 'vout' rising-edge, and place markers at points where the voltage passes 1.0V, we notice that the deterministic timing jitter of our PLL as shown in Figure 7.39 is 5.62ps.

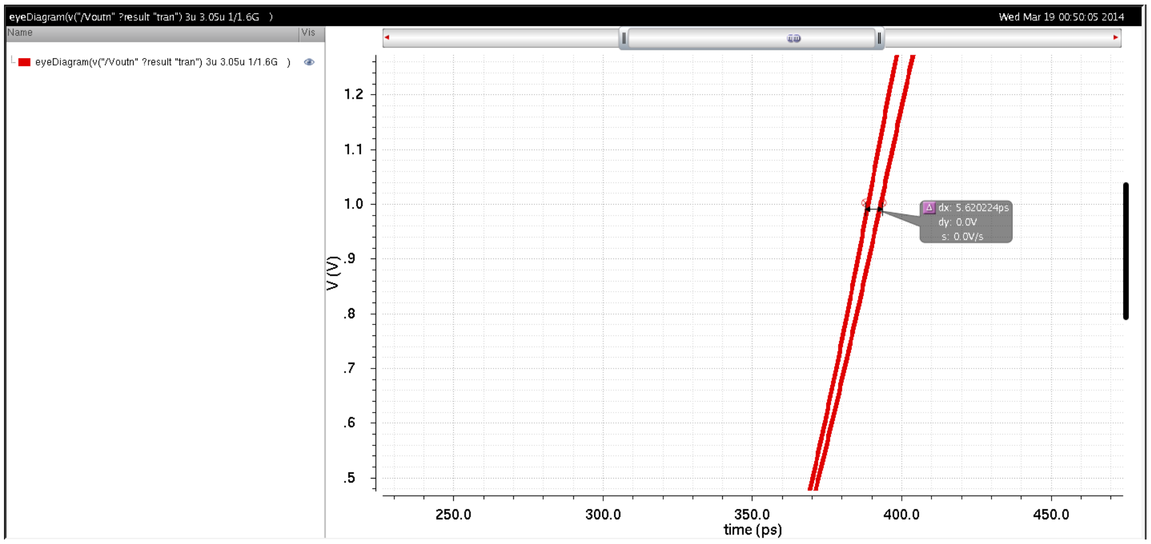
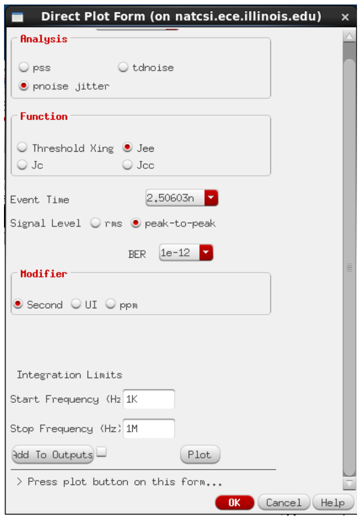


Figure 7.39: PLL Deterministic Jitter Plot

Finally, to calculate the random edge-to-edge jitter of our PLL we need to re-run the PNoise simulation but this time with jitter. In order to do so change the 'Noise-type' in the PNoise setup shown in Figure 7.35(b) to 'jitter'. Once you have run the PNoise simulation with jitter, navigate to *Results* $\rightarrow$ *Direct Plot* $\rightarrow$ *Main Form* from the main ADE window and a window like Figure 7.40 will pop up. Choose 'Jee' and pick an event-time at a point where PLL is in steady-state lock. You can also choose a specified BER. In modern links the BER is typically 10^{-12}, but we plot the jitter over various BER ranges as shown in Figure 7.41.



Direct Plot Form (on natcsi.ece.illinois.edu)

Analysis
pss   tdnoise
[✓ pnoise jitter]

Function
Threshold Xing   [✓ Jee]
Jc   Jcc

Event Time	2.50603n
Signal Level	rms [✓ peak-to-peak]
BER	1e-12

Modifier
[✓ Second]   UI   ppm

Integration Limits

Start Frequency (Hz)	1K
Stop Frequency (Hz)	1M

> Press plot button on this form...

Figure 7.40: PLL Random Jitter Plot Setup

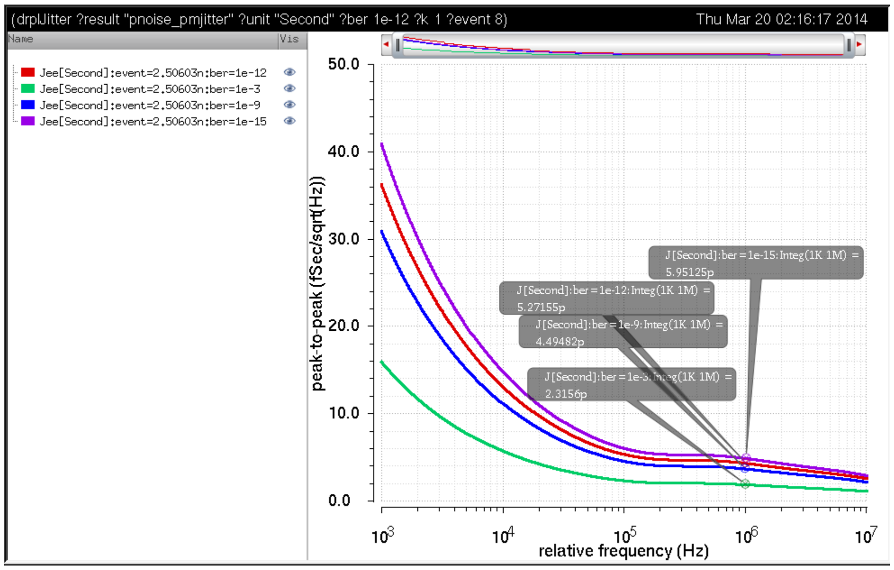


Figure 7.41: PLL Random Jitter Plot

This concludes the characterization of the clock-generator circuit at a transistor level and we have successfully verified its proper functionality.

CHAPTER 8

DISCUSSION

8.1 Conclusion

Overall, thus far in this thesis the foundational motivation for the use of High-Speed Serial Links has been described, fundamentals of PLLs as well as Charge-Pump PLLs have been covered, and an in-depth step-by-step tutorial on design/simulation of an on-chip clock synthesizer at both a behavioral level using Verilog-AMS and transistor level using Cadence Spectre have been described. The designed PLL based clock-generator circuit described in Chapter 5 operates at an output frequency of 1.6GHz at lock with -113dBc/Hz phase-noise, 5.62ps deterministic jitter and 5.27ps edge-to-edge random jitter at a BER level of 10^{-12}. Although many design improvements can be made at the circuit level to optimize the phase noise and jitter performance of the clock-generator circuit, the motivation for this thesis was to provide a tutorial style training manual for a student pursuing mixed-signal IC design at the beginning of their graduate studies; thus, the circuits used for the Integer-N synthesizer are very basic/standard. In this last chapter to conclude the thesis, a summary of future design improvements for the circuit designed/simulated in this thesis is outlined from both a system as well as circuit architecture level. Lastly, an outline of the potential areas of research to explore in the field of high-speed serial links design with a signal integrity focus is presented. Often times when pursuing graduate work in a diverse and mature field of Electrical and Computer Engineering such as Mixed-Signal Circuit design, especially with a focus on Signal Integrity, a new student needs some guidance and initial training to jump-start their careers. Therefore, the final section of this thesis concludes with a few words of advice for new students pursuing this field of study to enable them in solving unexplored areas within this field.

8.2 Future Work

8.2.1 Design Improvements

The PLL based clock-generator circuit designed and simulated in this thesis is very basic and is not optimized for optimal phase-noise and jitter performance for use as an on-chip clock signal in modern day high-speed serial links. Such a simple circuit topology was however chosen in order to simplify the complexity of the system and allow non-circuit designers, especially engineers in the field of signal integrity, to understand the basics of the mixed-signal circuit design/simulation flow using the ubiquitous tool like Cadence Virtuoso. The following list outlines some of the design changes that enhance the speed as well as robustness by optimizing the phase-noise, jitter and power consumption of the on-chip synthesizer:

1. Replace the NAND-PFD with other PFD topologies such as Pass-Transistor or Glitch-Latch Flip-Flop.

2. Implement an actual biasing current-sink in the Charge-Pump and drive this sink by an external off-chip voltage signal. Additionally, implement a differential charge-pump to minimize the CP induced noise and UP/DN current mismatch.

3. If using a ring-oscillator VCO topology, bias the circuit using a self-biasing circuit to reduce the control voltage ripples as well as suppress the supply-induced noise. Additionally, the oscillator should be made differential instead of single-ended as this would greatly improve the phase-noise and jitter performance [9].

4. Implement a TSPC Split-Output latch within the divider or a modified variant of it to reduce the divider-induced delay which translates into additional jitter at the output.

5. Consider implementing a LC-tank based VCO. Although it will increase the area of the circuit, the phase noise and jitter performance will be much better than even a differential ring-VCO design [10, 1].

6. Perform Monte Carlo analysis within Cadence ADE XL to optimize the design over PVT corners to select the optimal sizing for the transistors.

7. The industry-wide trend is to move towards all-digital PLLs. Digital PLLs offer many advantages over analog PLLs mainly in the fact that they eliminate the need for an analog loop-filter as well as a charge-pump, thereby saving area and power consumption. The disadvantage currently however is that quantization noise associated with the Time-to-Digital Converter (TDC) inside a digital PLL severely degrades the phase noise of the system. Thus, even the current state-of-the-art DPLLs are no match in terms of spectral purity performance at high-speeds compared to the analog PLLs. However, with improvements in transistor scaling and machine learning algorithmic noise-tolerance methodologies in digital signal processing, as well as stochastically enhanced circuit design techniques, the future really lies in the study of synthesizable DPLLs. Thus, this area of study should definitely be of utmost priority while performing research on clocking circuit design for high-speed serial link applications [11, 12].

8.2.2 Signal Integrity Focus

In the realm of signal integrity (SI) engineering, the principle of utmost importance is to ensure robust signaling between a driver transmitter and a receiver across a channel medium. As clock frequencies and the associated data rates keep rising with technology scaling and advances in SFT while the channel bandwidth remains roughly the same and package sizes rapidly scale down, the need for superior SI designs in high-speed digital systems is at an all-time high with demand only increasing in the upcoming years. Since the off-chip I/O BWs become the major design bottleneck while demand for low-power designs becomes ubiquitous, it is critical to study SI problems in high-speed serial links.

Typical high-speed serial links are limited by the electrical PCB channels; thus, being able to model all the effects of the channel is key to designing robust systems. The skin-effect has become a major problem in PCB trace channels as the high-frequency signals experience a large series resistance due to current migration toward conductor outer surfaces, and traditional modeling techniques are no longer useful as they assume the metal surfaces are perfectly smooth, whereas in reality there is a significant roughness present which is actually random in nature. Furthermore, the frequency-dependent dielectric-losses experienced along these PCB substrates are difficult to characterize due to lack of the required sophistication in measurement techniques as well as statistical modeling. Measurement is a challenge because for accurate models the setup needs to be passive as well as causal, both of which are very difficult to ensure in practice. Therefore, statistical modeling of the substrate effects would be extremely valuable to enhance the understanding of the channels used in high-speed interface systems. Recall that a backplane or PCB trace can essentially be treated as a transmission line; thus, being able to quantitatively characterize propagation delay, system characteristic impedance, as well as discontinuities during high speed signaling will be very valuable in easing out equalization requirements on both TX as well as RX sides [13].

Overall link performance is analyzed in terms of the TX/RX timing jitter as well as BER specifications. Since the channel is fixed, as mentioned earlier, one prominent method to combat the jitter and ISI-inducing effects of the channel is to perform equalization. Sophisticated TX pre-emphasis equalization and RX side adaptive DFE are vital to link designers as the data rates approach the Tb/s ranges over the upcoming years while the off-chip I/O BW remains about the same. Lastly, formulation of a robust statistical BER analysis and time-domain empirical jittery analysis framework involving all interference sources for any given HSSL will be key to fully characterizing the non-idealities present in links today without the need to over/under design at a circuit level.

8.2.3 Steps for New Students

When embarking on a journey to study and solve new problems in the field of High-Speed Serial Link design, whether it be specifically in the area of clocking circuits, equalization or signal integrity, it is critical to have strong system-level understanding of the HSSL architecture. Once familiarized with the system level basics of links, the student (if interested in circuit design) should take ownership of a specific block within the overall link and focus on optimizing it for low-power, high-speed applications. Conversely, students interested in exploring signal integrity issues in HSSLs should familiarize themselves with Behavioral modeling as well as basics of transistor-level design and simulation analysis so that they explore new HSSL architectures that have high signal integrity even at multi-GHz to THz speeds.

Ideally, in order to make meaningful contributions in the field of HSSL designs, a strong knowledge-base in fields of Integrated Circuits, Electromagnetics, RF/Microwave theory and Digital Signal Processing is key. Thus, at the onset of their graduate career students performing research in the area of robust, fast-signaling HSSLs should take the fundamental graduate-level courses in areas of Digital IC Design, Analog IC Design, Phase-Locked Loop Design, Electromagnetics and DSP. Lastly, ability to exercise the EDA tools like Cadence Virtuoso, Agilent ADS, Ansys HFSS as well as programming in MATLAB and Verilog are essential in order to gain hands-on experience and perform rapid-prototyping of new research ideas.

APPENDIX A

CADENCE VIRTUOSO INSTALLATION GUIDE

A.1 Introduction

The motivation for this manual is to provide a step-by-step tutorial on installing Cadence Virtuoso IC 6.15 tools from scratch, configuring the environment and using the tool to design and simulate circuits. In this short-tutorial users are exposed to the complete steps involved in configuring their machine to run the Cadence Virtuoso IC 6.15 design environment along with its ancillary softwares, converting their host computer into a server, remotely connecting to it and launching the Virtuoso simulator engine from the terminal window followed by a detailed guide to create their own custom circuits and simulate them using the Cadence Spectre circuit simulator.

