Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

Optimization of Electrical Validation and Debug time in Reference Clocks

---

ESTUDIO DE CASO que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **LUIS EDUARDO ARREDONDO SANDOVAL**

Director: **DR. MANUEL SALIM MAZA**

Tlaquepaque, Jalisco. Junio de 2023.

Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

Optimización de la validación eléctrica y reducción de tiempos de diagnóstico en relojes

---

ESTUDIO DE CASO que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **LUIS EDUARDO ARREDONDO SANDOVAL**

Tutor: **DR. MANUEL SALIM MAZA**

Tlaquepaque, Jalisco. Junio de 2023.

ACKNOWLEDGEMENTS

The author would like to express gratitude to the ITESO school for their tuition discount and for facilitating class progress during the COVID pandemic, to INTEL company, including my manager for approving financial support to cover part of the cost, to the professors for their dedication and patience, to my director, Dr. Manuel Salim Maza for sharing his knowledge and guiding me in the development of this work.

AGRADECIMIENTOS

El autor desea dar las gracias a la escuela ITESO por el descuento en la colegiatura y por facilitar el progreso de las clases durante la pandemia de COVID, a la empresa INTEL y a mi mánager por aprobar el apoyo económico para cubrir una parte del costo, a los profesores por su dedicación y paciencia, a mi director, el Dr. Manuel Salim Maza por compartir su conocimiento y guiarme en el desarrollo de este trabajo.

DEDICATION

I dedicate this work to my wife María de Jesús Ávila Rodriguez, for all her support, for taking care of the children while I concentrated and for feeding me while I was taking classes or doing homework; to my eldest son Erick Eduardo for encouraging me by telling me that this was possible; to my younger sons Marco Leonardo and Luis Angel for waiting long hours for me to play with them and letting me study; to my mother María de los Ángeles, my father Ramiro Arredondo and my siblings for their moral support.

DEDICATORIA

Dedico este trabajo a mi esposa María de Jesús Ávila Rodriguez, por todo su apoyo, por cuidar a los niños, por dejarme concentrar y por alimentarme mientras tomaba clases o hacia tarea; a mi hijo mayor Erick Eduardo por darme ánimos diciéndome que si se podía; a mis hijos más pequeños Marco Leonardo y Luis Angel por esperarme largas horas para jugar con ellos y dejarme estudiar; a mi madre Maria de los Ángeles, a mi padre Ramiro Arredondo y a mis hermanos por su apoyo moral.

ABSTRACT

The Electrical Validation and Debugging, at the system level, of the Reference Clock signals, requires a lot of Signal Integrity and High-Frequency knowledge and skills. This work intends to improve the actual validation methodology and to increase technical knowledge, so our validation team could find defects and root causes sooner. We are analyzing, modeling, and simulating the top four debug cases seen during past validation cycles, by designing and using clock buffers with controlled impedance, as well as transmission lines. The result obtained is that, besides the great knowledge and skills that have been created, we're also observing a very good correlation between simulation and real product behavior.

RESUMEN

La Validación Eléctrica y depuración, a nivel de sistema, de las señales de reloj de referencia, requiere muchos conocimientos y habilidades de Integridad de Señal y Alta Frecuencia. La intención de este trabajo es mejorar la metodología de validación actual e incrementar el conocimiento técnico, de esta manera nuestro equipo de validación podrá encontrar defectos y causas raíz rápidamente. Estaremos analizando, modelando y simulando los principales cuatro casos de depuración vistos en los ciclos de validación anteriores, diseñando y utilizando búferes de reloj con impedancia controlada, así como líneas de transmisión. El resultado obtenido es que, aparte de haber creado habilidades y conocimiento, también estamos observando muy buena correlación entre la simulación y el comportamiento real de nuestros productos.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **4**

AGRADECIMIENTOS **5**

DEDICATION **6**

DEDICATORIA **7**

ABSTRACT **8**

RESUMEN **9**

TABLE OF CONTENTS **10**

LIST OF FIGURES **13**

LIST OF TABLES **16**

LIST OF ACRONYMS AND ABBREVIATIONS **17**

1. INTRODUCTION **18**

1.1. BACKGROUND 19
1.2. JUSTIFICATION 19
1.3. THE PROBLEM 20
1.4. HYPOTHESIS 20
1.5. OBJECTIVES 21
1.5.1. General Objective: 21
1.5.2. Specific Objectives: 21
1.6. CONTRIBUTION OF SCIENTIFIC AND TECHNOLOGICAL INNOVATION 21

2. THE STATE OF THE ART **22**

2.1. THE ELECTRICAL VALIDATION 23
2.2. THE DEBUG 25

3. THEORETICAL FRAMEWORK AND HISTORICAL DEBUG CASES **26**

3.1. THE ELECTRICAL VALIDATION OF CLOCKS AT THE SYSTEM LEVEL 27
3.2. DEBUGGING CLOCK DEFECTS 27
3.3. IMPACT ANALYSIS 28
3.4. ELECTRICAL PARAMETERS REVIEW 29

4. CLOCK BUFFER ARCHITECTURE **33**

4.1. THE PMOS AND NMOS MODELS 34
4.1.1. Configuring ADS to use the Pmos/Nmos model 34
4.2. THE INVERTER 35
4.3. NMOS LEG TUNING 37
4.4. PMOS LEG TUNING 40
4.5. PROGRAMMABLE IMPEDANCE DRIVER 42
4.6. NMOS RCOMP FOR A 50Ω DRIVER 42
4.7. PMOS RCOMP FOR A 50Ω DRIVER 45

4.8. NMOS AND PMOS RCOMP FOR A 33Ω DRIVER 47
4.9. SINGLE-ENDED NON-IDEAL 50Ω TRANSMISSION LINE 48
4.10. DIFFERENTIAL NON-IDEAL 100Ω TRANSMISSION LINE 51
4.11. SINGLE-ENDED BUFFER 52
4.12. COMPLEMENTARY BUFFER 56

5. CASE STUDIES 60

5.1. THE "UNDERSHOOT" CASE STUDY 61
5.1.1. Simulation Scenario # 1 62
5.1.2. Simulation Scenario # 2 63
5.1.3. Simulation Scenario # 3 64
5.1.4. Conclusion 64
5.2. THE "JITTER" CASE STUDY 65
5.2.1. Noise Frequency Effect 65
5.2.2. System Variation Effect 67
5.2.3. Conclusion 68
5.3. THE "FREQUENCY PPM" CASE STUDY 69
5.3.1. Parts per Million 69
5.3.2. Colpitts LC Oscillator, Frequency Tuning 70
5.3.3. Ppm and Jitter, Inverters vs. CML, noise in supply voltage 72
5.3.4. Conclusion 74
5.4. THE "SLEW RATE" CASE STUDY 75
5.4.1. Slew Rate 75
5.4.2. Simulations to Affect Slew Rate 75
5.4.3. Conclusion 77

6. CONCLUSIONS 78

6.1. CONCLUSIONS 79
6.2. FUTURE WORK 79

BIBLIOGRAPHY 80

APPENDIX A. MEASUREMENTS IN LTSPICE 82

A.1 Measurement in Colpitts circuit at 32.768kHz 82
A.2 Measurement in Colpitts circuit at 100MHz 84
A.3 Measurement in cascaded Inverters circuit 84
A.4 Measurement in cascaded CMLs circuit 87

APPENDIX B. PHASE LOCK LOOP (PLL) 91

B.1 Supply Voltage and Reference Input 91
B.2 Phase Detector and Charge Pump 92
B.3 Loop Filter 92
B.4 Voltage Controlled Oscillator (VCO) 93
B.5 Measurements 94
B.6 Size of the transistors 95
B.7 Spice directives for the measurements 95
B.8 Spread Spectrum Clock (SSC) 97
B.9 Conclusion 98

APPENDIX C. WCS, TYP, BCS, AND MONTE-CARLO ANALYSIS 99

C.1 Wcs, Typ, Bcs Simulations..... 99
C.2 Monte Carlo Simulations..... 102
C.3 Conclusion..... 105

LIST OF FIGURES

Figure 1: 100MHz Clock Signal. a) Few clock-cycles. b) More than 100,000 clock cycles. 23
Figure 2: Results report. a) Config screen. b) Phase Jitter. c) Peak to Peak and RMS Jitter. 23
Figure 3: Measurements performed with Dpojet from Tektronix. 24
Figure 4: Overshoot and Undershoot. 29
Figure 5: Jitter. 30
Figure 6: Frequency ppm. 30
Figure 7: Slew Rate. 30
Figure 8: Basic PLL Configuration. 31
Figure 9: Cross Talk. 31
Figure 10: Phase Noise. 31
Figure 11: Spread Spectrum Clock Profile. 32
Figure 12: Power Management States. 32
Figure 13: Duty Cycle. 32
Figure 14: NMOS and PMOS models. 34
Figure 15: Configuration of Model in ADS. 34
Figure 16: CMOS Inverter. 35
Figure 17: Pmos to Nmos compensation. 35
Figure 18: In vs. Out plot, sweeping "In" voltage. 36
Figure 19: In vs. Out, applying a square signal. 37
Figure 20: Nmos leg. 37
Figure 21: Nmos Width adjustment circuit. 38
Figure 22: Selected Nmos Width. 39
Figure 23: Pmos leg. 40
Figure 24: Pmos Width adjustment circuit. 40
Figure 25: Selected Pmos Width. 42
Figure 26: Nmos Leg. a) Circuit. b) Symbol. 43
Figure 27: Nmos Rcomp verification circuit. 44
Figure 28: Nmos Rcomp circuit simulation result. 45
Figure 29: Pmos Leg. a) Circuit. b) Symbol 45
Figure 30: Pmos Rcomp verification circuit. 46
Figure 31: Pmos Rcomp circuit simulation result. 47
Figure 32: Nmos Rcomp simulation for a 33Ω driver. 47
Figure 33: Pmos Rcomp simulation for a 33Ω driver. 48
Figure 34: JLCPCB track structure for Single-Ended TL. 49
Figure 35: JLC7628 PCB Stack-up. 49
Figure 36: Line Calculator, in ADS. 50
Figure 37: LineCalc in ADS, 50Ω Single-Ended TL. 50
Figure 38: JLCPCB track structure for Differential TL. 51
Figure 39: LineCalc in ADS, 100Ω Differential TL. 52
Figure 40: Single-Ended buffer. a) Impedance Control. b) Single-Ended Buffer. 53
Figure 41: MLIN Transmission Line. 53
Figure 42: 10pF Capacitor Load. 54
Figure 43: Smallest Inverter, Sinusoidal instead of Square. 54

FIGURE 44: INVERTER FOR BUFFER, THE ADJUSTMENT CIRCUIT. 55
FIGURE 45: INVERTER ADJUSTED, THE SQUARE SIGNAL AT FAR END. 56
FIGURE 46: INVERTER. A) CIRCUIT. B) SYMBOL. 56
FIGURE 47: COMPLEMENTARY BUFFER. 57
FIGURE 48: IMPEDANCE CONTROL FOR 50Ω. A) PMOS. B) NMOS. 58
FIGURE 49: COMPLEMENTARY BUFFER DRIVING A 100Ω DIFFERENTIAL TL. 59
FIGURE 50: UNDERSHOOT, SCENARIO 1. 62
FIGURE 51: UNDERSHOOT, SCENARIO 2. 63
FIGURE 52: UNDERSHOOT, SCENARIO 3. 64
FIGURE 53: POWER SUPPLY PLUS NOISE. 65
FIGURE 54: JITTER PP VS. NOISE FREQUENCY. 66
FIGURE 55: WAVEFORM AND DENSITY, FREQUENCY EFFECT. 67
FIGURE 56: JITTER PP VS. SYSTEM CONFIGURATION. 67
FIGURE 57: WAVEFORM AND DENSITY, BUFFER, LENGTH, LOAD, AND NOISE EFFECT. 68
FIGURE 58: COLPITTS OSCILLATOR TANK CIRCUIT. 70
FIGURE 59: OSCILLATOR LC COLPITTS AT 32.768KHZ. 70
FIGURE 60: OSCILLATOR LC COLPITTS AT 100MHZ. 71
FIGURE 61: THREE CASCADED INVERTERS. 72
FIGURE 62: THREE CASCADED CML. 73
FIGURE 63: PPM RESULTS, CML VS. INVERTER. 74
FIGURE 64: JITTER RESULTS, CML VS. INVERTER. 74
FIGURE 65: EYE DIFF PROBE SETUP. A) EYE DIFF PROBE. B) CONFIG SCREEN. 76
FIGURE 66: RISE SLEW RATE VS. SYSTEM CONFIGURATION. 76
FIGURE 67: SPICE DIRECTIVES. A) MENU. B) EDITOR. 82
FIGURE 68: COLPITTS FOR 32.768KHZ SIMULATION. 82
FIGURE 69: OSCILLATOR LC COLPITTS AT 100MHZ. 84
FIGURE 70: THREE CASCADED INVERTERS. 85
FIGURE 71: THREE CASCADED CML. 87
FIGURE 72: BASIC PLL CONFIGURATION. 91
FIGURE 73: PLL CIRCUIT, IMPLEMENTED IN LTSPICE. 91
FIGURE 74: PHASE FREQUENCY DETECTOR AND CHARGE PUMP. 92
FIGURE 75: PLL BANDWIDTH. 93
FIGURE 76: VOLTAGE CONTROLLED OSCILLATOR. 94
FIGURE 77: SPREAD SPECTRUM ADDITION TO PLL. 97
FIGURE 78: TRIANGULAR SSC SIGNAL. 97
FIGURE 79: VREF VS. VOUT WITH SSC. A) FFT ANALYSIS. B) ZOOM AT 100MHZ. 98
FIGURE 80: VREF VS. VOUT WITHOUT SSC. A) FFT ANALYSIS. B) ZOOM AT 100MHZ. 98
FIGURE 81: THREE CASCADED CML. 99
FIGURE 82: MODELS TWEAKED. TOP LEFT: WCS. TOP RIGHT: TYP. BOTTOM: BCS 100
FIGURE 83: INCLUSION OF THE MODEL. A) WCS. B) TYP. C) BCS. 100
FIGURE 84: BCS, TYP, WCS, PPM RESULTS. 100
FIGURE 85: BCS, TYP, WCS, JITTER RESULTS. 101
FIGURE 86: BCS, TYP, WCS, RISE TIME, FALL TIME, MAX RESULTS. 101
FIGURE 87: BCS, TYP, WCS, RISE TIME, FALL TIME, MIN RESULTS. 102
FIGURE 88: MONTE CARLO CONFIGURATION OF THE NMOS. A) M3. B) M7. C) M9. 102

FIGURE 89: MONTE CARLO RESULTS, PART 1. 103
FIGURE 90: MONTE CARLO RESULTS, PART 2. 103
FIGURE 91: MONTE CARLO RESULTS, PART 3. 103
FIGURE 92: MONTE CARLO RESULTS, PART 4. 104
FIGURE 93: MONTE CARLO RESULTS, PART 5. 104

LIST OF TABLES

TABLE 1: RESULTS REPORT, GENERATED WITH THE SKYWORKS TOOL. 23
TABLE 2: ELECTRICAL PARAMETER FAILED VS. IMPACT. 28
TABLE 3: PMOS WIDTH ADJUSTMENT OF THE INVERTER. 36
TABLE 4: NMOS WIDTH ADJUSTMENT. 38
TABLE 5: PMOS WIDTH ADJUSTMENT 41
TABLE 6: L, W, AND R, FOR 50Ω AND 250Ω NMOS LEGS. 42
TABLE 7: L, W, AND R, FOR 50Ω AND 250Ω PMOS LEGS. 45
TABLE 8: TUNING OF THE INVERTER FOR THE BUFFER. 54
TABLE 9: COLPITTS 32P768KHZ, SIMULATING 240MS 71
TABLE 10: COLPITTS 100MHZ, SIMULATING 400US 72
TABLE 11: PARAMETER CHANGE VS. SLEW RATE. 77
TABLE 12: ELMORE'S DELAY. 77
TABLE 13: DESIGN GOAL VS. BANDWIDTH. 93
TABLE 14: PLL LOOP FILER ADJUSTMENT, C1 SWEEP. 94
TABLE 15: PLL LOOP FILTER ADJUSTMENT, C2 SWEEP 94
TABLE 16: PLL LOOP FILTER ADJUSTMENT, R1 SWEEP 95
TABLE 17: MONTE CARLO, JITTER AT OUTPUT V4, AN EXAMPLE OF RESULTS. 104

LIST OF ACRONYMS AND ABBREVIATIONS

ADS	Advanced Design System (simulation tool from Keysight)
Client segment	The market segment for Laptop and Desktop computers
CML	Current Mode Logic
CMOS	Complementary Metal Oxide Semiconductor
DC	Direct Current
Debug	Process of identifying and correcting errors or issues in electronic circuits
Device segment	The market segment for mobile devices like Tablets and Cellphones
EMI	Electromagnetic Interference
FR4	Flame Retardant level 4
GS/s	Giga Samples per second
IC	Integrated Circuit
Imgclk	Image Clock, for camera circuit
ITESO	Instituto Tecnológico y de Estudios Superiores de Occidente Western Institute of Technology and Higher Education
Nmos	N-channel Metal Oxide Semiconductor
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
PLL	Phase Lock Loop
Pmos	P-channel Metal Oxide Semiconductor
Post-silicon	Silicon physically built
PPM	Parts per Million
Pre-silicon	Before building silicon
PVT	Process, Voltage, Temperature
Rcomp	Compensating Resistor
RVP	Reference Validation Platform
Rx	Receiver
SCLK	System Clock
Server segment	The market segment for server systems
SoC	System on Chip
SR	Slew Rate
SSC	Spread Spectrum Clock
Stepping	The version or release of the IC
Test Plan	Plan or list of electrical tests for validation
Tx	Transmitter
VCO	Voltage Controlled Oscillator
Vdd	Drain Voltage, a typical name for a circuit's supply voltage
Z	Impedance

1. INTRODUCTION

Summary: *In this chapter, we will discuss the electrical validation of clock signals at the system level. We will explain the problem, justify it, and present hypotheses. We will also define general and specific objectives to achieve in this work.*

1.1. Background

System Clock (SCLK) is the name of a circuit, which is inside of an Integrated Circuit (IC), that generates and distributes clock signals to several components, inside and outside of the IC, so they can perform their function. The clock signals in a system are, comparably, what the heart is for the human body, these are pulses that set the rhythm at which diverse circuits get synchronized and work in harmony.

The SCLK circuit, besides clocking internal circuitries within the IC, provides clock signals to several components in a system, at the printed circuit board (PCB) level, also known as a "motherboard" or platform, with multiple ICs, and connectors for external electronic cards. At the system level, the SCLK must manage several frequencies, trace lengths, and market segments, such as device, client, or server segments. More details are in section 3.1.

Similarly, to any commercialized product, the SCLK circuit must pass through the electrical validation process before mass production. The validation process, at a high level, consists of several phases, such as readiness or planning (during pre-silicon), test plan execution (during post-silicon), repetitions, and approval.

The execution phase includes debugging in case of unsatisfactory results. The repetitions occur when improvements are done to the prototype, and approval is for mass production when all technical requirements are met.

During the test plan execution, it is expected that the SCLK works within electrical specifications when it is subjected to diverse stress types, for example, supply voltage and temperature variations. Also, we must test ICs that resulted to be faster or slower than the average speed of the produced units, this is called process corners. By testing at several PVT combinations (process, voltage, temperature), we can discover design problems, sooner, during the validation [1].

1.2. Justification

It is very important to uncover problems in the SCLK circuit in the early phases of the design or validation and to look for the root cause. If defects are found in late phases, it is very expensive to correct them, and customers would be impacted. One example of the cost of not discovering a problem in the design, prototype, or validation phases, is the so-called "Pentium flaw", which cost $475 million to the company [2].

The phases of the product design process are:

The idea – although building something "new" can be creative, many of the best ideas are the result of working on a product that already exists.

Investigation – We need to test the idea first, before producing the product, this way we can guarantee that we are creating a product that people will buy, for example, by making online surveys and asking for opinions in forums, among others.

Planning – to draw an initial draft of the product, as detailed as possible, with labels explaining its characteristics and functions, to make it clear what components or materials will be needed to create the product.

Prototypes – the objective is to create a product, that will be used as a sample for mass production. It is unlikely to occur, to have a finished product on the first try, it's more probable to experiment with several versions of the product and make improvements until we get what we're looking for.

Validation – every prototype must pass through a validation phase, this is, to put the product to the test, and discover its functionality and specifications meeting level, as well as to uncover defects that must be corrected and implemented in the next prototype version.

Supplies – When creating the prototype, we need also to create the supply chain, with suppliers, resources, and necessary activities to create the product, and put it in our customer's hands, storage, and shipment.

Look for the lower cost – when completing the previous phases, we'll have a better idea of the cost of producing the product, this way we can determine a price and margin [3].

1.3. The Problem

The electrical validation at the system level, of the SCLK's clocks, requires a lot of Signal Integrity knowledge [4], as well as to perform measurements in several PVT combinations, topology, clock types, etc. This makes the validation cycle so long that it often fits within the validation window, leaving no time for debugging.

The validation timeframe goes from Power-On, which is when the product is just built and ready to get powered, to the Tape-In, which occurs when all needed changes have been applied to create the next stepping of the product [5]. This time usually takes from 7 to 13 weeks, leaving very little or null time to analyze defects. Similarly, being a very complex branch, when there are problems and debugging needs to be done, it takes a lot of time due to the complexity of the circuit and the implementation at the system level.

1.4. Hypothesis

We may currently be running too many tests, such as many PVT combinations that do not provide new information about the health of the SCLK, that is, validating more than needed. It's also probable that the specific tests that we are running, not all are the best to find defects in the clock's circuits.

It is also believed that the lack of certain technical knowledge, about the SCLK and the implementation at the system level, is preventing us from finding the root causes of problems in a more efficient manner. Such technical knowledge could be, electrical parameters or physical phenomenon comprehension, about Jitter, Phase Noise, Slew Rate, Reflections, Impedance mismatches, etc.

We believe that, by developing technical knowledge and implementing some root cause analysis techniques, we could execute a more efficient validation and debugging.

1.5. Objectives

1.5.1. General Objective:

To improve the actual validation methodology and to increase the technical knowledge about SCLK architecture, so our team could easily find defects and their root cause within the validation time frame.

1.5.2. Specific Objectives:

To reduce the clock's electrical validation time from 20% to 30% of the actual time. This is, from 10 weeks down to 7 or 8 weeks, so we can dedicate 2 to 3 weeks to look for the root cause of the defects found, before the tape-in of the next stepping. We're also trying to reduce the debug time by 20% to 30% of the current time.

1.6. Contribution of Scientific and Technological Innovation

This work is a contribution, aiming to optimize the actual clocks electrical validation methodology, as well as, to generate technical knowledge, which helps to debug and root cause the defects more efficiently.

2. THE STATE OF THE ART

Summary: *This chapter shows a summary of the current electrical validation techniques, and some electrical parameters, that once understood, debugging tends to be easier.*

2.1. The Electrical Validation

Currently, there exist several tools to calculate electrical parameters, one example is the "PCIe Clock Jitter Tool" from Skyworks company [6], which analyzes reference clock signals for PCIe and provides a report with an extensive variety of electrical parameters and their values or measurements. The clock signal to be entered in the tool must be taken with an Oscilloscope, able to capture 100,000 or more clock cycles, with a minimum sample rate of 25 GS/s. Figure 1 shows a 100MHz reference clock signal for PCIe.

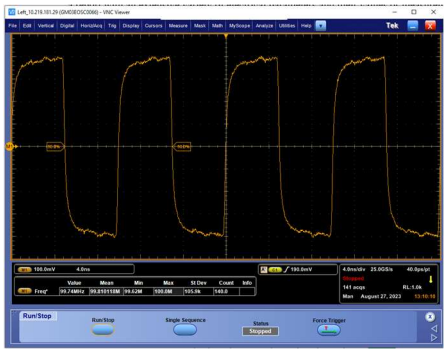
a)  b) 

Figure 1: 100MHz Clock Signal. a) Few clock-cycles. b) More than 100,000 clock cycles.

The signal is saved and introduced into the Skyworks tool, which will perform an analysis and generate a report with the measurements, Table 1 shows an example.

Filter Compliance Summary

Class	Data Rate	Architecture	Specs	Measured Jitter (1)		Spec Limit	Compliance Summary
GEN1	2.5 Gb/s	Common Clock	1.1 , 2.1 , 3.1	33.15 ps	peak-to-peak	86 ps	All PASS
GEN2	5 Gb/s	Common Clock	1.1 , 2.1 , 3.1	1.18 ps	HF RMS	3.1 ps	All PASS
				1.14 ps	LF RMS	3 ps	
GEN3	8 Gb/s	Common Clock	3.1 , 4.0	343.97 fs	RMS	1 ps	All PASS
GEN4	16 Gb/s	Common Clock	4.0	329.75 fs	RMS	500 fs	All PASS
GEN5	32 Gb/s	Common Clock	5.0	120.14 fs	RMS	150 fs	All PASS
GEN6	64 Gb/s	Common Clock	6.0	72.66 fs	RMS	100 fs	All PASS

Table 1: Results report, generated with the Skyworks tool.

Another tool is the "Clock Jitter Tool" from Intel [7], which analyzes the clock signal and provides the rms jitter value, along with a "pass" or "fail" statement. Figure 2 shows an example of the analysis generated by this tool.

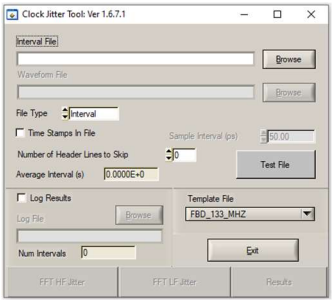
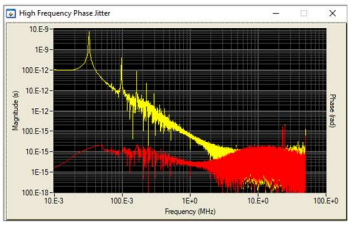
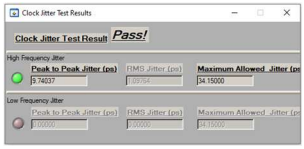
a)  b)  c) 

Figure 2: Results report. a) Config screen. b) Phase Jitter. c) Peak to Peak and RMS Jitter.

Another tool is the Dpojet software from Tektronix [8], which can perform several types of analyses of the clock signal, Figure 3 is an example of the measurements obtained with this tool.

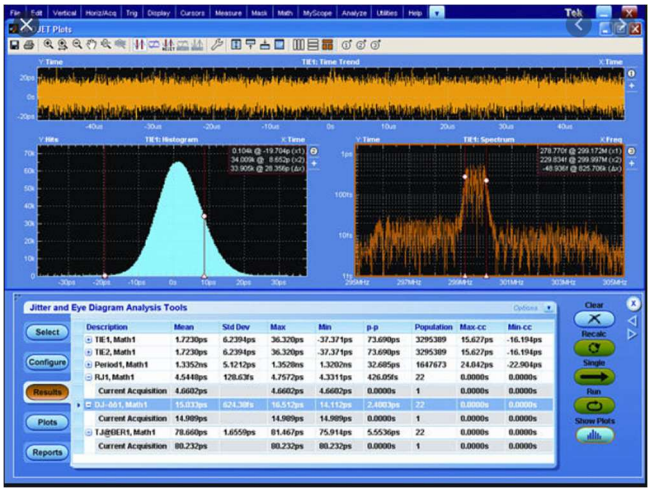


Figure 3: Measurements performed with Dpojet from Tektronix.

2.2. The Debug

Debug is the ability to root cause problems, in this case, on clocks at the system level. It is essential to understand most of the physical phenomena that occur in electronic systems, such as Jitter, Phase Noise, Slew Rate, Reflections, Impedance Mismatch, etc. There exists a lot of bibliographies that explain these phenomena, many times these are complicated themes, and it is difficult to associate them with the system.

In this work, it is planned to cover multiple electrical parameters, and how they are affected by the different characteristics of the system, such as topologies, trace lengths, terminations, etc. With this knowledge, the validator can do it reversed way, to attack a problem by knowing what could be affecting that specific physical phenomenon, then make some experiments or simulations, and find the root cause sooner.

There are many methodologies to root cause problems, for example, the 5 whys, which consists of making consecutive "why" questions on why something is occurring, then asking another "why" to the answer, and continuing up to the root cause. Two iterations would be enough, or typically 5, but more could be needed.

Another methodology is the "changes analysis" or "events analysis", which consists of carefully analyzing the changes made, from the moment that the failure didn't exist up to this moment, this way we can find which of the changes provoked the failure to appear.

Another method is the cause-and-effect fishbone diagram, also known as Ishikawa's diagram, which visually helps to map causes and effects. It can help to identify the possible causes of a problem and encourage us to follow categoric branches to the possible causes until we find the correct one. It is like the 5 whys, but much more visual [9].

3. THEORETICAL FRAMEWORK AND HISTORICAL DEBUG CASES

Summary: *In this chapter, the theoretical and conceptual foundations for the electrical validation of clocks are presented, as well as the parameters or electrical phenomena whose lack of knowledge makes it difficult to find root causes in clock integrated circuits and their implementation in a system. We can also see the historic list of defects that have been debugged during the validation of past products, in which electrical parameters involved are explained to generate technical knowledge.*

3.1. The electrical validation of clocks at the system level

Performing system-level electrical validation of clocks means, validating that clock signals arrive with signal integrity to the receiver circuits that require them. This document is focused on systems of the "Client" market segment. Although it is not focused on the "Devices" or "Server" segments, many of the concepts can be applied as well. The Client segment refers to computer systems like Laptops and Desktop. The Devices segment refers to mobile devices like Tablets and Cellphones. Similarly, the Server segment refers to Server systems.

Electrical Validation consists of two main phases, pre-silicon, and post-silicon. Pre-silicon is when the integrated circuit hasn't been built yet, many activities occur in this phase for our group, such as product specification study, board design guideline review, specifications, schematic and motherboard analysis, and test plan creation. Post-silicon starts at Power-On, which is the first time the physical Integrated Circuit is being powered up, and The Test Plan is executed, which is when several tests are performed on the product, and it finishes at the Production Release milestone. During the execution of the Test Plan, when defects are found, debugging must be done to find the root cause as soon as possible. Though the debug is not a planned phase, it is inherently added to the process when failures occur.

This work focuses mainly on reference clocks for PCIe, which are differential clocks that have a frequency of 100MHz and travel distances of 3 to 12 inches across the motherboard before reaching the receiver, using FR4 board material in either micro-strip or strip-line traces. Clocks, like any other signal, are susceptible to being contaminated by other signals as they travel across the board, which generates physical phenomena such as jitter, crosstalk, and noise in general.

In a "Client" system there are many other clocks with several frequencies and topologies, including Single-Ended clocks.

In this document, some failure cases will be reviewed on both, differential and single-ended clocks.

3.2. Debugging clock defects

There is no specific debugging methodology or step-by-step guide to finding the root cause of a clock problem on the platform. Instead, based on the electrical parameter where the fault is observed, a series of experiments are implemented to help expose and understand the defect. There are many methodologies to find the root causes of problems, as described in section 0.

A more efficient debug can be performed when technical knowledge of parameters or phenomena that occur in electronic systems is available, such as jitter, crosstalk, slew rate, and reflections, among others.

Just like these parameters, many others will be analyzed throughout this work to generate knowledge that serves to discover the root causes of failures more efficiently and promptly in the validation phase.

3.3. Impact Analysis

The historical debug cases are analyzed in Table 2, The table contains six columns, and the meaning of each column is:
"Electrical Parameter Failed" – This is the parameter or specification that failed to meet the spec during validation.
"Clock affected" – This is the name of the clock or the circuit that showed the malfunction.
"Impact" – It shows the actions to do to fix the problem, either redesigning the board or silicon or re-validation efforts.
"Recurrence" – In this column there's a quantification of how frequently the issue has been seen in previous projects.
"Time to fix" – This is the average time that it takes to fix the problem.
"Rating" – This is the weighing of every parameter, the value in this column shows the result of multiplying the factors from columns: Impact, Recurrence, and Time to fix, the higher the value, the more problematic issue.

Table 2: Electrical parameter failed vs. Impact.

Electrical Parameter Failed	Clock affected	Impact	Recurrence	Time to fix (weeks)	Rating
Undershoot	Imaging gpio clock for Camera	5- Very high, it might trigger board redesign.	3- Medium, only seen in one project	20 weeks	300
Jitter	PCIe Serial Reference Clock (SRC)	5- Very high, we need to perform experiments to find the cause, it might trigger board or Silicon circuit redesign or settings workaround.	4- High, seen in some projects	Few to several weeks (10 weeks average)	200
Frequency ppm	Susclk from Real Time Clock	4- High, we need to tune the capacitor value for the board. Replace and remeasure.	5- Very high, seen in many projects	7 weeks	140
Slew Rate	PCIe Serial Reference Clock (SRC)	4- High, we need to collect a lot of data, in several topologies, involve many stakeholders in meetings	5- Very high, seen in many projects	7 weeks	140
Other PLL issues	PLL itself	2- Low impact on our team because we don't own the PLL validation. It could trigger settings changes or Silicon circuit changes. Either way, we need at least to understand PLL basics.	5- Very high, seen in many projects	Few to several weeks (10 weeks average)	100
Cross talk	Crystal circuit, latest in 2015	4- High, it might trigger board redesign.	1- Very low, issue not seen since 2015	20 weeks	80
Phase Noise	Universal Flash Storage (UFS)	4- High, We can't afford the cost of a PXA because it is used only a few weeks per year, and	3- Medium, only seen in one project	5 weeks	60

	Reference Clock	we only have an oscilloscope.			
SSC profile from PLL	PCIe Serial Reference Clock (SRC)	2- Low, we typically need to tune two settings in the PLL.	5- Very high, seen in many projects	3 weeks	30
Power Management states	It could be any	2- Low impact on our team because we don't own the PM validation. It could trigger Software flow changes. We could study more on this in future analyses.	1- Very low, our team hasn't been involved in these debugs	A few weeks (3 weeks)	6
Duty Cycle	Imaging gpio clock for Camera	1- Very low, last time it was seen by the pre-silicon team, it was not seen during post-silicon validation	1- Very Low, issue not seen yet during validation	3 weeks	3

3.4. Electrical Parameters Review

Explaining every electrical parameter from Table 2.

