

MOS CURRENT MODE LOGIC (MCML) ANALYSIS FOR QUIET DIGITAL CIRCUITRY
AND CREATION OF A STANDARD CELL LIBRARY FOR REDUCING THE
DEVELOPMENT TIME OF MIXED-SIGNAL CHIPS

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

David Marusiak

June 2014

© 2014

David Marusiak

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: MOS Current Mode Logic (MCML) Analysis for Quiet Digital Circuitry and Creation of a Standard Cell Library for Reducing the Development Time of Mixed Signal Chips

AUTHOR: David Marusiak

DATE SUBMITTED: June 2014

COMMITTEE CHAIR: Tina Smilkstein, Ph.D.
Assistant Professor
Electrical Engineering Department

COMMITTEE MEMBER: John Y. Oliver, Ph.D.
Associate Professor, Director of CPE Program
Electrical Engineering Department

COMMITTEE MEMBER: Bridget Benson, Ph.D.
Assistant Professor
Electrical Engineering Department

ABSTRACT

MOS Current Mode Logic (MCML) Analysis for Quiet Digital Circuitry and Creation of a Standard Cell Library for Reducing the Development Time of Mixed Signal Chips

David Marusiak

Many modern digital systems use forms of CMOS logical implementation due to the straight forward design nature of CMOS logic and minimal device area since CMOS uses fewer transistors than other logic families. To achieve high-performance requirements in mixed-signal chip development and quiet, noiseless circuitry, this thesis provides an alternative to CMOS in the form of MOS Current Mode Logic (MCML). MCML dissipates constant current and does not produce noise during value changing in a circuit CMOS circuits do. CMOS logical networks switch during clock ticks and with every device switching, noise is created on the supply and ground to deal with the transitions. Creating a noiseless standard cell library with MCML allows use of circuitry that uses low voltage switching with 1.5V between logic levels in a quiet or mixed-signal environment as opposed to the full rail to rail swinging of CMOS logic. This allows cohesive implementation with analog circuitry on the same chip due to constant current and lower switching ranges not creating rail noise during digital switching. Standard cells allow for the Cadence tools to automatically generate circuits and Cadence serves as the development platform for the MCML standard cells.

The theory surrounding MCML is examined along with current and future applications well-suited for MCML are researched and explored with the goal of highlighting valid candidate circuits for MCML. Inverters and NAND gates with varying current drives are developed to meet these specialized goals and are simulated to prove

viability for quiet, mixed-signal applications. Analysis and results show that MCML is a superior implementation choice compared to CMOS for high speed and mixed signal applications due to frequency independent power dissipation and lack of generated noise during operation. Noise results show rail current deviations of 50nA to 300nA during switching over an average operating current of 20 μ A to 80 μ A respectively. The multiple order of magnitude difference between noise and signal allow the MCML cells to dissipate constant power and thus perform with no noise added to a system. Additional simulated results of a 31-stage ring oscillator result in a frequency for MCML of 1.57GHz simulated versus the 150.35MHz that MOSIS tested on a fabricated 31-stage CMOS oscillator. The layouts designed for the standard cell library conform to existing On Semiconductor ami06 technology dimensions and allow for design of any logical function to be fabricated. The I/O signals of each cell operate at the same input and output voltage swings which allow seamless integration with each other for implementation in any logical configuration.

Keywords: MCML, MOS Current Mode Logic, Digital Logic, High Speed, Standard Cell, Standard Cell Library, Current Mode Logic, Cadence virtuoso, Cadence, Low-Noise, Mixed-Signal, Constant Power

ACKNOWLEDGEMENTS

I would like to express my appreciation to my committee chair, Dr. Tina Smilkstein, who spent countless hours throughout the course of this thesis assisting me and providing the guidance necessary to make this project possible. Her dedication to the students she advised is remarkable and paramount in projects and theses like this reaching culmination at Cal Poly.

I also want to thank my committee members Dr. John Oliver and Dr. Bridget Benson who provided excellent experiences and friendships both inside and outside of the classroom. Their feedback on my thesis draft and questions and concerns during my thesis defense helped focus my thoughts in finalizing the project and creating the final polished product and thesis report.

Thanks to my family, friends, and everyone who supported me throughout my journey towards my master's degree. Your support kept the academic experience fun and exciting and helped keep me motivated on completing everything necessary to graduate within reasonable time.

And thanks to Digi-Key for providing a free schematic entry tool used for creating the schematics shown in this thesis.