Cadence is an Electronic Design Automation (EDA) environment that integrates various circuit design and verification applications and tools (both in-house proprietary as well as external third party vendor tools) in a single framework allowing unified IC design and verification in a single environment. The tools are generic and allow the designer to configure the environment depending on the fabrication technology of choice by installing the appropriate PDK (Process-Design Kit).

This tutorial document is not intended to be a one-stop reference for all the features available in Cadence Virtuoso Design Environment. Instead, it is only meant to be a quick-start guide for circuit designers to be able to use the EDA tool to effectively simulate their designs for quick prototyping and verification of their designs.

A.2 Environment Setup

A.2.1 Installing Cadence Virtuoso

1. Cadence Virtuoso design tools only work on Linux OS and best on RedHat based systems. In the scientific community a stable OS capable of running Cadence well is Scientific Linux which is an open-source Linux OS inspired from RedHat Enterprise Linux OS. Install the SL 6.4 64-bit x86 version from [`http://ftp1.scientificlinux.org/linux/scientific/6.4/x86_64/iso/`]. Make sure you install the *SL-64-x86_64-2013-03-21-Everything-DVD1.iso* and *SL-64-x86_64-2013-03-21-Everything-DVD2.iso* as it is the full enterprise version.
   Note: If your machine is 32-bit you can also install the 32-bit version from [`http://ftp1.scientificlinux.org/linux/scientific/6.5/i386/iso/http://ftp1.scientificlinux.org/linux/scientific/6.5/i386/iso/`].

2. Once you have installed the OS, make a new directory under the path */home/EEAPPS* and name it *C*ADENCE_INSTALL. In this tutorial we will be installing 'Virtuoso IC6.15' suite, 'MMSIM 11.1' (required for Spectre/Spectre-RF simulators), 'IUS8' (used for Verilog simulations in Cadence Design Suite), 'HSPICE' and a few 'PDKs' (Process Design Kit). Download the Cadence Virtuoso IC6.15 files (from your ftp file sever) and store them in your computer under */home/EEAPPS/CADENCE_INSTALL/IC615* folder. This folder is the location where you will keep the raw installation files during installation.
   Note: When you are downloading your Cadence Virtuoso files from your ftp server location they will most likely be in *tar* file formats. You will have 7 files for Base version and 8 files for the Hotfix version. First download them into your *D*ownloads folder and then extract the files one by one into the */home/EEAPPS/CADENCE_INSTALL*. Make sure while you *untar* your files you 'untar' each 'Base' file into the same folder and each 'Hotfix' file into the same folder so at the end of the whole process you will have two folders inside the 'CADENCE_INSTALL' folder named *IC06.15.011_lnx86.Base* and *IC06.15.132-615_lnx86.Hotfix*. Although all the installation will be performed from

the Hotfix files it is **very important** to also have the Base files extracted as a path to them will be needed during the installation setup.

3. Before you start the installation process open up a terminal window and type in *su* to make sure that you have **root** user privileges.

4. A key feature of Scientific Linux environment to note is that if you ever have any missing packages that cause an error you just have to type in **yum install** *PackageName* in the terminal window. We will be using this throughout the installation process when we encounter such situations.

5. Install the following packages:

   (a) **yum install elfutils elfutils-libelf libXp**

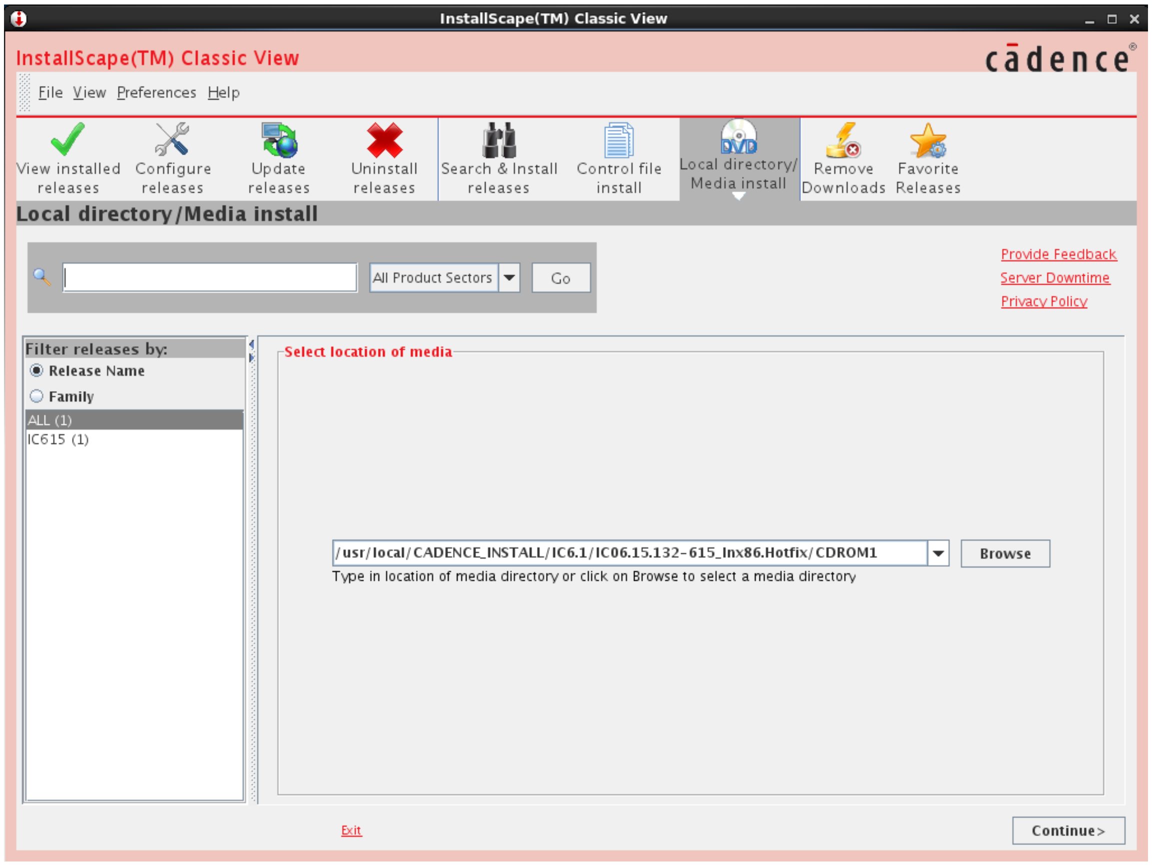
   (b) **yum install libXext.i686**

   (c) **yum install libelf.so.1**

   (d) **yum install libXrender.so.1**
   Note: You need these packages for InstallScape (Cadence Installation Wizard) to work.

6. Create a new directory by typing:
   mkdir -p /home/EEAPPS/CADENCE_INSTALL/IC615/. Now move the extracted Base and Hotfix folders to the 'IC6.15' folder you just created.

7. In the terminal window browse to the following folder: *c*d /home/EEAPPS/CADENCE_INSTALL/IC615/ IC06.15.132-615_lnx86.Hotfix/CDROM1 and then type in *sh SETUP.SH* to start the installation process [14].



/usr/local/CADENCE_INSTALL/IC6.1/IC06.15.132-615_lnx86.Hotfix/CDROM1

Type in location of media directory or click on Browse to select a media directory

Figure A.1: Installing Virtuoso from Installscape Step 1

8. You will now see the following instructions so follow the steps indicated below very carefully:

   (a) Specify path of install directory [OR type [RETURN] to exit]: **/home/EEAPPS/IC615**

   (b) Directory /home/EEAPPS/IC615 does not exist. Create? [y/n]: **y**

   (c) Do you have InstallScape for lnx86 platform installed somewhere [y/n]? **n**

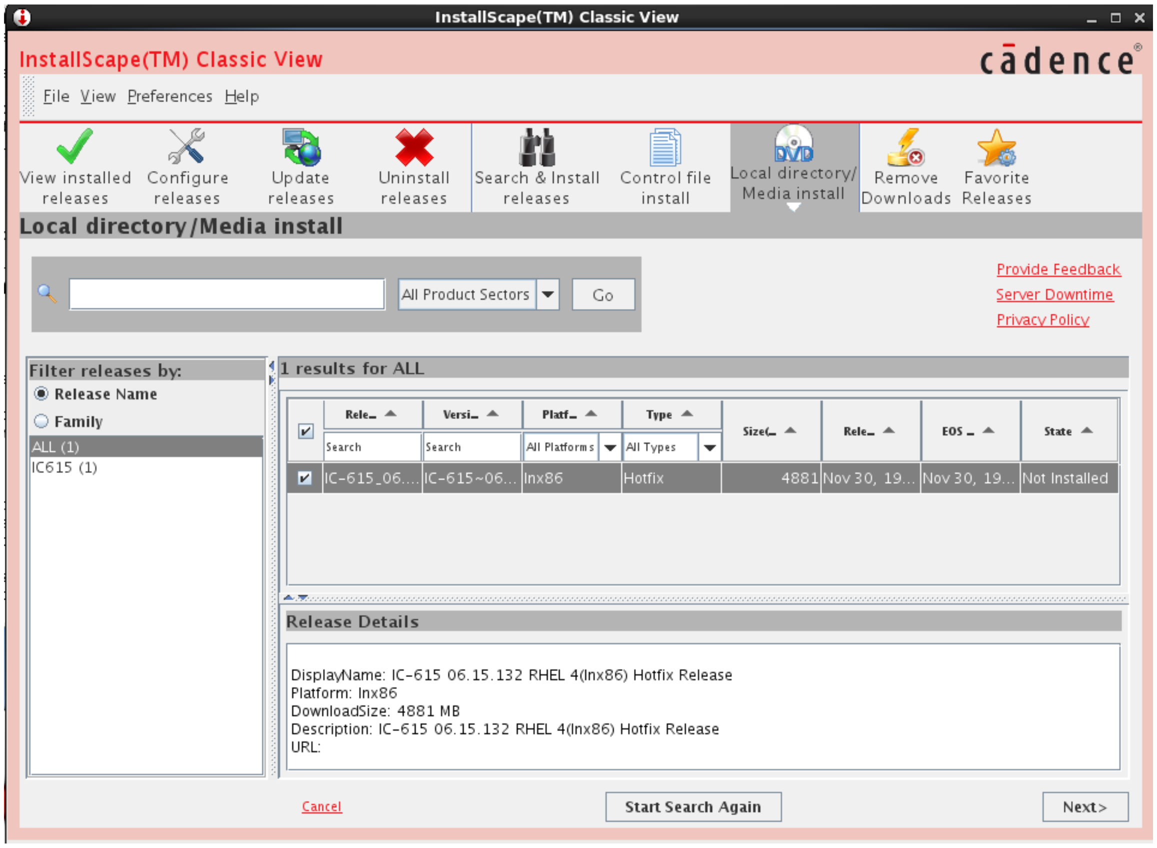
   (d) Do you want to install InstallScape for lnx86 [y/n]? **y**

   (e) Type the path to InstallScape installation directory [(q to exit)]: **/home/Cadence/InstallScape**

9. Now a window like Figure A.1 will pop up so follow the instructions shown in it to browse into the correct folder that contains the installation files.

10. From this point onwards follow the instructions shown in Figures A.2 through A.14 very carefully to complete the installation process for Virtuoso. Make sure you do exactly as shown in these figures to ensure the software gets installed properly.

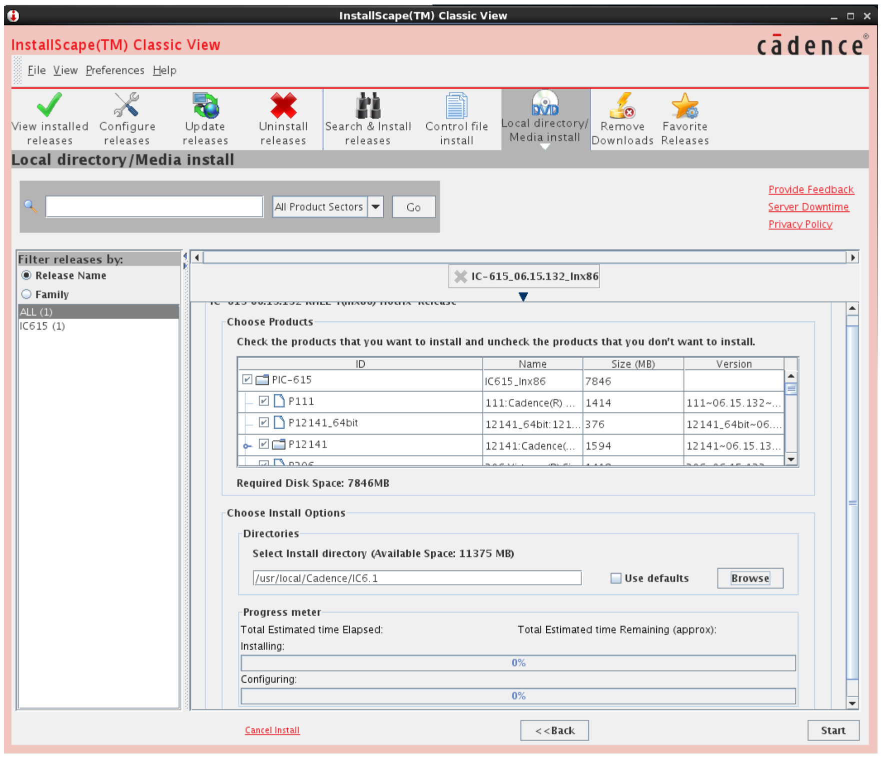
11. Once you reach the last step as shown in Figure A.14 hit 'Done'.



✓	Rele...	Versi...	Platf...	Type	Size(...	Rele...	EOS ...	State
	Search	Search	All Platforms	All Types				
✓	IC-615_06....	IC-615~06...	lnx86	Hotfix	4881	Nov 30, 19...	Nov 30, 19...	Not Installed

DisplayName: IC-615 06.15.132 RHEL 4(lnx86) Hotfix Release
Platform: lnx86
DownloadSize: 4881 MB
Description: IC-615 06.15.132 RHEL 4(lnx86) Hotfix Release
URL:

Figure A.2: Installing Virtuoso from Installscape Step 2



Choose Products

Check the products that you want to install and uncheck the products that you don't want to install.