Undershoot: This is a type of signal integrity issue that occurs when a signal's voltage drops below its intended low level, Figure 4, It can cause incorrect data transmission, especially in digital circuits, or even damage to the circuits [4]. Undershoot occurs when the impedance of the signal path is not matched properly, leading to reflections that cause the voltage to drop below the expected level. This can be caused by factors such as improper termination, incorrect trace lengths, or insufficient drive strength. To mitigate undershoot, designers use techniques such as adding series resistors to limit the current flow and prevent overshoot/undershoot, adjusting the drive strength of the signal source, or using impedance matching techniques to ensure that the signal path is properly matched.

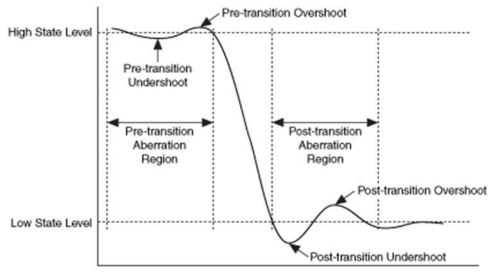


Figure 4: Overshoot and Undershoot.

Jitter: It is a measure of significant variations of a digital signal from its ideal position in time, affecting signal phase, width, and period, Figure 5. Jitter is an unwanted factor that can lead to an increase in the bit error rate of a serial link [10]. One type of jitter is Random, which is unbounded and can take any shape, also known as Gaussian jitter, the other type is Deterministic, which is a predictable jitter, it is further sub-categorized as period jitter, bounded uncorrelated jitter, and data-dependent jitter.

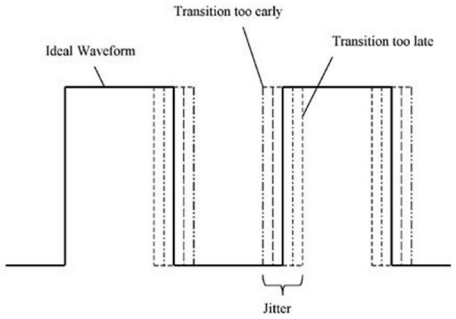


Figure 5: Jitter.

Frequency ppm: This is a measure of frequency precision, Figure 6, represents the output variation due to external conditions like temperature, voltage, and load changes, as well as aging. It is typically expressed in parts per million (ppm) [11]. For example – if a quartz oscillator has an output frequency of 1MHz (1,000,000 Hz) and it has a Frequency Stability of 5ppm, it will vary in frequency by 5 Hz.

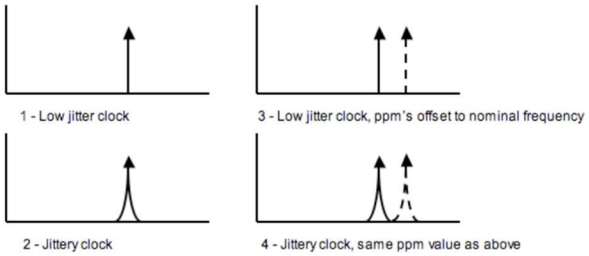


Figure 6: Frequency ppm.

Slew Rate: It is defined as the rate at which an electronic circuit or device can change its output voltage or current, Figure 7. It is often expressed in volts per nanosecond (V/ns) or amperes per nanosecond (A/ns) [12].

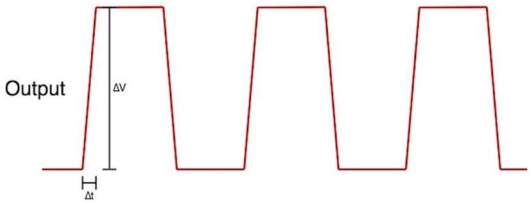


Figure 7: Slew Rate.

Other PLL issues: These refer to a variety of issues seen in PLLs, Figure 8, such as locking, lock time, bandwidth, stability, and others. This document doesn't intend to explain PLL debugging since it is out of our team validation scope, but still, we're trying to explain PLL's functionality for engineers to have a basic understanding of it [13].

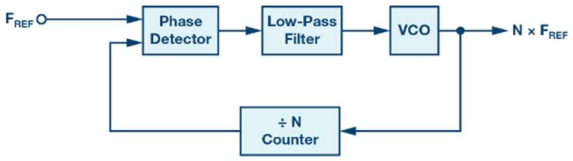


Figure 8: Basic PLL Configuration.

Cross talk: It is one of the signal-integrity problems. It is the transfer of an unwanted signal from one net to an adjacent net and will occur between every pair of nets, Figure 9. The net with the source of the noise is typically called the active net of the aggressor net. The net receiving the noise is called the quiet net or the victim net [14].

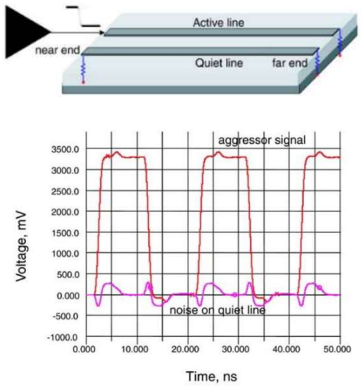


Figure 9: Cross Talk.

Phase Noise: It is typically characterized by a frequency offset from the carrier frequency, and it is usually expressed in units of decibels relative to the carrier (dBc), Figure 10. The lower the dBc value, the better the phase noise performance of the oscillator. Phase noise is defined as the ratio of the noise in a 1-Hz bandwidth at a specified frequency offset, fm, to the oscillator signal amplitude at frequency fo [15].

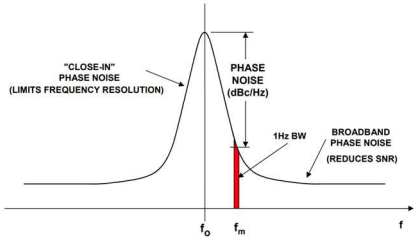


Figure 10: Phase Noise.

SSC profile from PLL: SSC is the variation of the frequency of a clock signal in a controlled way, Figure 11. In the frequency domain, the SSC reduces the peak amplitude of a digital clock signal by shifting the frequency. In other words, the energy of the clock is spread, decreasing the peak values of the radiation, and it can help with EMI reduction [16].

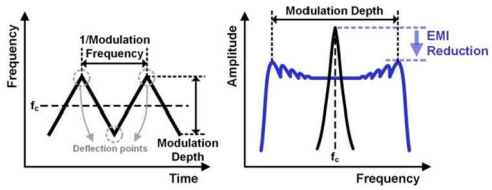


Figure 11: Spread Spectrum Clock Profile.

Power Management states: These relate to states where the power consumption, of a system, is lower or higher. To the user, a computer system appears to be either on or off, there are no other detectable states, Figure 12. However, the system supports multiple power states, such as working, standby, sleep, and hibernating, among others [17].

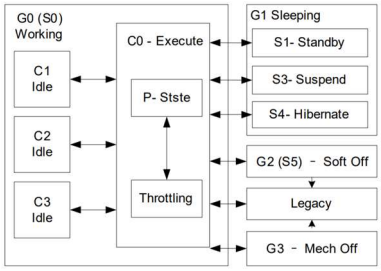


Figure 12: Power Management States.

Duty Cycle: It is the ratio of time a load or circuit is "on" compared to the time the load or circuit is "off", Figure 13. It is sometimes called the "duty factor", and it is expressed as a percentage of "on" time [18]. A 60% duty cycle is a signal that is "on" 60% of the time and "off" the other 40%.

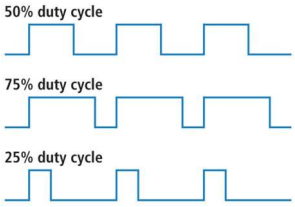


Figure 13: Duty Cycle.

4. CLOCK BUFFER ARCHITECTURE

Summary: *Many circuits are being designed and explained within this chapter, from CMOS inverters to single-ended and complementary buffers with controlled impedance. The design of the transmission lines is also explained here.*

4.1. The Pmos and Nmos models

One of the tools used during this work is ADS from Keysight company, this tool was learned from the course “High Frequency Electronic Design” in ITESO.

Many of the circuits that we’ll simulate use Pmos and Nmos transistors, and we need to get their model, shown in Figure 14, these models are obtained from the “Predictive Technology Model” web page [19]. I decided to use the 180nm transistor model [20] because this is the technology we studied in the course “Design of Analog Integrated Circuits” so I could use the license owned by the ITESO if needed.

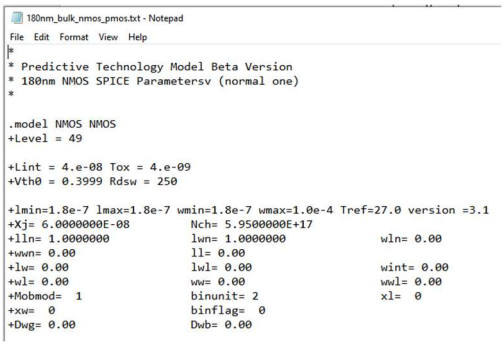


Figure 14: NMOS and PMOS models.

4.1.1. Configuring ADS to use the Pmos/Nmos model

In ADS, under “Data Items”, select “Netlist”, then double click on the “Netlist Include” object from the schematic, select “IncludeFiles[1]=”, browse for the model’s .txt file, click apply and Ok. Figure 15.

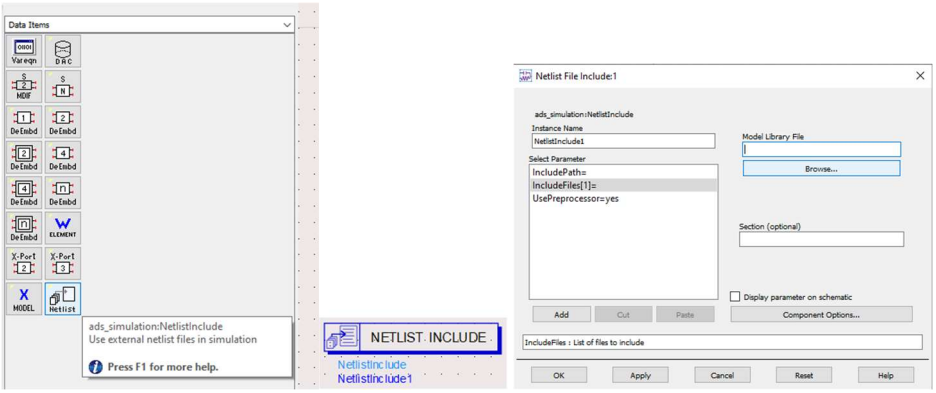


Figure 15: Configuration of Model in ADS.

4.2. The Inverter

One of the basic CMOS circuits is the Inverter, which is built with one Pmos and one Nmos transistor, connected as in Figure 16. When one transistor is On, the other is Off [21].

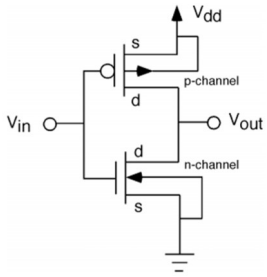


Figure 16: CMOS Inverter.

Though Nmos and Pmos supposedly work the same, in reverse to each other, this is not physically true, electrons in Nmos move faster than holes in a Pmos; therefore, Pmos transistors must be compensated, and this is done by increasing its "width" to more than twice the Nmos "width". The circuit in Figure 17 is used to make this compensation.

According to my notes from the course "Design of Analog Integrated Circuits", for a 180nm technology, the minimum channel "length" is $L_{min} = 180nm$, and it is recommended to use $L = 200nm$ for layout. The minimum channel "width" is $W_{min} = 320nm$, which is derived from the rules of escalation of the technology, and it takes "distance to edge", "contact size" and "contact separation" into consideration, it is recommended to use $W = 360nm$ for layout, which is twice the L_{min}, so we avoid manufacturing issues by separating from the minimum sizes.

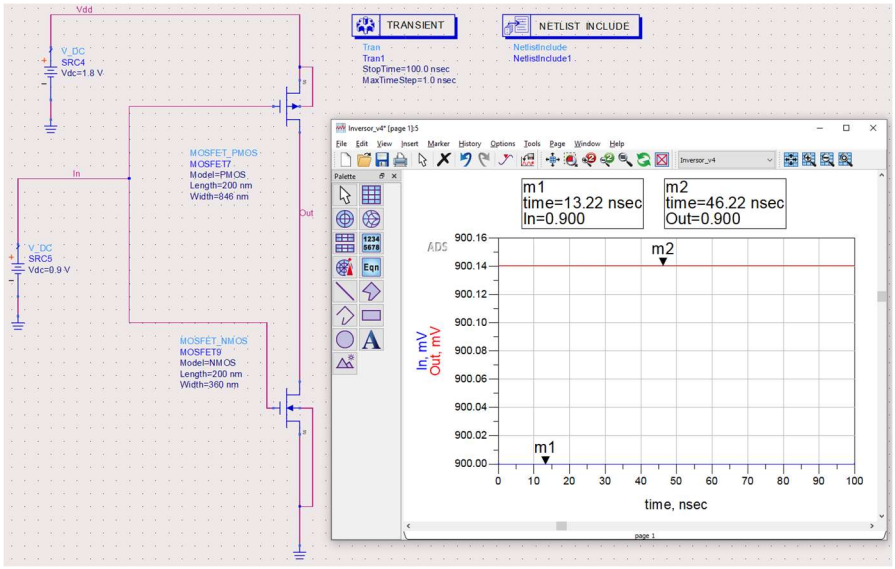


Figure 17: Pmos to Nmos compensation.

Steps to do the compensation: use the smaller recommended Length/Width values for the Nmos, which is 200nm/360nm, then use the smaller recommended Length and twice the Nmos Width for the Pmos, which is 200nm/720nm. Apply 0.9V (which is half of Vdd) to input "In" and measure Output "Out".

Increase or decrease the Pmos Width until "Out" is equal to "In". In this example, "Out" equal to "In" was obtained in Try # 8, Table 3. And Pmos width is 2.35 times Nmos width.

Table 3: Pmos Width adjustment of the Inverter.

Try #	In	Out	Nmos L/W	Pmos L/W
1	0.9V	0.646V	200nm/360nm	200nm/720nm
2	0.9V	0.813V	200nm/360nm	200nm/800nm
3	0.9V	0.994V	200nm/360nm	200nm/900nm
4	0.9V	0.907V	200nm/360nm	200nm/850nm
5	0.9V	0.889V	200nm/360nm	200nm/840nm
6	0.9V	0.898V	200nm/360nm	200nm/845nm
7	0.9V	0.902V	200nm/360nm	200nm/847nm
8	**0.9V**	**0.90014V**	**200nm/360nm**	**200nm/846nm**

Another way to do the compensation is by sweeping Input "In" from 0 to 1.8V, plotting In vs. Out signals, then increasing or decreasing Pmos Width till "In" and "Out" cross at 0.9V. Figure 18.

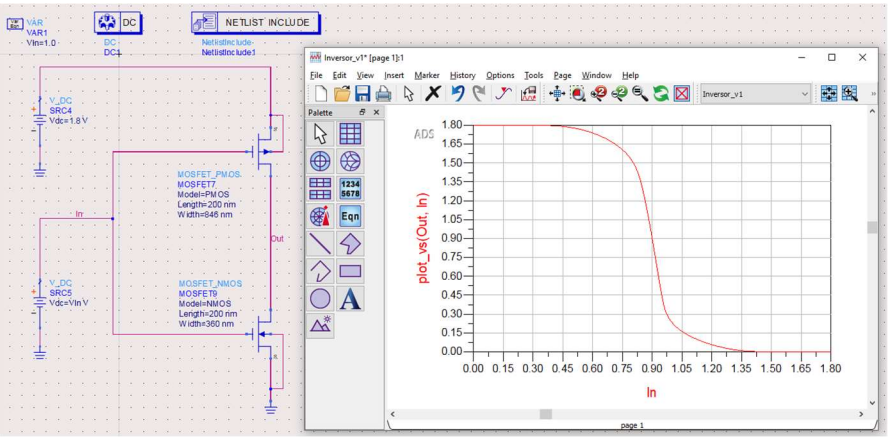


Figure 18: In vs. Out plot, sweeping "In" voltage.

The "Out" signal should have the same amplitude as the "In" signal but delayed 180°, so when the Input is a 1 logic, the Output is a 0 logic, and the other way around. This can be verified by applying a square signal to the Inverter and measuring the output. Also, the cross point, where signals are crossing each other, occurs at half Vdd, which is 0.9V. Figure 19.

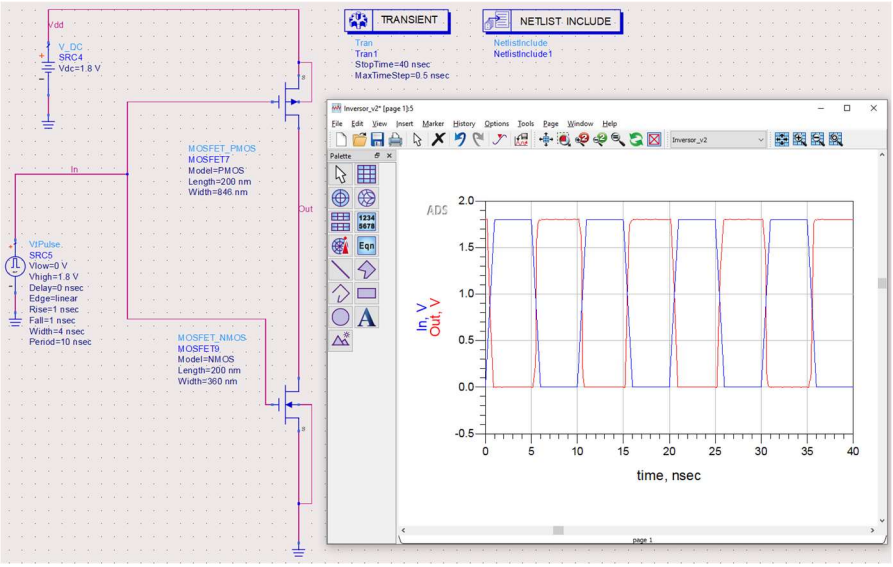


Figure 19: In vs. Out, applying a square signal.

4.3. Nmos Leg Tuning

The Nmos leg is one of the circuits of a complementary buffer with controlled impedance. When designing a 50Ω driver, the three typical designs are: 90/10 design, which means 90% resistor, 10% transistor, similarly, an 80/20 design means 80% resistor, 20% transistor, as well as 70/30 design meaning 70% resistor, 30% transistor. The resistors in silicon show fewer variations to process and temperature because their response is linear to voltage, while the transistors show more variations to process, voltage, and temperature, because the response is not linear, it passes through different operation regions.

In this case, I'm focusing on an 80/20 design, in which, the driver consists of one resistor of 40Ω in series with two Nmos transistors, from Out to Gnd as in Figure 20.

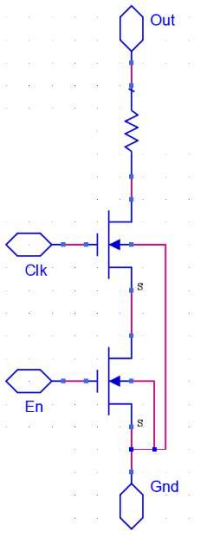


Figure 20: Nmos leg.

One of the Nmos transistors receives the clock signal from the controller, to be transmitted out of the buffer. The other Nmos transistor is gated by an “En” control signal, it helps to enable or disable a specific leg, so we can create a controlled impedance buffer.

The width of the Nmos transistors must be adjusted to get ~5 Ohm dc resistance on each, and both added to the 40Ω resistor, create a 50Ω driver. The circuit in Figure 21 is used to adjust the Width of the Nmos transistors. When using the minimum recommended Length/Width, whose are 200nm/360nm, we are getting 1.793V at Vpad.

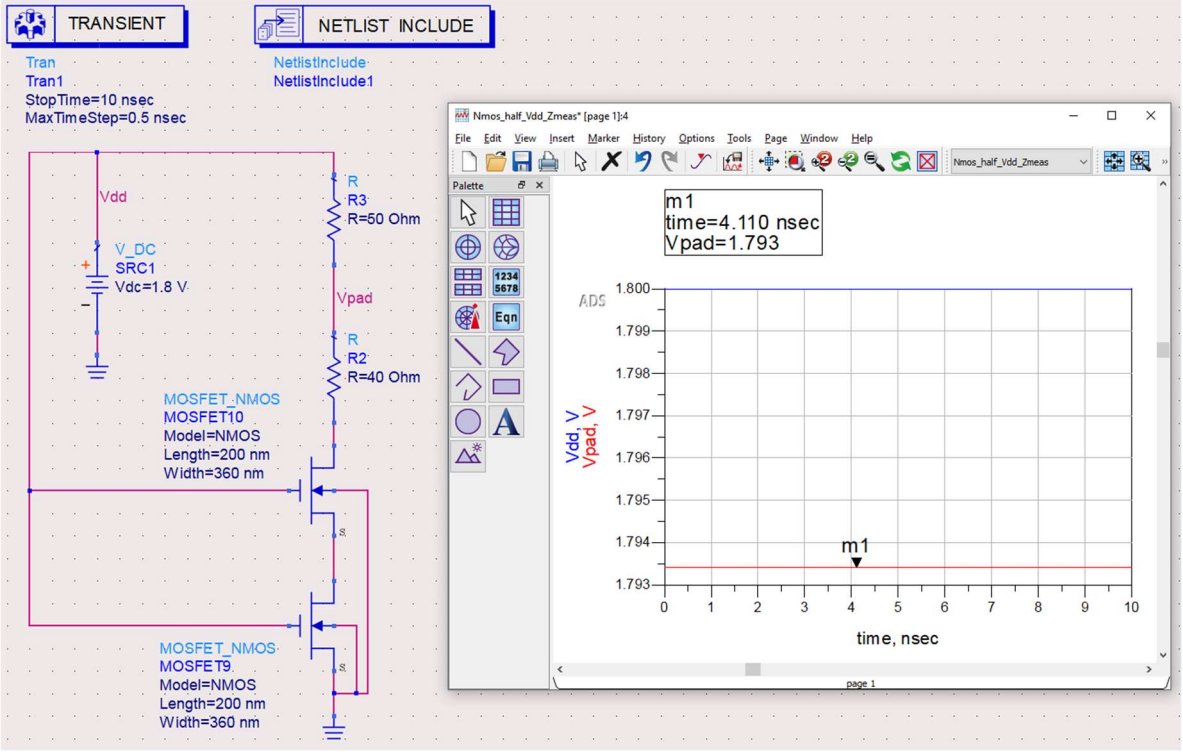


Figure 21: Nmos Width adjustment circuit.

The objective is to have half Vdd, which is 0.9V, at the Vpad node. This can be done by increasing the Width of the Nmos transistors while keeping the minimum recommended Length, of 200nm. Table 4 shows the iterations tried, increasing the Width of the Nmos transistors, and measuring Vpad until we get 0.9V.

Table 4: Nmos Width adjustment.

Nmos 1 and 2, Length	Nmos 1 and 2, Width	Vpad	Error (Vpad – Target)
200nm	360nm	1.793V	893mV
200nm	720nm	1.787V	887mV
200nm	1.44um	1.774V	874mV
200nm	2.88um	1.748V	848mV
200nm	5.76um	1.696V	796mV
200nm	11.52um	1.595V	695mV
200nm	23.04um	1.402V	502mV

200nm	46.08um	1.146V	246mV
200nm	92.16um	0.983V	83mV
200nm	184.32um	0.893V	-7mV
200nm	180.00um	0.896V	-4mV
200nm	175.00um	0.898V	-2mV
200nm	170.00um	0.901V	1mV
200nm	171.00um	0.90049V	0.49mV
200nm	172.00um	0.899929220V	0.07078mV
200nm	173.00um	0.899V	-1mV
200nm	171.90um	0.899985966V	-0.014034mV
200nm	171.85um	0.900014363V	0.014363mV
200nm	**171.875um**	**0.900000162V**	**0.000162mV - Selected**
200nm	172.876um	0.899999594V	-0.000406mV

Figure 22 shows the circuit with the selected values, Length=200nm and Width=171.875um for the Nmos leg and measuring Vpad.

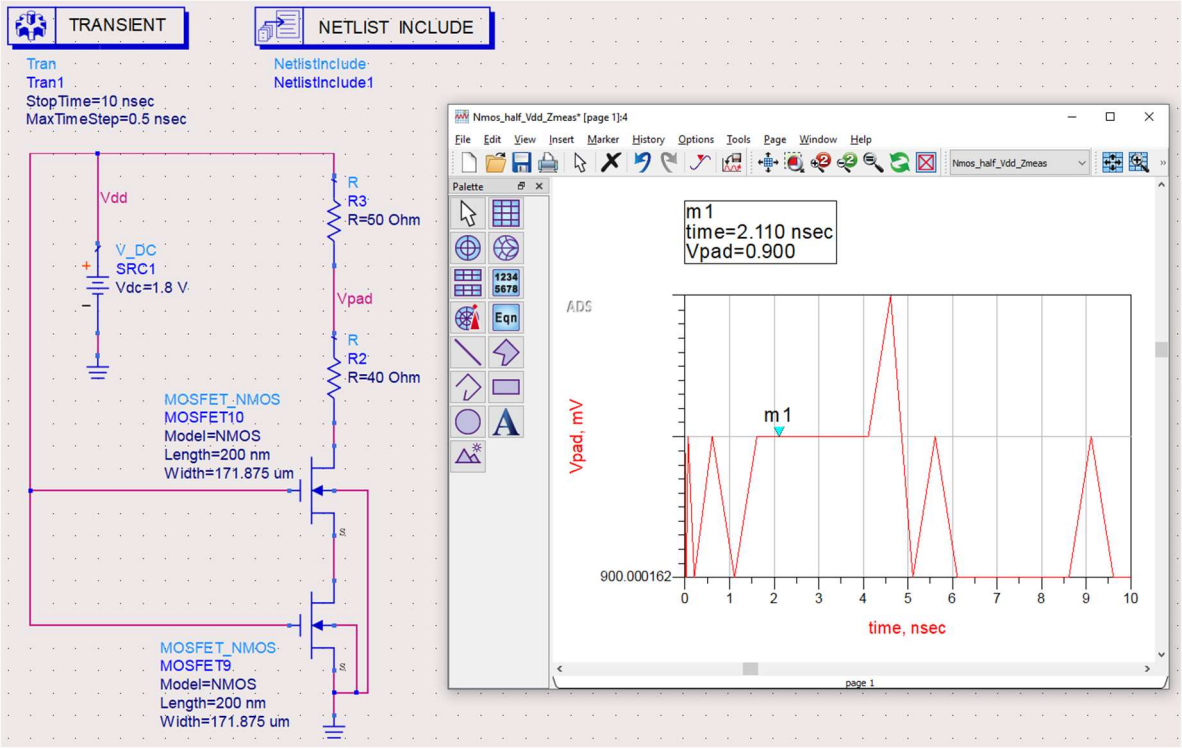


Figure 22: Selected Nmos Width.

4.4. Pmos Leg Tuning

Similarly, the Pmos leg is a circuit as shown in Figure 23, for an 80/20 design, the driver consists of two Pmos transistors in series with one resistor of 40Ω, from Vdd to Out.

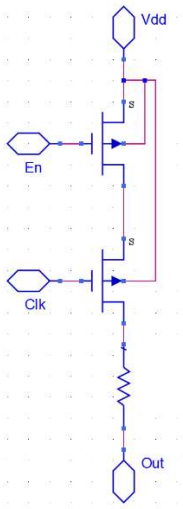


Figure 23: Pmos leg.

The circuit in Figure 24 is used to adjust the Width of the Pmos transistors. When using the minimum recommended Length/Width, whose are 200nm/360nm, we are getting 3mV at Vpad.

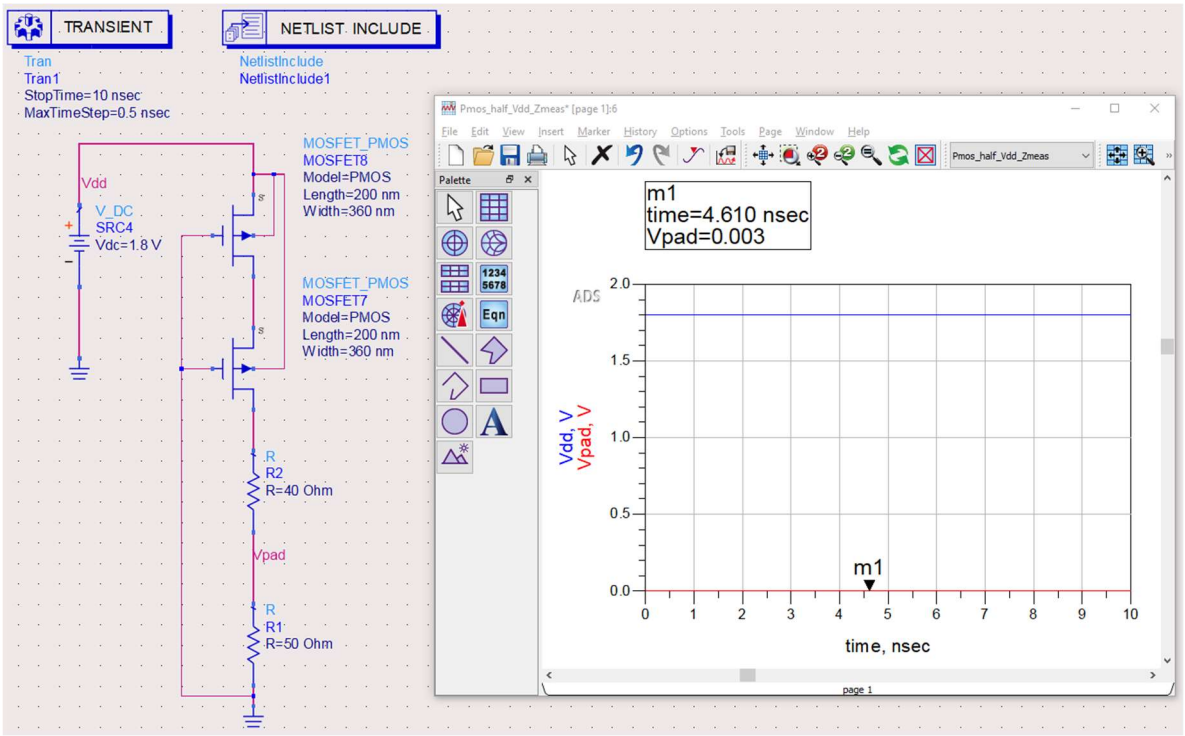


Figure 24: Pmos Width adjustment circuit.

Similarly, as in the Nmos leg adjustment, the objective is to have half Vdd, which is 0.9V, at the Vpad node. This can be done by increasing the Width of the Pmos transistors while keeping the minimum recommended Length, of 200nm. Table 5 shows the iterations by increasing the Width of the Pmos transistors and measuring the Vpad until we get 0.9V.

Table 5: Pmos Width adjustment.

Pmos 1 and 2, Length	Pmos 1 and 2, Width	Vpad	Error (Vpad – Target)
200nm	360nm	0.003V	-897mV
200nm	720nm	0.005V	-895mV
200nm	1.44um	0.011V	-889mV
200nm	2.88um	0.021V	-879mV
200nm	5.76um	0.043V	-857mV
200nm	11.52um	0.084V	-816mV
200nm	23.04um	0.166V	-734mV
200nm	46.08um	0.317V	-583mV
200nm	92.16um	0.550V	-350mV
200nm	184.32um	0.750V	-150mV
200nm	368.64um	0.869V	-31mV
200nm	737.28um	0.933V	33mV
200nm	700um	0.930V	30mV
200nm	600um	0.919V	19mV
200nm	500um	0.903V	3mV
200nm	450um	0.892V	-8mV
200nm	475um	0.898V	-2mV
200nm	485um	0.8997753784V	-0.2246216mV
200nm	486um	0.899975262V	-0.024738mV
200nm	487um	0.900174352V	0.174352mV
200nm	486.2um	0.9000151435V	-0.0151435mV
200nm	486.1um	0.8999952068V	-0.0047932mV
200nm	486.15um	0.900005176V	0.005176mV
200nm	486.14um	0.900003182V	0.003182mV
200nm	486.13um	0.900001189V	0.001189mV
200nm	486.12um	0.899999195V	-0.000805mV
200nm	486.121um	0.8999993941V	-0.0006059mV
200nm	486.122um	0.8999995935V	-0.0004065mV
200nm	486.123um	0.899999793V	-0.000207mV
200nm	**486.124um**	**0.899999992V**	**-0.000008mV - Selected**
200nm	486.125um	0.900000192V	0.000192mV

Figure 25 shows the circuit with the selected values, Length=200nm and Width=486.124um for the Pmos leg and measuring Vpad.