Available: <http://www.digikey.com/schemeit>

TABLE OF CONTENTS

	Page
LIST OF TABLES	x
LIST OF FIGURES	xi
 CHAPTER	
1. INTRODUCTION	1
1.1 – Motivation & Goals	1
1.2 – Thesis Layout.....	3
2. AN OVERVIEW OF MCML	5
2.1 – Ideal MCML Gate	5
2.2 – Basic MCML Gates	10
2.2.1 – MCML Inverter/Buffer	10
2.2.2 – MCML AND/NAND/OR/NOR Gates.....	12
2.2.3 – MCML XOR Gate	14
2.2.4 – MCML MUX.....	15
2.2.5 – MCML D-Latch	16
2.2.6 – Arbitrary Logic Functions.....	17
2.3 – MCML to CMOS & CMOS to MCML Conversion.....	20
3. STANDARD CELL LIBRARY BASICS	23
3.1 – Overview	23
3.1.1 – Components of a Standard Cell	26
3.1.2 – Traditional Standard Cell Creation Process	31
3.1.3 – Automated Standard Cell Creation Process	33
3.1.4 – Created MCML Standard Cell Library for this Thesis	38
4. MCML CELL OPTIMIZATION AND TRANSISTOR SIZING	39
4.1 – Challenges in the Optimization of Standard Cell Libraries	39
4.2 – General Optimization Techniques	41

4.2.1 – Area.....	41
4.2.2 – Power	44
4.3 – MCML Sizing in This Project.....	46
5. MCML IMPLEMENTATION OF A STANDARD CELL LIBRARY	48
5.1 – Overview.....	48
5.2 – MCML Inverter Schematic	49
5.2.1 – 1x Inverter Schematic	50
5.2.2 – 2x Inverter Schematic	54
5.2.3 – 4x Inverter Schematic	58
5.2.4 – MCML Inverter Schematic Remarks.....	61
5.3 – MCML NAND Gate Schematic.....	63
5.3.1 – 1x NAND Gate Schematic.....	65
5.3.2 – 2x NAND Gate Schematic.....	69
5.3.3 – 4x NAND Gate Schematic.....	72
5.3.4 – MCML NAND Gate Schematic Remarks	75
5.4 – MCML Inverter Layout	77
5.4.1 – 1x Inverter Layout.....	77
5.4.2 – 2x Inverter Layout.....	80
5.4.3 – 4x Inverter Layout.....	81
5.5 – MCML NAND Gate Layout.....	83
5.5.1 – 1x NAND Gate Layout	83
5.5.2 – 2x NAND Gate Layout	85
5.5.3 – 4x NAND Gate Layout	86
5.6 – MCML Layout Remarks.....	87
5.7 – MCML Ring Oscillator.....	89
5.7.1 – MCML Gate Chaining Proof of Concept.....	89
5.7.2 – MCML Ring Oscillator Design and Test.....	93

5.8 – MCML Results Summary	96
6. CONCLUSION AND FUTURE WORK	99
6.1 – Summary	99
6.2 – Future Work	100
WORK CITED.....	103
APPENDICES	
A. Standard Cell Library Cadence Schematics and Layouts	104
B. Cadence Setup & SKILL Scripts Used For Development	124

LIST OF TABLES

Table	Page
2.1. Arbitrary 3-Input Truth Table.....	18
5.1. 1x Inverter Parameter Settings for Simulation.....	54
5.2. 2x Inverter Parameter Settings for Simulation.....	58
5.3. 4x Inverter Parameter Settings for Simulation.....	61
5.4. 1x NAND Gate Parameter Settings for Simulation	68
5.5. 2x NAND Gate Parameter Settings for Simulation	72
5.6. 4x NAND Gate Parameter Settings for Simulation	75
5.7. Inverter Results Summary Detailing Simulated Operation	96
5.8. NAND Gate Results Summary Detailing Simulated Operation	96
5.9. Frequency Sweep for the 1x MCML Inverter.....	97
5.10. Frequency Sweep for the 1x MCML NAND Gate	97

LIST OF FIGURES

Figure	Page
2.1. General Layout of an Ideal MCML Gate.....	5
2.2. MCML Inverter/Buffer Schematic	10
2.3. MCML AND/NAND/OR/NOR Schematic	13
2.4. MCML 2-Input XOR Gate & 3 Input XOR Gate Schematics.....	14
2.5. MCML 2:1 MUX Schematic	15
2.6. MCML D-Latch Schematic	17
2.7. Binary Decision Tree for Table 2.1	18
2.8. MCML Gate for Table 2.1	19
2.9. Simplified MCML gate for Table 2.1	20
2.10. MCML to CMOS & CMOS to MCML Converters.....	21
3.1. Flowchart of Various Design Paths Using Cadence Tools [10]	25
3.2. Unconnected MCML Inverter Layout & Copy of Figure 2.2 Schematic as Reference	27
3.3. Connected Layout for an MCML Inverter with no DRC Errors	28
3.4. Automatically created data sheet for D flip flop using compilation [8]	37
4.1 Reducing Component Count through Elimination of Logically Redundant Transistors that do not Affect the Final Output of the Circuit.....	43
5.1. Recreated MCML Inverter/Buffer Schematic	49
5.2. MCML Inverter Schematic with Transistor Sizing Variables	51
5.3. 1x Inverter Simulated Results Showing Correct Logical Operation	52
5.4. 1x Inverter Simulation with Current through the Tail Transistor	53
5.5. 2x Inverter Simulated Results Showing Correct Logical Operation	56
5.6. 2x Inverter Simulation	57