ID	Name	Size (MB)	Version
☑ PIC-615	IC615_lnx86	7846	
☑ P111	111:Cadence(R) ...	1414	111~06.15.132~...
☑ P12141_64bit	12141_64bit:121...	376	12141_64bit~06....
☑ P12141	12141:Cadence(...	1594	12141~06.15.13...

Required Disk Space: 7846MB

Choose Install Options

Directories

Select Install directory (Available Space: 11375 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

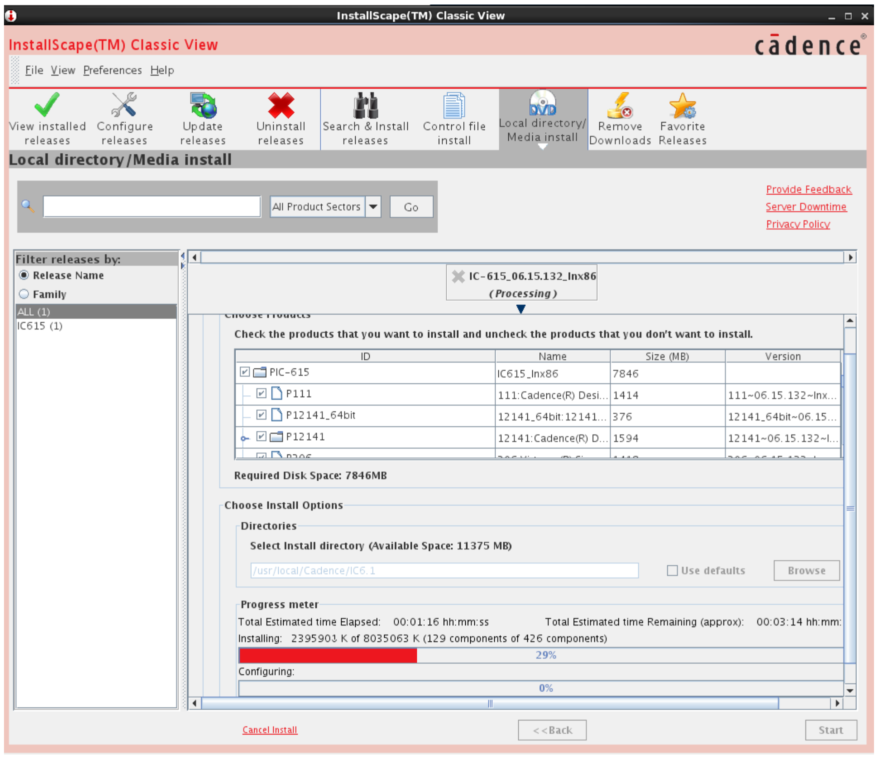
Progress meter

Total Estimated time Elapsed: Total Estimated time Remaining (approx):

Installing: 0%

Configuring: 0%

Figure A.3: Installing Virtuoso from Installscape Step 3



Check the products that you want to install and uncheck the products that you don't want to install.

ID	Name	Size (MB)	Version
☑ PIC-615	IC615_lnx86	7846	
☑ P111	111:Cadence(R) Desi...	1414	111~06.15.132~lnx...
☑ P12141_64bit	12141_64bit:12141...	376	12141_64bit~06.15...
☑ P12141	12141:Cadence(R) D...	1594	12141~06.15.132~l...

Required Disk Space: 7846MB

Choose Install Options

Directories

Select Install directory (Available Space: 11375 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:01:16 hh:mm:ss Total Estimated time Remaining (approx): 00:03:14 hh:mm:

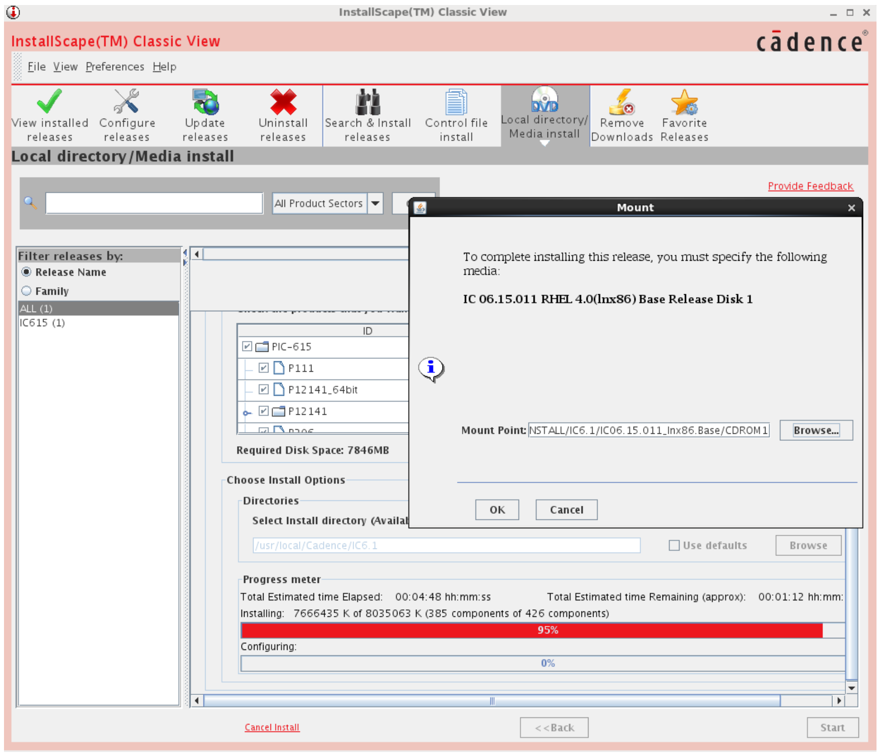
Installing: 2395903 K of 8035063 K (129 components of 426 components)

29%

Configuring:

0%

Figure A.4: Installing Virtuoso from Installscape Step 4



Local directory/Media install

Filter releases by:
● Release Name
○ Family
ALL (1)
IC615 (1)

ID
☑ PIC-615
☑ P111
☑ P12141_64bit
☑ P12141

Required Disk Space: 7846MB

Choose Install Options

Directories

Select Install directory (Availal

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:04:48 hh:mm:ss Total Estimated time Remaining (approx): 00:01:12 hh:mm:

Installing: 7666435 K of 8035063 K (385 components of 426 components)

95%

Configuring:

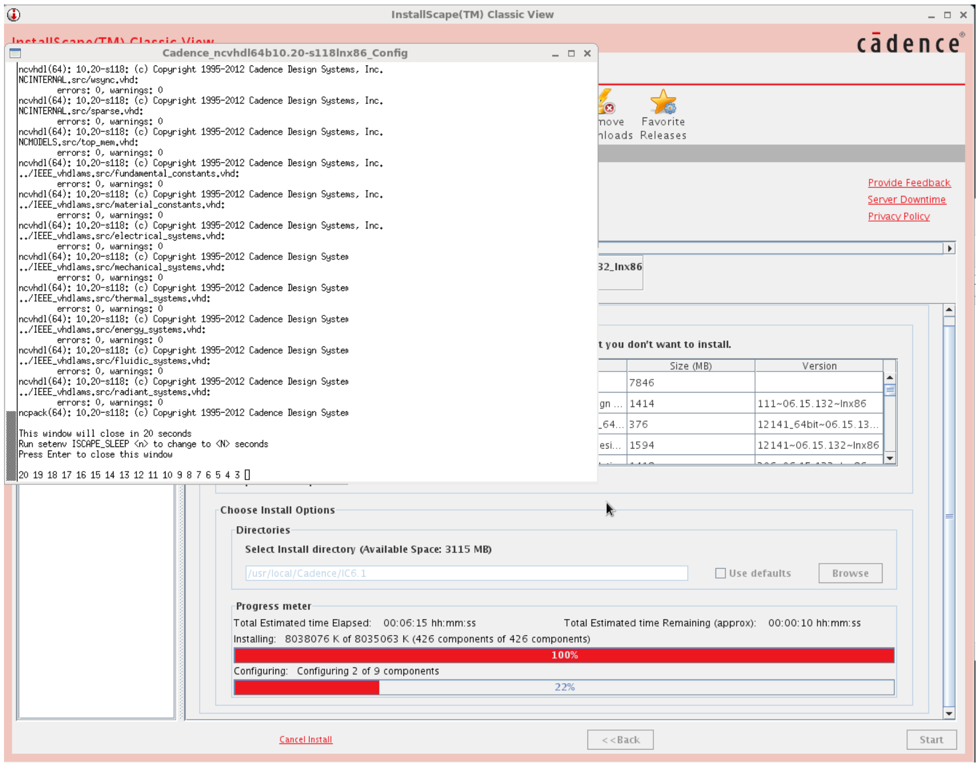
0%

To complete installing this release, you must specify the following media:

IC 06.15.011 RHEL 4.0(lnx86) Base Release Disk 1

Mount Point: NSTALL/IC6.1/IC06.15.011_lnx86.Base/CDROM1

Figure A.5: Installing Virtuoso from Installscape Step 5



Cadence_ncvhdl64b10.20-s118lnx86_Config

```
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
NCINTERNAL.src/wsync.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
NCINTERNAL.src/sparse.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
NCMODELS.src/top_mem.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
../IEEE_vhdlams.src/fundamental_constants.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
../IEEE_vhdlams.src/material_constants.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
../IEEE_vhdlams.src/electrical_systems.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systen
../IEEE_vhdlams.src/mechanical_systems.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systen
../IEEE_vhdlams.src/thermal_systems.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systen
../IEEE_vhdlams.src/energy_systems.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systen
../IEEE_vhdlams.src/fluidic_systems.vhd:
        errors: 0, warnings: 0
ncvhdl(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systen
../IEEE_vhdlams.src/radiant_systems.vhd:
        errors: 0, warnings: 0
ncpack(64): 10.20-s118: (c) Copyright 1995-2012 Cadence Design Systen

This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 
```

32_lnx86

t you don't want to install.

	Size (MB)	Version
	7846	
gn ...	1414	111~06.15.132~lnx86
_64...	376	12141_64bit~06.15.13...
esi...	1594	12141~06.15.132~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 3115 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:06:15 hh:mm:ss Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss

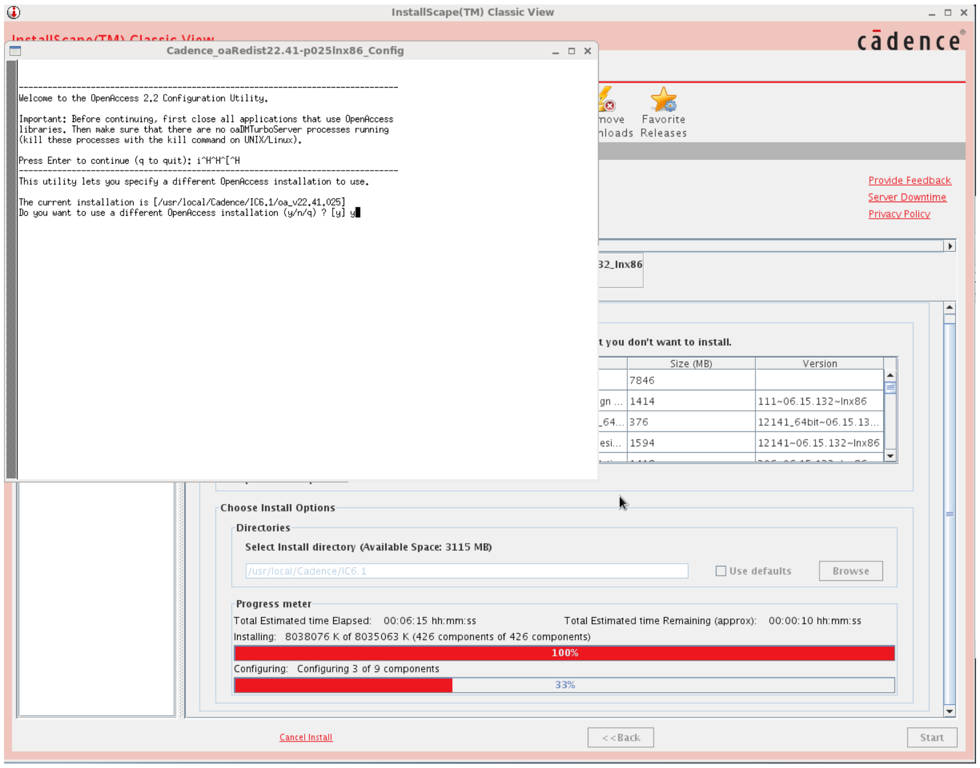
Installing: 8038076 K of 8035063 K (426 components of 426 components)

100%

Configuring: Configuring 2 of 9 components

22%

Figure A.6: Installing Virtuoso from Installscape Step 6



InstallScape(TM) Classic View

Cadence_oaRedist22.41-p025lnx86_Config

```
--------------------------------------------------------------------------------
Welcome to the OpenAccess 2.2 Configuration Utility.

Important: Before continuing, first close all applications that use OpenAccess
libraries. Then make sure that there are no oaDMTurboServer processes running
(kill these processes with the kill command on UNIX/Linux).

Press Enter to continue (q to quit): i^H^H^[^H
--------------------------------------------------------------------------------
This utility lets you specify a different OpenAccess installation to use.

The current installation is [/usr/local/Cadence/IC6.1/oa_v22.41.025]
Do you want to use a different OpenAccess installation (y/n/q) ? [y] y
```

	Size (MB)	Version
	7846	
gn ...	1414	111~06.15.132~lnx86
_64...	376	12141_64bit~06.15.13...
esi...	1594	12141~06.15.132~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 3115 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:06:15 hh:mm:ss Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss