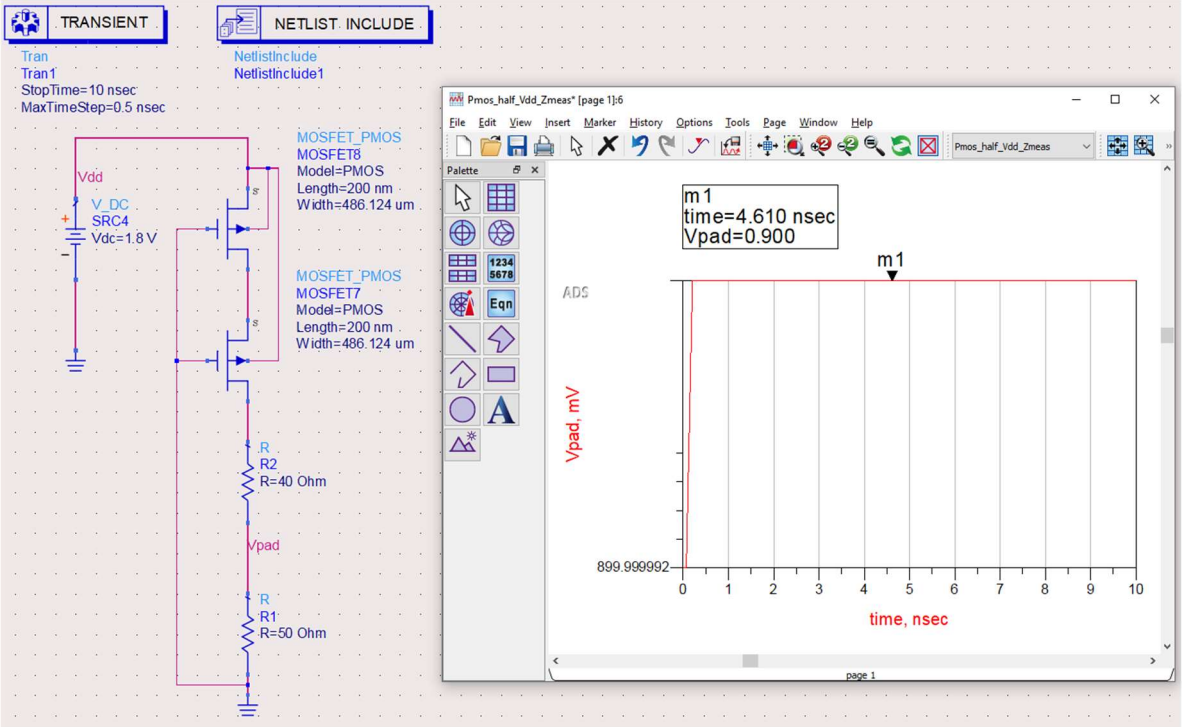


Figure 25: Selected Pmos Width.

4.5. Programmable Impedance Driver

We can create a programmable impedance driver by putting some replicas of the Nmos and Pmos tuned legs. In this example we're putting 10 legs of each, Nmos and Pmos, we're selecting the values of W, L, and R to get a 50Ω driver when 5 of the 10 legs are used. Every one of the 10 legs must have a DC resistance of 250Ω, so when 5 legs are On, the equivalent DC resistance will be 50Ω, this is because they are in parallel to each other.

We want the driver to be tuned at 50Ω with 5 of the 10 legs, in typical, this is in the middle, so when working with best and worst cases, we can have the margin to add or remove legs.

4.6. Nmos Rcomp for a 50Ω Driver

From the tuned values, found at the Nmos leg tuning, section 4.3, we can create a 250Ω leg by multiplying the Resistor by 5 and dividing the Width by 5. Length mustn't change. Values can be seen in Table 6 and the Nmos leg is shown in Figure 26.

Table 6: L, W, and R, for 50Ω and 250Ω Nmos Legs.

Length	Width	Resistor	Nmos Leg DC resistance
200nm	171.875um	40Ω	50Ω
200nm	34.375um	200Ω	250Ω

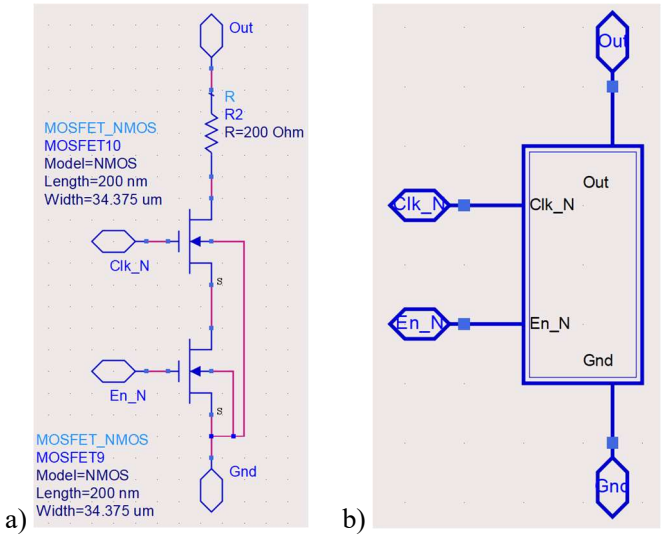


Figure 26: Nmos Leg. a) Circuit. b) Symbol.

The circuit in Figure 27 helps to verify that the Nmos driver is tuned to 50Ω when 5 of the 10 legs are turned On. The "Nmos_Leg" X1<0:9> block contains 10 Nmos legs in parallel, the Nmos leg Clk_N and En_N pins are controlled with VtStep sources, which transition from 0V to 1.8V in a sequence, after 100usec each. The Nmos are activated with 1.8V. The OpAmp is an Operational Amplifier used as a comparator, it compares the 0.9V Vref to the voltage at Vpad. We're putting a 2pF load at the output of the comparator, this is only to have a small load and not an open circuit and to avoid errors in the simulation tool. Vdd is 1.8V.

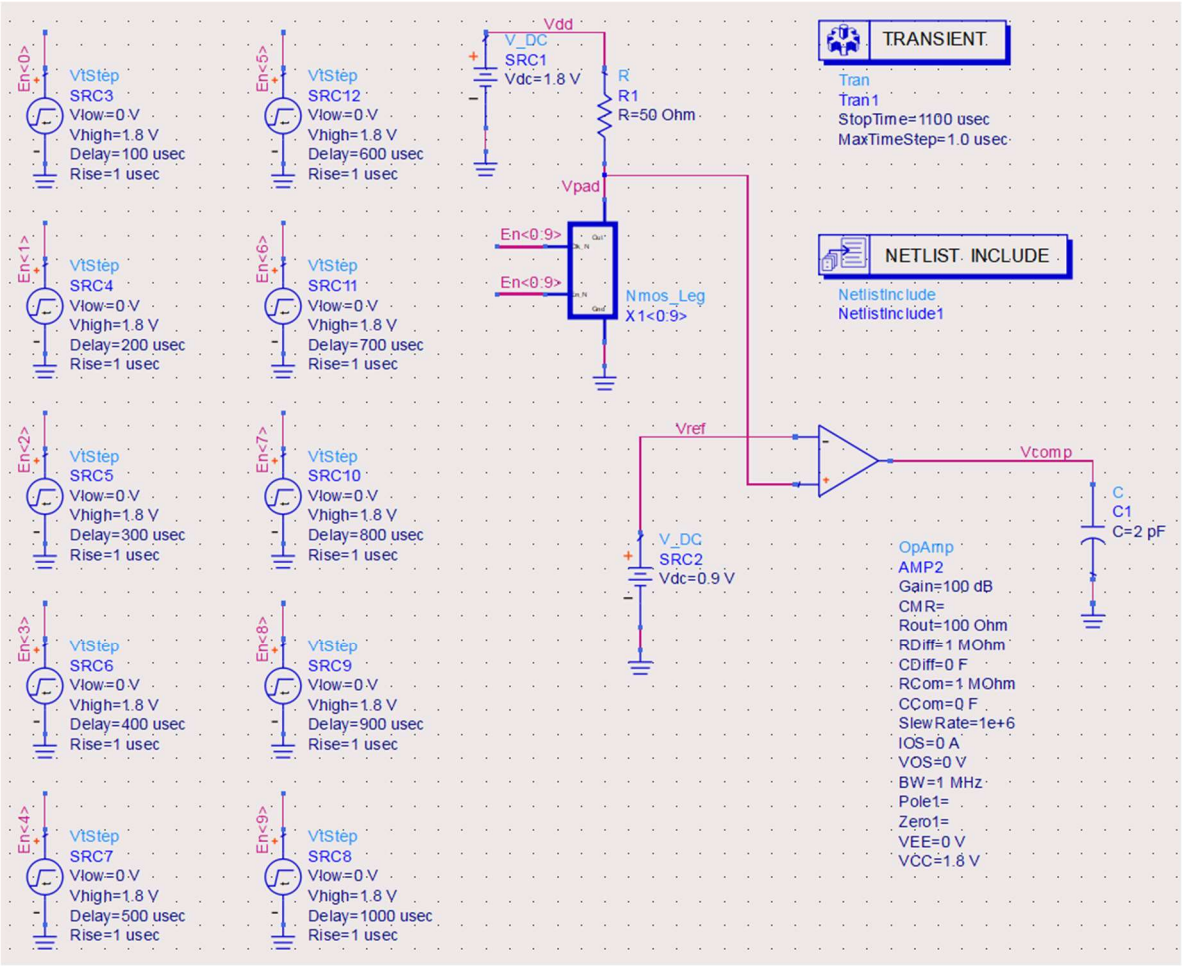


Figure 27: Nmos Rcomp verification circuit.

The expectation is that the Vcomp output transitions from 1.8V to 0V when Vpad is equal to Vref=0.9V, this is expected to happen when 5 of the 10 legs are activated. Figure 28 shows the simulation of the circuit, we can see that effectively, the Vpad is equal to Vref when 5 legs are active, and Vcomp transitions correctly. Every step on the Vpad signal represents one more leg that has turned On.

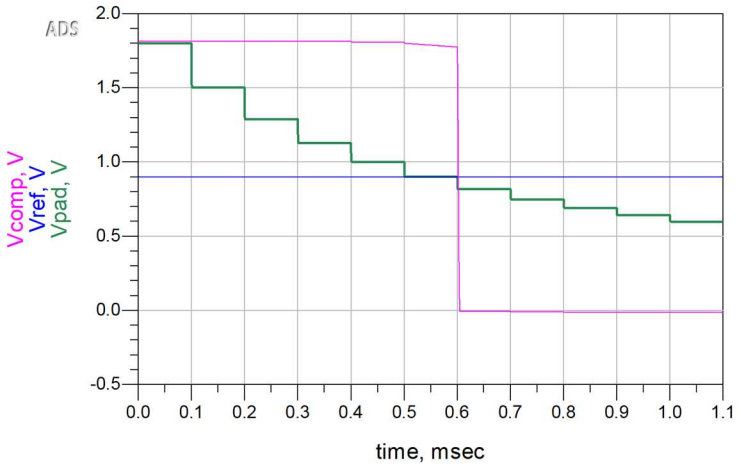


Figure 28: Nmos Rcomp circuit simulation result.

4.7. Pmos Rcomp for a 50Ω Driver

Similarly, from the tuned values, found at the Pmos leg tuning, section 4.4, Table 7 shows the Length, Width, and Resistor values for a 250Ω Pmos leg. This is done by multiplying the Resistor by 5 and dividing the Width by 5. Length mustn't change. The Pmos leg is shown in Figure 29.

Table 7: L, W, and R, for 50Ω and 250Ω Pmos Legs.

Length	Width	Resistor	Nmos Leg DC resistance
200nm	486.124um	40Ω	50Ω
200nm	97.225um	200Ω	250Ω

a) b)

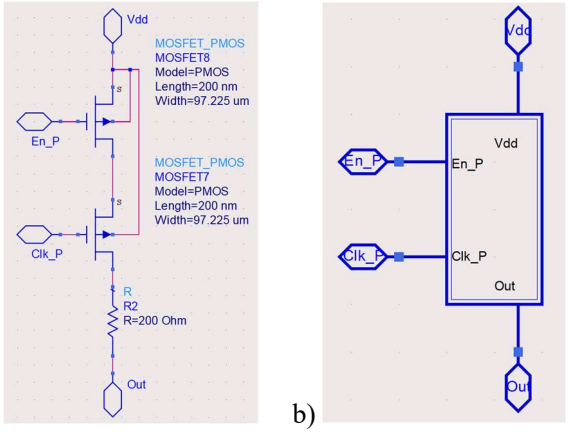


Figure 29: Pmos Leg. a) Circuit. b) Symbol.

Figure 30 shows the circuit to verify that the Pmos driver is tuned to 50Ω when 5 of the 10 legs are turned On. The "Pmos_Leg" X1<0:9> block contains 10 Pmos legs in parallel, the Pmos leg Clk_P and En_P pins are controlled with VtStep sources, which transition from 1.8V to 0V in sequence, after 100usec each.

The Pmos are activated with 0V. The OpAmp is comparing the 0.9V Vref to the voltage at Vpad. The 2pF load capacitor avoids errors in the simulation tool. Vdd is 1.8V.

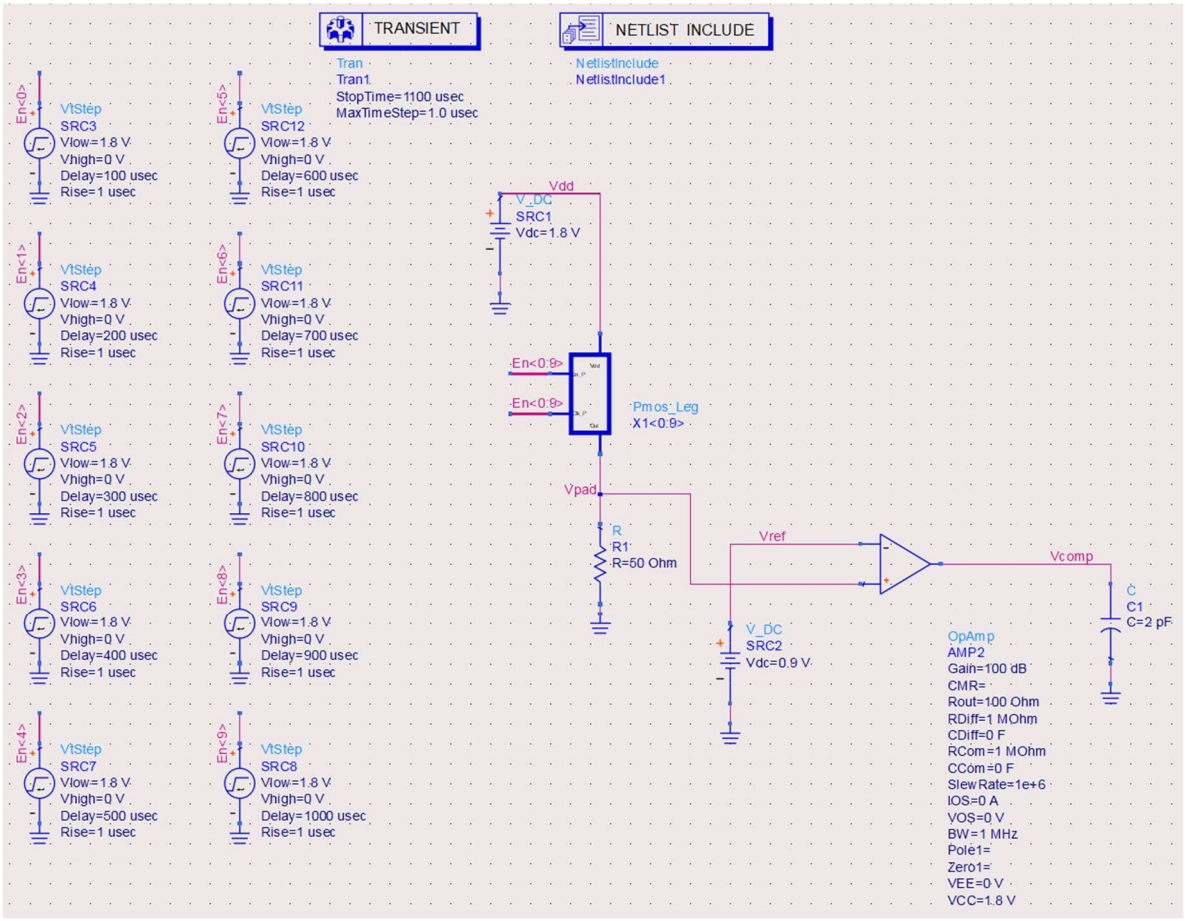


Figure 30: Pmos Rcomp verification circuit.

As expected, the Vcomp output transitions from 0V to 1.8V when Vpad is equal to Vref=0.9V, this occurs when 5 of the 10 Pmos legs are activated. Every step on the Vpad signal represents one more leg that has turned On. Figure 31 shows the result of the simulation.

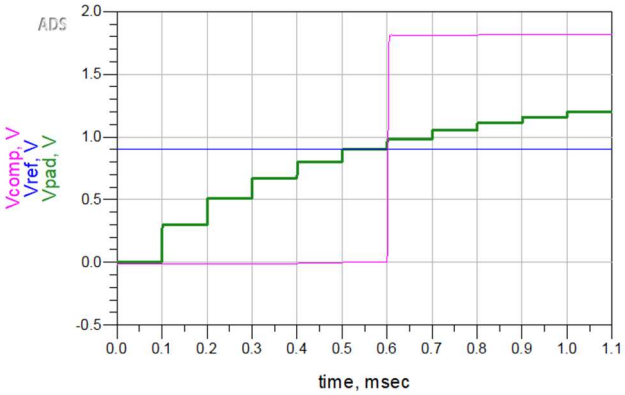


Figure 31: Pmos Rcomp circuit simulation result.

4.8. Nmos and Pmos Rcomp for a 33Ω Driver

The image clock (Imgclk) is a Single-Ended 33Ω driver. By using the "Programmable Impedance Driver" reviewed in section 4.5, we only need to run the Rcomp simulations and check how many legs are needed to be enabled to get a 33Ω driver. From the simulation in Figure 32 and Figure 33, we can see that we need to activate 8 of the 10 legs to obtain a 33Ω driver.

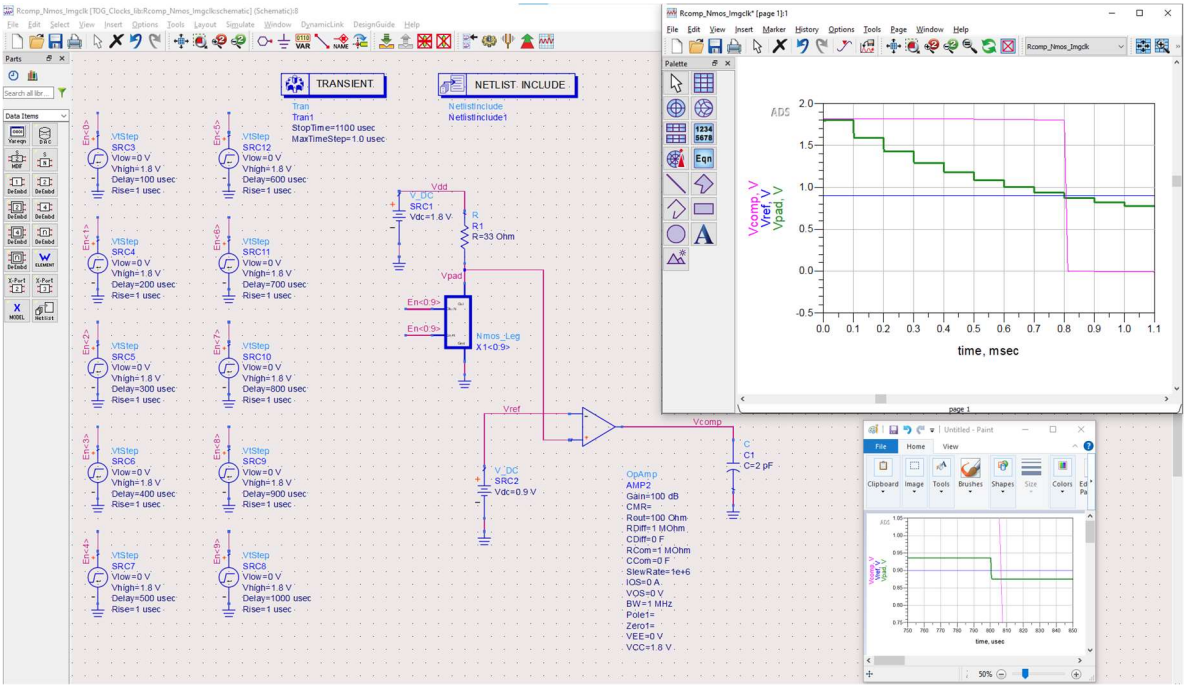


Figure 32: Nmos Rcomp simulation for a 33Ω driver.

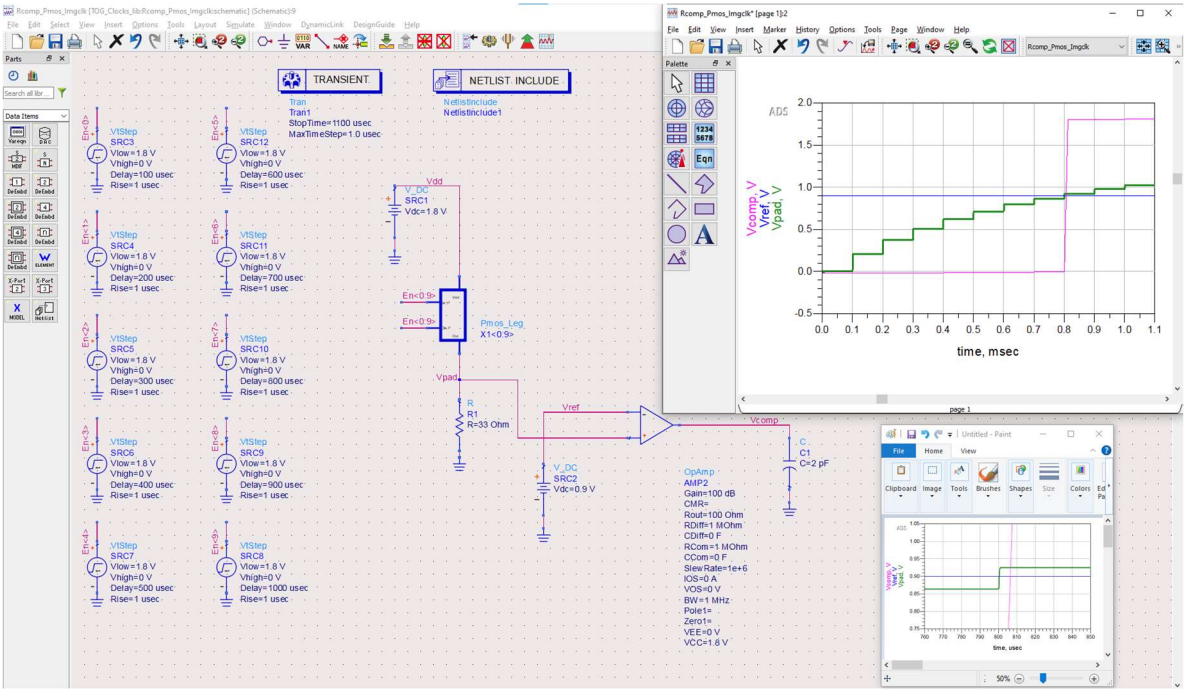


Figure 33: Pmos Rcomp simulation for a 33Ω driver.

4.9. Single-Ended Non-Ideal 50Ω Transmission Line

In this section we'll review how to create a non-ideal Single-Ended Transmission Line in ADS, We need it to make simulations with a driver and a transmission line and understand how the clock signal arrives at the Far-End receiver, which will be closer to the reality.

In one of the classes at the ITESO, named "Printed Circuit Board Design Workshop", I learned that we could use specific tools from the PCB suppliers, calculators that help to understand the physical dimensions of the tracks for a specific Impedance, one of such calculators is from JLCPCB company [22]. In this tool, we need to specify the Impedance that we want, In this example we want a 4-layer PCB because we're considering a simple circuit, we're selecting the Thickness 1.6mm for a cheaper PCB, the thicker the PCB, the more expensive it is. We are considering routing the clock in the outer layer, so the inner layer will act as the Return Path, and selecting Single-ended, which is the type on the required transmission line.

The result of the calculation is shown in Figure 34, we're focusing on the JLC7628 stack-up, which is shown in Figure 35, we can see that our 50Ω Single-Ended Transmission Line can be built with a trace width of 11.55mil, a height H1 of 7.10mil, a Dielectric Constant of 4.6, and a copper thickness T1 of 1.4mil.

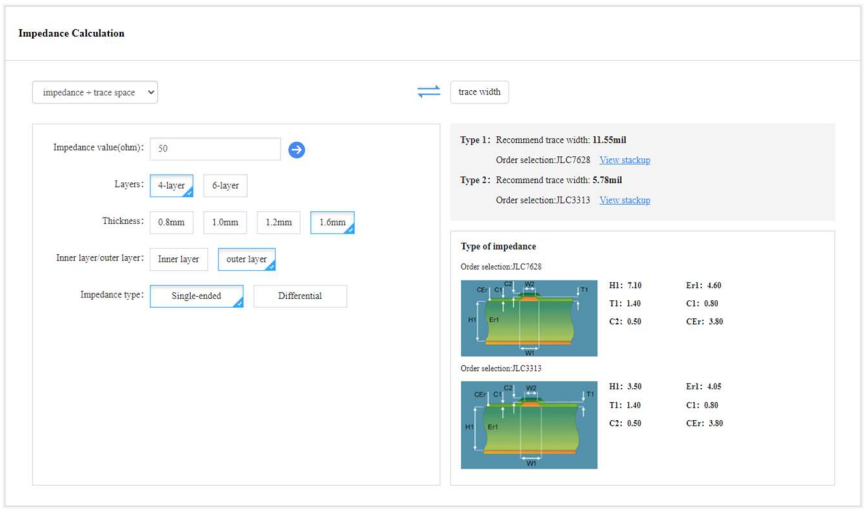


Figure 34: JLCPCB track structure for Single-Ended TL.

JLC7628 Stack-up is shown in Figure 35. In the stack-up, we can see the dimensions of every layer.

Controlled Impedance PCB Parameters and Stackup

1.Prepreg dielectric constant:

Prepreg type	Dielectric constant
7628	4.6

2.Solder mask Parameters

Solder mask thickness over the laminate	Solder mask thickness over traces	Solder mask dielectric constant
0.8mil	0.5mil	3.8

3. Core dielectric constant

Core dielectric constant
4.6

4-Layer Impedance Control Stackup

Thickness

0.8mm 1.0mm 1.2mm 1.6mm 2.0mm

a) JLC7628 Stackup:

Layer	Material Type	Thickness	
Top Layer1	Copper	0.035 mm	
Prepreg	7628*1	0.2104 mm	
Inner Layer2	Copper	0.0152 mm	1.1mm (with copper core)
Core	Core	1.065 mm	
Inner Layer3	Copper	0.0152 mm	
Prepreg	7628*1	0.2104 mm	
Bottom Layer4	Copper	0.035 mm	

0.2mm (7.87 mil) is nominal thickness of 7628 prepreg. Use 7.1 mil as the thickness when the controlled impedance tracks are on top/bottom, use 8.1 mil when tracks are inside.

Figure 35: JLC7628 PCB Stack-up.

Another tool to create a Transmission Line is the built-in calculator, which is built-in in the ADS software, it can be opened as shown in Figure 36.

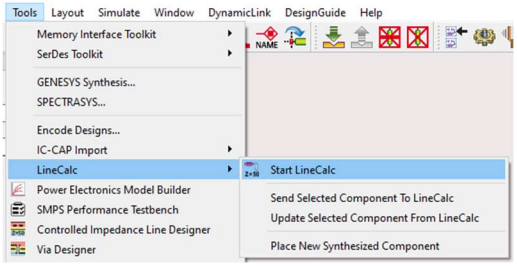


Figure 36: Line Calculator, in ADS.

In this tool, we want to create an MLIN transmission line, we're inputting some values that were previously obtained with the JLCPCB calculator, such as the height H=7.1mil, the dielectric constant Er=4.6, and the copper thickness T=1.4mil, other parameters are left as default. We need to set the Z0 impedance to 50Ω, the Frequency at 5GHz, which is the recommended bandwidth for a high-speed clock, and then click on "Synthesize".

The tool calculates the physical width W and length L, and the electrical length E_Eff, which is related to the frequency value, see Figure 37.

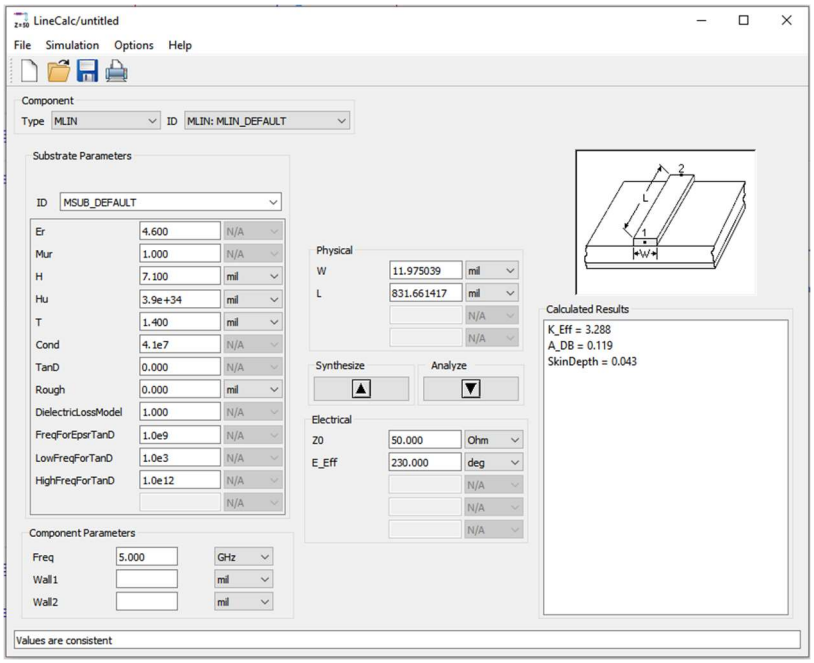


Figure 37: LineCalc in ADS, 50Ω Single-Ended TL.

As we can compare, the width W=11.55mil from the JLCPCB calculator is very similar to the width W=11.975mil obtained with the LineCalc tool. This is a good indicator that we're designing our Transmission Line correctly.

The value of width W=11.975039mil will be used in ADS to create the non-ideal MLIN Transmission Line.

4.10. Differential Non-Ideal 100Ω Transmission Line

Following the same tools, as in a single-ended transmission line, we're now building a Differential non-ideal transmission line. In the calculator from JLCPCB company [22], we need to specify the impedance at 100Ω, 4 layers for this example so the PCB is simple, we're selecting a thickness of 1.6mm for a cheaper PCB. We are considering routing the clock in the outer layer, so the inner layer will act as the Return Path, and selecting Differential as the Impedance type, For the Trace space we're selecting 4mil, this is the minimum suggested by the supplier and with this trace space we can save space in our board for more traces.

The result of the calculation is shown in Figure 38, similar to the single-ended transmission line, we're focusing on JLC7628 stack-up, we can see that our 100Ω Differential transmission line can be built with a trace width of 4.96mil, a height H1 of 7.10mil, a trace separation S1 of 4.0mil, a dielectric constant of 4.6, and copper thickness T1 of 1.4mil

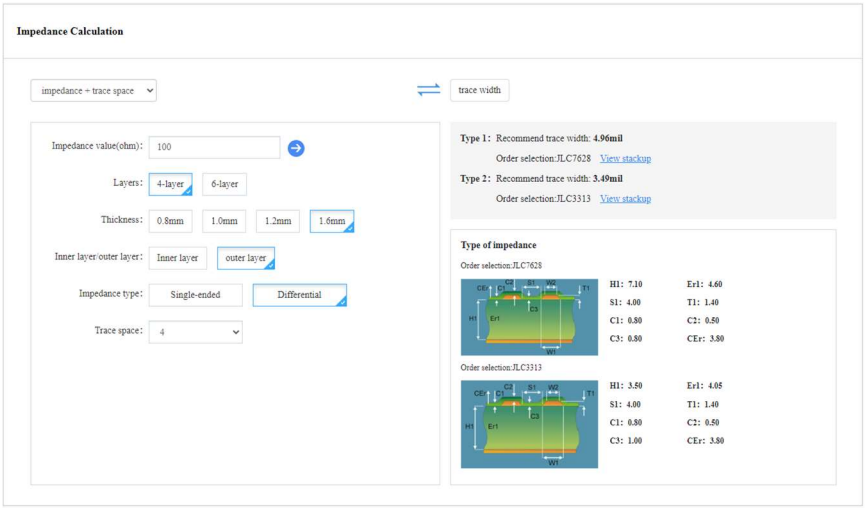


Figure 38: JLCPCB track structure for Differential TL.