5.7.	4x Inverter Simulated Results Showing Correct Logical Operation	59
5.8.	4x Inverter Simulation	60
5.9.	Recreated MCML AND/NAND/OR/NOR Schematic	64
5.10.	MCML NAND Gate Schematic with Variable Sized Transistors	65
5.11.	1x NAND Gate Simulated Results Showing Logical Performance.....	66
5.12.	1x NAND Gate Simulated Results with Biasing Current Visible	67
5.13.	2x NAND Gate Simulation Showing Logical Performance	70
5.14.	2x NAND Gate Simulated Results with Current Performance Visible	71
5.15.	4x NAND Gate Simulation Showing Logical Performance	73
5.16.	4x NAND Gate Simulated Results with Current Performance Visible	74
5.17.	MCML 1x Inverter Block Layout with Pcells and Routing	79
5.18.	MCML 1x Inverter Full Routed Layout with All Layers and Pathing	79
5.19.	MCML 2x Inverter Block Layout with Pcells and Routing	81
5.20.	MCML 2x Inverter Full Routed Layout with All Layers and Pathing	81
5.21.	MCML 4x Inverter Block Layout with Pcells and Routing	82
5.22.	MCML 4x Inverter Full Routed Layout with All Layers and Pathing	82
5.23.	MCML 1x NAND Gate Block Layout with Pcells and Routing	84
5.24.	MCML 1x NAND Gate Full Routed Layout with All Layers & Pathing	85
5.25.	MCML 2x NAND Gate Block Layout with Pcells and Routing	86
5.26.	MCML 2x NAND Gate Full Routed Layout with All Layers & Pathing	86
5.27.	MCML 4x NAND Gate Block Layout with Pcells and Routing	87
5.28.	MCML 4x NAND Gate Full Routed Layout with All Layers & Pathing	87
5.29.	Three 1x MCML Inverters in Series for Concept Testing	89
5.30.	Open-Ended Three 1x MCML Inverter Chain Logical Test Results.....	90
5.31.	Open-Ended Three 1x MCML Inverter Chain with Current Data.....	91

5.32.	Three Stage MCML Ring Oscillator Concept Test	92
5.33.	Three Stage MCML Ring Oscillator Output Results.....	93
5.34.	31-Stage Ring Oscillator Block Diagram Schematic.....	94
5.35.	Steady State Peaks of the 31-Stage Ring Oscillator	95
A.1.	Model File Configuration Window for SCMOS ami06 Transistors.....	105
A.2.	Stimuli Windows for vdd and Pbias – Both Use DC Voltage Functions	106
A.3.	Stimuli Windows for the In & !In Signals – The Signals are Complements.....	107
A.4.	Highlighted MCML Inverter Schematic with Highlighted Pin List	108
A.5.	Final ADE L Window Before Running Simulation.....	109
A.6.	4x Inverter Full Logical Performance of Inverter/Buffer	110
A.7.	Vdd and Pbias Stimuli Windows for the MCML NAND Gates.....	111
A.8.	A and !A Stimuli Windows for the MCML NAND Gates	111
A.9.	B and !B Stimuli Windows for the MCML NAND Gates.....	112
A.10.	Final Parameter Window in ADE L for NAND Gate Simulation	113
A.11.	Highlighted NAND Gate Schematic with Highlighted Pin List.....	113
A.12.	1x MCML NAND Gate Full Logical Performance	114
A.13.	2x MCML NAND Gate Full Logical Performance	115
A.14.	4x MCML NAND Gate Full Logical Performance	115
A.15.	Dual Differential Pbias Control Circuitry Attempt.....	116
A.16.	MCML Pbias Control Circuitry Symbol.....	117
A.17.	1x MCML NAND Gate Schematic Ready for Layout	118
A.18.	Physical Configuration Window for an MCML Inverter	119
A.19.	Physical Configuration Window for an MCML NAND Gate	120
A.20.	Cadence Virtuoso Toolbar with ‘Create Via’ Button Indicated	121
A.21.	16x16 Multiplier Layout Generated Using 7rf Standard Cells	122