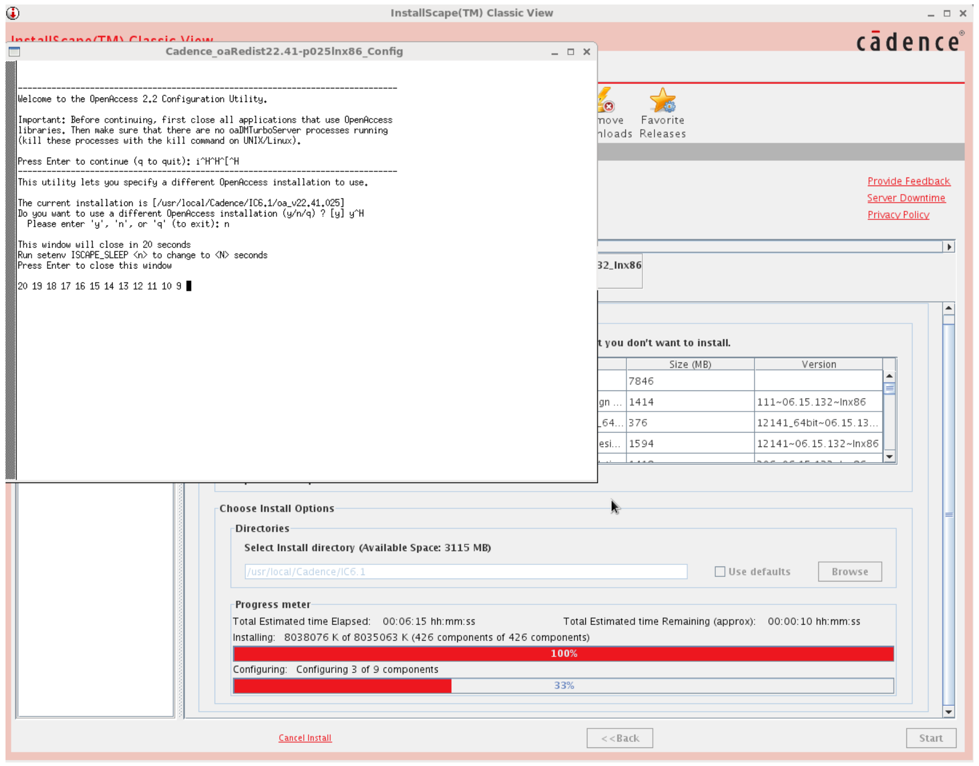
Installing: 8038076 K of 8035063 K (426 components of 426 components)

100%

Configuring: Configuring 3 of 9 components

33%

Figure A.7: Installing Virtuoso from Installscape Step 7



InstallScape(TM) Classic View

Cadence_oaRedist22.41-p025lnx86_Config

```
--------------------------------------------------------------------------------
Welcome to the OpenAccess 2.2 Configuration Utility.

Important: Before continuing, first close all applications that use OpenAccess
libraries. Then make sure that there are no oaDMTurboServer processes running
(kill these processes with the kill command on UNIX/Linux).

Press Enter to continue (q to quit): i^H^H^[^H
--------------------------------------------------------------------------------
This utility lets you specify a different OpenAccess installation to use.

The current installation is [/usr/local/Cadence/IC6.1/oa_v22.41.025]
Do you want to use a different OpenAccess installation (y/n/q) ? [y] y^H
  Please enter 'y', 'n', or 'q' (to exit): n

This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 10 9
```

	Size (MB)	Version
	7846	
gn ...	1414	111~06.15.132~lnx86
_64...	376	12141_64bit~06.15.13...
esi...	1594	12141~06.15.132~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 3115 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:06:15 hh:mm:ss Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss

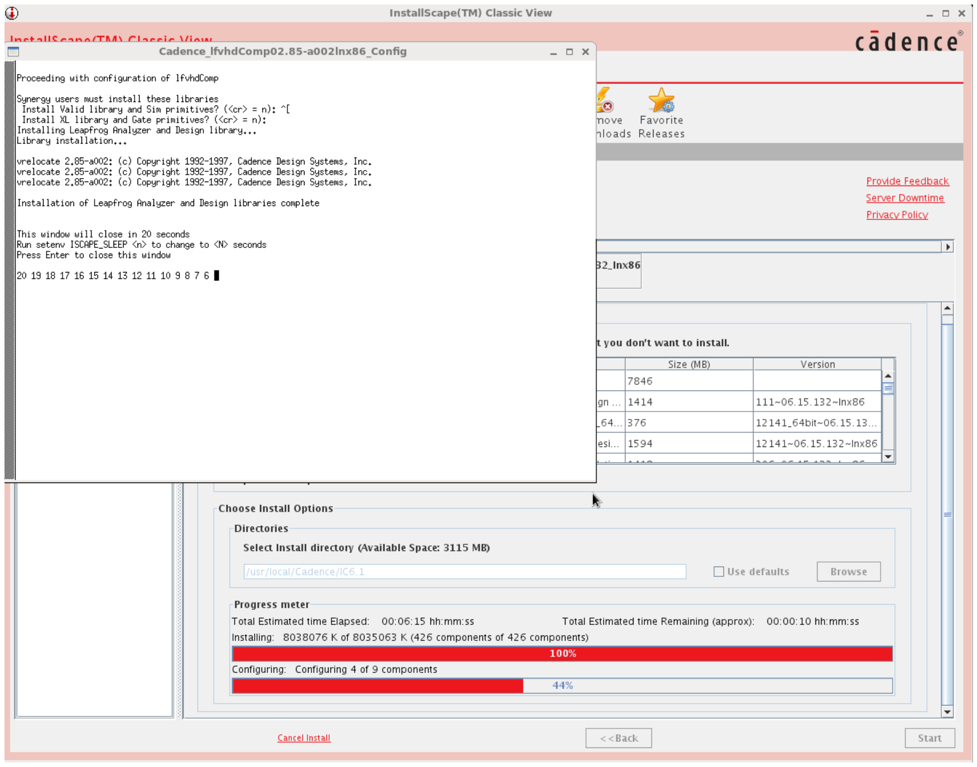
Installing: 8038076 K of 8035063 K (426 components of 426 components)

100%

Configuring: Configuring 3 of 9 components

33%

Figure A.8: Installing Virtuoso from Installscape Step 8



```
Proceeding with configuration of lfvhdComp

Synergy users must install these libraries
 Install Valid library and Sim primitives? (<cr> = n): ^[
 Install XL library and Gate primitives? (<cr> = n):
Installing Leapfrog Analyzer and Design library...
Library installation...

vrelocate 2.85-a002: (c) Copyright 1992-1997, Cadence Design Systems, Inc.
vrelocate 2.85-a002: (c) Copyright 1992-1997, Cadence Design Systems, Inc.
vrelocate 2.85-a002: (c) Copyright 1992-1997, Cadence Design Systems, Inc.

Installation of Leapfrog Analyzer and Design libraries complete


This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 █
```

	Size (MB)	Version
	7846	
gn ...	1414	111~06.15.132~lnx86
_64...	376	12141_64bit~06.15.13...
esi...	1594	12141~06.15.132~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 3115 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:06:15 hh:mm:ss Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss

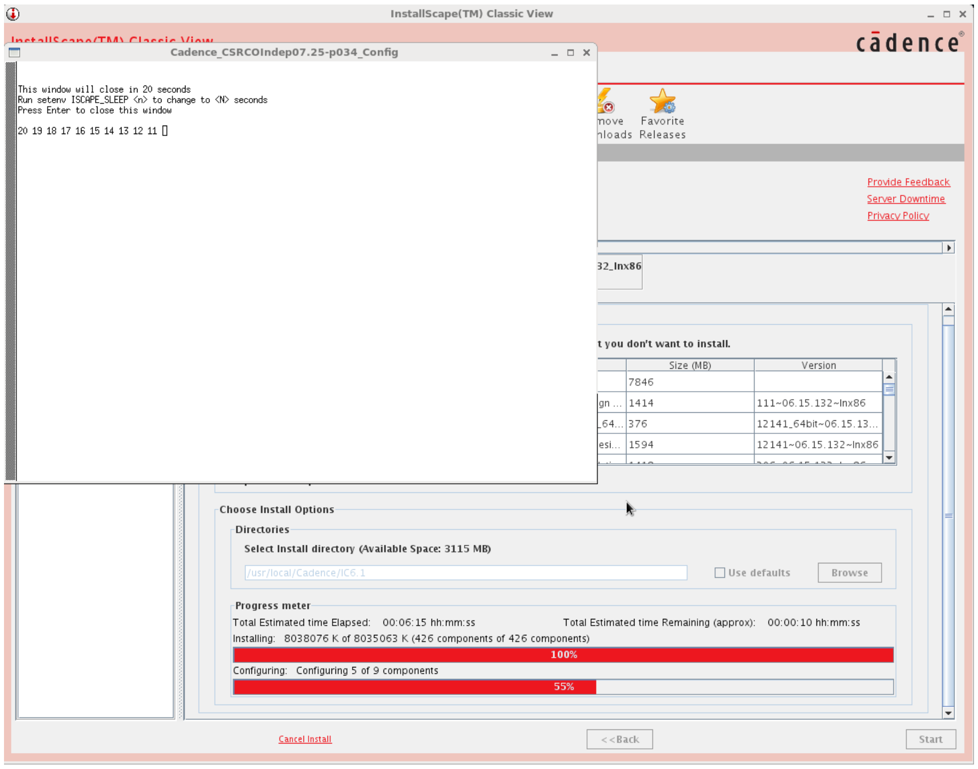
Installing: 8038076 K of 8035063 K (426 components of 426 components)

100%

Configuring: Configuring 4 of 9 components

44%

Figure A.9: Installing Virtuoso from Installscape Step 9



```
This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 ▯
```

	Size (MB)	Version
	7846	
gn ...	1414	111~06.15.132~lnx86
_64...	376	12141_64bit~06.15.13...
esi...	1594	12141~06.15.132~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 3115 MB)

/usr/local/Cadence/IC6.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:06:15 hh:mm:ss Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss

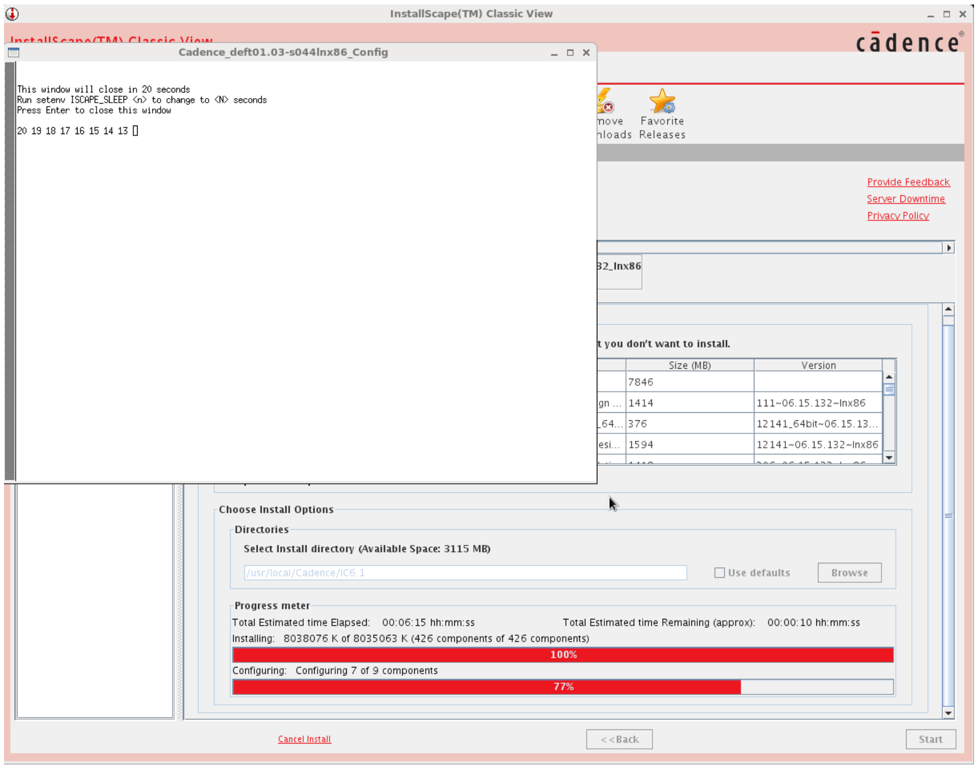
Installing: 8038076 K of 8035063 K (426 components of 426 components)

100%

Configuring: Configuring 5 of 9 components

55%

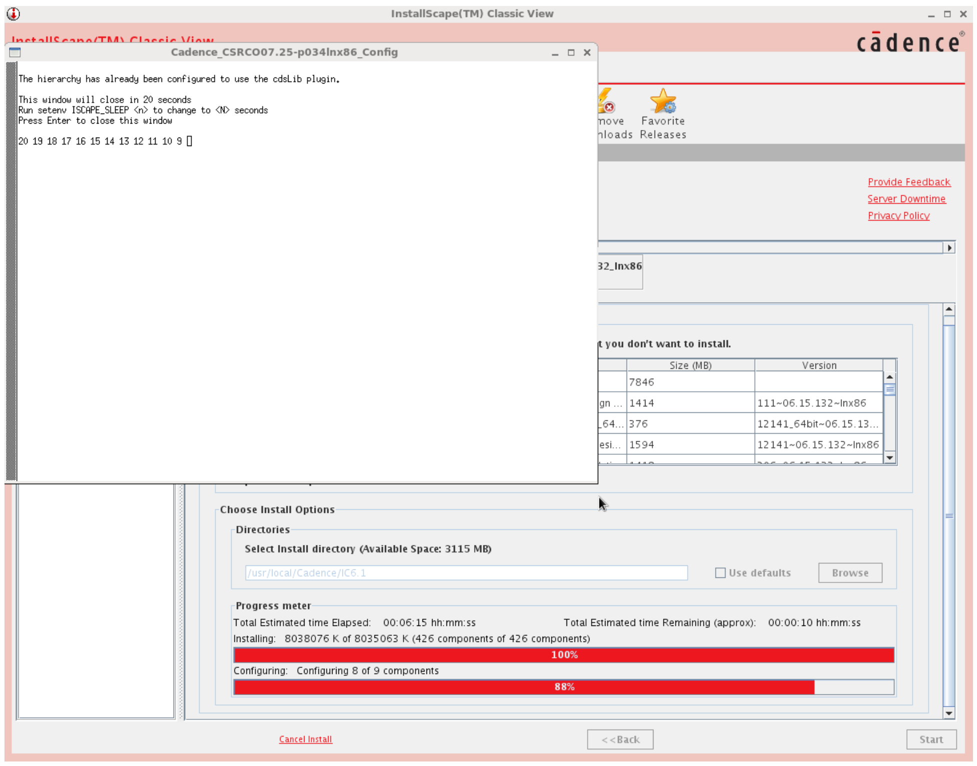
Figure A.10: Installing Virtuoso from Installscape Step 10



```
This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13
```