Next, we're creating an MCLIN, which is a Differential transmission line with the ADS built-in Line Calculator, we're inputting some values previously obtained with the JLCPCB calculator, such as the height H=7.1mil, the dielectric constant Er=4.6, the copper thickness T=1.4mil, other parameters are left as default. We need to set the ZE=100Ω, the ZO=50Ω, and the Frequency at 5GHz, and then click on "Synthesize". We're changing the L to 5507mil which represents a 1ns time delay in the TL, then clicking on "Analyze" and observing no change in ZE, ZO, W, and S. This is because the length of the track doesn't affect the impedance.

We're obtaining the width W and space S for our transmission line, see Figure 39.

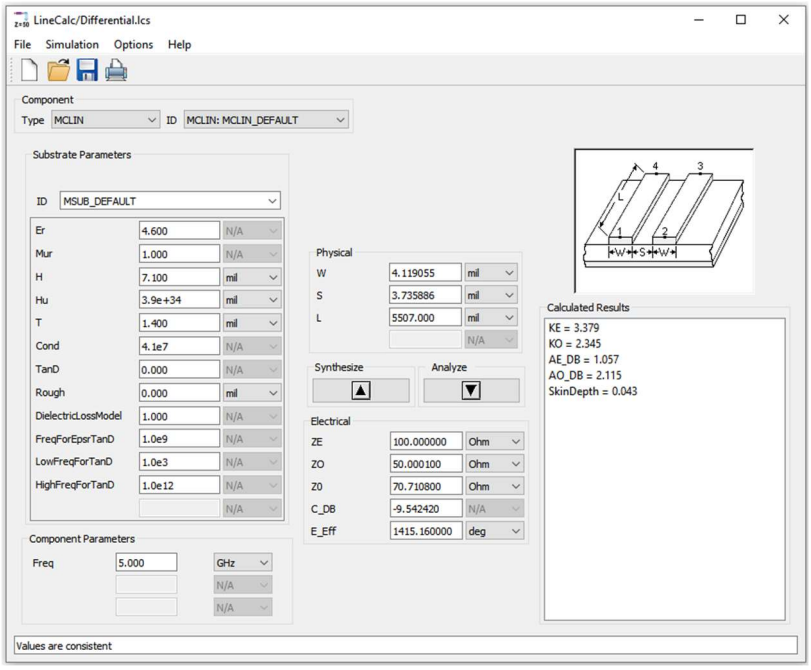


Figure 39: LineCalc in ADS, 100Ω Differential TL.

As we can compare, the width W=4.96mil and the S=4mil from the JLCPCB calculator are very similar to the width W=4.119055mil and the S=3.735886mil obtained with the LineCalc tool. This is a good indicator that we're designing our Transmission Line correctly.

The values of width W=4.119055mil and S=3.735886mil will be used in ADS to create the non-ideal MCLIN Transmission Line.

4.11. Single-Ended Buffer

It was created a Single-Ended buffer in ADS, for Image Clock simulations, shown in Figure 40, which is built with 10 Pmos and 10 Nmos legs, the impedance is controlled by providing or not providing voltage at the En_P and En_N pins with the V_DC sources, Pmos turns On with 0V and turns Off with 1.8V, while Nmos turns On with 1.8V and turns Off with 0V. The driver impedance is set to 33Ω by enabling 8 of the 10 legs, in each Pmos and Nmos arrays. We're putting an inverter to the Clk_In signal, it's just acting like a buffer to provide enough power to the Buffer's Input

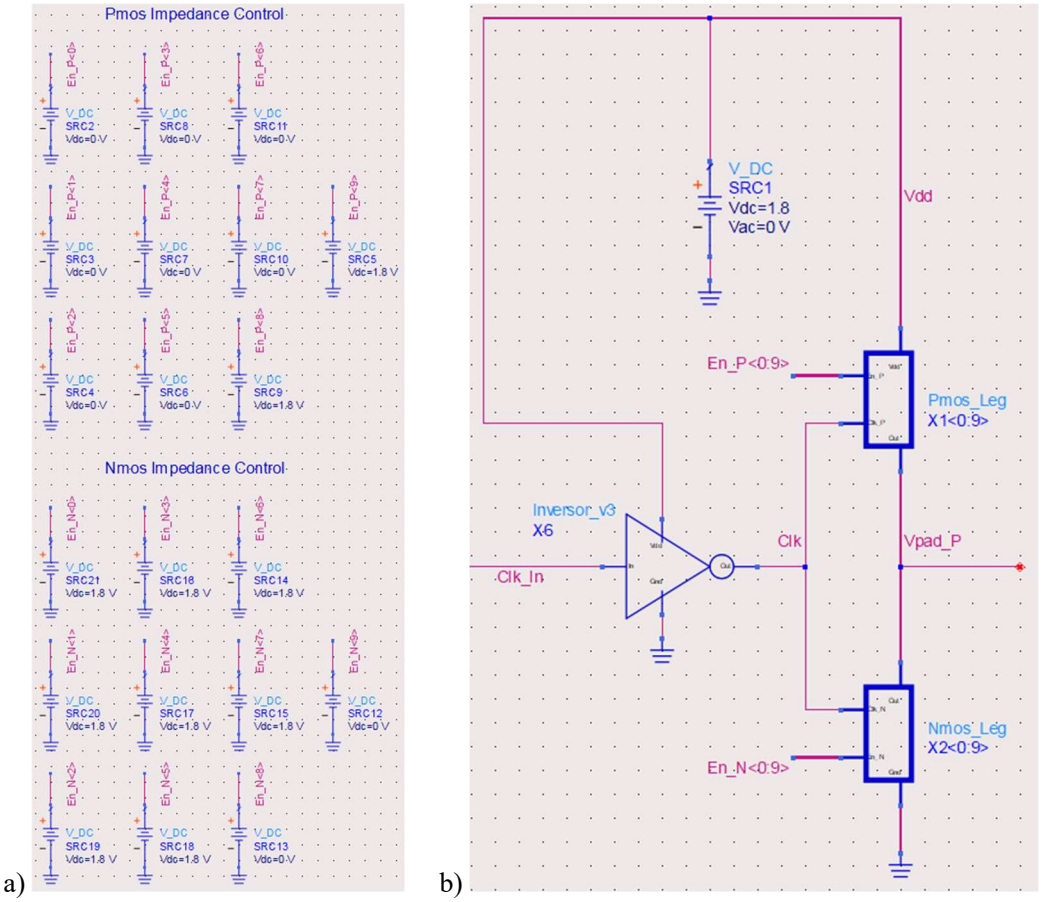


Figure 40: Single-Ended buffer. a) Impedance Control. b) Single-Ended Buffer.

The Image Clock drives a PDB's trace of 15 inches in the motherboard, We're using a non-ideal MLIN Transmission Line in ADS Figure 41, and we're setting W=11.975039 as obtained in section 4.9.

Figure 41: MLIN Transmission Line.

The typical load of an Image Clock receiver is 10pF, we're putting a capacitor at the Far End, Figure 42, this is noted as the Imgclk node.

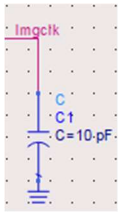


Figure 42: 10pF Capacitor Load.

As we saw in the Inverter section, 4.2, the smallest inverter is made with an Nmos of L/W 200nm/360nm, and a Pmos of L/W 200nm/846nm. An inverter of this size is not powerful enough to drive the transmission line, As seen in Figure 43, the signal at the Far End is sinusoidal instead of square. VtPulse is injecting a clean 19.2MHz clock signal into the buffer

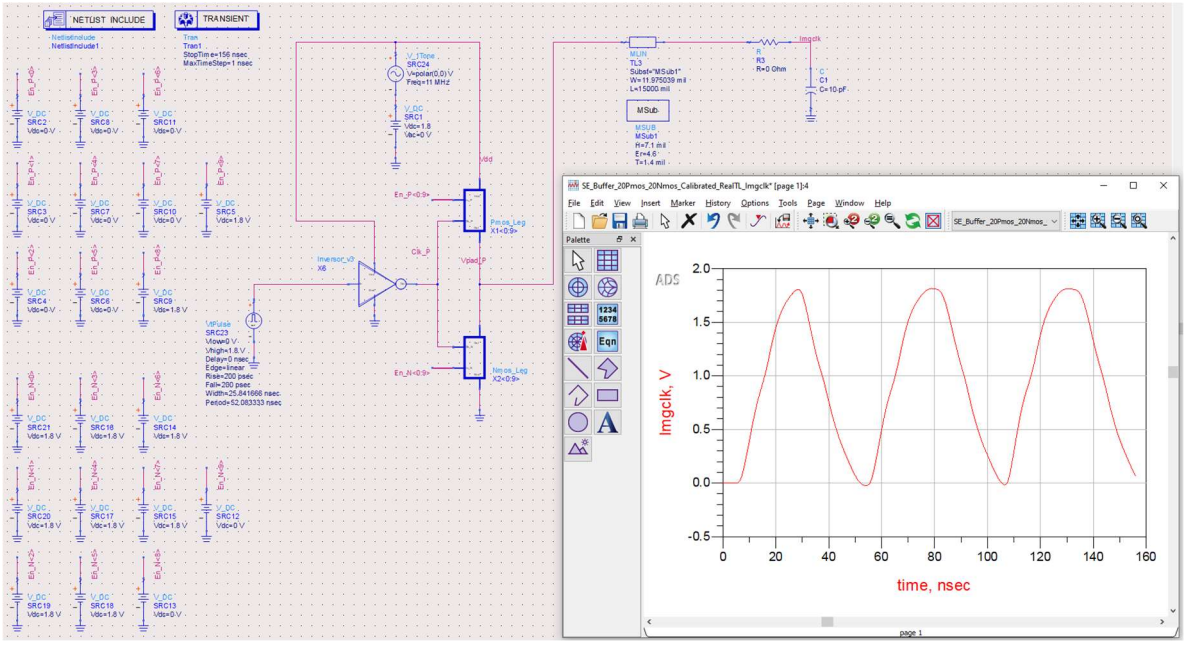


Figure 43: Smallest Inverter, Sinusoidal instead of Square.

We need to adjust the inverter, we can use the Nmos and Pmos from the tuned legs as a base, Nmos L/W of 200nm/34.375um and Pmos L/W of 200nm/97.225um, and adjust Pmos' width to get Out=In, similarly as seen in inverter section, 4.2. Table 8 shows the iterations of the adjustment.

Table 8: Tuning of the Inverter for the Buffer.

Try #	In	Out	Nmos L/W	Pmos L/W
1	0.9V	1.171V	200nm/34.375um	200nm/97.225um
2	0.9V	1.062V	200nm/34.375um	200nm/90.0um
3	0.9V	0.885V	200nm/34.375um	200nm/80.0um
4	0.9V	0.904V	200nm/34.375um	200nm/81.0um
5	0.9V	0.895V	200nm/34.375um	200nm/80.5um
6	0.9V	0.89954	200nm/34.375um	200nm/80.75um
7	0.9V	0.89956	200nm/34.375um	200nm/80.751um
8	0.9V	0.899735	200nm/34.375um	200nm/80.76um
9	0.9V	0.9005	200nm/34.375um	200nm/80.8um

10	0.9V	0.90031	200nm/34.375um	200nm/80.79um
11	0.9V	0.900021	200nm/34.375um	200nm/80.775um
12	**0.9V**	**0.9000023 error: 2.3uV**	**200nm/34.375um**	**200nm/80.774um**
13	0.9V	0.899983 error: -17uV	200nm/34.375um	200nm/80.773um

From the iterations, we obtained an Output equal to the Input with a Pmos width of 80.774um, as seen in Figure 44.

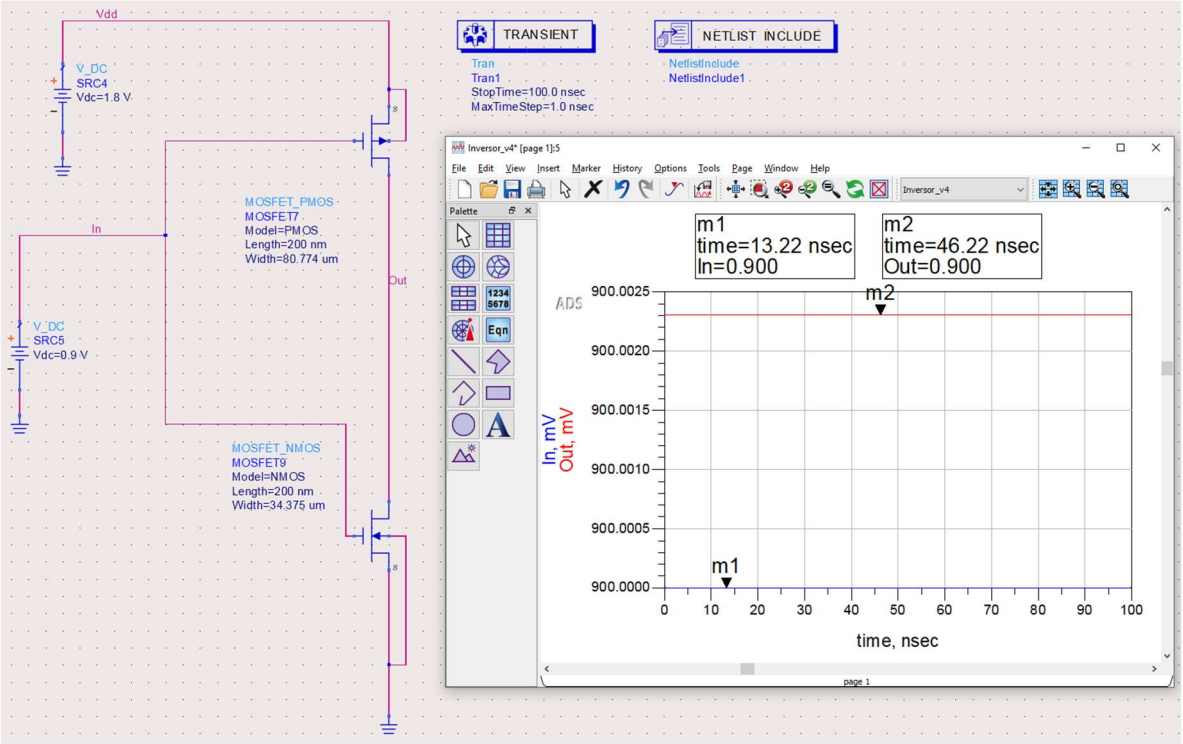


Figure 44: Inverter for Buffer, the adjustment circuit.

Now the signal at the Far End looks square, although it seems to have a big Overshoot and Undershoot, which will be studied in section 5.1, This is the correct waveform, Figure 45.

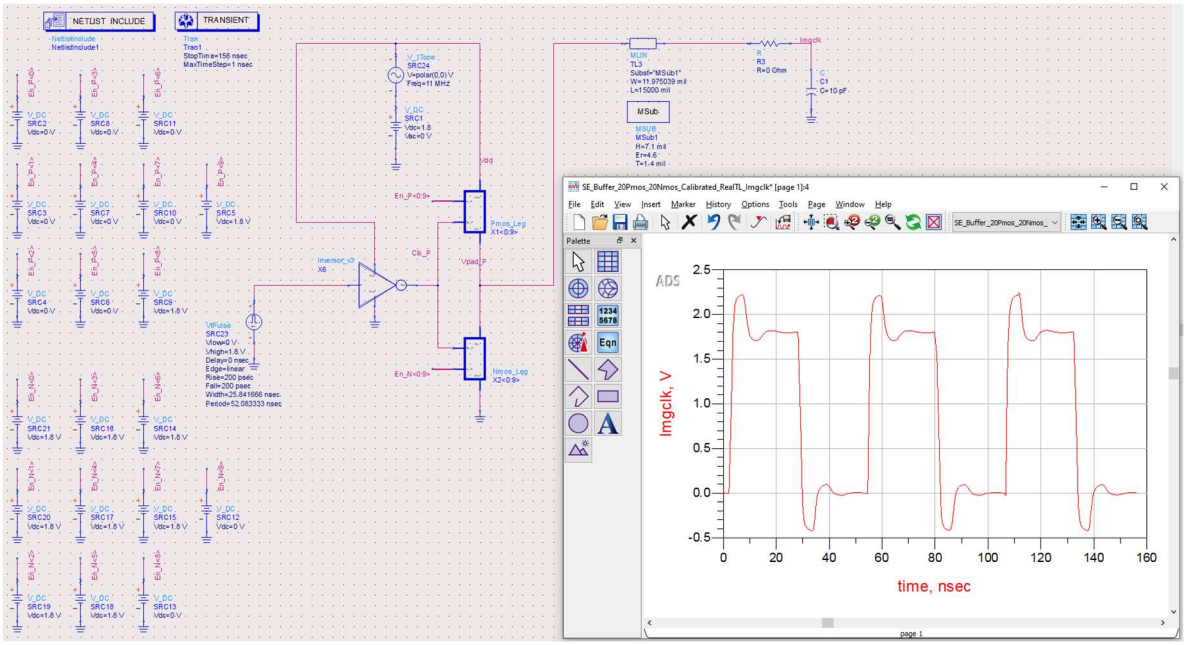


Figure 45: Inverter adjusted, the square signal at Far End.

Figure 46 shows the adjusted inverter, which is now able to drive the buffer, and its symbol.

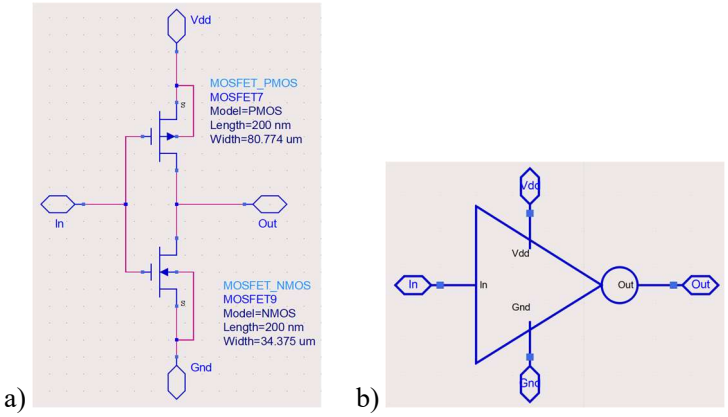


Figure 46: Inverter. a) Circuit. b) Symbol.

4.12. Complementary Buffer

Also, a complementary buffer was created, which is used for PCIe reference clock simulations, shown in Figure 47, This is built with two Single-Ended buffers, it has two sets of Pmos and Nmos legs. Two inverters are needed to drive the two buffers, Clk_P signal is inverted related to Clk_N, so the buffers will drive a differential transmission line in Odd mode, which is, when one signal is high, the other signal is low, and vice versa.

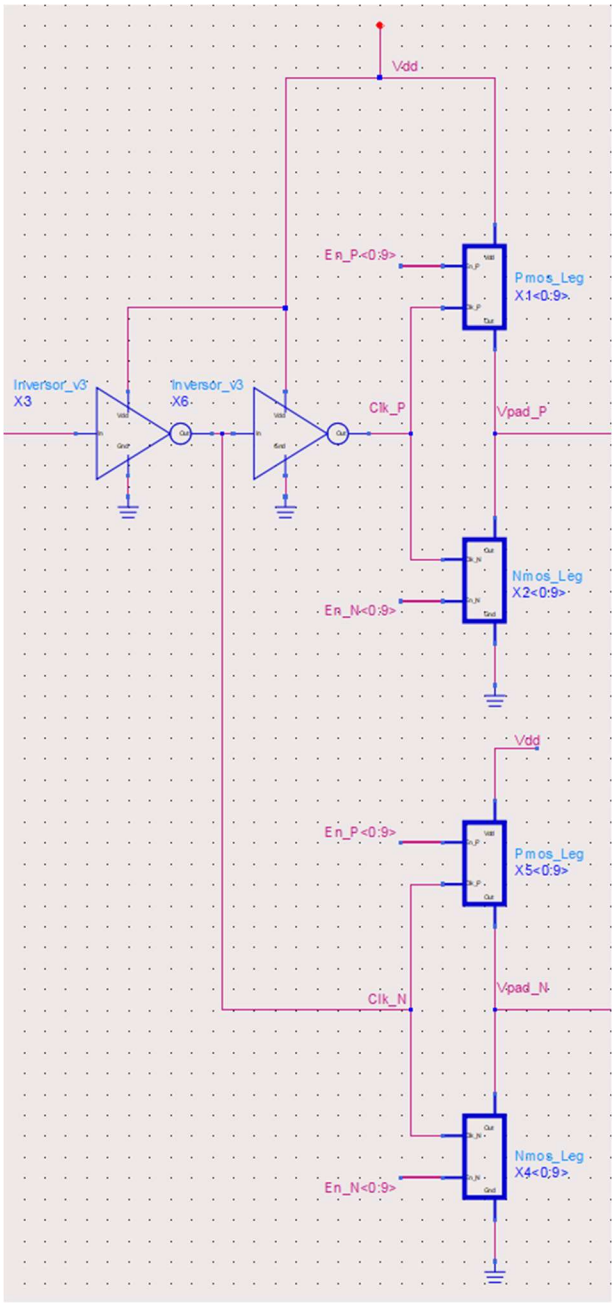
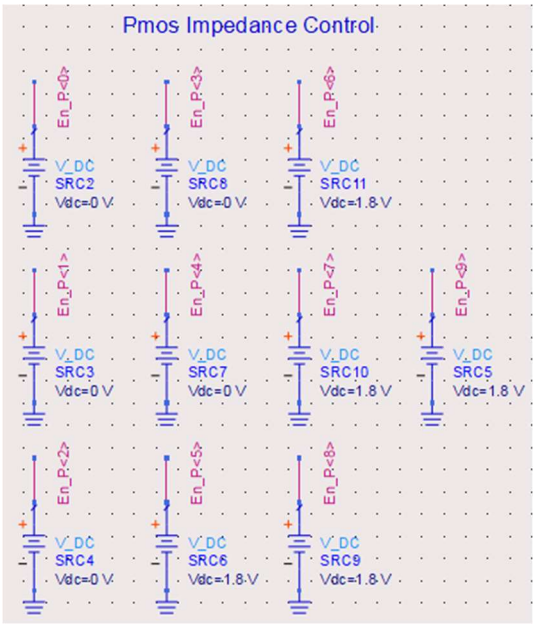


Figure 47: Complementary Buffer.

Similarly, the impedance is controlled with 20 DC power supplies, 10 for the Pmos impedance control, and 10 for the Nmos side. To have 50Ω drivers, it's needed to activate 5 of the 10 legs, as seen in sections 4.6 and 4.7. Pmos are activated with 0V and deactivated with 1.8V, while Nmos are activated with 1.8V and deactivated with 0V, see Figure 48.

a)



b)

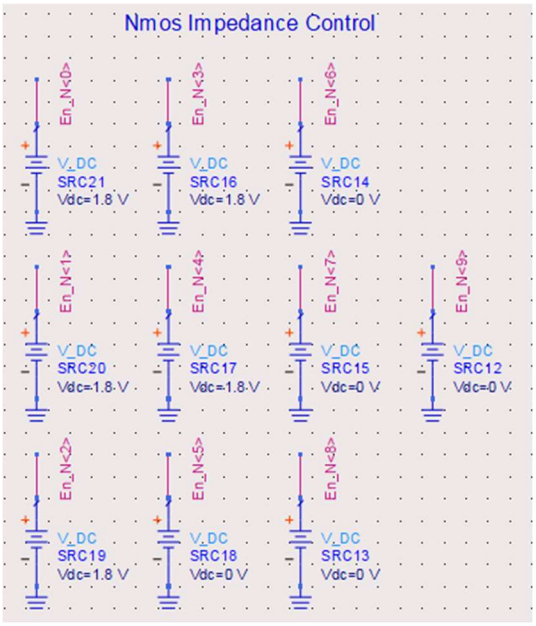


Figure 48: Impedance Control for 50Ω. a) Pmos. b) Nmos.

This buffer can drive a differential transmission line, like the one designed in section 4.10. Figure 49 shows the eye-diagram response, V_DC supplies 1.8Vdc, and V_1Tone is an AC source that will be used to inject noise to Vdd, we'll explore this later. VtPulse is injecting a clean 100MHz clock signal into the buffer. MCLIN is the non-ideal 100Ω differential transmission line, with a length of 5.507 inches, which represents a Time Delay of 1ns. SRC_P and SRC_N are the Far End destiny, we're mimicking the PCIe's input receiver load with the 2pF capacitors on each line. EyeDiff_Probe is the tool to generate the eye diagram from the signal being measured, while EyeProbeSummary records the selected measurements from the eye in an Excel output file. NetlistInclude is adding the Pmos/Nmos models, and Transient is the directive to simulate 300ns in this case.

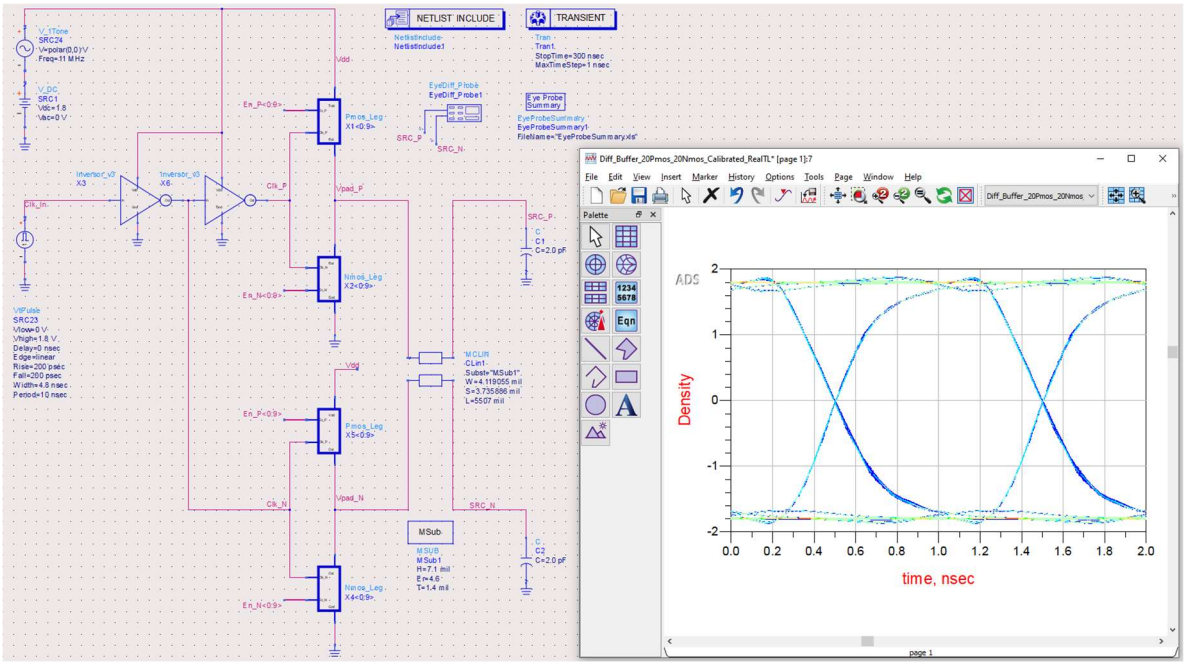


Figure 49: Complementary Buffer driving a 100Ω Differential TL.

5. CASE STUDIES

Summary: *The top four debug cases are studied in this chapter, undershoot, jitter, ppm, and slew rate. From these, we can learn what physical changes affect each parameter and therefore improve our analysis skills to debug real products.*

5.1. The “Undershoot” Case Study

In one of the projects, when we measured overshoot/undershoot in the Image clock, which is the clock that synchronizes the camera circuit, we discovered that undershoot was lower than -300mV, which is the minimum acceptable voltage. The Specification says, for all input/output voltages, with respect to ground, the absolute maximum rating is from -0.3V to Vdd + 1V.

When debugging, we learned that the Image clock’s driver impedance is 33Ω, and it cannot be changed from the SoC, so the impedance matching had to be done in the motherboard, it was also discovered that the RVP didn’t intercept a change in the topology, there must be a 10Ω resistor within 1 inch near the SoC’s transmitter, and it was not there. There was room for a resistor but only near the End connector, We performed measurements in the lab, placing the 10Ω resistor but the undershoot was not fixed.

We had to wait around 20 weeks for an RVP with the correct topology, this is, with a 10Ω resistor near the SoC’s transmitter to re-validate. After this time, when the board was received, we saw that the undershoot was within spec in the RVP with the correct location of the resistor.

In this work, we are making simulations to understand how the “Undershoot” behaves regarding the physical location of the resistor in the motherboard. Below we can see, in a simulation environment, the undershoot effect in three scenarios, first: when the 10Ω resistor is not in the path, second: when the 10Ω resistor is near the End connector, and third: when the 10Ω resistor is near the SoC’s transmitter. We can observe that, the “Undershoot” is only passing the spec in the “near to the SoC’s transmitter” case. These scenarios were simulated in the ADS tool.

5.1.1. Simulation Scenario # 1

This is the implementation in the initial motherboard that was received. In Figure 50, we have the 33Ω Tx driver at the Near End, which is at the left of the figure, then we have a 15 inches 50Ω transmission line in the board, all lengths L add up to 15,000mil, ending in a connector at the Far End, at the right, the connector is not part of the simulation, one 10pF capacitor termination is emulating the receiver's input load; This value of the capacitor was mentioned by an engineer from Signal Integrity team, this is the load they use in their simulations.

As we can see from the simulation, there are big Overshoot and Undershoot in the signal at Imgclk Far End. In this case, only the Undershoot is violating the specification, the lower voltage is -407mV, which is less than the minimum spec -300mV.

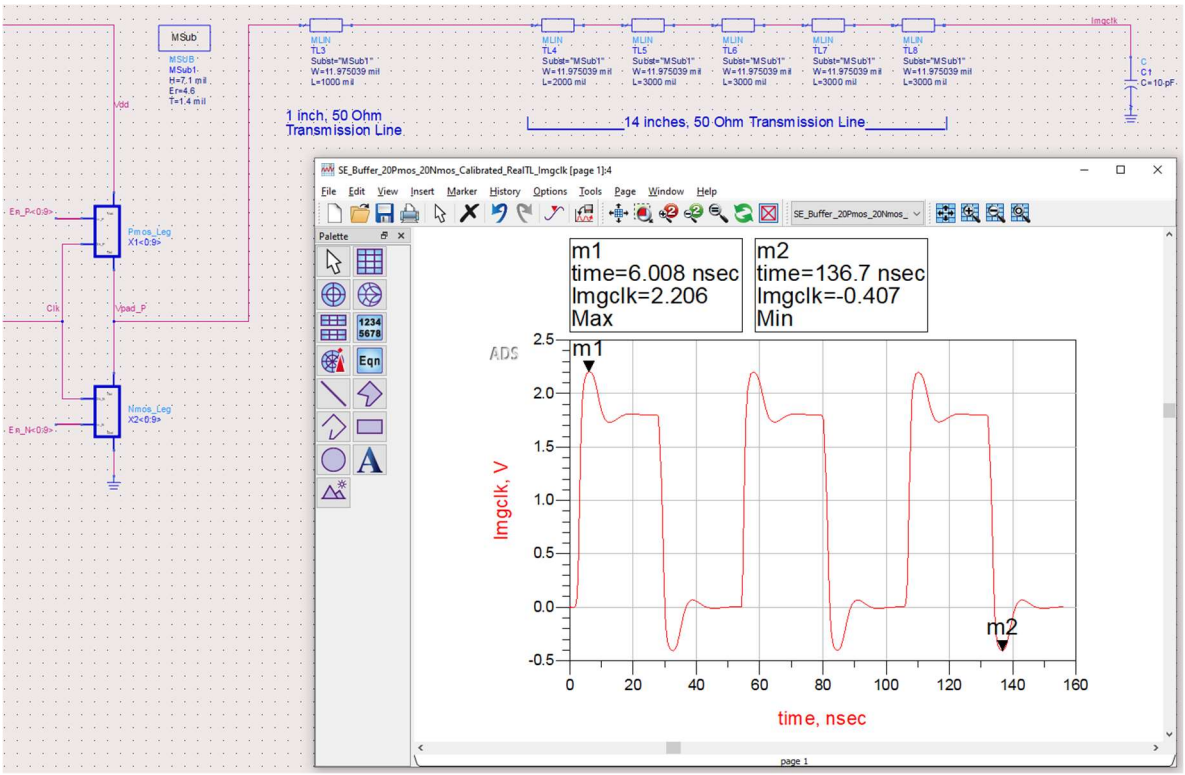


Figure 50: Undershoot, scenario 1.

5.1.2. Simulation Scenario # 2

In the initial board received, the only physical place where we could put a resistor is at the Far End, which is at the right, close to the connector. We put a 10Ω resistor there and performed measurements, but the Overshoot was not fixed. Figure 51 shows the simulation of this scenario, we are seeing -395mV, which is still lower than the minimum spec -300mV. The problem was not fixed at this point.