InstallScape(TM) Classic View
Cadence_deft01.03-s044lnx86_Config
cādence
Favorite Releases
Provide Feedback
Server Downtime
Privacy Policy
Choose Install Options
Directories
Select Install directory (Available Space: 3115 MB)
/usr/local/Cadence/IC6.1
Use defaults
Browse
Progress meter
Total Estimated time Elapsed: 00:06:15 hh:mm:ss
Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss
Installing: 8038076 K of 8035063 K (426 components of 426 components)
100%
Configuring: Configuring 7 of 9 components
77%
Cancel Install
<<Back
Start

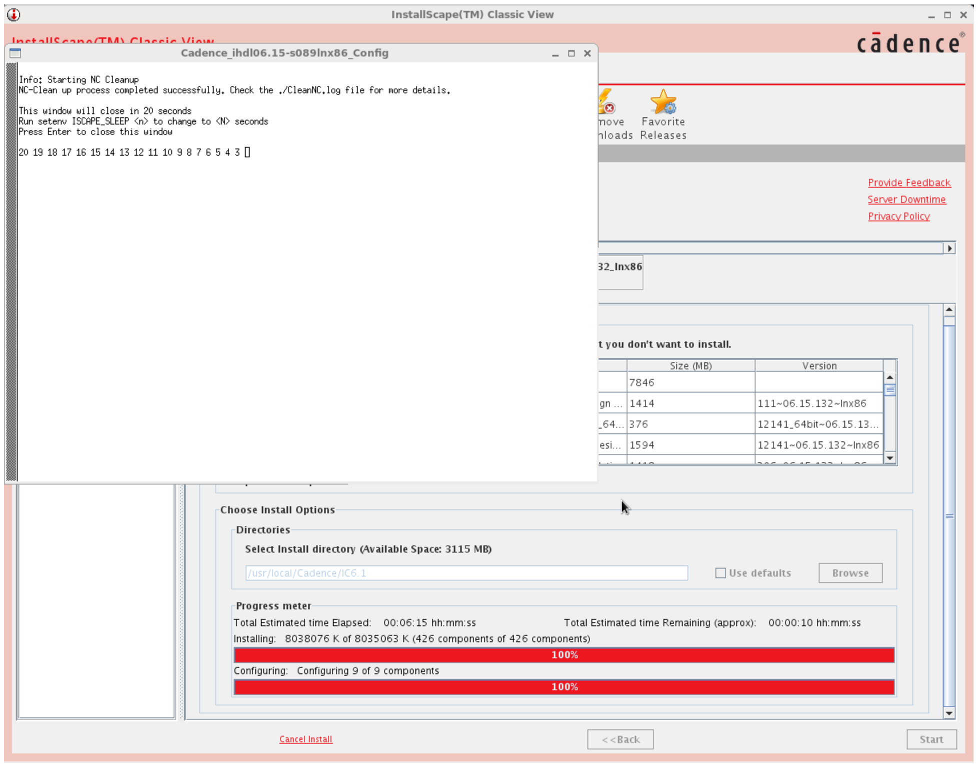
Figure A.11: Installing Virtuoso from Installscape Step 11



```
The hierarchy has already been configured to use the cdsLib plugin.

This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 10 9
```

InstallScape(TM) Classic View
Cadence_CSRC007.25-p034lnx86_Config
cādence
Favorite Releases
Provide Feedback
Server Downtime
Privacy Policy
Choose Install Options
Directories
Select Install directory (Available Space: 3115 MB)
/usr/local/Cadence/IC6.1
Use defaults
Browse
Progress meter
Total Estimated time Elapsed: 00:06:15 hh:mm:ss
Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss
Installing: 8038076 K of 8035063 K (426 components of 426 components)
100%
Configuring: Configuring 8 of 9 components
88%
Cancel Install
<<Back
Start

Figure A.12: Installing Virtuoso from Installscape Step 12



```
Info: Starting NC Cleanup
NC-Clean up process completed successfully. Check the ./CleanNC.log file for more details.

This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 
```

32_lnx86

t you don't want to install.

	Size (MB)	Version
	7846	
gn ...	1414	111~06.15.132~lnx86
_64...	376	12141_64bit~06.15.13...
esi...	1594	12141~06.15.132~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 3115 MB)

/usr/local/Cadence/IC6.1

Use defaults

Browse

Progress meter

Total Estimated time Elapsed: 00:06:15 hh:mm:ss Total Estimated time Remaining (approx): 00:00:10 hh:mm:ss

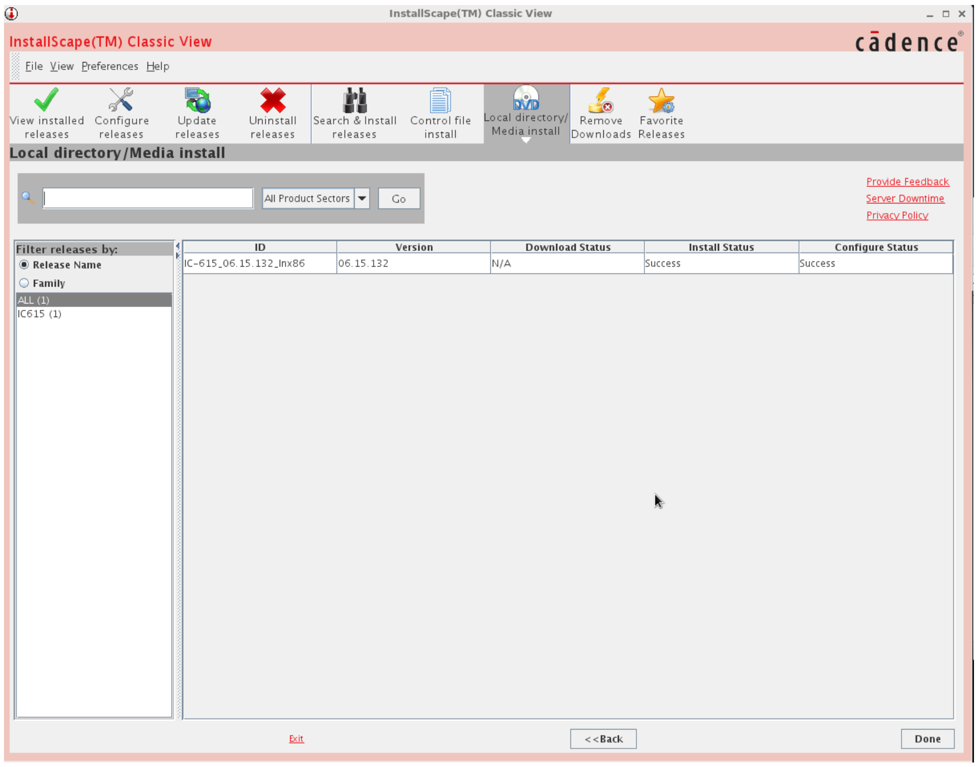
Installing: 8038076 K of 8035063 K (426 components of 426 components)

100%

Configuring: Configuring 9 of 9 components

100%

Figure A.13: Installing Virtuoso from Installscape Step 13



Local directory/Media install

Filter releases by:

Release Name [✓] Family

ALL (1)

IC615 (1)

ID	Version	Download Status	Install Status	Configure Status
IC-615_06.15.132_lnx86	06.15.132	N/A	Success	Success

Figure A.14: Installing Virtuoso from Installscape Step 14

A.2.2 Installing MMSIM (Spectre/SpectreRF/HSpice)

1. Now that you have installed virtuoso in order to actually use the HSPICE or Spectre simulation engines you need to install the MMSIM package.

2. Download the MMSIM installation files from your ftp server and keep the *tar* files in the *Downloads* folder. You will have 3 files for Base version and 3 files for the Hotfix version. Extract the files one by one into the */home/EEAPPS/CADENCE_INSTALL*. Make sure that while you *untar* your files you 'untar' each 'Base' file into the same folder and each 'Hotfix' file into the same folder so at the end of the whole process you will have two folders inside the 'CADENCE_INSTALL' folder named *MMSIM11.10.214_lnx86.Base* and *MMSIM11.10.617_lnx86.Hotfix* [14]. Although all the entire installation will be performed from the Hotfix files it is **very important** to also have the Base files extracted as a path to them will be needed during the installation setup just like you did during Virtuoso installation.

3. Open up the terminal window and create a new directory inside the CADENCE_INSTALL folder by typing in *mkdir -p /home/EEAAPS/-CADENCE_INSTALL/MMSIM11.1/*. Now move the extracted Base and Hotfix folders to the 'MMSIM11.1' folder you just created.

4. In the terminal window browse to the following folder: *cd /home/EEAPPS/CADENCE_INSTALL/ MMSIM11.1/MMSIM11.10.617_lnx86.Hotfix/CDROM1* and then type in *sh SETUP.SH* to start the installation process.

5. You will now see the following instructions so once again follow the steps indicated below very carefully:

   (a) Specify path of install directory [OR type [RETURN] to exit]: **/home/EEAPPS/MMSIM11.1**

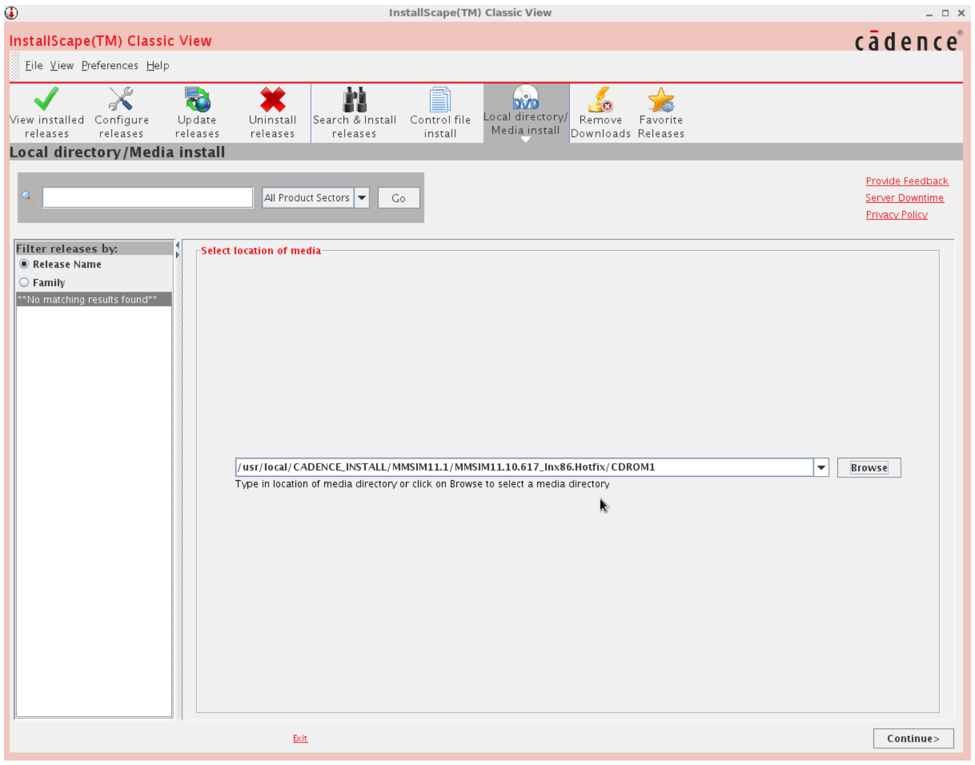
   (b) Do you have InstallScape for lnx86 platform installed somewhere [y/n]? **y**

   (c) Type the path to InstallScape installation directory [(q to exit)]: **/home/Cadence/InstallScape**

6. Now a window like Figure A.15 will pop up so follow the instructions shown in it to browse into the correct folder that contains the installation files.

7. From this point onwards follow the instructions shown in Figures A.16 through A.22 very carefully to complete the installation process for Virtuoso. Make sure you do exactly as shown in these figures to ensure the software gets installed properly.

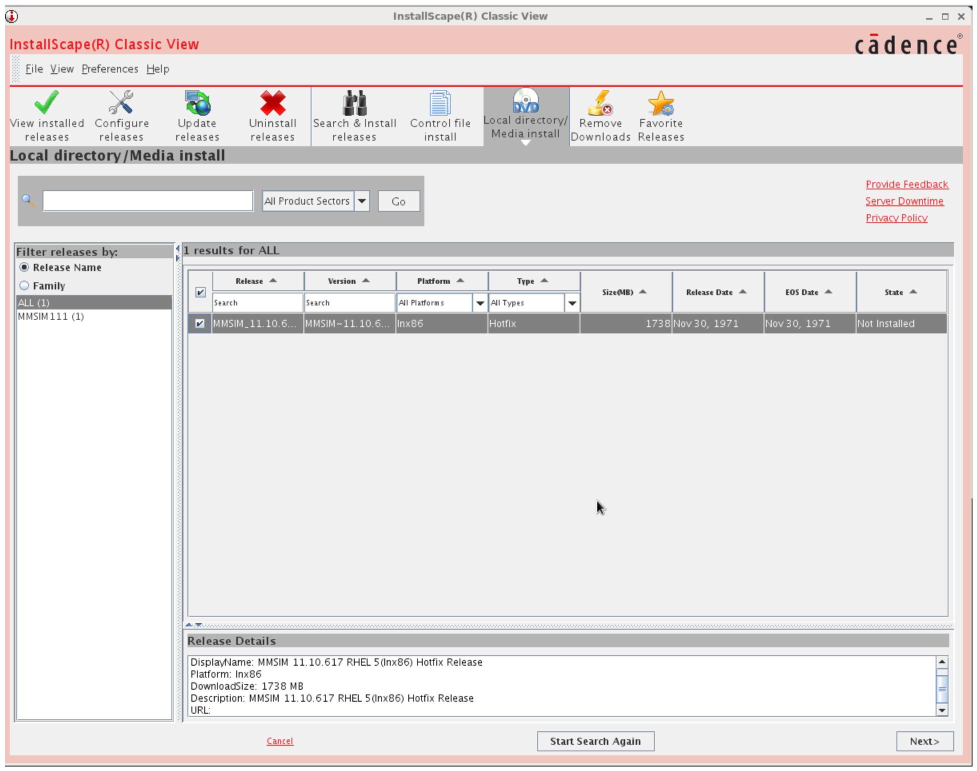
8. Once you reach the last step as shown in Figure A.22 hit ‘Done’.



/usr/local/CADENCE_INSTALL/MMSIM11.1/MMSIM11.10.617_lnx86.Hotfix/CDROM1

Type in location of media directory or click on Browse to select a media directory

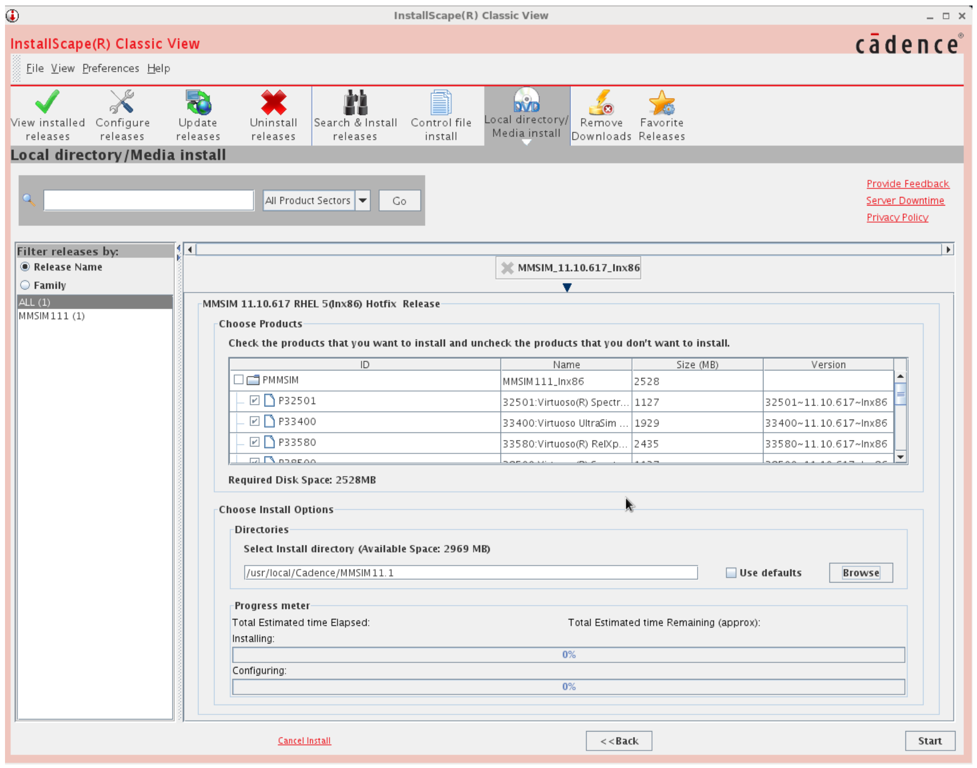
Figure A.15: Installing MMSIM from Installscape Step 1



Release	Version	Platform	Type	Size(MB)	Release Date	EOS Date	State
MMSIM_11.10.6...	MMSIM~11.10.6...	lnx86	Hotfix	1738	Nov 30, 1971	Nov 30, 1971	Not Installed

DisplayName: MMSIM 11.10.617 RHEL 5(lnx86) Hotfix Release
Platform: lnx86
DownloadSize: 1738 MB
Description: MMSIM 11.10.617 RHEL 5(lnx86) Hotfix Release
URL:

Figure A.16: Installing MMSIM from Installscape Step 2



MMSIM 11.10.617 RHEL 5(lnx86) Hotfix Release

Choose Products

Check the products that you want to install and uncheck the products that you don't want to install.

ID	Name	Size (MB)	Version
☐ PMMSIM	MMSIM111_lnx86	2528	
✓ P32501	32501:Virtuoso(R) Spectr...	1127	32501~11.10.617~lnx86
✓ P33400	33400:Virtuoso UltraSim ...	1929	33400~11.10.617~lnx86
✓ P33580	33580:Virtuoso(R) RelXp...	2435	33580~11.10.617~lnx86

Required Disk Space: 2528MB

Choose Install Options

Directories

Select Install directory (Available Space: 2969 MB)

/usr/local/Cadence/MMSIM11.1 ☐ Use defaults

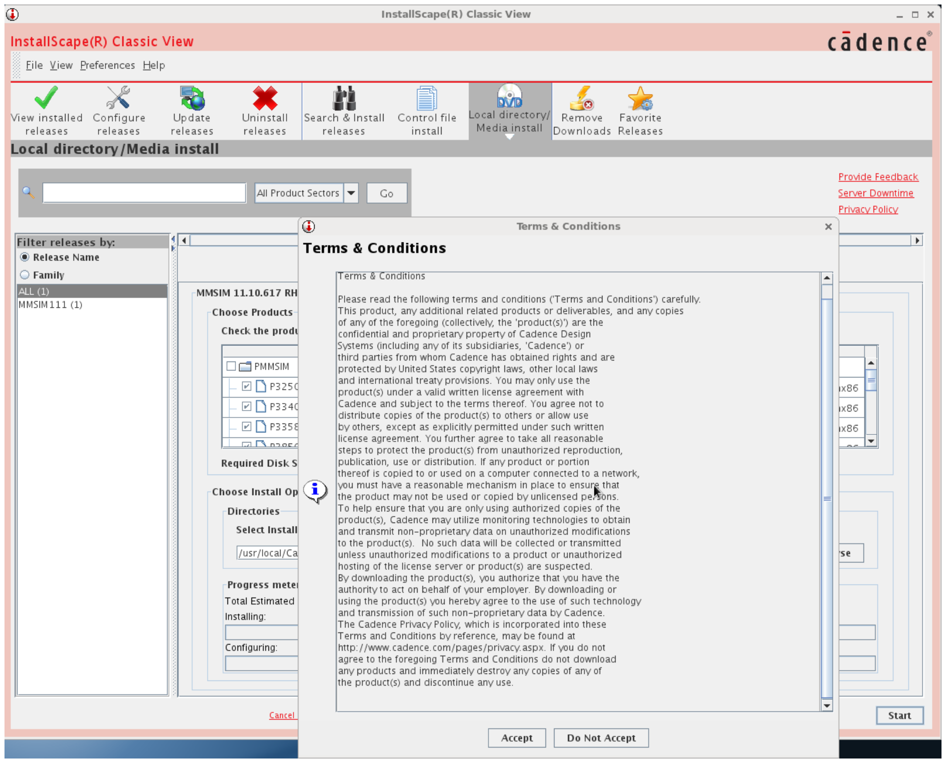
Progress meter

Total Estimated time Elapsed: Total Estimated time Remaining (approx):

Installing: 0%

Configuring: 0%

Figure A.17: Installing MMSIM from Installscape Step 3

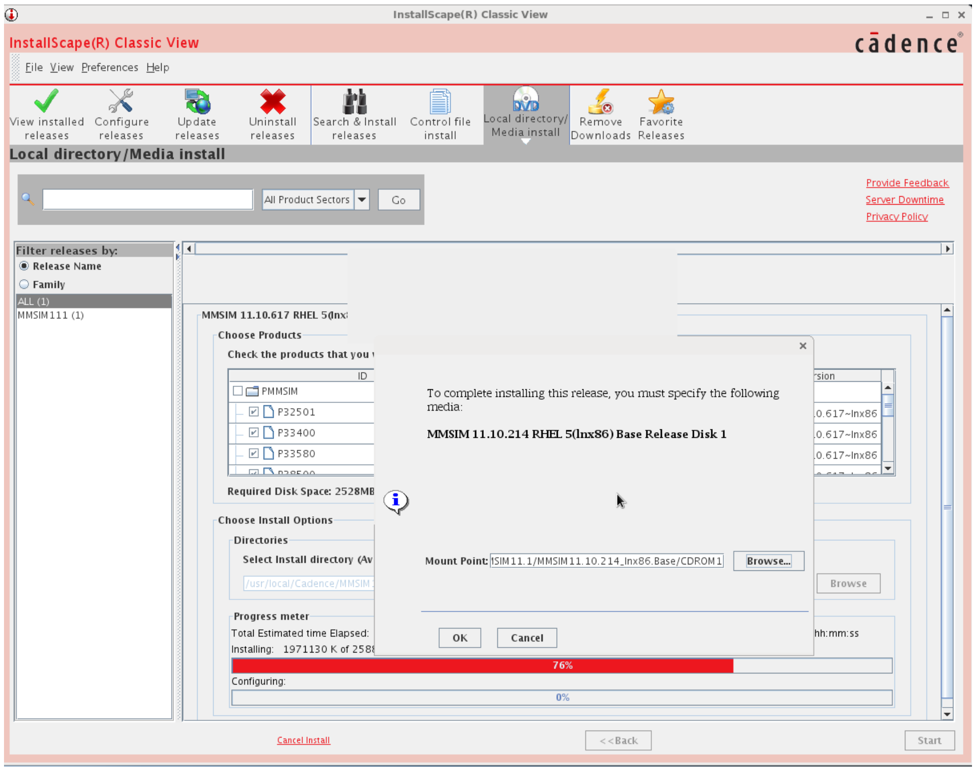


Terms & Conditions

Terms & Conditions

Please read the following terms and conditions ('Terms and Conditions') carefully.
This product, any additional related products or deliverables, and any copies
of any of the foregoing (collectively, the 'product(s)') are the
confidential and proprietary property of Cadence Design
Systems (including any of its subsidiaries, 'Cadence') or
third parties from whom Cadence has obtained rights and are
protected by United States copyright laws, other local laws
and international treaty provisions. You may only use the
product(s) under a valid written license agreement with
Cadence and subject to the terms thereof. You agree not to
distribute copies of the product(s) to others or allow use
by others, except as explicitly permitted under such written
license agreement. You further agree to take all reasonable
steps to protect the product(s) from unauthorized reproduction,
publication, use or distribution. If any product or portion
thereof is copied to or used on a computer connected to a network,
you must have a reasonable mechanism in place to ensure that
the product may not be used or copied by unlicensed persons.
To help ensure that you are only using authorized copies of the
product(s), Cadence may utilize monitoring technologies to obtain
and transmit non-proprietary data on unauthorized modifications
to the product(s). No such data will be collected or transmitted
unless unauthorized modifications to a product or unauthorized
hosting of the license server or product(s) are suspected.
By downloading the product(s), you authorize that you have the
authority to act on behalf of your employer. By downloading or
using the product(s) you hereby agree to the use of such technology
and transmission of such non-proprietary data by Cadence.
The Cadence Privacy Policy, which is incorporated into these
Terms and Conditions by reference, may be found at
http://www.cadence.com/pages/privacy.aspx. If you do not
agree to the foregoing Terms and Conditions do not download
any products and immediately destroy any copies of any of
the product(s) and discontinue any use.

Figure A.18: Installing MMSIM from Installscape Step 4



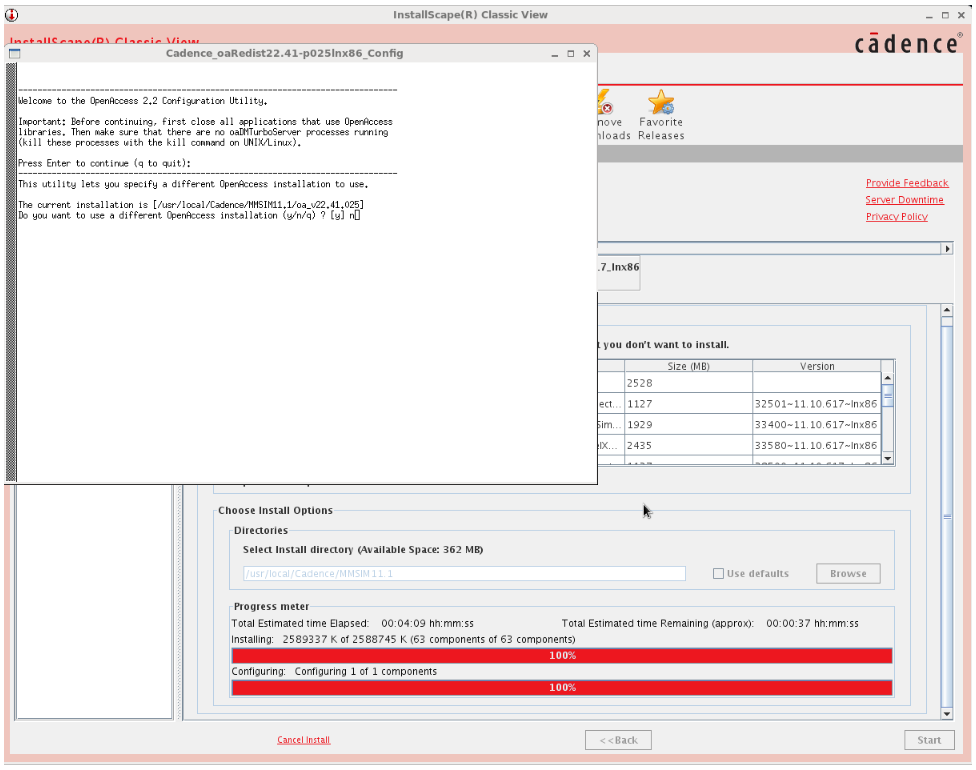
MMSIM 11.10.617 RHEL 5(lnx
Choose Products
Check the products that you
PMMSIM
P32501
P33400
P33580
Required Disk Space: 2528MB
Choose Install Options
Directories
Select Install directory (Av
/usr/local/Cadence/MMSIM
Progress meter
Total Estimated time Elapsed: hh:mm:ss
Installing: 1971130 K of 258
76%
Configuring:
0%

To complete installing this release, you must specify the following media:

MMSIM 11.10.214 RHEL 5(lnx86) Base Release Disk 1

Mount Point: 1SIM11.1/MMSIM11.10.214_lnx86.Base/CDROM1

Figure A.19: Installing MMSIM from Installscape Step 5



```
--------------------------------------------------------------------------
Welcome to the OpenAccess 2.2 Configuration Utility.

Important: Before continuing, first close all applications that use OpenAccess
libraries. Then make sure that there are no oaDMTurboServer processes running
(kill these processes with the kill command on UNIX/Linux).

Press Enter to continue (q to quit):
--------------------------------------------------------------------------
This utility lets you specify a different OpenAccess installation to use.

The current installation is [/usr/local/Cadence/MMSIM11.1/oa_v22.41.025]
Do you want to use a different OpenAccess installation (y/n/q) ? [y] n
```

you don't want to install.