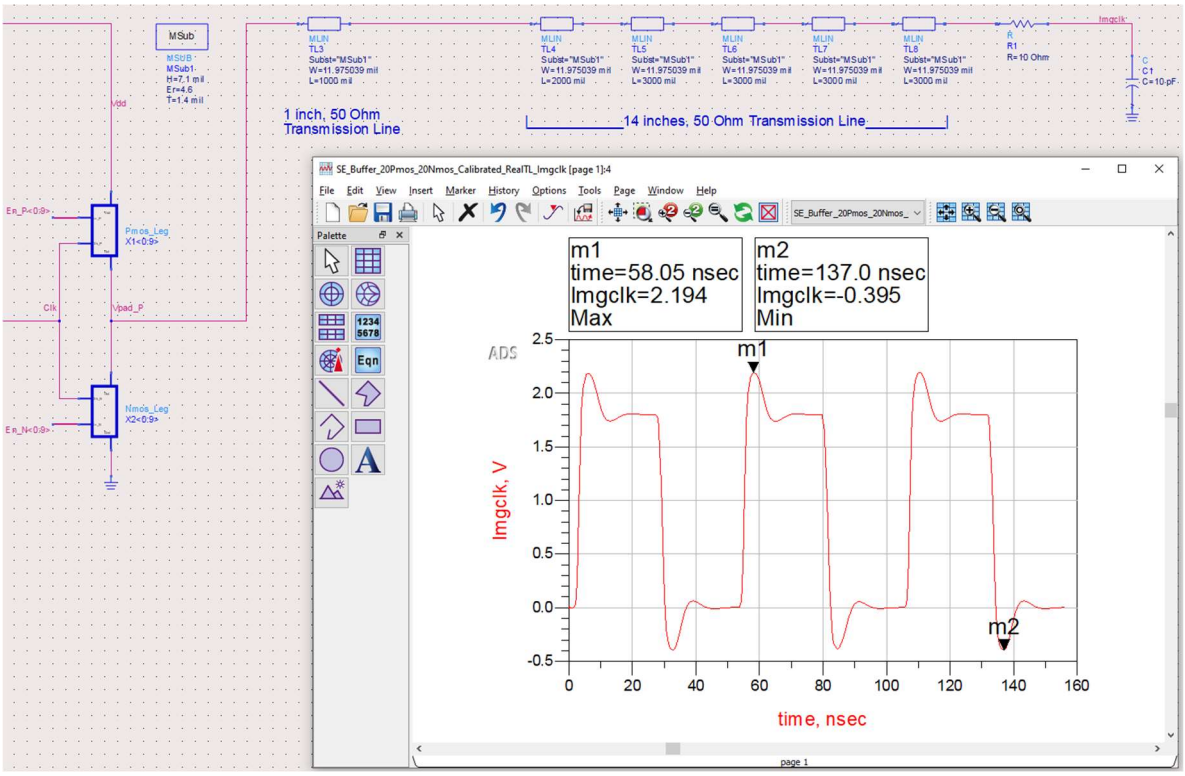


Figure 51: Undershoot, scenario 2.

5.1.3. Simulation Scenario # 3

Scenario 3 shows the final resolution, which is to put a 10Ω resistor at a maximum of 1 inch from the Tx driver. As commented above, we had to wait 20 weeks for a motherboard with this implementation. When received, we performed measurements and saw that the Overshoot was not violating the specification anymore. As shown in Figure 52, the simulation result is -175mV, which is higher than the minimum spec -300mV.

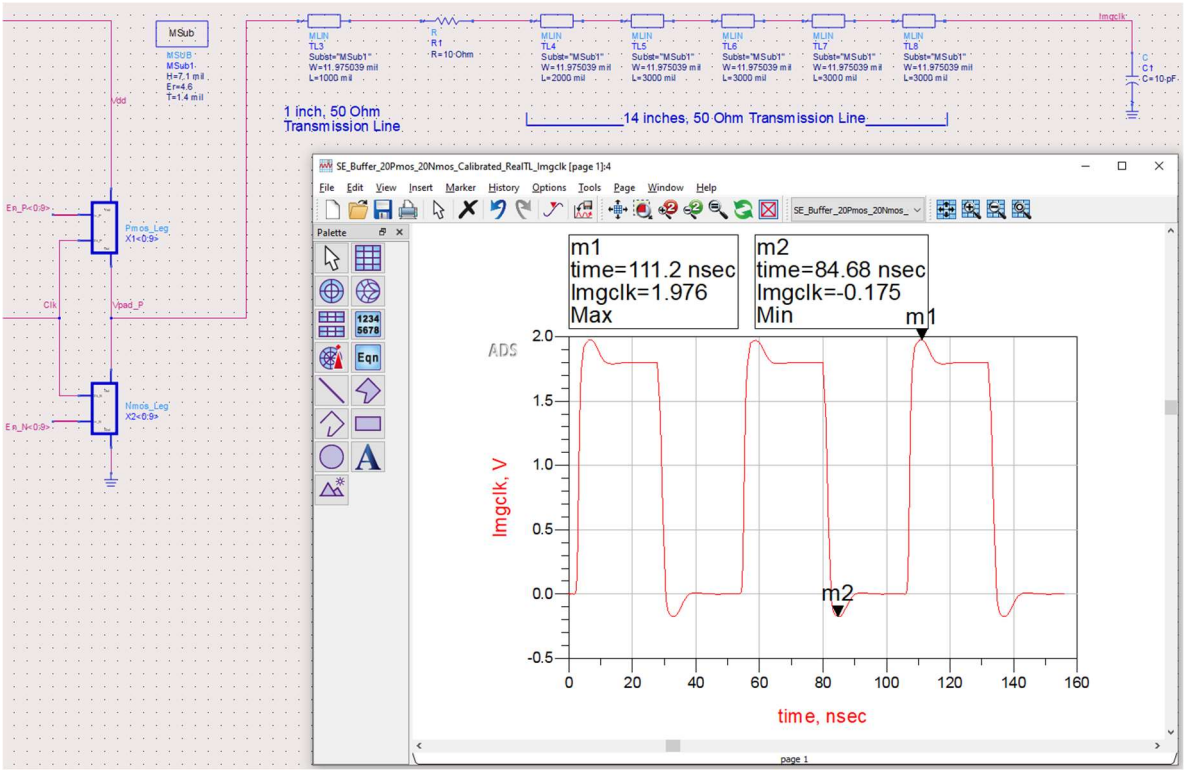


Figure 52: Undershoot, scenario 3.

5.1.4. Conclusion

The most important point from this case study is that we could try this simulation in the Readiness phase, at design board reviews, which occurs a long time before the board is built, and catch these problems earlier so we can reduce or eliminate the debug time for Overshoots and Undershoots. We just need to know what the Tx driver impedance is, the impedance of the PCB tracks, any resistor value in the path, and the impedance of the load.

5.2. The “Jitter” Case Study

It is very common to see excessive jitter in clocks or data lines, and this is one of the more difficult issues to root cause. Some of the common causes of high jitter are:

- Noisy power supplies, This is noise that can cause fluctuations in the voltage of the clock signal, leading to jitter.

- Signal reflections, When the clock signal is transmitted along a transmission line, signal reflections can occur due to impedance mismatch, causing jitter.

- Crosstalk, is the unwanted coupling of signals between two adjacent conductors. When a clock signal is routed too close to other high-speed signals, it can lead to crosstalk and jitter.

- Ground bounce, is the voltage difference between two points in a ground plane. When the ground plane has high impedance, the current flowing through it can cause ground bounce, which in turn causes jitter.

- Temperature variations, can cause variations in the physical properties of the components, leading to variations in the propagation delay of the clock signal.

To reduce high jitter in clock signals, it is important to minimize the effects of these factors by careful circuit design, layout, and component selection.

5.2.1. Noise Frequency Effect

In this section we’ll purposedly induce jitter to a buffer by applying noise to its power supply, it will be applied to the complementary buffer created in section 4.12. This is done by adding a “V_1Tone” source in series with the “V_DC” power supply, so Vdd is the addition of both sources, Figure 53. We can control the amplitude and frequency of the noise added.

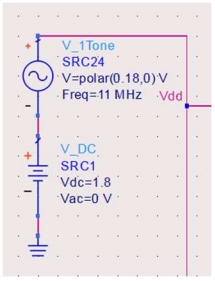


Figure 53: Power supply plus noise.

The suggestion is to add noise with a frequency of 11MHz, which is not a direct multiple of the 100Mhz clock, and so not being in phase with it, this way we can find bad jitters sooner during simulation. Few other frequency values were simulated to see the effect of the jitter. Every simulation captures one microsecond of signal, and it takes about ten seconds to execute, A few cases were run capturing more time and it was observed that the jitter didn’t change considerably but took much more time to execute. So, one microsecond seemed to be a good time for the analysis. There was no necessity to remove a piece

of the signal from the start of the simulation, they ran very clean since the very beginning, which was not the case from some simulations we did in LTspice in the early phases of this work.

The graph in Figure 54 shows the Jitter peak to peak versus the Frequency of the noise, We can see that, effectively, 11MHz noise produces a high jitter, around 75ps. While other frequencies produce jitter in the range of 70ps to 80ps, we can observe that closer multiples of the clock frequency produce less jitter, these are the cases of 25MHz, 50MHz, 100MHz, and 200MHz, with jitter values of 58ps, 18ps, 8ps, and 12ps respectively, this could be explained because these values are some ways in phase with the clock.

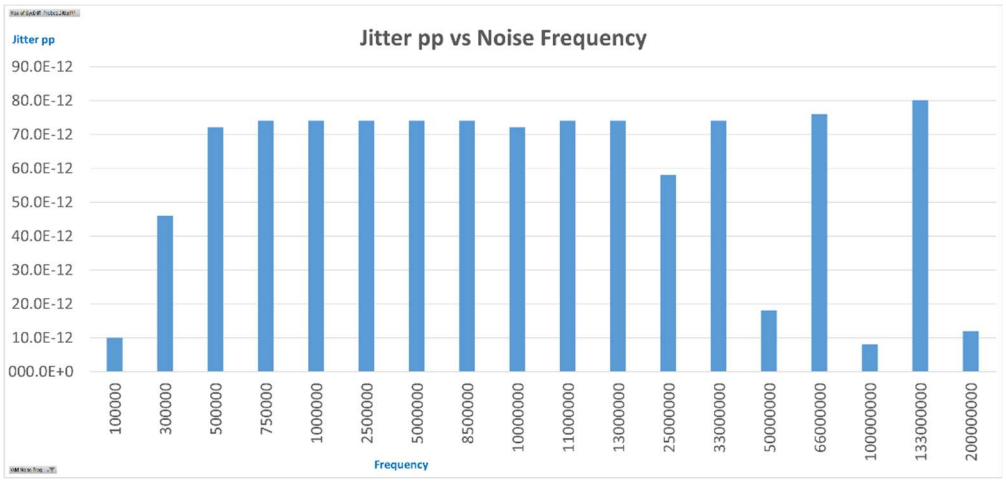


Figure 54: Jitter pp vs. Noise Frequency.

It can also be observed that 100kHz and 300kHz cause low jitter, The explanations for these could be that these are low frequencies compared to the simulation window, which is 1us, This can be better appreciated in Figure 55, which shows waveform and eye diagrams in various frequencies.

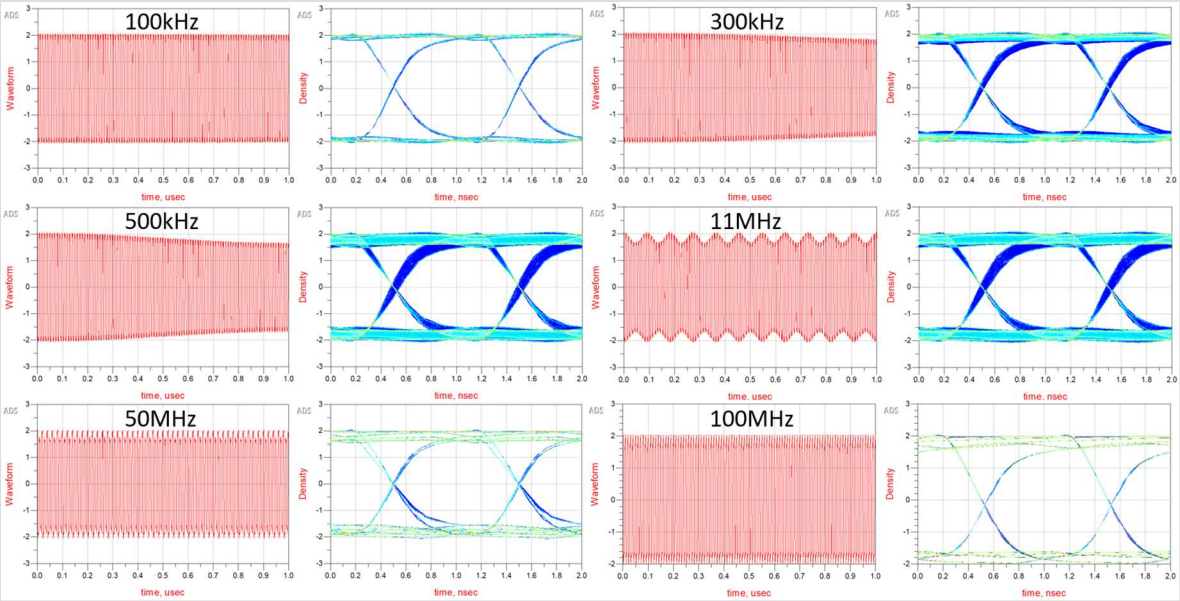


Figure 55: Waveform and Density, frequency effect.

5.2.2. System Variation Effect

Several simulations were run, making some variations to the system, the results can be seen in Figure 56, it was iterated by changing the Buffer Impedance to 33Ω and 50Ω; the length of the transmission line to 3in, 5.5in, and 12in; the far-end load between 2pF capacitors and 50Ω resistors; the noise related to Vdd in percentage with values of 0%, 2,5%, 5% and 10%; and every combination was tried 3 times, just to observe run to run variation. The impedance of the transmission line was 100Ω differential in all runs.

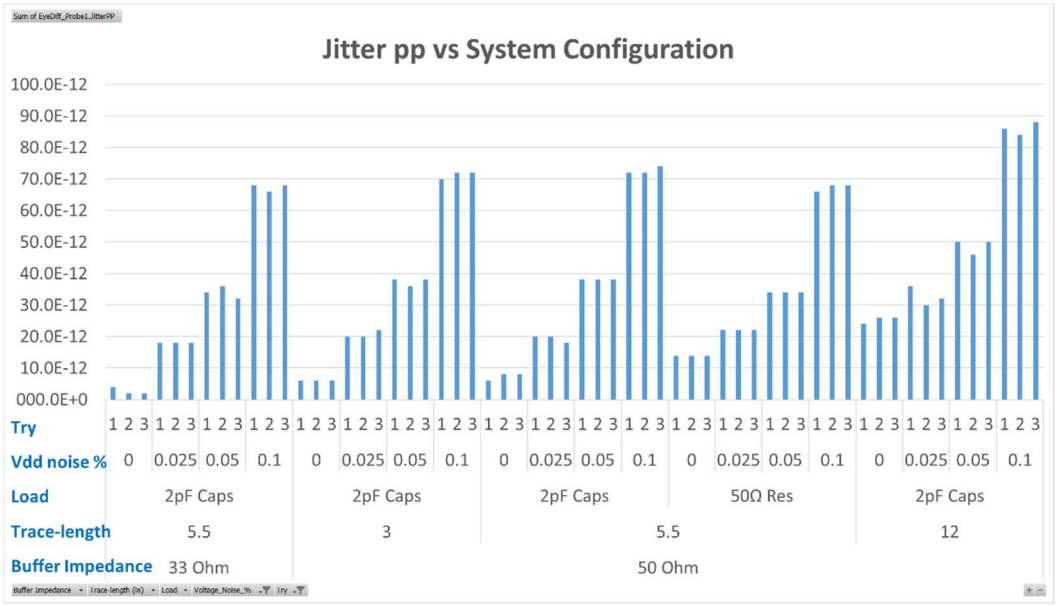


Figure 56: Jitter pp vs. System Configuration.

We can observe that the larger the transmission line, the greater the jitter pp; it can also be observed how jitter goes higher as Vdd noise increases; it was observed a small or null run-to-run variation, this gives good confidence in the results; when the line is terminated with the 50Ω load we get the lowest jitter values; by changing the Buffer Impedance to 33Ω to force mismatch, the jitter didn't increase excessively as expected, maybe the mismatch was not big enough. The waveforms and eye diagrams for some of the combinations can be seen in Figure 57.

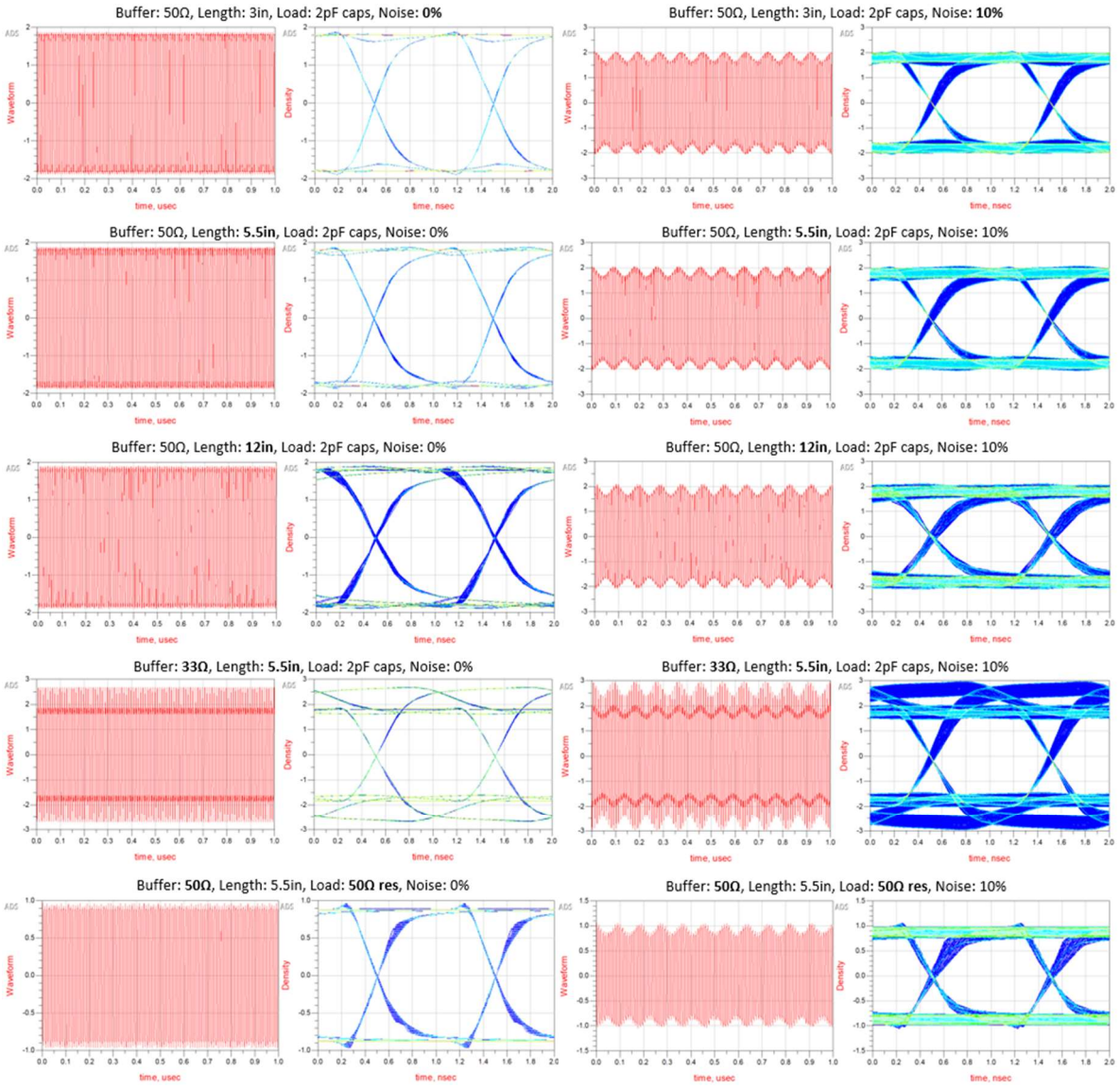


Figure 57: Waveform and Density, Buffer, Length, Load, and Noise effect.

5.2.3. Conclusion

We were able to get jitter increased in our circuit, we applied noise to the power supply at diverse frequencies and amplitudes, we tried to mismatch the impedance, increased the length of the transmission line, as well as changed the load. Other modifications to the system like crosstalk and temperature

variations could be considered in future work. We observed how the clock signal gets jittery by changing some of the architecture of the system, This learning can be applied to real systems in the laboratory, and by performing some experiments, and changing variables in the system, we could more easily understand and find the root cause of real problems.

5.3. The "Frequency ppm" Case Study

In some of the projects we have seen that ppm is out of the specification, in this chapter, we're trying to understand the effect of frequency ppm when modifying the value of certain components in an oscillator circuit.

5.3.1. Parts per Million

Frequency stability in a crystal oscillator is usually represented in ppm (parts per million). However, it is easier to understand this in terms of frequency. The formula below gives the variation in Hz for a specific frequency and ppm. There exist many calculators on the web to ease the analysis [11].

$$Variation\ in\ Hz = \frac{f * ppm}{10^6}$$

Equation 1: Variation in Hz.

Where, f = center frequency (in Hz)

Ppm = frequency variation in ppm

The frequency stability of a crystal oscillator represents the variation in output frequency due to external conditions like temperature variation, voltage variation, output load variation, and frequency aging.

Using the formula, for a 32.768kHz crystal with 20ppm, we can find that:

Frequency variation: 0.65536 Hz
Min Frequency: 32.76734464 kHz
Max Frequency: 32.76865536 kHz

Similarly, the formula below can calculate the actual ppm for a measured frequency:

$$ppm = \frac{(measured\ Freq - ideal\ Freq) * 10^6}{ideal\ Freq}$$

Equation 2: PPM equation.

Stability tolerance can also be expressed as a percentage of frequency deviation rather than as parts per million (ppm).

0.5% = 5000 ppm
0.01% = 100 ppm
0.005% = 50 ppm
0.001% = 10 ppm
0.0001% = 1 ppm

5.3.2. Colpitts LC Oscillator, Frequency Tuning

During electrical validation, we debugged some issues related to the Crystal oscillator. In this section we are studying, and simulating, the Colpitts oscillator, because they are modeled very similarly to Crystals [23].

The Colpitts Oscillator design uses two capacitors in series with a parallel inductor to form its resonance tank circuit producing sinusoidal oscillations, Figure 58. This LC resonance circuit is connected between the collector and the base of a single-stage transistor amplifier, producing a sinusoidal output waveform.

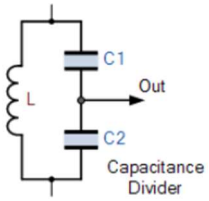


Figure 58: Colpitts Oscillator Tank Circuit.

The condition for oscillations is: $X_{C1} + X_{C2} = X_L$. The advantage of this configuration is that, with less self and mutual inductance within the tank circuit, frequency stability of the oscillator is improved along with a simple design [24].

My director helped me to create two Colpitts circuits in Tspice, one of them is for 32.768kHz, the same frequency as an RTC crystal, Figure 59. As an exercise, we tried to tune the frequency to 0ppm, this is done by tuning C1, C2, and L1.

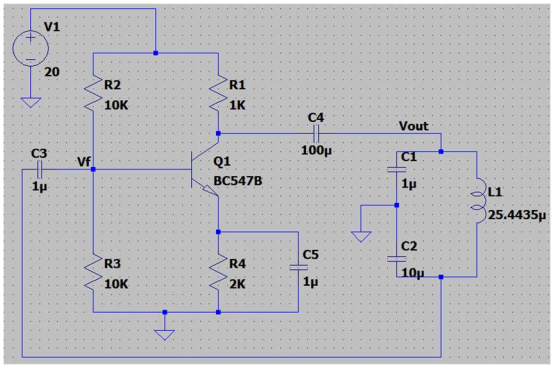


Figure 59: Oscillator LC Colpitts at 32.768kHz.

Table 9 shows the iterations trying to tune the circuit to 32.768kHz. We started by changing C1 in one or a few pico-Farads and see the effect in ppm; as we can observe in the table, a better ppm can be obtained when C1 is increased by only 1pF. Then, taking the C1 value as a base, C2 was changed in a few pico-Farads, observing that a better ppm is obtained by decreasing C2 1pF. Finally, L1 was changed in a few nano-Henry, observing that the best ppm is obtained with no change in L1. Ppm is calculated with Equation 2. Every run takes approximately 30 seconds to execute in LTspice.

Table 9: Colpitts 32p768kHz, simulating 240ms.

C1	C2	L1	Average mid-Freq	Ppm	Note
0.999995uF	10.0uF	25.4435uH	32782.7	448.6	
0.999999uF	10.0uF	25.4435uH	32749.1	-576.8	
1.0uF	10.0uF	25.4435uH	32774	183.1	Default
1.000001uF	10.0uF	25.4435uH	32764.9	-94.6	better
1.000002uF	10.0uF	25.4435uH	32756	-366.2	
1.001uF	10.0uF	25.4435uH	32757	-335.7	
1.1uF	10.0uF	25.4435uH	31383.6	-42248.5	
1.000001uF	9.999997uF	25.4435uH	32781.9	424.2	
1.000001uF	9.999999uF	25.4435uH	32.766	-61.0	better
1.000001uF	10.0uF	25.4435uH	32764.9	-94.6	Base
1.000001uF	10.000003uF	25.4435uH	32.763	-152.6	
1.000001uF	9.999999uF	25.442uH	32747.6	-622.6 (480ms)	
1.000001uF	9.999999uF	25.442uH	32757.1	-332.6	
1.000001uF	9.999999uF	25.4435uH	32.766	-61.0	Base/best
1.000001uF	9.999999uF	25.445uH	32764.4	-109.9	

The other Colpitts circuit is for 100MHz, the same frequency as a PCIe reference clock, Figure 60. The same exercise was executed, trying to tune the frequency to 0ppm, which is done by tuning C1, C2, and L1.

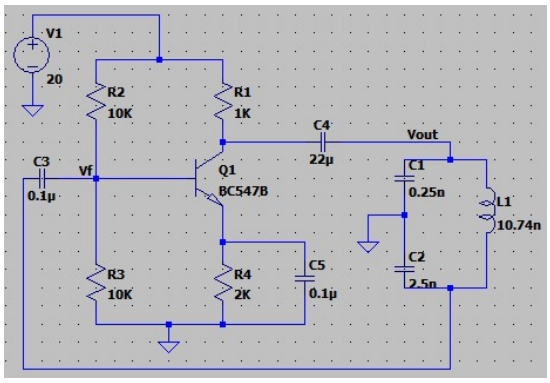


Figure 60: Oscillator LC Colpitts at 100MHz.

Table 10 shows the iterations when tuning the second circuit to 100MHz. Similarly, we started by changing C1 in one pico-Farad step and see the effect in ppm; as we can observe in the table, a better ppm can be obtained when C1 is increased by 2pF. Then, taking the C1 value as a base, C2 was changed in a few pico-Farads, observing that a better ppm is obtained by increasing C2 1pF. Finally, L1 was changed in a few pico-Henry, observing that the perfect 0 ppm is obtained by decreasing 10 pH in L1. Similarly, ppm is calculated with Equation 2. In this circuit, every run takes approximately 3 minutes to execute in LTspice.

Table 10: Colpitts 100MHz, simulating 400us.

C1	C2	L1	Average mid-Freq	Ppm	Note
0.249n	2.5n	10.74n	100480000	4800	
0.25n	2.5n	10.74n	100304000	3040.0	Default-Base
0.251n	2.5n	10.74n	100123000	1230	
0.252n	2.5n	10.74n	99929400	-706	better
0.253n	2.5n	10.74n	99761300	-2387	
0.252n	2.498n	10.74n	99928800	-712	
0.252n	2.5n	10.74n	99929400	-706	Base
0.252n	2.501n	10.74n	99937700	-623	better
0.252n	2.503n	10.74n	99936800	-632	
0.252n	2.505n	10.74n	99936000	-640	
0.252n	2.501n	10.725n	100011000	110	
0.252n	2.501n	10.729n	99989000	-110	
0.252n	2.501n	10.73n	100000000	0	Perfect match
0.252n	2.501n	10.731n	99990100	-99	
0.252n	2.501n	10.74n	99937700	-623	Base
0.252n	2.501n	10.75n	99890800	-1092	

5.3.3. Ppm and Jitter, Inverters vs. CML, noise in supply voltage

Two circuits were created by my director, in these circuits, we're injecting an ideal signal to either three cascaded inverters or three cascaded cml circuits, the intention is to compare their performance about ppm and jitter.

Figure 61 shows the circuit with three cascaded inverters, V1 supplies 1.0V and V2 is sourcing a 9MHz signal with different amplitudes to provoke 0%, 1%, 5%, and 10% noise to Vdd. V3 injects an ideal 100MHz square signal into the first inverter, then its output is entered into the second inverter, and successively, its output is entered into the third inverter, and the final output is v4. All Cmiller were put to experiment about the parasitic capacitance between the gate and drain, but in the end, all of them have a value of 0f. All Pmos have L=180nm and W=8.0um, and all Nmos have L=180nm and W=5.77um.

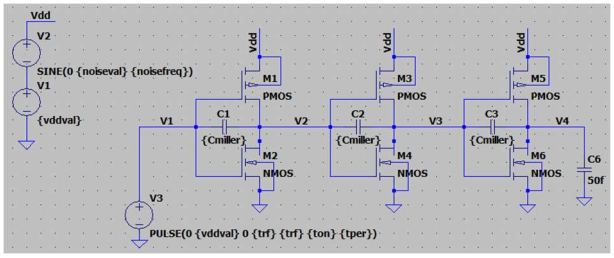


Figure 61: Three Cascaded Inverters.

Figure 62 shows the three cascaded CML circuits, similarly, V1 supplies 1.0V and V2 is sourcing a 9MHz signal with various amplitudes, This helps to provoke 0%, 1%, 5%, and 10% noise to Vdd. V3 and V4 inject an ideal differential 100MHz square signal into the first CML, then its differential output is entered into the second CML, and successively, its differential output is entered into the third CML, the final differential output is obtained at V4p and V4m.

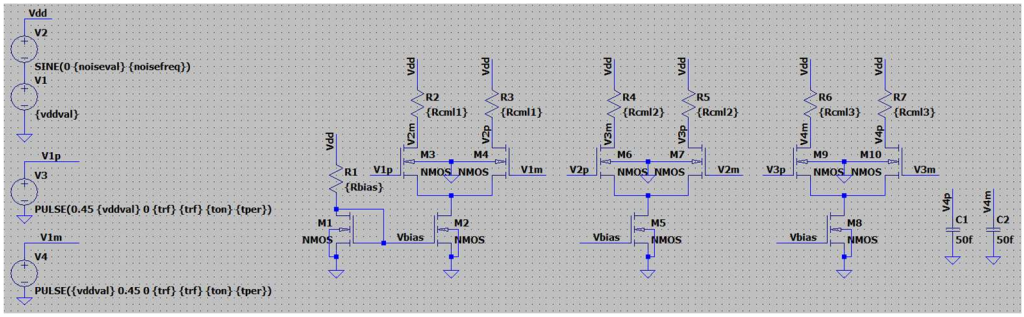


Figure 62: Three Cascaded CML.

The current at each CML phase is delimited with Current Mirrors. The phases are scaled, the width of the transistors is doubled and the Rcml resistors are divided by two, this way The output voltage of every phase is maintained, following the formula below.

$$V_{out} = IR = 2I\frac{R}{2} = 4I\frac{R}{4}$$

Equation 3: Vout formula.

In the circuit, Rbias is 10k**Ω**, M1 L/W is 180nm/6um, and ignoring the Ron of the M1, Rbias is limiting the current to 100uA, and when measured we're seeing 58.2uA, the difference is because the real Ron of M1 seems to be ~7k**Ω**.

M2 L/W is 180nm/6um, and since M2 has the same size as M1, it is mirroring and setting the same current to the 1st CML amplifier, We measured 56.3uA when M3 or M4 is active and up to 58.2uA when they transition, Rcml1 are 10k**Ω**, same as Rbias.

M5 L/W is 180nm/12.5um, its width is nearly twice as M1, so M5 is mirroring and doubling the current to the 2nd CML amplifier, we measured 117.2uA when M6 or M7 is active and up to 121uA when they transition, Rcml2 are 5k**Ω**, which is half Rbias.

M8 L/W is 180nm/25um, and its width is nearly four times the M1's width, so M8 is mirroring and quadrupling the current to the 3rd CML amplifier, we measured 234uA when M9 or M10 is active and up to 241.7uA then they transition, Rcml3 are 2.5k**Ω**, which is a quarter of Rbias.

Figure 63 shows the ppm results, v1 is the ideal source, v2 is the output of the first cml/inverter, v3 is the output of the second cml/inverter, and v4 is the output of the third cml/inverter. The values 0, 0.01, 0.05, and 0.1 is the noise, in percentage, applied to the Vdd.
It can be observed that the inverters affect the ppm, as high as ~700ppm, while the cml ppm values are under 100ppm in general.

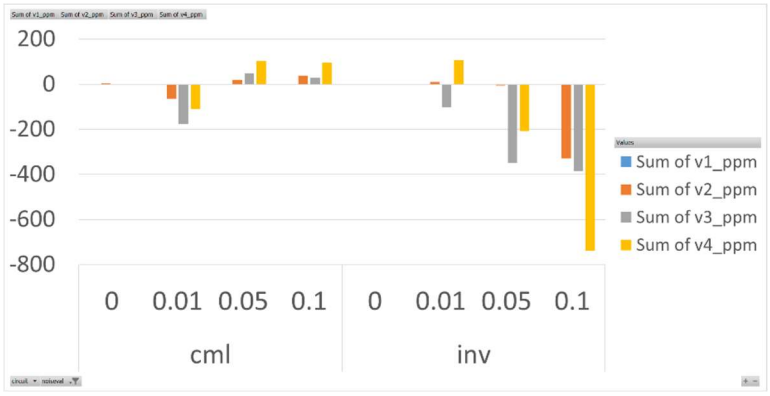


Figure 63: Ppm results, cml vs. inverter.

Figure 64 shows the jitter results, v1 is the ideal source, v2 is the output of the first cml/inverter, v3 is the output of the second cml/inverter, and v4 is the output of the third cml/inverter. The values 0, 0.01, 0.05, and 0.1 is the noise, in percentage, applied to the Vdd.
It can be observed that the inverters are noisier than cml, as they generate more jitter, 80ps vs. 10ps.

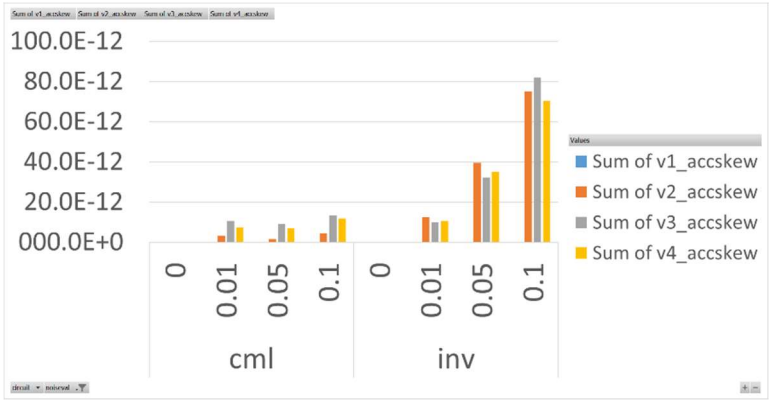


Figure 64: Jitter results, cml vs. inverter.

5.3.4. Conclusion

It was observed that the Colpitts circuit, oscillating at 32.768kHz, was very unstable, or very sensitive. A very small change in any of the components of the tank circuit provokes a large change in ppm; in one of the tries, we doubled the simulation time, from 240ms to 480ms, expecting to see a more stable result, but we observed the opposite, the ppm almost doubled. The theory about the Colpitts circuit states that this circuit is more stable at high frequencies, so 32.768kHz seems to be low for it.

Contrarily, the Colpitts circuit, oscillating at 100MHz, was observed to be very stable, a small change in the components provokes a small change in ppm. In this case, it was possible to tune the circuit to 0ppm, which is the ideal case.

In the lab, we have seen that, by changing the capacitor loads, of the crystal circuit in that case, in a few pico-farads, provokes a small change in ppm, we have been able to effectively tune our platforms to low ppm by tuning the capacitors.

CML has better performance than Inverters, in ppm and jitter. One advantage of inverters is that, if they don't oscillate, they save energy, while a disadvantage is the high noise. One disadvantage of CML is that, in silicon, a resistor takes much more area than transistors, while its advantage is the low noise, Another disadvantage is that CML is always draining current from Vdd.

5.4. The "Slew Rate" Case Study

In this section, we want to make clearer what slew rate is, and how specific characteristics of the system provoke changes in slew rate, so debugging can be accelerated.

5.4.1. Slew Rate

In the context of transmission lines, slew rate refers to the rate of change of the voltage or current signal at the output of the line. The slew rate is typically measured in volts per nanosecond (V/ns) for voltage signals and amperes per nanosecond (A/ns) for current signals.

In practical terms, a high slew rate in a transmission line means that the signal can change rapidly, which can be useful for transmitting high-frequency signals with fast transitions. However, a high slew rate can also cause problems such as overshoot, ringing, and signal distortion. Therefore, it is important to design transmission lines with appropriate impedance and termination to minimize the effects of slew rate distortion.

5.4.2. Simulations to Affect Slew Rate

In some of our products, we have seen slew rate, either lower or higher than specification, in this section, we made simulations and purposedly changed some parameters in our system, we're using the complementary buffer created in section 4.12. To get the Slew Rate, we need to obtain an Eye Diagram, and we need to measure the Rise Time and its Amplitude, this is done with the EyeDiff_Probe tool in ADS, where the desired "Measurements" can be added with a double click, Figure 65.

a) b)

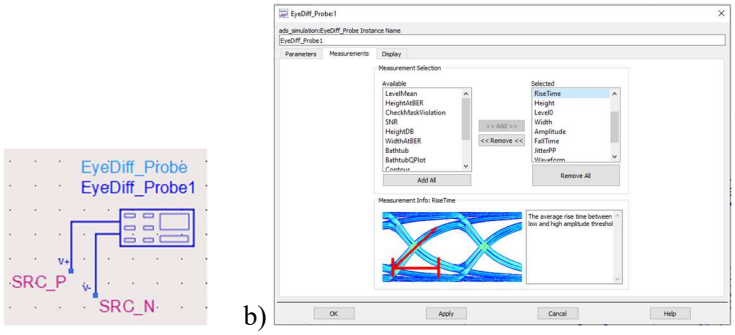


Figure 65: Eye Diff Probe setup. a) Eye Diff Probe. b) Config screen.

By default, the Rise Time is measured from 20% to 80% of the Amplitude, this threshold, if needed, can be changed in "Parameters". To obtain the Slew Rate, we need to multiply the Amplitude by 0.6, and with this, we obtain the voltage between 20% and 80%; Then we need to divide this voltage between the Rise Time, and the result is multiplied by 10^{-9} so the units will be V/ns, Equation 4.

$$Slew\ Rate = \frac{Voltage\ from\ 20\%\ to\ 60\%}{Rise\ Time\ from\ 20\%\ to\ 60\%} * 10^{-9}$$

Equation 4: Slew Rate equation.

In this experiment, we want to understand the effect on slew rate when changing three parameters of the system, the supply voltage Vdd, the length of the transmission line, and the buffer's impedance. The results can be seen in Figure 66.

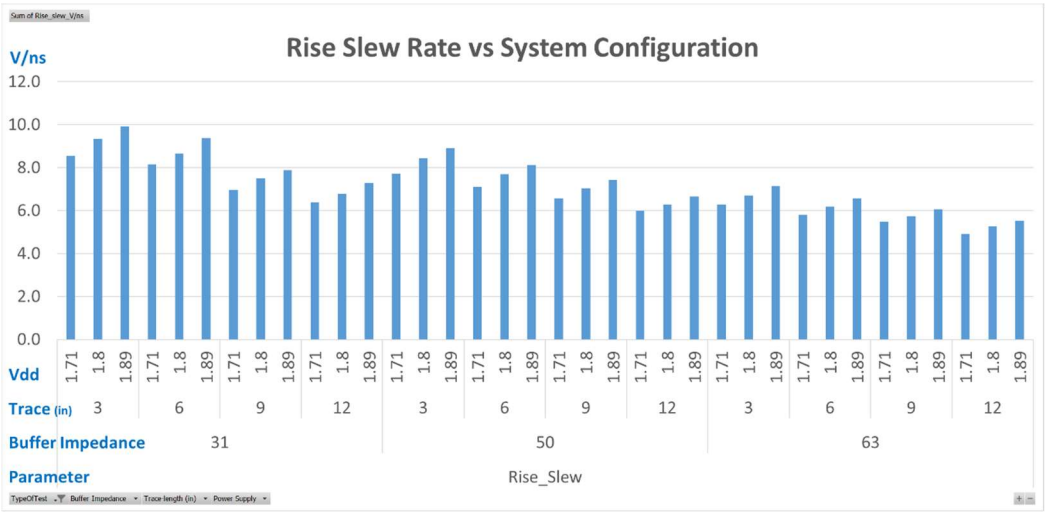


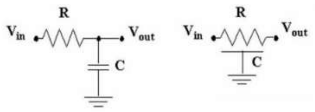
Figure 66: Rise Slew Rate vs. System Configuration.

We can observe that higher Vdd produces a higher slew rate. Another observation is that, as trace length increases, the slew rate gets lower. By reducing the buffer's impedance we can get a higher slew rate, more details in Table 11.

Table 11: Parameter change vs. Slew Rate.

Parameter	Change	Slew rate (SR) effect	Explanation
Vdd	↑↑↑	↑↑↑	The higher the Vdd, the higher the SR
Vdd	↓↓↓	↓↓↓	The lower the Vdd, the lower the SR
Trace-length	↑↑↑	↓↓↓	The larger the trace length, the lower the SR
Trace-length	↓↓↓	↑↑↑	The shorter the trace length, the higher the SR
Buffer's Z	↑↑↑	↓↓↓	The higher the buffer's Z, the lower the SR
Buffer's Z	↓↓↓	↑↑↑	The lower the buffer's Z, the higher the SR

Another way of understanding slew rate is with Elmore's delay [25], which states that the rise time depends on a constant multiplied by the resistance and capacitance of an RC network, tr = kRC, Table 12. The bigger the capacitance or resistance, the bigger the rise time, and since the slew rate relates to velocity, the bigger the rise time, the smaller the slew rate.



Voltage range	Lumped RC network	Distributed RC network
0→50% (t_p)	0.69 RC	0.38 RC
0→63% (τ)	RC	0.5 RC
10→90% (t_r)	2.2 RC	0.9 RC

Table 12: Elmore's Delay.

5.4.3. Conclusion

By understanding how certain parameters affect slew rate when debugging SR failures, we can rapidly review the configuration or architecture of our system and execute a few experiments to find the root cause or propose a fix sooner, i.e., to change the buffer's impedance, to measure and compare another clock with shorter or larger trace-length, or to increase/decrease the Vdd to observe if the failure follows a specific condition, this way we get more hints for the root cause.

6. CONCLUSIONS

Summary: *In this chapter, we're presenting the conclusions and future work related to the Optimization of Electrical Validation and Debugging for Reference Clocks.*

6.1. Conclusions

Within this work, we have analyzed the main electrical parameters that are measured to clock signals during the electrical validation, we made a table with the most common failures historically found and modeled and simulated the most important cases. These simulations helped to have a better understanding of why failures occur, what could be the most common root causes, and what experiments to execute when we find similar failures in future validations. The circuit models created in this work will be used as part of our debug process in real products to ease the root cause finding.

We have a clearer understanding of what to do when we see failures like Undershoot, Jitter, Ppm, and Slew Rate, we need to quickly review things like Impedance matching, power supply quality, circuit capacitive loading, and longitude of PCB tracks, principally.

We also observed a perfect match, comparing our simulations of the Undershoot to the measurements obtained on a real product in the laboratory, this is a good indication that we can catch this type of error during the readiness phase.

We want to capture errors before they appear, and if show up, we want to have a very good understanding of failures and quickly propose some debug experiments to find the root cause soon.

We believe that, if we catch the first four failures from the debug table, sooner during the validation or even before it starts, we can reduce the validation window, including debug, down to 70% or 80% of the actual time.

6.2. Future Work

We consider all items below can apport significant value to our team knowledge and they could be developed in the future.

- To simulate the undershoot at the Image clock in LTspice, the value we see is that LTspice is an open tool, we don't need to pay for a license, which is not the case for ADS.
- To add more variables for Jitter simulations, so we can understand how it is affected by Crosstalk, Temperature variations, Process variations, and bigger Impedance mismatches. We could also add vias and connectors to the path and observe the effect.
- To execute experiments with Even Impedances (ZE) and Odd Impedances (ZO), so we can have a better understanding of how Impedance works in Transmission Lines.
- In our real products, sometimes we can make improvements by correcting a few settings, for future work, we could add those circuits to the complementary buffer created in this work, simulate those changes, and observe their effect sooner than executing real measurements in the laboratory.
- To simulate the Bouncing created by Impedance mismatches, along with the theory and calculations, including the time delay of the Transmission Line.

BIBLIOGRAPHY

[1] M. Patel, “Fundamentals of EDVT (Electrical Design Validation/Verification Test)” Linkedin.com. https://www.linkedin.com/pulse/fundamentals-edvt-electrical-design-test-milind-patel (accessed May 7, 2023)

[2] M. Hall, “Intel” Britannica.com. https://www.britannica.com/topic/Intel (accessed May 7, 2023)

[3] E. Gonzalez, “Proceso de diseño de packaging” Esdesignbarcelona.com. https://www.esdesignbarcelona.com/int/expertos-diseno/proceso-de-diseno-de-un-producto-que-fases-existen (accessed May 7, 2023)

[4] E. Bogatin, Signal Integrity - Simplified, Upper Saddle River, NJ, USA: Prentice Hall, 2004

[5] Hdwang, “What does CPU stepping mean?” Arstechnica.com. https://arstechnica.com/civis/viewtopic.php?t=155183 (accessed May 7, 2023)

[6] Skyworks, “Timing - PCIe Clock Jitter Tool” Skyworksinc.com. https://www.skyworksinc.com/en/Application-Pages/pcie-clock-jitter-tool (accessed May 7, 2023)

[7] Intel, “Clock Jitter Tool” Designintools.intel.com. https://designintools.intel.com/Clock_Jitter_Tool_p/stlgrn124.htm (accessed May 7, 2023)

[8] Tektronix, “DPOJET” Tek.com. https://www.tek.com/en/support/software/application/dpojet-dpomso5000b-dpo7000c-dpodsamso70000cddx-dpo70000sx-v1030 (accessed May 7, 2023)

[9] Tableau, “El análisis de la causa raíz, explicado con ejemplos y metodos” Tableau.com. https://www.tableau.com/es-mx/learn/articles/root-cause-analysis (accessed May 7, 2023)

[10] D. Derickson and M. Muller, “Jitter Basics” in Digital Communications Test and Measurement: High-Speed Physical Layer Characterization, Pearson, 2007

[11] Editorial Team, “What is Frequency Stability” Everythingrf.com. https://www.everythingrf.com/community/what-is-frequency-stability#:~:text=Parts%20per%20Million%20(ppm)%3A,our%20PPM%20to%20Hz%20Calculator (accessed May 7, 2023)

[12] “What is Slew Rate: Working & its Applications” Elprocus.com. https://www.elprocus.com/slew-rate/ (accessed May 7, 2023)

[13] I. Collins, “Phase-Locked Loop (PLL) Fundamentals” Analog.com. https://www.analog.com/en/analog-dialogue/articles/phase-locked-loop-pll-fundamentals.html (accessed May 7, 2023)

[14] S. H. Hall and H. L. Heck, Advanced Signal Integrity for High-Speed Digital Designs, Wiley, 2009

[15] W. Kester, “Converting Oscillator Phase Noise to Tie Jitter” Analog.com. https://www.analog.com/media/en/training-seminars/tutorials/mt-008.pdf (accessed May 7, 2023)

[16] J. Lapiedra, “Spread Spectrum Clocking Using the CDCS502/503” Ti.com. https://www.ti.com/lit/ml/scaa103/scaa103.pdf?ts=1680035486925 (accessed May 7, 2023)

[17] J. Kehr, “System Power States” Learn.microsoft.com. https://learn.microsoft.com/en-us/windows/win32/power/system-power-states (accessed May 7, 2023)

[18] “What is duty cycle?” Fluke.com. https://www.fluke.com/en-us/learn/blog/electrical/what-is-duty-cycle (accessed May 7, 2023)

[19]"Predictive Technology Model" Ptm.asu.edu. https://ptm.asu.edu/ (accessed May 7, 2023)
[20]"180nm NMOS PMOS Spice Parametersv" Ptm.asu.edu. http://ptm.asu.edu/modelcard/180nm_bulk.txt (accessed May 7, 2023)
[21]Tertulien Ndjountche, CMOS Analog Integrated Circuits, CRC Press, 2011
[22]"Impedance Calculation" Cart.jlcpcb.com https://cart.jlcpcb.com/impedanceCalculation (accessed May 7, 2023)
[23]"Crystal Oscillator with Colpitts Configuration" ee-diary.com https://www.ee-diary.com/2022/01/crystal-oscillator-with-colpitts.html (accessed Jun 25, 2023)
[24]"The Colpitts Oscillator" Electronics-tutorials.ws https://www.electronics-tutorials.ws/oscillator/colpitts.html (accessed May 7, 2023)
[25]"The Elmore delay" inf.ufsc.br http://www.inf.ufsc.br/~luiz.santos/ine5442/slides/aulas15-16.pdf (accessed May 16, 2023)
[26]"Simulation Command" ltwiki.org https://ltwiki.org/index.php?title=Simulation_Command (accessed Jun 25, 2023)
[27]"Phase Lock Loop Fundamentals" ti.com https://www.ti.com/video/series/phase-lock-loop-fundamentals.html (accessed Jun 25, 2023)
[28]Kenneth Wyatt "Spread Spectrum Clock Generation – Theory and Debate" interferencetechnology.com Spread Spectrum Clock Generation – Theory and Debate (interferencetechnology.com) (accessed Jun 25, 2023)
[29]"Corner models: Inaccurate at best, and it only get worst" researchgate.net https://www.researchgate.net/publication/261126425_Corner_models_Inaccurate_at_best_and_it_only_gets_worst (accessed Jun 25, 2023)
[30]Will Kenton "Monte Carlo Simulation: History, How it Works, and 4 Key Steps" investopedia.com https://www.investopedia.com/terms/m/montecarlosimulation.asp (accessed Jun 25, 2023)

APPENDIX A. Measurements in LTspice

The measurements in LTspice are performed with Spice directives [26], which are created by right-clicking in an empty area of the schematic, then selecting the “Draft” option and “SPICE Directive”, and adding spice commands line by line, Figure 67.

a) b)

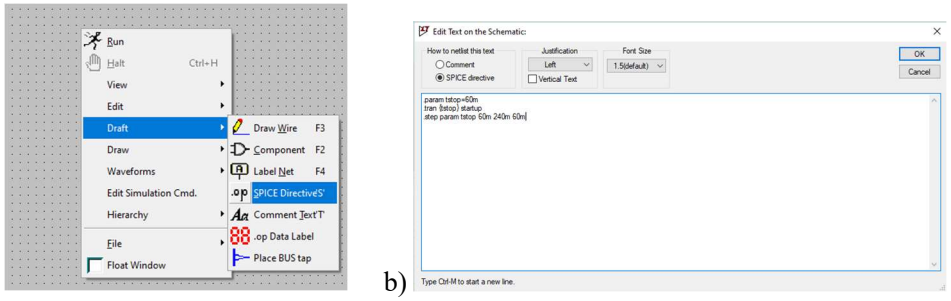


Figure 67: Spice Directives. a) Menu. b) Editor.

A.1 Measurement in Colpitts circuit at 32.768kHz

The Colpitts oscillator for 32.768kHz starts to oscillate after 24ms of simulation. We’re letting it run for 240ms and measuring several parameters, including period, frequency, duty cycle, and jitter, among others at 60ms, 120ms, 180ms, and 240ms, Figure 68. The output voltage of this oscillator can be as high as ~21.6V.

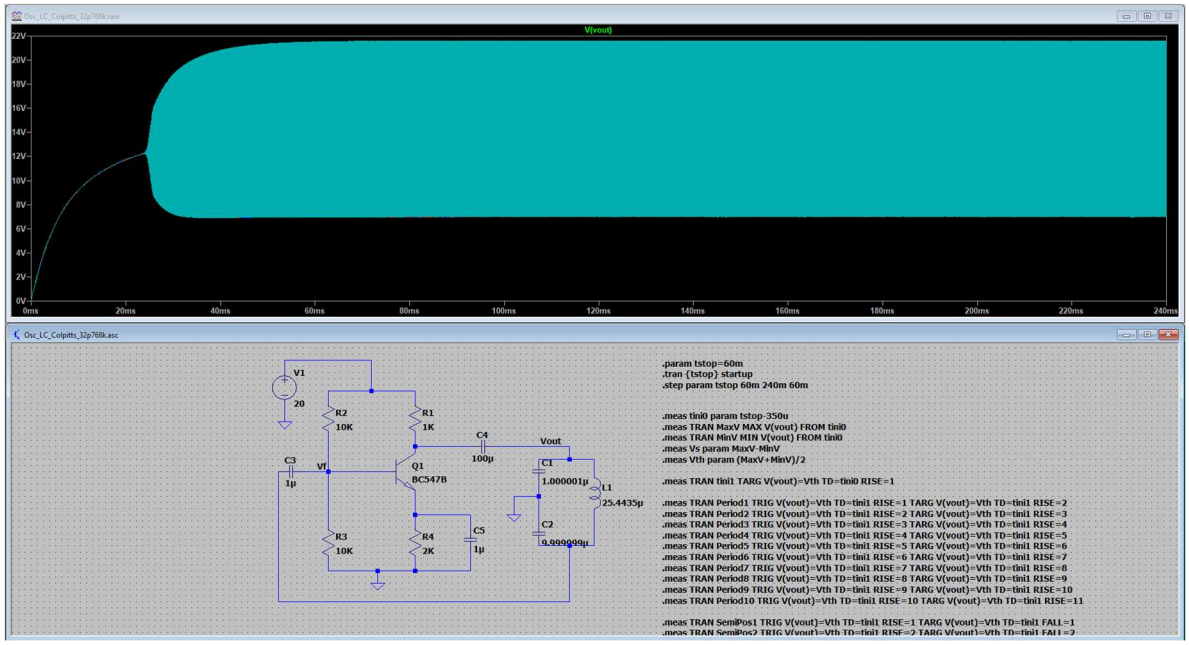


Figure 68: Colpitts for 32.768kHz simulation.

Spice Directives and Explanation:

.param tstop=60m*; variable tstopinitialized to 60ms*
.tran {tstop} startup*; time domain analysis, with step-size “tstop”, “startup” solves the initial operating point with*

```
                    * Independent voltage and current sources turned off or set to zero.
.step param tstop 60m 240m 60m; step is sweeping the variable tstop, similar to a for-loop, the sweep starts at 60ms,
                                * stops at 240ms with a step of 60ms

* tstop will be equal to 60ms, 120ms, 180ms and 240ms, one at a time
.meas tini0 param tstop-350u; tini0 equal to tstop minus 350us (getting ~11 clock cycles), clock period: 30.5us
.meas TRAN MaxV MAX V(vout) FROM tini0; measuring MaxV in vout, within 11 cycles near tstop
.meas TRAN MinV MIN V(vout) FROM tini0; measuring MinV in vout, within 11 cycles near tstop
.meas Vs param MaxV-MinV; calculating range between MinV and MaxV
.meas Vth param (MaxV+MinV)/2; calculating Vth, this is the midpoint between MaxV and MinV

.meas TRAN tini1 TARG V(vout)=Vth TD=tini0 RISE=1; tini1: time when vout first rises after tini0

.meas TRAN Period1 TRIG V(vout)=Vth TD=tini1 RISE=1 TARG V(vout)=Vth TD=tini1 RISE=2; 1st period
.meas TRAN Period2 TRIG V(vout)=Vth TD=tini1 RISE=2 TARG V(vout)=Vth TD=tini1 RISE=3; 2nd period
.meas TRAN Period3 TRIG V(vout)=Vth TD=tini1 RISE=3 TARG V(vout)=Vth TD=tini1 RISE=4; 3rd period
.meas TRAN Period4 TRIG V(vout)=Vth TD=tini1 RISE=4 TARG V(vout)=Vth TD=tini1 RISE=5; 4th period
.meas TRAN Period5 TRIG V(vout)=Vth TD=tini1 RISE=5 TARG V(vout)=Vth TD=tini1 RISE=6; 5th period
.meas TRAN Period6 TRIG V(vout)=Vth TD=tini1 RISE=6 TARG V(vout)=Vth TD=tini1 RISE=7; 6th period
.meas TRAN Period7 TRIG V(vout)=Vth TD=tini1 RISE=7 TARG V(vout)=Vth TD=tini1 RISE=8; 7th period
.meas TRAN Period8 TRIG V(vout)=Vth TD=tini1 RISE=8 TARG V(vout)=Vth TD=tini1 RISE=9; 8th period
.meas TRAN Period9 TRIG V(vout)=Vth TD=tini1 RISE=9 TARG V(vout)=Vth TD=tini1 RISE=10; 9th period
.meas TRAN Period10 TRIG V(vout)=Vth TD=tini1 RISE=10 TARG V(vout)=Vth TD=tini1 RISE=11; 10th period

.meas TRAN SemiPos1 TRIG V(vout)=Vth TD=tini1 RISE=1 TARG V(vout)=Vth TD=tini1 FALL=1; 1st duty cycle
.meas TRAN SemiPos2 TRIG V(vout)=Vth TD=tini1 RISE=2 TARG V(vout)=Vth TD=tini1 FALL=2; 2nd duty cycle
.meas TRAN SemiPos3 TRIG V(vout)=Vth TD=tini1 RISE=3 TARG V(vout)=Vth TD=tini1 FALL=3; 3rd duty cycle
.meas TRAN SemiPos4 TRIG V(vout)=Vth TD=tini1 RISE=4 TARG V(vout)=Vth TD=tini1 FALL=4; 4th duty cycle
.meas TRAN SemiPos5 TRIG V(vout)=Vth TD=tini1 RISE=5 TARG V(vout)=Vth TD=tini1 FALL=5; 5th duty cycle
.meas TRAN SemiPos6 TRIG V(vout)=Vth TD=tini1 RISE=6 TARG V(vout)=Vth TD=tini1 FALL=6; 6th duty cycle
.meas TRAN SemiPos7 TRIG V(vout)=Vth TD=tini1 RISE=7 TARG V(vout)=Vth TD=tini1 FALL=7; 7th duty cycle
.meas TRAN SemiPos8 TRIG V(vout)=Vth TD=tini1 RISE=8 TARG V(vout)=Vth TD=tini1 FALL=8; 8th duty cycle
.meas TRAN SemiPos9 TRIG V(vout)=Vth TD=tini1 RISE=9 TARG V(vout)=Vth TD=tini1 FALL=9; 9th duty cycle
.meas TRAN SemiPos10 TRIG V(vout)=Vth TD=tini1 RISE=10 TARG V(vout)=Vth TD=tini1 FALL=10; 10th dc

.meas Cyc2Cyc1 param Period2-Period1; Calculating cycle to cycle jitter, which is the delta between adjacent periods
.meas Cyc2Cyc2 param Period3-Period2
.meas Cyc2Cyc3 param Period4-Period3
.meas Cyc2Cyc4 param Period5-Period4
.meas Cyc2Cyc5 param Period6-Period5
.meas Cyc2Cyc6 param Period7-Period6
.meas Cyc2Cyc7 param Period8-Period7
.meas Cyc2Cyc8 param Period9-Period8
.meas Cyc2Cyc9 param Period10-Period9

.meas DutyCycle1 param 100*SemiPos1/Period1; Calculating duty cycle in percentage of the period
.meas DutyCycle2 param 100*SemiPos2/Period2
.meas DutyCycle3 param 100*SemiPos3/Period3
.meas DutyCycle4 param 100*SemiPos4/Period4
.meas DutyCycle5 param 100*SemiPos5/Period5
.meas DutyCycle6 param 100*SemiPos6/Period6
.meas DutyCycle7 param 100*SemiPos7/Period7
.meas DutyCycle8 param 100*SemiPos8/Period8
.meas DutyCycle9 param 100*SemiPos9/Period8
.meas DutyCycle10 param 100*SemiPos10/Period10

.meas PeriodMax param
max(Period1,max(Period2,max(Period3,max(Period4,max(Period5,max(Period6,max(Period7,max(Period8,max(Period9,
Period10)))))))))); Calculating Maximum of the 10 periods
.meas PeriodMin param
min(Period1,min(Period2,min(Period3,min(Period4,min(Period5,min(Period6,min(Period7,min(Period8,min(Period9,Peri
od10)))))))))); Calculating Minimum of the 10 periods
```

```
.meas FreqMax param 1/PeriodMin; Calculating Max Frequency
.meas FreqMin param 1/PeriodMax; Calculating Min Frequency
.meas FreqMid param 2/(PeriodMax+PeriodMin); Calculating Mid-Frequency
.meas AccSkew param PeriodMax-PeriodMin; Calculating Accumulated Skew, this is the jitter
.meas AccSkewPer param 100*AccSkew*FreqMid; Calculating Accumulated Skew in percentage
.meas Cyc2CycMax param
max(Cyc2Cyc1,max(Cyc2Cyc2,max(Cyc2Cyc3,max(Cyc2Cyc4,max(Cyc2Cyc5,max(Cyc2Cyc6,max(Cyc2Cyc7,max(Cy
c2Cyc8,Cyc2Cyc9)))))))); Calculating Max of cycle-to-cycle jitter
.meas Cyc2CycMin param
min(Cyc2Cyc1,min(Cyc2Cyc2,min(Cyc2Cyc3,min(Cyc2Cyc4,min(Cyc2Cyc5,min(Cyc2Cyc6,min(Cyc2Cyc7,min(Cyc2
Cyc8,Cyc2Cyc9)))))))); Calculating Min of cycle-to-cycle jitter
.meas DutyCycleMax param
max(DutyCycle1,max(DutyCycle2,max(DutyCycle3,max(DutyCycle4,max(DutyCycle5,max(DutyCycle6,max(DutyCycl
e7,max(DutyCycle8,max(DutyCycle9,DutyCycle10)))))))); Calculating Max of duty cycle
.meas DutyCycleMin param
min(DutyCycle1,min(DutyCycle2,min(DutyCycle3,min(DutyCycle4,min(DutyCycle5,min(DutyCycle6,min(DutyCycle7,
min(DutyCycle8,min(DutyCycle9,DutyCycle10)))))))); Calculating Min of duty cycle
```

A.2 Measurement in Colpitts circuit at 100MHz

The Colpitts oscillator for 100MHz starts to oscillate after 3us of simulation. We're letting it run for 400us. The same measurements are collected as in the 32.768kHz oscillator. The output voltage can be as high as ~21.3V, Figure 69.

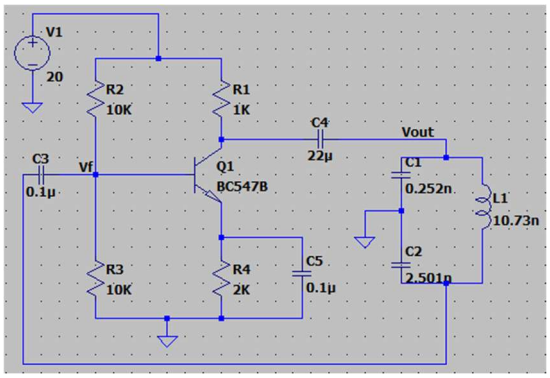


Figure 69: Oscillator LC Colpitts at 100MHz.

The commands from the 32.768kHz oscillator can be reused, only below 4 lines are different:

```
.param tstop=100u; variable tstop initialized to 100us
.step param tstop 100u 400u 100u; step is sweeping the variable tstop, similar to a for-loop, the sweep starts at 100us,
                                  * stops at 400us with a step of 100us
.meas tini0 PARAM tstop-1u; tini0 equal to tstop minus 1us (getting ~100 clock cycles), clock period: 10ns
```

A.3 Measurement in cascaded Inverters circuit

Below we can see the Spice directives to make measurements in the Inverters circuit from Figure 70.

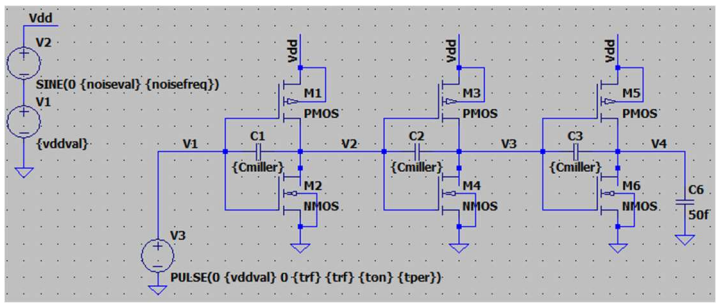


Figure 70: Three Cascaded Inverters.

We have a set of measurements for V1, V2, V3, and V4, including period, frequency, duty cycle, and jitter, among others.