	Size (MB)	Version
	2528	
ect...	1127	32501~11.10.617~lnx86
Sim...	1929	33400~11.10.617~lnx86
IX...	2435	33580~11.10.617~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 362 MB)

/usr/local/Cadence/MMSIM11.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:04:09 hh:mm:ss Total Estimated time Remaining (approx): 00:00:37 hh:mm:ss

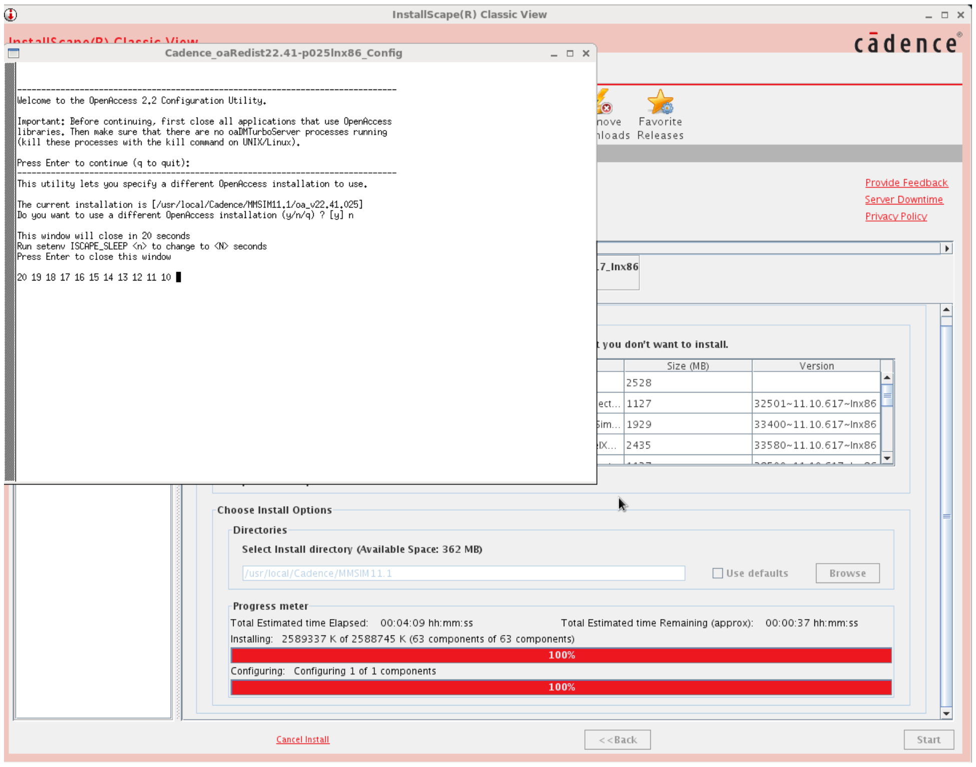
Installing: 2589337 K of 2588745 K (63 components of 63 components)

100%

Configuring: Configuring 1 of 1 components

100%

Figure A.20: Installing MMSIM from Installscape Step 6



```
--------------------------------------------------------------------------
Welcome to the OpenAccess 2.2 Configuration Utility.

Important: Before continuing, first close all applications that use OpenAccess
libraries. Then make sure that there are no oaDMTurboServer processes running
(kill these processes with the kill command on UNIX/Linux).

Press Enter to continue (q to quit):
--------------------------------------------------------------------------
This utility lets you specify a different OpenAccess installation to use.

The current installation is [/usr/local/Cadence/MMSIM11.1/oa_v22.41.025]
Do you want to use a different OpenAccess installation (y/n/q) ? [y] n

This window will close in 20 seconds
Run setenv ISCAPE_SLEEP <n> to change to <N> seconds
Press Enter to close this window

20 19 18 17 16 15 14 13 12 11 10
```

you don't want to install.

	Size (MB)	Version
	2528	
ect...	1127	32501~11.10.617~lnx86
Sim...	1929	33400~11.10.617~lnx86
IX...	2435	33580~11.10.617~lnx86

Choose Install Options

Directories

Select Install directory (Available Space: 362 MB)

/usr/local/Cadence/MMSIM11.1 ☐ Use defaults Browse

Progress meter

Total Estimated time Elapsed: 00:04:09 hh:mm:ss Total Estimated time Remaining (approx): 00:00:37 hh:mm:ss

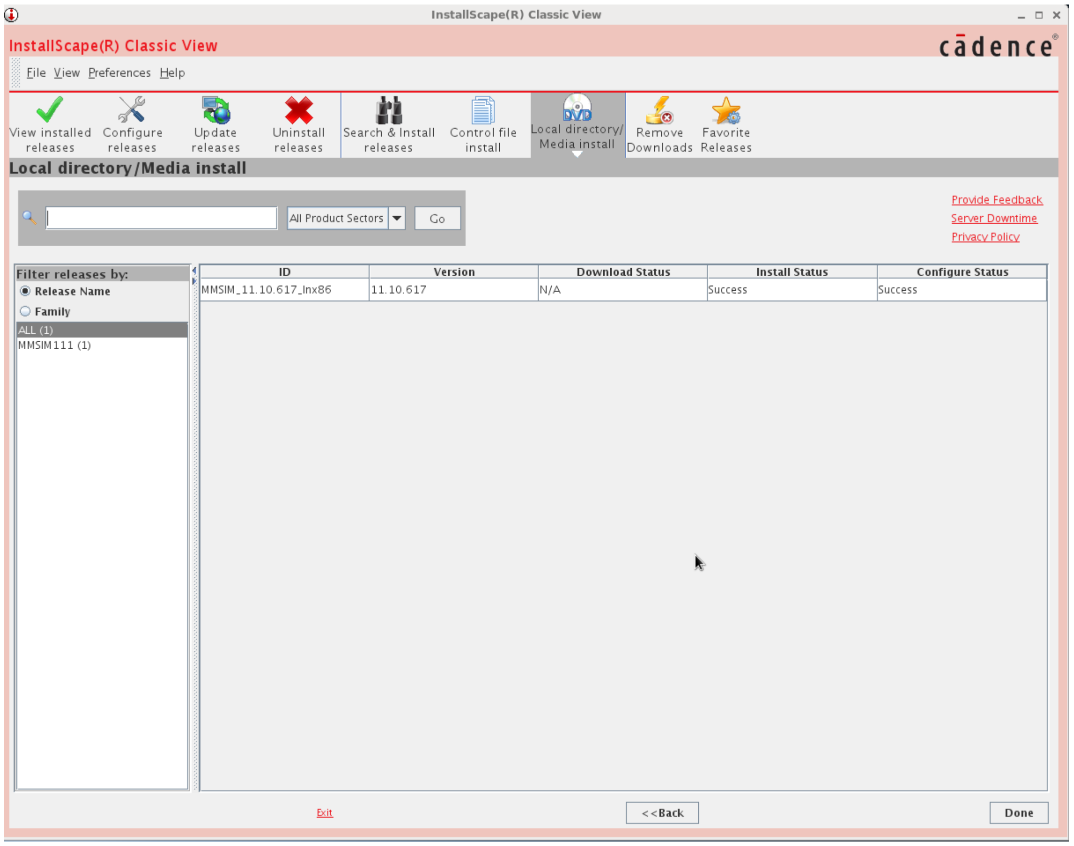
Installing: 2589337 K of 2588745 K (63 components of 63 components)

100%

Configuring: Configuring 1 of 1 components

100%

Figure A.21: Installing MMSIM from Installscape Step 7



ID	Version	Download Status	Install Status	Configure Status
MMSIM_11.10.617_lnx86	11.10.617	N/A	Success	Success

Figure A.22: Installing MMSIM from Installscape Step 8

9. Now that you have installed both Virtuoso and MMSIM the most critical step is to configure the environment variables correctly. In order to do so you will need to change your OS's shell to *bash*. To figure out the current shell of your OS open up a terminal and type in *echo $SHELL*. To change the shell to **bash** if it is not set by default type in chsh -s /bin/bash. If you actually were successful in changing the shell type in *echo $SHELL*, and you should get */bin/bash* as an output.

10. Open up your current *bash* file by typing *gedit .bashrc &* in the terminal window. Now replace the text with that of Section 2.3. Save the updated file and close it.

11. In the terminal window type in **source .bashrc** to update your *bash* settings.

A.2.3 Installing a PDK

1. Download the PDK from your foundry vendor and extract the files in a new directory called *P*DK under the path: */home/EEAPPS*.

2. Create a working directory in your home folder as this will be the directory where you will launch Virtuoso from and will store all your files. For the purposes of this tutorial call your work directory *ckt180* under your 'Documents' folder.

3. Copy the *cds.lib* file from the PDK you installed above and open it up in a text editor.

4. Make sure your 'cds.lib' has the following items before you launch virtuoso. We have to do this to include the in-built libraries that come with the Virtuoso software.

   (a) SOFTINCLUDE /home/EEAPPS/IC615/share/cdssetup/cds.lib

   (b) SOFTINCLUDE /home/EEAPPS/IUS08.20.015/tools/inca/files/cds.lib

   (c) SOFTINCLUDE /home/EEAPPS/TSMC018/cds.lib

   (d) DEFINE ahdlLib $CDSHOME/tools/dfII/samples/artist/ahdlLib

5. At this point Virtuoso is ready for launch so browse to your work directory and type **virtuoso &**. See Figure A.23.

A.2.4 Remote Connections Setup

In order to remotely login to the Server from your machine follow the instructions provided below:

1. **Windows OS Users:**

   (a) Install the SSH client **MobaXterm**[`http://mobaxterm.mobatek.net/download-home-edition.html`] depending on your preference.

   (b) Install **Xming X Server** [`http://sourceforge.net/projects/xming/`] for Windows to allow X-forwarding during the SSH session. Also, install **Xming-fonts** from [`http://sourceforge.`

`net/projects/xming/files/Xming-fonts/`]
<u>Note:</u> Without installing Xming you will not be able to open Virtuoso or for that matter any application with a GUI.

   (c) Launch your SSH client, type **ssh -X username@natcsi.ece.illinois.edu**, hit 'Enter'. You will prompted to type in a password so type it in and again hit 'Enter'. Now you can follow the steps outlined in Figure A.23.

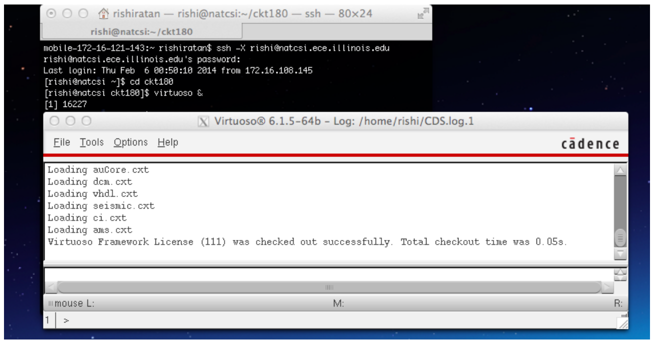
2. **Mac OSX Users:**

   (a) Install **XQuartz 2.7.5** for Mac OSX if you are using OSX Mountain Lion or later. If you have an older OS then you will already have X11 pre-installed in your system. Check your 'System Preferences' to check whether X11 is turned on.
   <u>Note:</u> Without installing XQuartz or enabling X11 (depending upon your OSX version) you will not be able to open Virtuoso or for that matter any application with a GUI.

   (b) Launch your SSH client and type **ssh -X username@natcsi.ece.illinois.edu**, hit 'Enter'. You will prompted to type in a password so type it in and again hit 'Enter'. Now you can follow the steps outlined in Figure A.23.