***** general parameters *****
.param vddval=1*; V1 power supply set to 1V*
.param noiseval=0.1*; voltage for noise injection to the power supply*
.param noisefreq=9Meg*; frequency for noise injection to the power supply*
.param tper=10n*; period of the clock V3 entering to the inverters*
.param trf=tper/10*; rise time of the clock V3 set to 10% of the period*
.param ton=tper/2-trf*; high time of the clock V3*
.param Cmiller=0f*; Miller capacitors in the inverters*
.param LoTh=0.2*; low threshold set to 20%*
.param HiTh=0.8*; high threshold set to 80%*
.param tstop=120n*; time to stop set to 120ns*
.tran {tstop}*; transient analysis stopping at tstop*
.meas tini0 PARAM tstop-115n*; tini0 equal to tstop minus 115ns*
.meas v0_Vth PARAM vddval/2*; v0_Vth equal to half vddval*

*****v1 measurements*****
.meas TRAN v1_MaxV MAX V(v1) FROM tini0*; measuring Max V on v1 node*
.meas TRAN v1_MinV MIN V(v1) FROM tini0*; measuring Min V on v1 node*
.meas v1_Vs param v1_MaxV-v1_MinV*; Range of voltage between MinV and MaxV on v1 node*
.meas v1_Vth param (v1_MaxV+v1_MinV)/2; *calculating v1_Vth, this is the midpoint between MaxV and MinV*
.meas TRAN v1_tini1 TARG V(v1)=v0_Vth TD=tini0 RISE=1*; v1_tini1: time when v1 first rises after tini0*

.meas TRAN v1_Period1 TRIG V(v1)=v0_Vth TD=v1_tini1 RISE=1 TARG V(v1)=v0_Vth TD=v1_tini1 RISE=2*; period 1*
.meas TRAN v1_Period2 TRIG V(v1)=v0_Vth TD=v1_tini1 RISE=2 TARG V(v1)=v0_Vth TD=v1_tini1 RISE=3*; period 2*
…
…
.meas TRAN v1_Period10 TRIG V(v1)=v0_Vth TD=v1_tini1 RISE=10 TARG V(v1)=v0_Vth TD=v1_tini1 RISE=11*; period 10*

.meas TRAN v1_SemiPos1 TRIG V(v1)=v0_Vth TD=v1_tini1 RISE=1 TARG V(v1)=v0_Vth TD=v1_tini1 FALL=1*; 1st duty cycle*
.meas TRAN v1_SemiPos2 TRIG V(v1)=v0_Vth TD=v1_tini1 RISE=2 TARG V(v1)=v0_Vth TD=v1_tini1 FALL=2*; 2nd duty cycle*
…
…
.meas TRAN v1_SemiPos10 TRIG V(v1)=v0_Vth TD=v1_tini1 RISE=10 TARG V(v1)=v0_Vth TD=v1_tini1 FALL=10*;10th duty cycle*

.meas TRAN v1_Rise1 TRIG V(v1)=vddval*LoTh TD=v1_tini1 RISE=1 TARG V(v1)=vddval*HiTh TD=v1_tini1 RISE=2*; 1st rise time on v1*
.meas TRAN v1_Rise2 TRIG V(v1)=vddval*LoTh TD=v1_tini1 RISE=2 TARG V(v1)=vddval*HiTh TD=v1_tini1 RISE=3*; 2nd rise time on v1*
…
…
.meas TRAN v1_Rise10 TRIG V(v1)=vddval*LoTh TD=v1_tini1 RISE=10 TARG V(v1)=vddval*HiTh TD=v1_tini1 RISE=11*; 10th rise time on v1*

.meas TRAN v1_Fall1 TRIG V(v1)=vddval*HiTh TD=v1_tini1 FALL=2 TARG V(v1)=vddval*LoTh TD=v1_tini1 FALL=2*; 1st fall time on v1*
.meas TRAN v1_Fall2 TRIG V(v1)=vddval*HiTh TD=v1_tini1 FALL=3 TARG V(v1)=vddval*LoTh TD=v1_tini1 FALL=3*; 2nd fall time on v1*
…
…
.meas TRAN v1_Fall10 TRIG V(v1)=vddval*HiTh TD=v1_tini1 FALL=11 TARG V(v1)=vddval*LoTh TD=v1_tini1 FALL=11*; 10th fall time on v1*

.meas v1_Cyc2Cyc1 param v1_Period2-v1_Period1*; Calculating cycle to cycle jitter, which is the delta between adjacent periods*
.meas v1_Cyc2Cyc2 param v1_Period3-v1_Period2
…

```
            …
.meas v1_Cyc2Cyc9 param v1_Period10-v1_Period9

.meas v1_DutyCycle1 param 100*v1_SemiPos1/v1_Period1; Calculating duty cycle in percentage of the period
.meas v1_DutyCycle2 param 100*v1_SemiPos2/v1_Period2
            …
            …
.meas v1_DutyCycle10 param 100*v1_SemiPos10/v1_Period10

.meas v1_PeriodMax param
max(v1_Period1,max(v1_Period2,max(v1_Period3,max(v1_Period4,max(v1_Period5,max(v1_Period6,max(v1_Period7,
max(v1_Period8,max(v1_Period9,v1_Period10)))))))); v1 period max
.meas v1_PeriodMin param
min(v1_Period1,min(v1_Period2,min(v1_Period3,min(v1_Period4,min(v1_Period5,min(v1_Period6,min(v1_Period7,min
(v1_Period8,min(v1_Period9,v1_Period10)))))))); v1 period min
.meas v1_FreqMax param 1/v1_PeriodMin; max Frequency on v1
.meas v1_FreqMin param 1/v1_PeriodMax; min Frequency on v1
.meas v1_FreqMid param 2/(v1_PeriodMax+v1_PeriodMin); mid-Frequency on v1
.meas v1_100Mppm param 1e6*(v1_FreqMid-100e6)/100e6; Frequency ppm on v1
.meas v1_AccSkew param v1_PeriodMax-v1_PeriodMin; Accumulated skew jitter on v1
.meas v1_AccSkewPer param 100*v1_AccSkew*v1_FreqMid; Accumulated skew jitter in percentage on v1
.meas v1_Cyc2CycMax param
max(v1_Cyc2Cyc1,max(v1_Cyc2Cyc2,max(v1_Cyc2Cyc3,max(v1_Cyc2Cyc4,max(v1_Cyc2Cyc5,max(v1_Cyc2Cyc6,m
ax(v1_Cyc2Cyc7,max(v1_Cyc2Cyc8,v1_Cyc2Cyc9)))))))); max of cycle-to-cycle jitter
.meas v1_Cyc2CycMin param
min(v1_Cyc2Cyc1,min(v1_Cyc2Cyc2,min(v1_Cyc2Cyc3,min(v1_Cyc2Cyc4,min(v1_Cyc2Cyc5,min(v1_Cyc2Cyc6,min(
v1_Cyc2Cyc7,min(v1_Cyc2Cyc8,v1_Cyc2Cyc9)))))))); min of cycle-to-cycle jitter
.meas v1_Cyc2CycMaxPer param 100*v1_Cyc2CycMax*v1_FreqMid; max cycle to cycle jitter in percentage
.meas v1_Cyc2CycMinPer param 100*v1_Cyc2CycMin*v1_FreqMid; min cycle to cycle jitter in percentage

.meas v1_DutyCycleMax param
max(v1_DutyCycle1,max(v1_DutyCycle2,max(v1_DutyCycle3,max(v1_DutyCycle4,max(v1_DutyCycle5,max(v1_Duty
Cycle6,max(v1_DutyCycle7,max(v1_DutyCycle8,max(v1_DutyCycle9,v1_DutyCycle10)))))))); max of duty cycle
.meas v1_DutyCycleMin param
min(v1_DutyCycle1,min(v1_DutyCycle2,min(v1_DutyCycle3,min(v1_DutyCycle4,min(v1_DutyCycle5,min(v1_DutyCy
cle6,min(v1_DutyCycle7,min(v1_DutyCycle8,min(v1_DutyCycle9,v1_DutyCycle10)))))))); min of duty cycle

.meas v1_RiseMax param
max(v1_Rise1,max(v1_Rise2,max(v1_Rise3,max(v1_Rise4,max(v1_Rise5,max(v1_Rise6,max(v1_Rise7,max(v1_Rise8,
max(v1_Rise9,v1_Rise10)))))))); max of rise time
.meas v1_RiseMin param
min(v1_Rise1,min(v1_Rise2,min(v1_Rise3,min(v1_Rise4,min(v1_Rise5,min(v1_Rise6,min(v1_Rise7,min(v1_Rise8,min
(v1_Rise9,v1_Rise10)))))))); min of rise time
.meas v1_FallMax param
max(v1_Fall1,max(v1_Fall2,max(v1_Fall3,max(v1_Fall4,max(v1_Fall5,max(v1_Fall6,max(v1_Fall7,max(v1_Fall8,max(
v1_Fall9,v1_Fall10)))))))); max of fall time
.meas v1_FallMin param
min(v1_Fall1,min(v1_Fall2,min(v1_Fall3,min(v1_Fall4,min(v1_Fall5,min(v1_Fall6,min(v1_Fall7,min(v1_Fall8,min(v1_
Fall9,v1_Fall10)))))))); min of fall time

*****v2 measurements*****
* same code as in v1 measurements, just need to rename all v1 as v2

*****v3 measurements*****
* same code as in v1 measurements, just need to rename all v1 as v3

*****v4 measurements*****
* same code as in v1 measurements, just need to rename all v1 as v4

*****Measurements Summary*****
.meas sv1_MaxV PARAM v1_MaxV; Max Voltages
.meas sv2_MaxV PARAM v2_MaxV
.meas sv3_MaxV PARAM v3_MaxV
```

.meas sv4_MaxV PARAM v4_MaxV
.meas sv1_MinV PARAM v1_MinV*; Min Voltages*
.meas sv2_MinV PARAM v2_MinV
.meas sv3_MinV PARAM v3_MinV
.meas sv4_MinV PARAM v4_MinV
.meas sv1_RiseMax PARAM v1_RiseMax*; Max, Min, Rise, Fall v1*
.meas sv1_RiseMin PARAM v1_RiseMin
.meas sv1_FallMax PARAM v1_FallMax
.meas sv1_FallMin PARAM v1_FallMin
.meas sv2_RiseMax PARAM v2_RiseMax*; Max, Min, Rise, Fall v2*
.meas sv2_RiseMin PARAM v2_RiseMin
.meas sv2_FallMax PARAM v2_FallMax
.meas sv2_FallMin PARAM v2_FallMin
.meas sv3_RiseMax PARAM v3_RiseMax*; Max, Min, Rise, Fall v3*
.meas sv3_RiseMin PARAM v3_RiseMin
.meas sv3_FallMax PARAM v3_FallMax
.meas sv3_FallMin PARAM v3_FallMin
.meas sv4_RiseMax PARAM v4_RiseMax*; Max, Min, Rise, Fall v4*
.meas sv4_RiseMin PARAM v4_RiseMin
.meas sv4_FallMax PARAM v4_FallMax
.meas sv4_FallMin PARAM v4_FallMin
.meas sv1_AccSkew PARAM v1_AccSkew*; Accumulated Skew, jitter*
.meas sv2_AccSkew PARAM v2_AccSkew
.meas sv3_AccSkew PARAM v3_AccSkew
.meas sv4_AccSkew PARAM v4_AccSkew
.meas sv1_100Mppm PARAM v1_100Mppm*; Frequency ppm*
.meas sv2_100Mppm PARAM v2_100Mppm
.meas sv3_100Mppm PARAM v3_100Mppm
.meas sv4_100Mppm PARAM v4_100Mppm

A.4 Measurement in cascaded CMLs circuit

These are the cascaded CMLs and the Spice directives to make measurements, Figure 71.

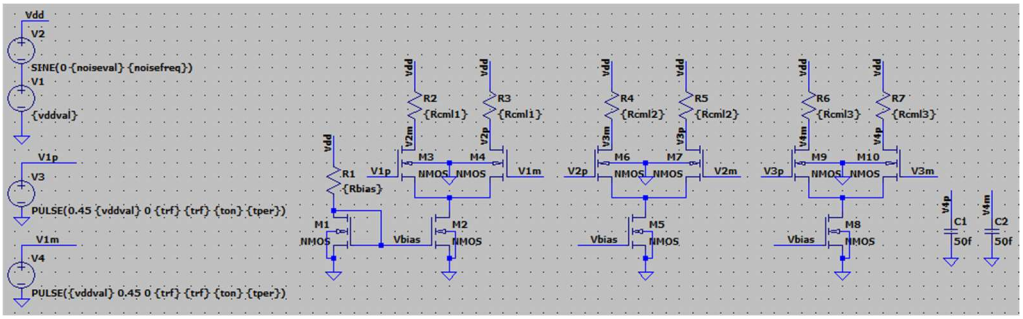


Figure 71: Three Cascaded CML.

***** general parameters *****
.param vddval=1*; supply voltage set to 1V*
.param noiseval=0.1*; noise voltage*
.param noisefreq=9Meg*; noise frequency*
.param tper=10n*; period of clock injected*
.param trf=tper/10*; rise/fall time*
.param ton=tper/2-trf*; high time*
.param Rbias=10K*; resistor for the current mirror*
.param Rcml1=10K*; cml1 load resistor*
.param Rcml2=5K*; cml2 load resistor*
.param Rcml3=2.5K*; cml3 load resistor*
.param Wdiode=6u*; M1 width*
.param Wtail1=6u*; M2 width*

```
.param Wdiff1=6u; M3 and M4 width
.param Wtail2=12.5u; M5 width
.param Wdiff2=12.5u; M6 and M7 width
.param Wtail3=25u; M8 width
.param Wdiff3=25u; M9 and M10 width
.param LoTh=-0.3; Low threshold
.param HiTh=0.3; High threshold
.param tstop=120n; tstop set to 120ns
.tran {tstop}; transient analysis ending at tstop

.meas tini0 PARAM tstop-115n; tini0 set to tstop minus 115n
.meas v0_Vth PARAM vddval/2; Vth set to half vddval

*****v1 measurements*****
.meas TRAN v1p_MaxV MAX V(v1p) FROM tini0; MaxV on v1p
.meas TRAN v1p_MinV MIN V(v1p) FROM tini0; MinV on v1p
.meas TRAN v1m_MaxV MAX V(v1m) FROM tini0; MaxV on v1m
.meas TRAN v1m_MinV MIN V(v1m) FROM tini0; MinV on v1m
.meas TRAN v1_MaxV MAX V(v1p)-V(v1m) FROM tini0; MaxV on differential v1
.meas TRAN v1_MinV MIN V(v1p)-V(v1m) FROM tini0; MinV on differential v1
.meas v1_Vs param v1_MaxV-v1_MinV; voltage range between Max and Min on differential v1
.meas v1_Vth param (v1_MaxV+v1_MinV)/2; threshold voltage, half of differential signal
.meas TRAN v1_tini1 TARG V(v1p)-V(v1m)=v1_Vth TD=tini0 RISE=1; v1_tini1: time at first rise after tini0

.meas TRAN v1_Period1 TRIG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=1 TARG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=2; period 1
.meas TRAN v1_Period2 TRIG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=2 TARG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=3; period 2
            …
            …
.meas TRAN v1_Period10 TRIG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=10 TARG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=11; period 10

.meas TRAN v1_SemiPos1 TRIG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=1 TARG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 FALL=1; 1st duty cycle
.meas TRAN v1_SemiPos2 TRIG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=2 TARG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 FALL=2; 2nd duty cycle
            …
            …
.meas TRAN v1_SemiPos10 TRIG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 RISE=10 TARG V(v1p)-V(v1m)=v1_Vth TD=v1_tini1 FALL=10; 10th duty cycl

.meas TRAN v1_Rise1 TRIG V(v1p)-V(v1m)=vddval*LoTh TD=v1_tini1 RISE=1 TARG V(v1p)-V(v1m)=vddval*HiTh TD=v1_tini1 RISE=2; 1st rise
.meas TRAN v1_Rise2 TRIG V(v1p)-V(v1m)=vddval*LoTh TD=v1_tini1 RISE=2 TARG V(v1p)-V(v1m)=vddval*HiTh TD=v1_tini1 RISE=3; 2nd rise
            …
            …
.meas TRAN v1_Rise10 TRIG V(v1p)-V(v1m)=vddval*LoTh TD=v1_tini1 RISE=10 TARG V(v1p)-V(v1m)=vddval*HiTh TD=v1_tini1 RISE=11; 10rd ri

.meas TRAN v1_Fall1 TRIG V(v1p)-V(v1m)=vddval*HiTh TD=v1_tini1 FALL=2 TARG V(v1p)-V(v1m)=vddval*LoTh TD=v1_tini1 FALL=2; 1st fall
.meas TRAN v1_Fall2 TRIG V(v1p)-V(v1m)=vddval*HiTh TD=v1_tini1 FALL=3 TARG V(v1p)-V(v1m)=vddval*LoTh TD=v1_tini1 FALL=3; 2nd fall
            …
            …
.meas TRAN v1_Fall10 TRIG V(v1p)-V(v1m)=vddval*HiTh TD=v1_tini1 FALL=11 TARG V(v1p)-V(v1m)=vddval*LoTh TD=v1_tini1 FALL=11; 10th f

.meas v1_Cyc2Cyc1 param v1_Period2-v1_Period1; Calculating cycle to cycle jitter, which is the delta between adjacent periods
.meas v1_Cyc2Cyc2 param v1_Period3-v1_Period2
            …
            …
.meas v1_Cyc2Cyc9 param v1_Period10-v1_Period9

.meas v1_DutyCycle1 param 100*v1_SemiPos1/v1_Period1; Calculating duty cycle in percentage of the period
.meas v1_DutyCycle2 param 100*v1_SemiPos2/v1_Period2
            …
            …
.meas v1_DutyCycle10 param 100*v1_SemiPos10/v1_Period10

.meas v1_PeriodMax param
max(v1_Period1,max(v1_Period2,max(v1_Period3,max(v1_Period4,max(v1_Period5,max(v1_Period6,max(v1_Period7,
max(v1_Period8,max(v1_Period9,v1_Period10)))))))); v1 period max
.meas v1_PeriodMin param
min(v1_Period1,min(v1_Period2,min(v1_Period3,min(v1_Period4,min(v1_Period5,min(v1_Period6,min(v1_Period7,min
(v1_Period8,min(v1_Period9,v1_Period10)))))))); v1 period min
.meas v1_FreqMax param 1/v1_PeriodMin; max frequency
```

.meas v1_FreqMin param 1/v1_PeriodMax*; min frequency*
.meas v1_FreqMid param 2/(v1_PeriodMax+v1_PeriodMin)*; mid-frequency*
.meas v1_100Mppm param 1e6*(v1_FreqMid-100e6)/100e6*; frequency ppm*
.meas v1_AccSkew param v1_PeriodMax-v1_PeriodMin*; accumulated skew jitter*
.meas v1_AccSkewPer param 100*v1_AccSkew*v1_FreqMid*; Accumulated skew jitter in percentage*

.meas v1_Cyc2CycMax param
max(v1_Cyc2Cyc1,max(v1_Cyc2Cyc2,max(v1_Cyc2Cyc3,max(v1_Cyc2Cyc4,max(v1_Cyc2Cyc5,max(v1_Cyc2Cyc6,max(v1_Cyc2Cyc7,max(v1_Cyc2Cyc8,v1_Cyc2Cyc9))))))))*; max of cycle to cycle jitter*
.meas v1_Cyc2CycMin param
min(v1_Cyc2Cyc1,min(v1_Cyc2Cyc2,min(v1_Cyc2Cyc3,min(v1_Cyc2Cyc4,min(v1_Cyc2Cyc5,min(v1_Cyc2Cyc6,min(v1_Cyc2Cyc7,min(v1_Cyc2Cyc8,v1_Cyc2Cyc9))))))))*; min of cycle to cycle jitter*
.meas v1_Cyc2CycMaxPer param 100*v1_Cyc2CycMax*v1_FreqMid*; max cycle to cycle jitter in percentage*
.meas v1_Cyc2CycMinPer param 100*v1_Cyc2CycMin*v1_FreqMid*; min cycle to cycle jitter in percentage*
.meas v1_DutyCycleMax param
max(v1_DutyCycle1,max(v1_DutyCycle2,max(v1_DutyCycle3,max(v1_DutyCycle4,max(v1_DutyCycle5,max(v1_DutyCycle6,max(v1_DutyCycle7,max(v1_DutyCycle8,max(v1_DutyCycle9,v1_DutyCycle10))))))))*; max of duty cycle*
.meas v1_DutyCycleMin param
min(v1_DutyCycle1,min(v1_DutyCycle2,min(v1_DutyCycle3,min(v1_DutyCycle4,min(v1_DutyCycle5,min(v1_DutyCycle6,min(v1_DutyCycle7,min(v1_DutyCycle8,min(v1_DutyCycle9,v1_DutyCycle10))))))))*; min of duty cycle*

.meas v1_RiseMax param
max(v1_Rise1,max(v1_Rise2,max(v1_Rise3,max(v1_Rise4,max(v1_Rise5,max(v1_Rise6,max(v1_Rise7,max(v1_Rise8,max(v1_Rise9,v1_Rise10))))))))*; max of rise time*
.meas v1_RiseMin param
min(v1_Rise1,min(v1_Rise2,min(v1_Rise3,min(v1_Rise4,min(v1_Rise5,min(v1_Rise6,min(v1_Rise7,min(v1_Rise8,min(v1_Rise9,v1_Rise10))))))))*; min of rise time*
.meas v1_FallMax param
max(v1_Fall1,max(v1_Fall2,max(v1_Fall3,max(v1_Fall4,max(v1_Fall5,max(v1_Fall6,max(v1_Fall7,max(v1_Fall8,max(v1_Fall9,v1_Fall10))))))))*; max of fall time*
.meas v1_FallMin param
min(v1_Fall1,min(v1_Fall2,min(v1_Fall3,min(v1_Fall4,min(v1_Fall5,min(v1_Fall6,min(v1_Fall7,min(v1_Fall8,min(v1_Fall9,v1_Fall10))))))))*; min of fall time*

*****v2 measurements*****
* same code as in v1 measurements, just need to rename all v1 as v2

*****v3 measurements*****
* same code as in v1 measurements, just need to rename all v1 as v3

*****v4 measurements*****
* same code as in v1 measurements, just need to rename all v1 as v4

*****Measurements Summary1*****
.meas sv1p_MaxV PARAM v1p_MaxV*; Max p Voltages*
…
.meas sv4p_MaxV PARAM v4p_MaxV
.meas sv1p_MinV PARAM v1p_MinV*; Min p Voltages*
…
.meas sv4p_MinV PARAM v4p_MinV
.meas sv1m_MaxV PARAM v1m_MaxV*; Max m Voltages*
…
.meas sv4m_MaxV PARAM v4m_MaxV
.meas sv1m_MinV PARAM v1m_MinV*; Min m Voltages*
…
.meas sv4m_MinV PARAM v4m_MinV
.meas sv1_MaxV PARAM v1_MaxV*; Max diff Voltages*
…
.meas sv4_MaxV PARAM v4_MaxV
.meas sv1_MinV PARAM v1_MinV*; Min diff Voltages*
…
.meas sv4_MinV PARAM v4_MinV

.meas sv1_RiseMax PARAM v1_RiseMax*; Max, Min, Rise, Fall v1, repeats for v2, v3 and v4*
.meas sv1_RiseMin PARAM v1_RiseMin

```
.meas sv1_FallMax PARAM v1_FallMax
.meas sv1_FallMin PARAM v1_FallMin

.meas sv1_FreqMid PARAM v1_FreqMid; Mid Freq
            …
.meas sv4_FreqMid PARAM v4_FreqMid
 .meas sv1_AccSkew PARAM v1_AccSkew; Accumulated Skew, jitter
            …
.meas sv4_AccSkew PARAM v4_AccSkew
.meas sv1_100Mppm PARAM v1_100Mppm; Frequency ppm
            …
.meas sv4_100Mppm PARAM v4_100Mppm
.meas sv1_AccSkewPer PARAM v1_AccSkewPer; Accumulated Skew, jitter in percentage
            …
.meas sv4_AccSkewPer PARAM v4_AccSkewPer
.meas sv1_Cyc2CycMaxPer PARAM v1_Cyc2CycMaxPer; Max cycle to cycle jitter
            …
.meas sv4_Cyc2CycMaxPer PARAM v4_Cyc2CycMaxPer
.meas sv1_Cyc2CycMinPer PARAM v1_Cyc2CycMinPer; Min cycle to cycle jitter
            …
.meas sv4_Cyc2CycMinPer PARAM v4_Cyc2CycMinPer
.meas sv1_DutyCycleMax PARAM v1_DutyCycleMax; Max Duty cycle
            …
.meas sv4_DutyCycleMax PARAM v4_DutyCycleMax
.meas sv1_DutyCycleMin PARAM v1_DutyCycleMin; Min Duty cycle
            …
.meas sv4_DutyCycleMin PARAM v4_DutyCycleMin
```

APPENDIX B. Phase Lock Loop (PLL)

The PLL is a circuit whose objective is to adjust to a desired Output frequency [27], the basic configuration can be seen in Figure 72.

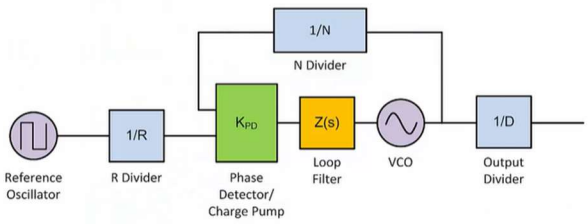


Figure 72: Basic PLL configuration.

My director helped me to make this PLL, The original circuit obtained from the internet was not working, which is very similar to this one, and it was very difficult and time-consuming to get it to work. In the end, this is the working configuration for LTspice, Figure 73.

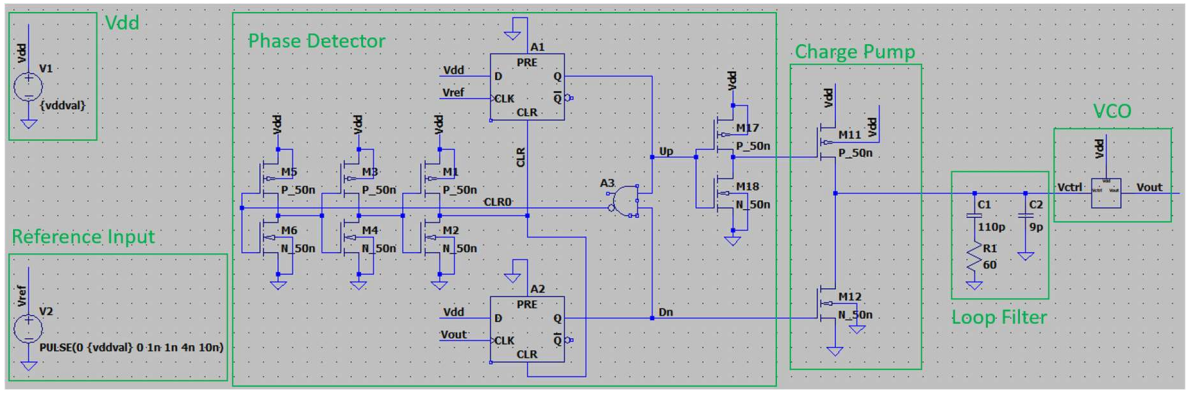


Figure 73: PLL circuit, implemented in LTspice.

For learning purposes, our PLL is receiving a 100MHz square signal, and the desired output is the same, so we don't have an "R Divider" at the input of the PLL, nor an "N Divider" in the feedback loop, nor an "Output Divider" at the output of the PLL.

R-Divider divides down the input Reference frequency, so the PLL can work with a lower frequency input. N-Divider divides down the output frequency of the VCO and creates the feedback to the Phase Detector, its output should match the frequency of the R-Divider's output, so the VCO can work at higher frequencies. Output-Divider divides down the frequency of the VCO to the final output frequency.

B.1 Supply Voltage and Reference Input

Vdd is supplying 1.0V to the circuit. Vref is a 100MHz square signal with rise and fall times set to 1ns, and a period equal to 10ns.

B.2 Phase Detector and Charge Pump

The Phase Detector consists of two D-flops, one Nand, and a Delay circuit, and the Charge Pump consists of one Pmos and one Nmos, Figure 74.

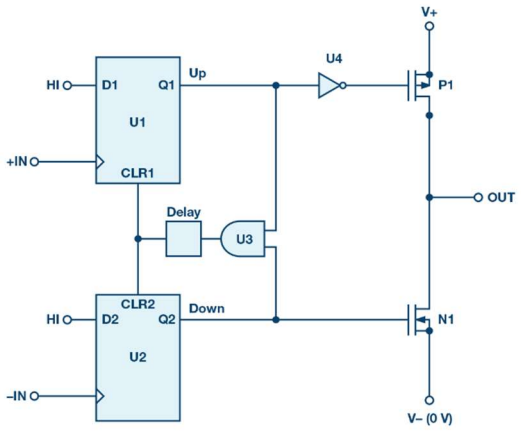


Figure 74: Phase Frequency Detector and Charge Pump.

The phase frequency detector compares the VCO's output (Vout) and the reference input (Vref) and converts this to a correction voltage. The charge pump converts this correction voltage into a correction current.

If the compared frequencies are the same, then the PLL is considered to be in "Lock" and theoretically the output of the charge pump should be in high impedance, in practice, there are narrow and alternating positive and negative pulses of current with a period equal to the phase detector period. When out of the lock, these pulses become much wider and in the same direction, this adjusts the voltage to the loop filter, which controls the VCO output frequency. In summary, the charge pump sources current if the output frequency/phase is too low, and it sinks current if the output frequency/phase is too high.

B.3 Loop Filter

The Loop Filter is a low pass filter, implemented with C1, R1, and C2. It integrates the correction currents and produces a tuning voltage to steer the VCO output frequency. The loop filter has a dramatic effect on performance, it determines the PLL loop bandwidth, which impacts the phase noise spurs and switching speed.

The PLL closed-loop transfer function has a huge impact on phase noise, spurs, and lock time. The PLL loop bandwidth (BW), is the most critical parameter that impacts these performance metrics, Figure 75.

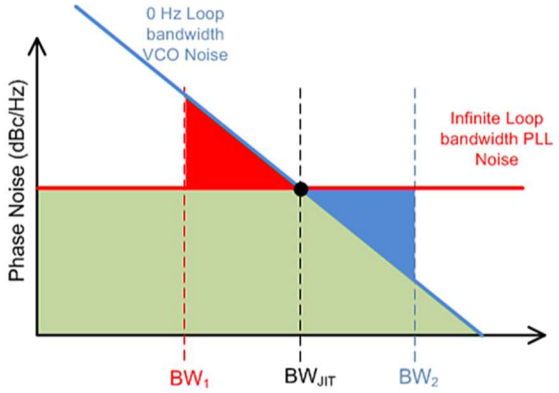


Figure 75: PLL Bandwidth.

To minimize jitter, choose the loop bandwidth to be BW_{JIT}, which minimizes the area under the curve, this is where the free-running VCO and phase noise are equal. BW can be increased or decreased to achieve a different goal [27], see Table 13.

Table 13: Design Goal vs. Bandwidth.

Design Goal	Bandwidth
Minimize Jitter	$BW = BW_{JIT}$
Minimize Phase Noise at offset < BW_{JIT}	$BW > BW_{JIT}$
Minimize Phase Noise at offset > BW_{JIT}	$BW < BW_{JIT}$
Minimize Lock Time	$BW > BW_{JIT}$
Minimize Spurs	$BW < BW_{JIT}$

B.4 Voltage Controlled Oscillator (VCO)

The VCO produces frequencies within a limited frequency band, typically higher than the reference clock. The VCO's frequency is tunable, and it's controlled by changing its input voltage, and it can vary significantly over temperature, supply voltage, or semiconductor process.

Our VCO is built with a ring oscillator, which is a circuit composed of an odd number of inverters in a ring, whose outputs switch between two values, low or high, Figure 76. The inverters are attached in a chain and the output of the last inverter is fed back into the first. Our ring oscillator is made with 5 inverters, built by M1/M2, M3/M4, M5/M6, M7/M8, M9/M10. The Pmos transistors M26, M15, M16, M17, M18, and M19 are controlling the current to the Pmos of the inverters from the "Vctrl" input. Similarly, the Nmos transistors M25, M20, M21, M22, M23, and M24 control the current to the Nmos of the inverters from the "Vctrl" input. We have 2 more inverters just before "Vout", formed by M11/M12 and M13/M14, they improve the rise/fall times of the "Vout".

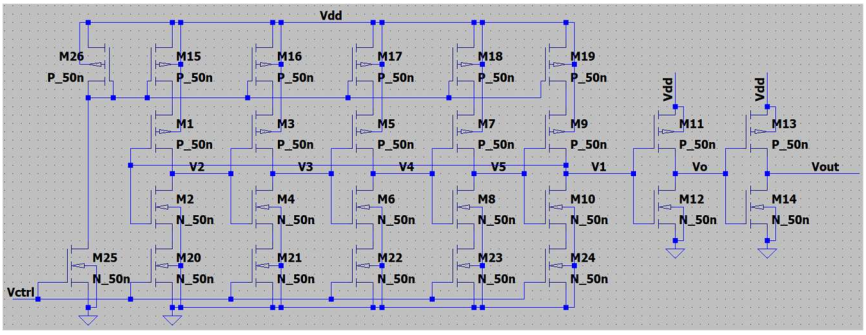


Figure 76: Voltage Controlled Oscillator.

B.5 Measurements

We tried to adjust the PLL as close as possible to 100MHz and less than 100ppm, which is the PCIe specification when SSC is off. While measuring ppm at the output of the VCO, C1, C2, and R1 in the loop filter were swept, see Table 14, Table 15, and Table 16. The average ppm was obtained by measuring approximately the last 10 periods of the simulation, calculating the frequencies of the 10 periods, calculating the average frequency of them, and the average ppm is calculated from the average frequency, more details can be seen in section B.7.

Table 14: PLL Loop Filer adjustment, C1 sweep.

C1 (pF)	C2 (pF)	R1 (Ω)	Avg PPM	Note
60	10	100	-102.057	
70	10	100	48.2754	
80	10	100	74.5155	
90	10	100	80.7398	
100	10	100	-69.9759	
110	10	100	13.3808	Base / Best
120	10	100	-186.818	
130	10	100	78.31	
140	10	100	58.6802	
150	10	100	-87.974	

Table 15: PLL Loop Filter adjustment, C2 sweep.

C1 (pF)	C2 (pF)	R1 (Ω)	Avg PPM	Note
110	5	100	-323.447	
110	6	100	82.5588	
110	7	100	-150.512	
110	8	100	-69.3328	
110	9	100	-0.12677	Best
110	10	100	13.3808	Base
110	11	100	-71.2855	
110	12	100	-114.401	
110	13	100	90.5293	
110	14	100	32.4274	

Table 16: PLL Loop Filter adjustment, R1 sweep.

C1 (pF)	C2 (pF)	R1 (Ω)	Avg PPM	Note
110	9	50	65.9354	
110	9	60	-2.43E-08	Best
110	9	70	124.955	
110	9	80	81.3133	
110	9	90	8.68422	
110	9	100	-0.126767	
110	9	110	-58.5055	
110	9	120	82.5245	
110	9	130	-86.3701	
110	9	140	11.6098	
110	9	150	88.5844	

The best values found are: C1: 110pF, C2: 9pF and R1: 60Ω, obtaining a ppm of -2.4e-8, which is practically 0ppm.

B.6 Size of the transistors

The size of the transistors in the PLL is shown below, in the form of Length / Width.

Phase Detector Pmos: M5, M3, M1, M17: 180n/1u
Phase Detector Nmos: M6, M4, M2, M18: 180n/1u
Charge Pump Pmos: M11: 180n/1.6u
Charge Pump Nmos: M12: 180n/1u
VCO Pmos: M26, M15, M16, M17, M18, M19, M11: 180n/2u
VCO Pmos: M1, M3, M5, M7, M9, M13: 180n/3.31u
VCO Nmos: M2, M4, M6, M8, M10, M14: 180n/1.66u
VCO Nmos: M25, M20, M21, M22, M23, M24, M12: 180n/1u

B.7 Spice directives for the measurements

Below we can see the Spice directives to make the measurements in the PLL.

```
***** general parameters *****
.include BSIM4_models.txt; adding the model of the Pmos/Nmos transistors
.param tstop=10u; tstop set to 10us
.param tmeas=9.88u; tmeas set to 9.88us
.param vddval=1; vddval set to 1V
.param P1val=3.31u; all P1val transistor width set to 3.31um
.param N1val=1.66u; all N1val transistor width set to 1.66um
.param P2val=2u; all P2val transistor width set to 2um
.param N2val=1u; all N2val transistor width set to 1um
.param P1val2=2u; all P1val2 transistor width set to 2um
.param N1val2=1u; all N1val2 transistor width set to 1um
.param LoTh=0.1; Low threshold set to 0.1V
.param HiTh=0.9; High threshold set to 0.9V
.tran {tstop}; tran analysis during tstop time

*****vref measurements*****
.meas v0_Vth PARAM vddval/2; Vth set to half vddval
```

```
.meas tini0 PARAM tmeas; tini0 set to tmeas time
.meas TRAN vref_Period1 TRIG V(vref)=v0_Vth TD=tmeas RISE=1 TARG V(vref)=v0_Vth TD=tmeas RISE=2; period of vref
.meas vref_freq PARAM 1/vref_Period1; frequency of Vref
.meas vref_100Mppm param 1e6*(vref_freq-100e6)/100e6; ppm of Vref

*****vout measurements*****
.meas TRAN vout_tini1 TARG V(Vout)=v0_Vth TD=tini0 RISE=1; vout_tini1: time at first rise after tini0

.meas TRAN vout_Period1 TRIG V(Vout)=v0_Vth TD=vout_tini1 RISE=1 TARG V(Vout)=v0_Vth TD=vout_tini1 RISE=2; period 1
.meas TRAN vout_Period2 TRIG V(Vout)=v0_Vth TD=vout_tini1 RISE=2 TARG V(Vout)=v0_Vth TD=vout_tini1 RISE=3; period 2
            …
            …
.meas TRAN vout_Period10 TRIG V(Vout)=v0_Vth TD=vout_tini1 RISE=10 TARG V(Vout)=v0_Vth TD=vout_tini1 RISE=11; period 10

.meas TRAN vout_SemiPos1 TRIG V(Vout)=v0_Vth TD=vout_tini1 RISE=1 TARG V(Vout)=v0_Vth TD=vout_tini1 FALL=1; 1st duty cycle
.meas TRAN vout_SemiPos2 TRIG V(Vout)=v0_Vth TD=vout_tini1 RISE=2 TARG V(Vout)=v0_Vth TD=vout_tini1 FALL=2; 2nd duty cycle
            …
            …
.meas TRAN vout_SemiPos10 TRIG V(Vout)=v0_Vth TD=vout_tini1 RISE=10 TARG V(Vout)=v0_Vth TD=vout_tini1 FALL=10; 10th duty cycle

.meas TRAN vout_Rise1 TRIG V(Vout)=vddval*LoTh TD=vout_tini1 RISE=1 TARG V(Vout)=vddval*HiTh TD=vout_tini1 RISE=2; vout rise time
.meas TRAN vout_Fall1 TRIG V(Vout)=vddval*HiTh TD=vout_tini1 FALL=2 TARG V(Vout)=vddval*LoTh TD=vout_tini1 FALL=2; vout fall time

.meas vout_Cyc2Cyc1 param vout_Period2-vout_Period1; Calculating cycle to cycle jitter, which is the delta between adjacent periods
.meas vout_Cyc2Cyc2 param vout_Period3-vout_Period2
            …
            …
.meas vout_Cyc2Cyc9 param vout_Period10-vout_Period9

.meas vout_DutyCycle1 param 100*vout_SemiPos1/vout_Period1; Calculating duty cycle in percentage of the period
.meas vout_DutyCycle2 param 100*vout_SemiPos2/vout_Period2
            …
            …
.meas vout_DutyCycle10 param 100*vout_SemiPos10/vout_Period10

.meas vout_PeriodMax param
max(vout_Period1,max(vout_Period2,max(vout_Period3,max(vout_Period4,max(vout_Period5,max(vout_Period6,max(v
out_Period7,max(vout_Period8,max(vout_Period9,vout_Period10)))))))); Vout period max
.meas vout_PeriodMin param
min(vout_Period1,min(vout_Period2,min(vout_Period3,min(vout_Period4,min(vout_Period5,min(vout_Period6,min(vout
_Period7,min(vout_Period8,min(vout_Period9,vout_Period10)))))))); Vout period min
.meas vout_FreqMax param 1/vout_PeriodMin; max frequency
.meas vout_FreqMin param 1/vout_PeriodMax; min frequency
.meas vout_FreqMid param 2/(vout_PeriodMax+vout_PeriodMin); mid-frequency
.meas vout_100MppmMax param 1e6*(vout_FreqMax-100e6)/100e6; ppm max
.meas vout_100MppmMin param 1e6*(vout_FreqMin-100e6)/100e6; ppm min
.meas vout_100MppmMid param 1e6*(vout_FreqMid-100e6)/100e6; ppm mid
.meas vout_AccSkew param vout_PeriodMax-vout_PeriodMin; accumulated skew jitter
.meas vout_AccSkewPer param 100*vout_AccSkew*vout_FreqMid; Accumulated skew jitter in percentage
.meas vout_Cyc2CycMax param
max(vout_Cyc2Cyc1,max(vout_Cyc2Cyc2,max(vout_Cyc2Cyc3,max(vout_Cyc2Cyc4,max(vout_Cyc2Cyc5,max(vout_C
yc2Cyc6,max(vout_Cyc2Cyc7,max(vout_Cyc2Cyc8,vout_Cyc2Cyc9)))))))); max of cycle to cycle jitter
.meas vout_Cyc2CycMin param
min(vout_Cyc2Cyc1,min(vout_Cyc2Cyc2,min(vout_Cyc2Cyc3,min(vout_Cyc2Cyc4,min(vout_Cyc2Cyc5,min(vout_Cyc
2Cyc6,min(vout_Cyc2Cyc7,min(vout_Cyc2Cyc8,vout_Cyc2Cyc9)))))))); min of cycle to cycle jitter
.meas vout_Cyc2CycMaxPer param 100*vout_Cyc2CycMax*vout_FreqMid; max cycle to cycle jitter in percentage
.meas vout_Cyc2CycMinPer param 100*vout_Cyc2CycMin*vout_FreqMid; min cycle to cycle jitter in percentage
.meas vout_DutyCycleMax param
max(vout_DutyCycle1,max(vout_DutyCycle2,max(vout_DutyCycle3,max(vout_DutyCycle4,max(vout_DutyCycle5,max(vout_DutyCyc
le6,max(vout_DutyCycle7,max(vout_DutyCycle8,max(vout_DutyCycle9,vout_DutyCycle10)))))))); max of duty cycle
.meas vout_DutyCycleMin param
min(vout_DutyCycle1,min(vout_DutyCycle2,min(vout_DutyCycle3,min(vout_DutyCycle4,min(vout_DutyCycle5,min(vout_DutyCycle
6,min(vout_DutyCycle7,min(vout_DutyCycle8,min(vout_DutyCycle9,vout_DutyCycle10)))))))); min of duty cycle

.meas vout_Freq1 param 1/vout_Period1; frequency of period 1
```

```
.meas vout_Freq2 param 1/vout_Period2; frequency of period 2
            …
            …
.meas vout_Freq10 param 1/vout_Period10; frequency of period 10
.meas vout_FreqAvg param
(vout_Freq1+vout_Freq2+vout_Freq3+vout_Freq4+vout_Freq5+vout_Freq6+vout_Freq7+vout_Freq8+vout_Freq9+vout
_Freq10)/10; average frequency
.meas vout_100MppmAvg param 1e6*(vout_FreqAvg-100e6)/100e6; average ppm
```

B.8 Spread Spectrum Clock (SSC)

Spread Spectrum helps to reduce power at the output peak frequency to reduce Electromagnetic Interference (EMI) and avoid disturbing other circuits [28].

We can inject an SSC signal to our PLL at the control voltage of the VCO, so we can force it to modulate its output frequency and follow the SSC form, see "Vssc" in Figure 77.

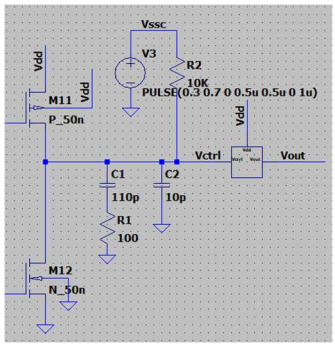


Figure 77: Spread Spectrum addition to PLL.

We tried sinusoidal, triangular, and Hershey's kiss SSC forms in our experiments, in this example, we're showing the triangular SSC case, see Figure 78.

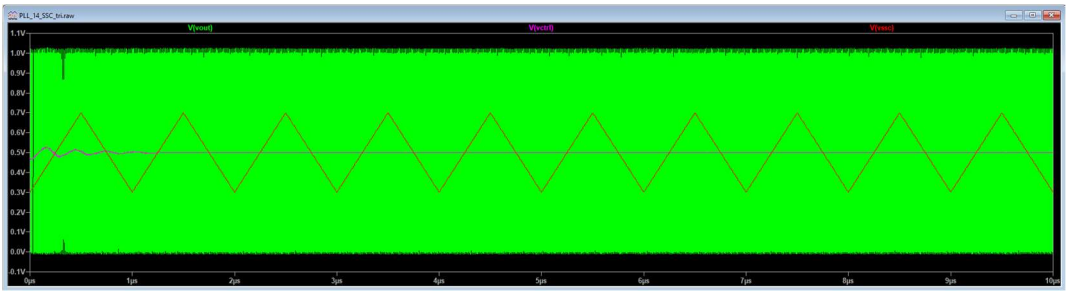


Figure 78: Triangular SSC signal.

With the Fast Fourier Transform (FFT) analysis, when SSC is applied, we can observe the effect of the SSC, the Vout power at 100MHz is slightly lower than the Vref power, Figure 79.

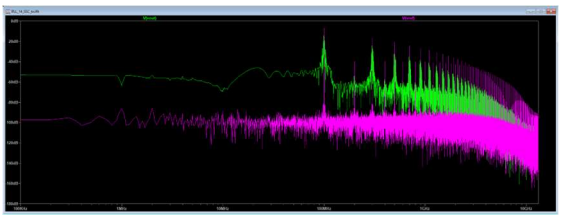
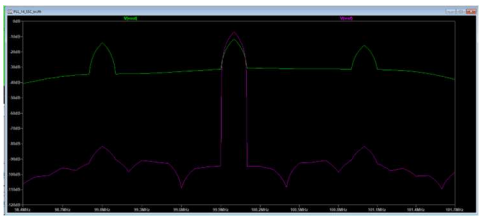
a)  b) 

Figure 79: Vref vs. Vout with SSC. a) FFT analysis. b) Zoom at 100MHz.

Contrarily, when SSC is not applied, we can observe that the Vout power at 100MHz is the same as the Vref power, Figure 80.

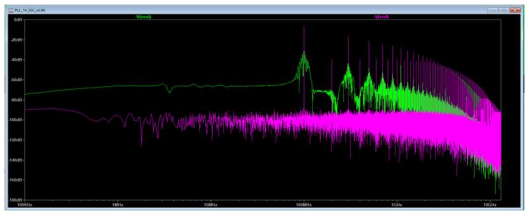
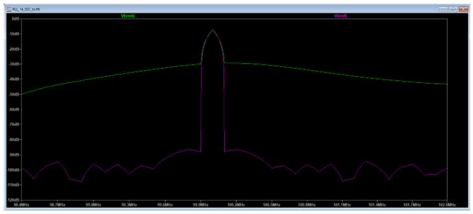
a)  b) 

Figure 80: Vref vs. Vout without SSC. a) FFT analysis. b) Zoom at 100MHz.

B.9 Conclusion

This PLL worked fine in the end, we were able to obtain 0 ppm concerning the desired output frequency of 100MHz, meaning that the PLL got locked and tuned. We were able to modulate the Pll's output frequency with an SSC signal and slightly reduce the power at the peak frequency, we could reduce it more by changing the frequency of the SSC signal. I obtained great knowledge by simulating this circuit, I now can understand how the blocks work, how the PLL is locked, how the bandwidth of the PLL is tuned with the Loop Filter, why dividers are needed in some designs, how phase detector outputs an error, that is converted to a correction current and later to a correction voltage, and how the VCO's input voltages and the output frequency is tuned. The PLL is a very interesting circuit. The knowledge obtained will help us to understand the debug scenarios seen during our validation.

APPENDIX C. Wcs, Typ, Bcs, and Monte-Carlo Analysis

Worst-Case-Speed (Wcs), Typical (Typ), and Best-Case-Speed (Bcs) analysis can be done by tweaking the parameters of the transistor from its model file or obtaining such files from the internet [29].

Monte Carlo (MC) analysis can be done by randomly changing the Length and Width of the transistors by a certain percentage and observing the effect in the simulation results [30].

Both analyses are being performed in our Cml circuit, which was previously analyzed in this document, and can be seen in Figure 81. V1p/V1m is the input signal, V2p/V2m is the output of the first Cml, V3p/V3m is the output of the second CML and V4p/V4m is the final output, which is the output of the third Cml.

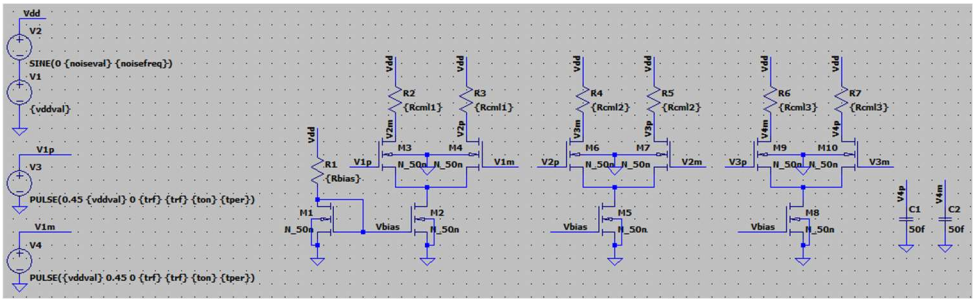


Figure 81: Three Cascaded CML.

C.1 Wcs, Typ, Bcs Simulations

As recommended by my director, we're tweaking two internal parameters of the transistor, as they seem to be the most important and he had already some experience with them from the past.

We're tweaking the threshold voltage (vth0) and the thickness of the gate oxide (toxref). The vth0 determines the voltage needed in the gate to turn it on, while the toxref determines the gate oxide capacitance (cox), by changing both parameters we can create the worst and best cases.

For Wcs, we're increasing vth0 by 20% and toxref by 8%.
For Typ, we're using the default values of the model.
For Bcs, we're reducing vth0 by 20% and toxref by 8%.

The tweaking of the models can be seen in Figure 82.

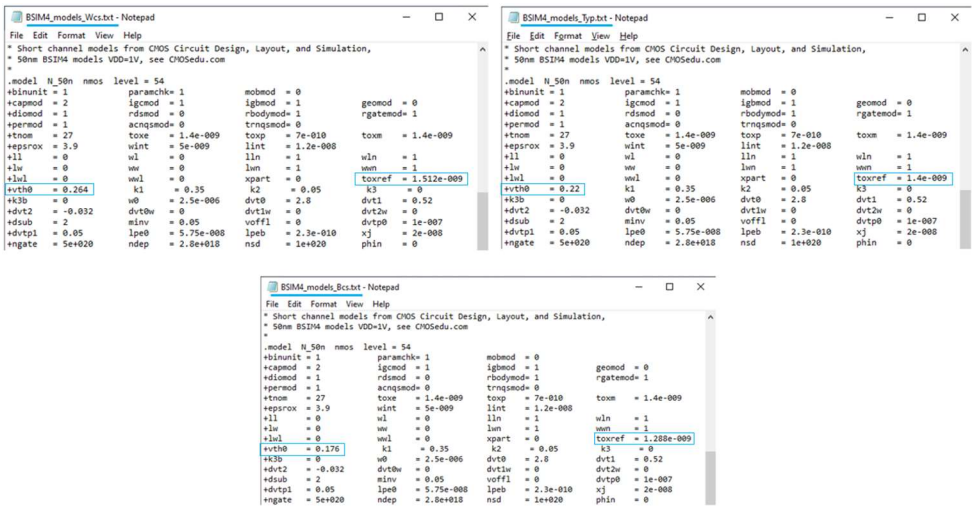


Figure 82: Models tweaked. Top left: Wcs. Top right: Typ. Bottom: Bcs

To simulate every case, we need to include the corresponding model with the spice directive ".include", see Figure 83.

a)
```
.include BSIM4_models_Wcs.txt

.param vddval=1
.param noiseval=0
.param noisefreq=9Meg
```

b)
```
.include BSIM4_models_Typ.txt

.param vddval=1
.param noiseval=0
.param noisefreq=9Meg
```

c)
```
.include BSIM4_models_Bcs.txt

.param vddval=1
.param noiseval=0
.param noisefreq=9Meg
```

Figure 83: Inclusion of the model. a) Wcs. b) Typ. c) Bcs.

The effort invested in this analysis could be exponentially increased, for example, in this case, we are also adding 0%, 1%, 5%, and 10% of noise to the power supply, so to collect all these scenarios we need to execute our simulation 12 times, this is 3 models times 4 noise-percentages.

In Figure 84, we can see the average ppm result, V2 has a higher ppm (worse) with Bcs transistors, V3 has a higher ppm with Typ transistors, and V4 has a higher ppm with Wcs transistors. V1 ppm is always 0 because this is an ideal signal from the generator source.

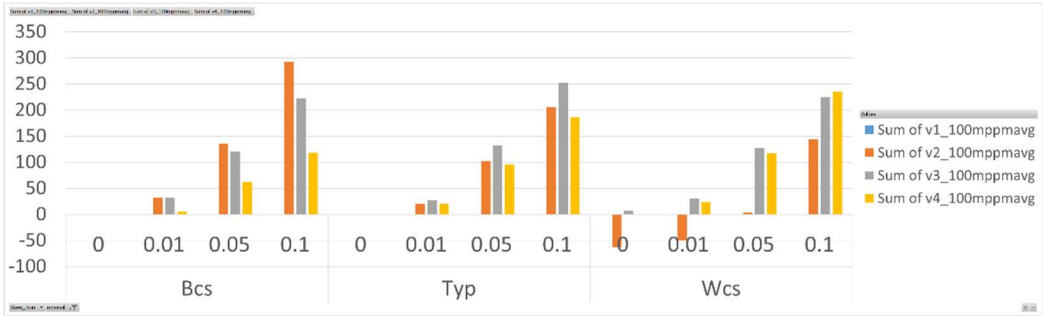


Figure 84: Bcs, Typ, Wcs, PPM results.

In Figure 85, we can see the accumulated skew or Jitter, V2 has a higher Jitter (worse) with Bcs transistors, V3 has a higher Jitter with Typ transistors, and V4 has a higher Jitter with Wcs transistors. V1 ppm is always 0 because this is an ideal signal from the generator source.

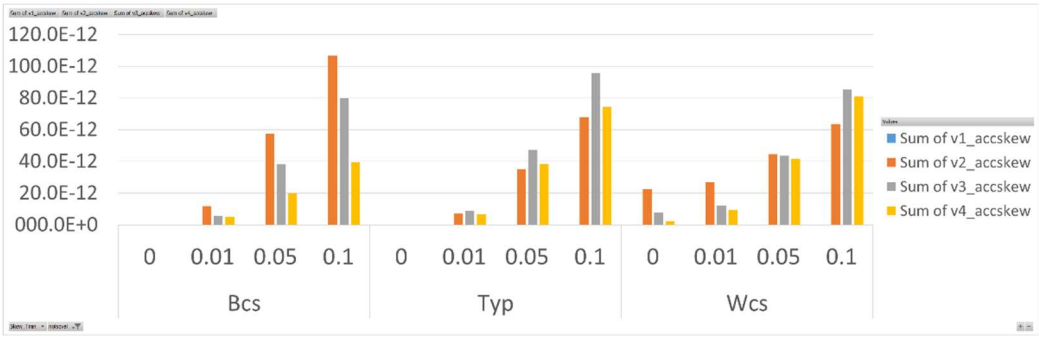


Figure 85: Bcs, Typ, Wcs, Jitter results.

In Figure 86, we can see the maximum rise and fall times, in this case, all, V2, V3, and V4 have higher maximum rise/fall (worse) with Wcs transistors, V1 rise/fall is always the same because this is an ideal signal from generator source.

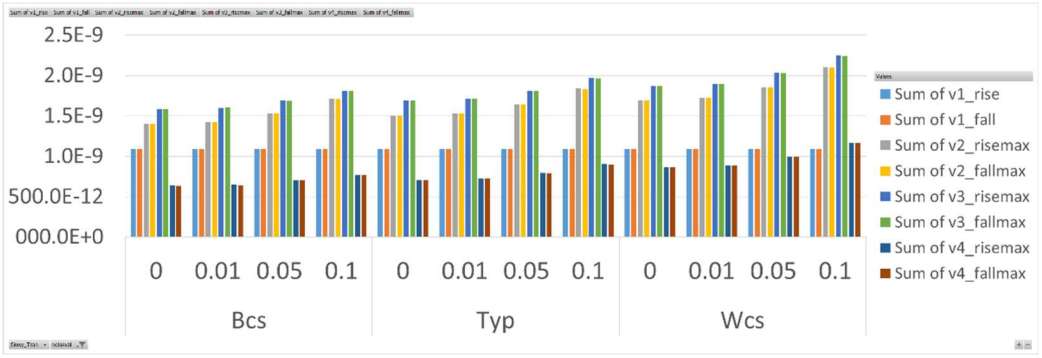


Figure 86: Bcs, Typ, Wcs, Rise time, Fall time, max results.

In Figure 87, we can see the minimum rise and fall times, like maximum cases, all, V2, V3, and V4 have higher minimum rise/fall (worse) with Wcs transistors, V1 rise/fall is always the same because this is an ideal signal from generator source.

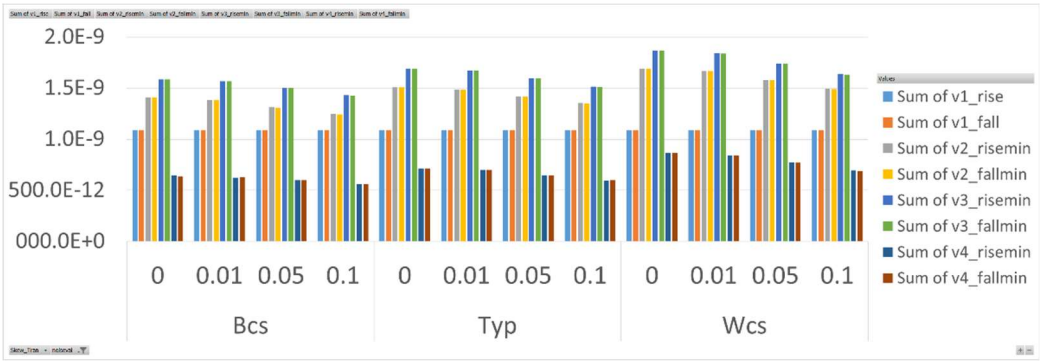


Figure 87: Bcs, Typ, Wcs, Rise time, Fall time, min results.

C.2 Monte Carlo Simulations

A Monte Carlo simulation is a mathematical technique that simulates the range of possible outcomes for an uncertain event. These predictions are based on an estimated range of values instead of a fixed set of values and evolve randomly [30].

We're making MC simulations in our CML circuit, which was analyzed in the previous section. In Figure 88 we can see the MC formula applied to the Length and Width of the transistors of the circuit.

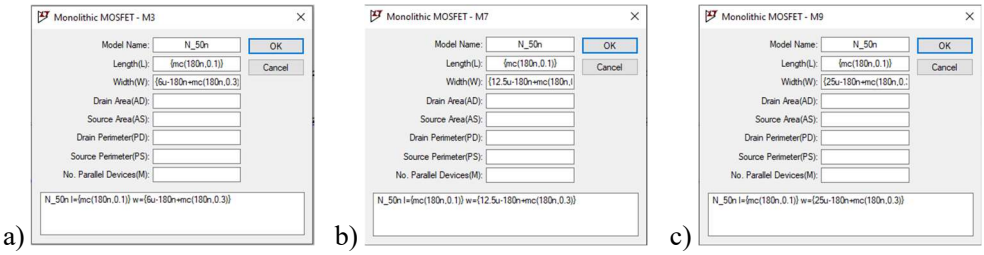


Figure 88: Monte Carlo configuration of the Nmos. a) M3. b) M7. c) M9.

From the MC formulas, we're telling LTspice to randomly change the Length of the transistors to the minimum 180nm +/- 10%, similarly, we're telling to randomly reduce and increase the Width by 30% of the minimum 180nm.

In our CML circuit, M1, M2, M3, and M4 have the same configuration, similarly, M5, M6, and M7 share the same parameters, and M8, M9, and M10 are configured the same way. The Resistors and Capacitors are simulated with their nominal values.

These are the possible MC iterations in our circuit:

Transistor models → Bcs, Typ, Wcs
Noise in the power supply → 0%, 1%, 5%, 10%
Quantity of runs → 30, 100, 200, 1024
Jitter measurement at → v1, v2, v3, v4
Ppm measurement at → v1, v2, v3, v4
Rise time measurement at → v1, v2, v3, v4
Fall time measurement at → v1, v2, v3, v4

For now, and for learning purposes, we're only executing the iteration: model Wcs, noise 10%, and Jitter at v4 output, with 200 runs.

The LTspice directive, to run MC, is: ".step param run 1 200 1" where 200 is the number of runs.

Execute the simulation, and once it's finished, select "View" and "SPICE Error Log", Figure 89.

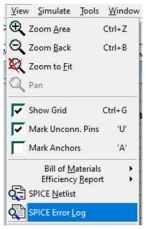


Figure 89: Monte Carlo results, part 1.

Right-click within the Log and select "Plot .step'ed .meas data", Figure 90.

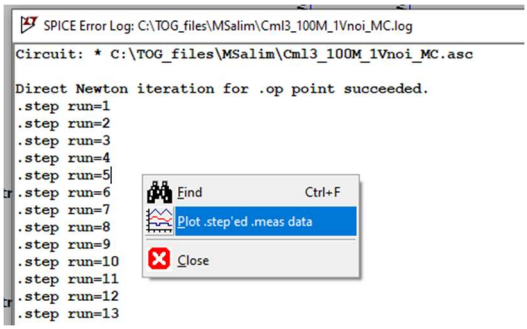


Figure 90: Monte Carlo results, part 2.

Right-click in the measurements pane and select "Add Traces", Figure 91.

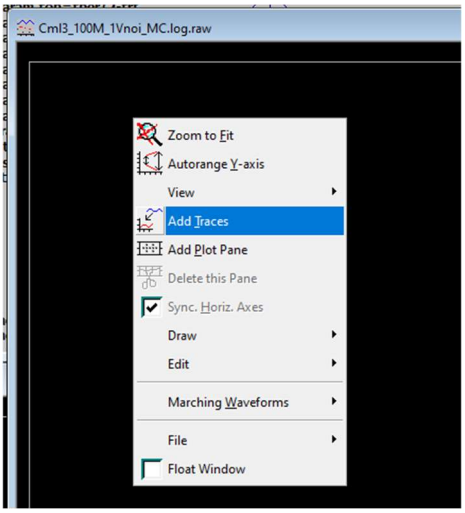


Figure 91: Monte Carlo results, part 3.

Select the measurement to plot, "sv4_accskew" in this example, for Jitter at v4 output, Figure 92.

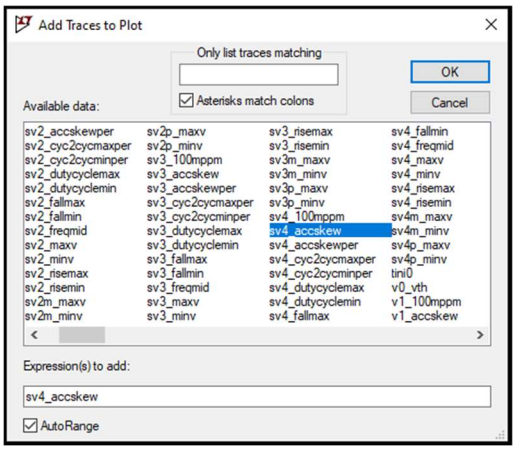


Figure 92: Monte Carlo results, part 4.

Finally, we obtain the Jitter at v4 output during the 200 runs of the Monte Carlo simulation. We can observe that the Jitter varied from 30.5ps to 44.5ps, Figure 93.

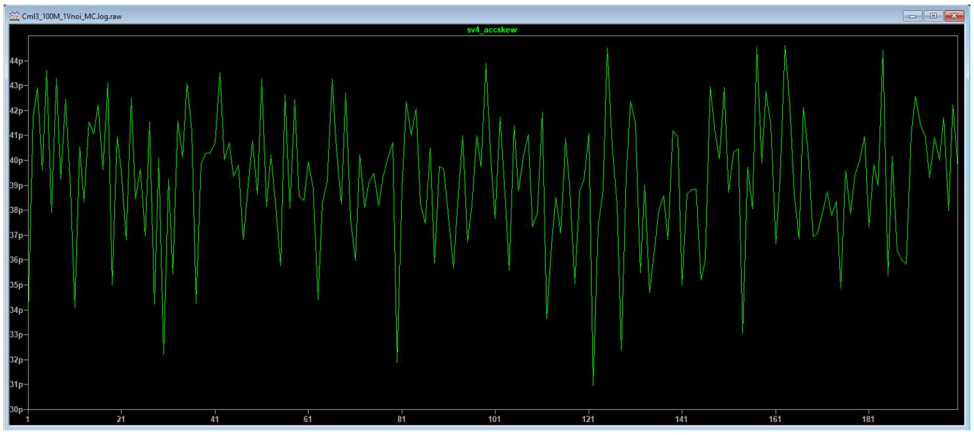


Figure 93: Monte Carlo results, part 5.

About how many runs to execute, it is not recommended very few since the precision will be low, but not so many because the simulation could last too much time. The minimum quantity could be 30, as a rule of thumb, and the maximum number could be 2 at the number of variables. For example, in our circuit we're doing MC in 10 transistors, so 2^{10} is 1024.

1024 runs would be very time-consuming, we observed that 10 runs are executed in 3.5 seconds, 30 runs in 9.2 sec, 100 runs in 34 sec, and 200 runs in 93 seconds. We also observed that minimum and maximum Jitter did not change significantly when executing a larger quantity of runs, see Table 17.

Table 17: Monte Carlo, Jitter at output v4, an example of results.

model	noise (mV)	runs	sv4_accskew_min (ps)	sv4_accskew_max (ps)	Time elapsed (s)
Wcs	0.1	10	33.84	43.62	3.5
Wcs	0.1	30	32.23	43.61	9.2
Wcs	0.1	100	31.88	43.88	34.3
Wcs	0.1	200	30.96	44.61	93

C.3 Conclusion

It is interesting how we can simulate better or worse transistors by shifting their internal parameters, saving the model, and calling it from LTspice. It is important to learn what parameters are the most representative in a real process and tweak those, another alternative is to look for best- and worst-case models from the suppliers. Then make measurements and observe the effect on specific electrical parameters of interest.

Monte Carlo analysis is another very useful tool, with this, we can easily understand how our circuit will behave when the components have skewed values, for example, values according to their tolerances from the production. We could put the MC formula to all components that need to be analyzed.