3. **Linux OS Users:**

   (a) Launch **Terminal** and type **ssh -X username@natcsi.ece.illinois.edu**, hit 'Enter'. You will prompted to type in a password so type it in and again hit 'Enter'. Now you can follow the steps outlined in Figure A.23.


rishiratan — rishi@natcsi:~/ckt180 — ssh — 80×24
rishi@natcsi:~/ckt180

```
mobile-172-16-121-143:~ rishiratan$ ssh -X rishi@natcsi.ece.illinois.edu
rishi@natcsi.ece.illinois.edu's password:
Last login: Thu Feb  6 00:50:10 2014 from 172.16.108.145
[rishi@natcsi ~]$ cd ckt180
[rishi@natcsi ckt180]$ virtuoso &
[1] 16227
```

Virtuoso® 6.1.5-64b - Log: /home/rishi/CDS.log.1
File
Tools
Options
Help
cādence

```
Loading auCore.cxt
Loading dcm.cxt
Loading vhdl.cxt
Loading seismic.cxt
Loading ci.cxt
Loading ams.cxt
Virtuoso Framework License (111) was checked out successfully. Total checkout time was 0.05s.
```

Figure A.23: Launch Instructions for Virtuoso

A.2.5 Configuring Bash Environment

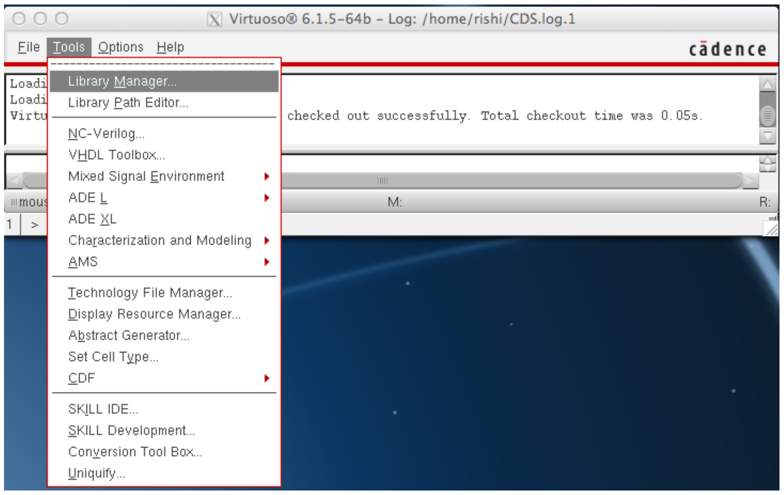
```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# User specific aliases and function
alias mat='cd /home/rishi/matlab; matlab &'
alias cscope='/home/EEAPPS/Cscope/ai_bin/cscope'
################## Hspice #####################
SYNOPSYS_HOME=/home/EEAPPS
HSP_HOME=$SYNOPSYS_HOME/HSPICE
SCL_HOME=$SYNOPSYS_HOME/SCL
HSP_BIN=$HSP_HOME/hspice/bin
SCL_BIN=$SCL_HOME/linux/bin
export LM_LICENSE_FILE=/home/EEAPPS/HSPICE/linmac.dat
export PATH=${HSP_HOME}/hspice/bin:$PATH
export PATH=/home/EEAPPS/Cscope/ai_bin/:$PATH
##################### IC ########################
export MMSIM_ROOT=/home/EEAPPS/MMSIM
export OA_HOME=/home/EEAPPS/IC615/oa
export CDSHOME=/home/EEAPPS/IC615
export CDSDIR=/home/EEAPPS/IC615
export CDS_ROOT=/home/EEAPPS/IC615
export CDS_INST_DIR=//home/EEAPPS/IC615
export DD_DONT_DO_OS_LOCKS=SET
export CDS_LIC_FILE=5280@cadence.webstore.illinois.edu
export CDS_Netlisting_Mode=``Analog"
export_PATH=${CDS_INST_DIR}/tools/bin:$PATH
export_PATH=${CDS_INST_DIR}/tools/dfII/bin:$PATH
export_PATH=${CDS_INST_DIR}/tools/plot/bin:$PATH
export_PATH=${CDS_INST_DIR}/tools/dracula/bin:$PATH
export_PATH=${CDS_ROOT}/tools/bin:$PATH
export_PATH=${CDS_ROOT}/tools/dfII/bin:$PATH
export_PATH=${CDS_ROOT}/tools/dracula/bin:$PATH
```

```
export PATH=${CDS_ROOT}/tools/plot/bin:$PATH
export PATH=${CDS_ROOT}/tools/iccraft/bin:$PATH
export PATH=/home/EEAPPS/InstallScape/iscape/bin:$PATH
export PATH=${MMSIM_ROOT}/tools/dfII/bin:$PATH
export PATH=${MMSIM_ROOT}/tools/spectre/bin:$PATH
export PATH=${MMSIM_ROOT}/tools/ultrasim/bin:$PATH
export PATH=${MMSIM_ROOT}/tools/bin:$PATH
export CDS_AUTO_64BIT=ALL
export CDS_LOAD_ENV=CSF
export EDITOR=/usr/bin/gedit
################# IUS #########################
export CADENCE_CURR_IUS=$SYNOPSYS_HOME/IUS08.20.015
export PATH=${CADENCE_CURR_IUS}/tools.lnx86/bin:$PATH
export PATH=${CADENCE_CURR_IUS}/tools/bin:$PATH
export PATH=${CADENCE_CURR_IUS}/bin:$PATH
################### MATLAB ############
export PATH=/home/MATLAB/R2013b/bin:$PATH
```

A.2.6 Creating a Library in Cadence Virtuoso

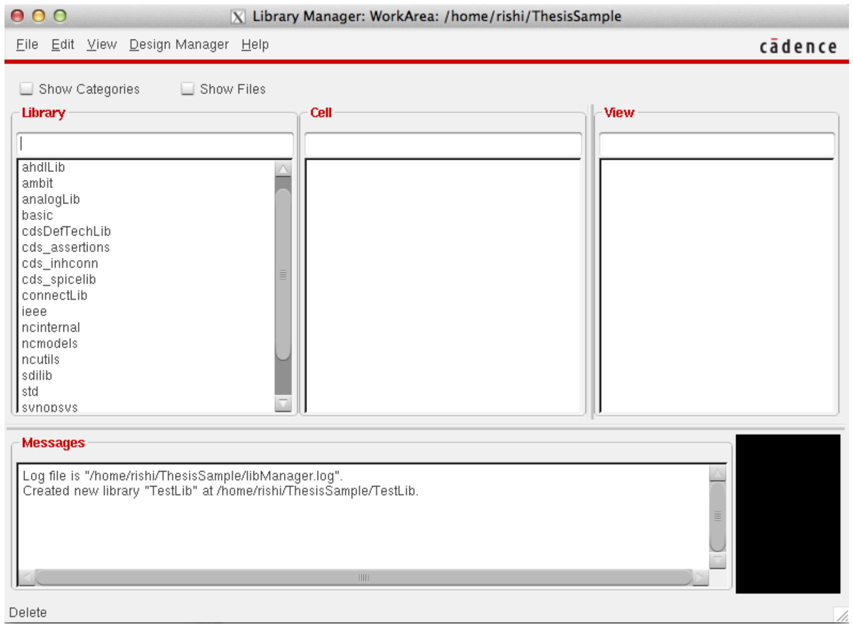
1. The first task after launching Virtuoso is to organize all your designs into appropriate libraries. To view all the libraries in the current work directory click on $Tools \rightarrow LibraryManager$ as outlined in Figure A.24, and the Library Manager window will pop up as shown in Figure A.25.

   Note: If you want to manually add a library that you copied from an external source into your Cadence work directory you would need to edit the *cds.lib* file found in your work directory folder by opening it in a text-editor.



```
Loadi
Loadi
Virtu                                 checked out successfully. Total checkout time was 0.05s.
```

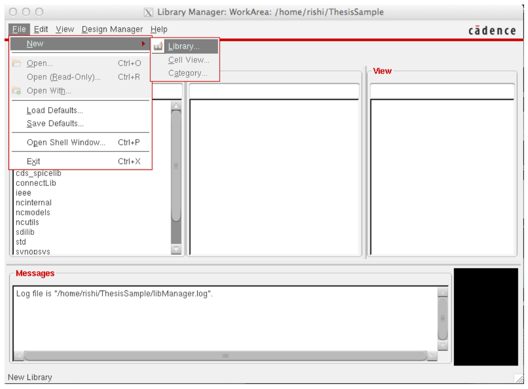
Figure A.24: Launch Instructions for Library Manager



```
Log file is "/home/rishi/ThesisSample/libManager.log".
Created new library "TestLib" at /home/rishi/ThesisSample/TestLib.
```

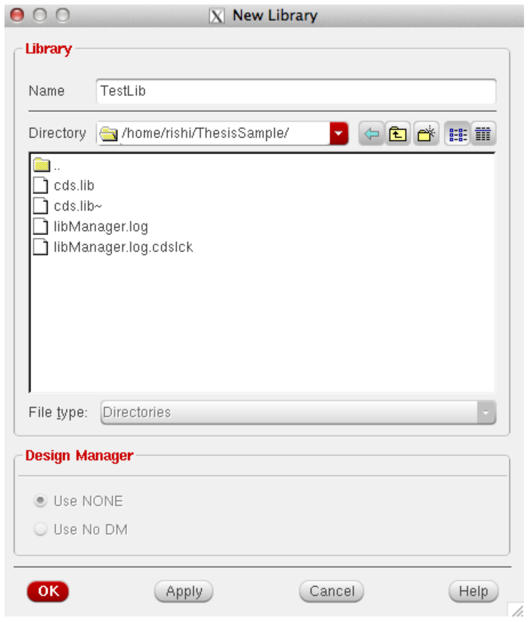
Figure A.25: Library Manager Window

2. To create a new library click on *File* → *New* → *Library* and name the library as *TestLib* as highlighted in Figure A.26. After creating the new library you need to specify the Technology File to be used in your respective PDK. In our case we will 'Attach an existing technology library', specifically the 'tsmc18rf' which corresponds to 180nm CMOS process. Figure A.27 shows the steps involved in attaching the appropriate technology file to a new library.



Log file is "/home/rishi/ThesisSample/libManager.log".

(a) Create New Library



Name TestLib

Directory /home/rishi/ThesisSample/

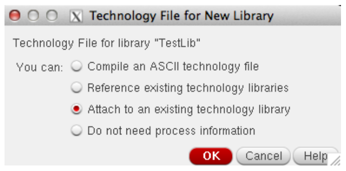
File type: Directories

[✓ Use NONE]

Use No DM

(b) New Library Name

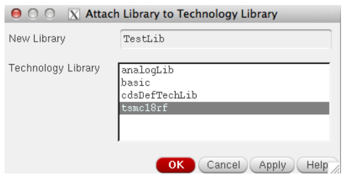
Figure A.26: Steps to Create New Library



Technology File for library "TestLib"

You can: Compile an ASCII technology file
Reference existing technology libraries
[✓ Attach to an existing technology library]
Do not need process information

(a)



New Library TestLib

Technology Library [✓ tsmc18rf]

(b)

Figure A.27: Attaching Tech File

Refer to Chapters 6 and 7 for full behavioral/circuit-level simulation guide.

A.3 Common Troubleshooting Tips

Some of the most commonly recurring errors are discussed below along with the possible solution to resolve them:

1. Often when working on a common server machine with multiple users logging onto the same account, the cadence filesystems gets "locked" preventing any edit operations on any of the files associated with the given user. The error will resemble the following statement in the terminal window if such an event occurs: **WARNING* file /home-/rishi/CDS.log File is already locked by some other process.* In order to fix this problem navigate to your home directory. Note that the home directory is not the directory from which virtuoso is launched, instead it is its the parent directory. Delete any of the files of the form 'CDS.log', 'CDS.log.1' and 'CDS.log.cdslck'. After deleting these files refresh Virtuoso by navigating to the Cadence Virtuoso 'Log' window and clicking on $File \rightarrow Refresh$.

2. When sharing libraries among users within a server make sure you add the library name in the 'cds.lib' file contained in your cadence launch directory. Additionally, make sure while copying the files, the destination user has write/edit privileges as lack of the write permission will limit edit capabilities within virtuoso.

3. During simulation of complex circuits the Cadence simulation folder gets full and often causes the entire machine to hang. In order to prevent this from happening periodically, delete the contents of the 'simulation' folder found within your Cadence launch directory.

4. Always save the one functional simulation setting by navigating to the *Session* $\rightarrow$ *SaveState* window within the ADE window. Make sure you select 'Cellview' as this would also save the simulation plots from the last simulation, thereby saving some simulation time for future use.

REFERENCES

[1] S. Palermo, *CMOS Nanoelectronics Analog and RF VLSI Circuits, Chapter 9.* New York City, N.Y.: McGraw-Hill, 2011.

[2] E. Alon, "High-speed electrical interface circuit design: Lecture 1," 2011. [Online]. Available: http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee290c_s11/lectures/Lecture01_Intro_2up.pdf

[3] V. Stojanović, "Channel-limited high-speed links: Modeling, analysis and design," Ph.D. dissertation, Stanford University, Palo Alto, 2004. [Online]. Available: http://chipgen.stanford.edu/papers/vs_thesis.pdf

[4] K. Kundert and O. Zinke, *Designer's Guide to Verilog-AMS, Chapter 3.* Boston, M.A.: Kluwer-Academic Publishers, 2004.

[5] D. Friedman, "International solid-state circuits conference trends 2013," 2013. [Online]. Available: http://isscc.org/doc/2013/2013_Trends.pdf

[6] J. C. Chen, "Mutli-gigabit serdes: The cornerstone of high speed serial interconnects," 2011. [Online]. Available: http://www.design-reuse.com/articles/10541/multi-gigabit-serdes-the-cornerstone-of-high-speed-serial-interconnects.html

[7] 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

[8] 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.

[9] U.Ku-Moon. P.K. Hanumolu, "Effect of power supply noise on ring osc phase noise," 2004. [Online]. Available: http://web.engr.oregonstate.edu/~moon/research/files/newcas04_supply.pdf

[10] M. Mansuri, “Low-power low-jitter on-chip clock generation,” Ph.D. dissertation, Univ. of California, Los-Angeles, 2003. [Online]. Available: http://www.ece.tamu.edu/~spalermo/ecen689/pll_thesis_mansuri_ucla_2003.pdf

[11] U.Ku-Moon. P.K. Hanumolu, G.Y. Wei and K. Mayaram, “Digitally-enhanced phase-locking circuits,” in *Proc. IEEE Custom Integrated Circuits Conference'07)*, San Jose, USA, Sep. 2007, pp. 361–368.

[12] V. Kratyuk et al., “A design procedure for all-digital phase-locked loops based on a charge-pump phase-locked-loop analogy,” *IEEE Transactions on Circuits and Systems-II*, vol. 54, no. 3, pp. 247–251, 2007.

[13] J. Fan et al., “Signal integrity design for high-speed digital circuits:progress and directions,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 52, no. 2, pp. 392–400, 2010.

[14] R. Helinski, “Installing cadence ic 6.1,” 2010. [Online]. Available: http://www.ece.unm.edu/~jimp/vlsiII/cadence_install/installing_cadence.pdf

[15] B. Razavi, *Monolithic Phase-Locked Loops and Clock Recovery Circuits, Chapter 1.* Piscataway, N.J.: IEEE Press, 1996.