CHAPTER 1

INTRODUCTION

1.1 – Motivation & Goals

As technology has advanced over the past decades, VLSI systems experienced miniaturization of devices which lead to increased processing speed and power with reduced overall chip area. This trend results in the modern handheld devices individuals use daily, from phones to laptops and even watches. As devices become portable, the chips on the devices contain more mixed-signal components to reduce overall device cost. As the chip number increases, the packaging cost becomes the main expense in modern devices. This makes the need for quiet digital circuitry of paramount importance for the engineers designing modern consumer electronics. Mixed signal chips allow for overall reduction in total chip count and resulting packaging cost. This means that having digital and analog devices on the same IC die is becoming necessary for the production of modern systems. Electronically, this produces a need for mixed signal interaction. Devices used day to day must be reliable and robust to succeed in the current technology market. MCML allows for reduced chip number without negative performance impacts between nearby analog and digital circuits.

Devices utilizing CMOS logic have issues with precise analog circuitry when placed near switching digital devices. This is due to the ground and supply noise induced during digital switching in a clocked system. These problems escalate as devices scale down in size and concurrently, supply voltages also decrease, increasing the likelihood of interference between analog and digital circuitry [3]. MOS Current Mode Logic is a

digital logic implementation that demonstrates noiseless operation which makes it useful for mixed signal applications and minimizes the effects of the switching noise on the ground and supply rails due to constant current biasing. Examining mixed-signal applications stems from the technical market constantly trying to reduce production cost. Packaging constitutes a major cost of electronics production and modern designs aim to reduce this cost through reduction of chips. The supply and ground noise incorporated from digital electronics can cause adverse effects on analog devices in close proximity, especially when such devices require precise voltages and inputs. Even if digital and analog circuits do not share the same supply and ground rails, coupling can also provide similar negative effects on a system. MCML logic provides a means of addressing these issues with evolving electronics and creates a means for reducing cheap count and allowing for mixed signal interaction.

There are design aspects and properties of MCML that may provide drawbacks to a system and that should be kept in mind along with the myriad of benefits. Standard cell implementation of MCML devices are more complicated than their CMOS counterparts due to the added number of transistors and more complex sizing that that goes along with MCML. The stacked nature of MCML logic gates may require a higher supply than the equivalent CMOS implantation to allow for necessary voltage drops on each transistor in the chain or path. Despite these complexities, MCML is less sensitive to process variation than CMOS logic due to the differential and digital nature of the devices. MCML also allows for higher frequency operation compared to CMOS devices and dissipates constant current regardless of operating frequency which allows for less power consumed

than equivalent CMOS devices at higher frequencies. The advantages and drawbacks should be kept in mind and compared when designing electronics and mixed signal chips.

Engineers have established a large knowledge base for CMOS technology. Conversely, MCML is dramatically less understood and the information pool regarding this technology remains limited. As a result, this thesis boils down to two main goals. First, is to provide broad analysis of MCML to build upon that knowledge base and investigate applications for MCML technology and how they compare with their CMOS counterparts. The second goal is to provide a design aid by implementing MCML in a standard cell library to allow designers to utilize MCML gates without necessarily having to master their properties and create their own gates on a silicon level. Along these lines, both goals are advanced through examination of optimization methods for MCML gates and providing schematics and layouts for basic logic gates in the on-semiconductor SCMOS technology. Layouts utilizing IBM 7rf 180nm technology are also detailed for attempted chip tape out but the simulations and results base on the On Semiconductor SCMOS 600nm technology.

1.2 – Thesis Layout

This thesis examines MCML and its ability to enable quick design for noise sensitive and mixed signal chips. This analysis requires some basis of comparison, so standard CMOS devices are used as baselines for performance. Chapter 2 investigates basic MCML operation and provides an overview of the technology for conceptual purposes and developing a base of MCML theory. This includes an examination of performance characteristics and common logical gates. Chapter 3 continues investigating MCML for practicality in real world applications. This section is heavily cited as implementation of

these systems goes beyond the scope of this project but sheds light on the possible avenues for MCML in the modern technical world. Chapter 4 acts as a bridge between MCML theory and standard cell library implementation. Here, an overview of standard cell libraries is provided and the SKILL language is detailed as it serves as the means of implementation for the MCML gates within the Cadence software and the language allows for utility and optimization scripts to be written as well. Chapter 5 details standard cell optimization challenges on the IC level and explores multiple avenues for optimization to serve as a baseline for potential further research. Chapter 6 details the implementation of the MCML standard cell library and examines simulated results of the template cells. These cells include MCML inverters and NAND gates with varying current capabilities with common sized layouts that snap into an existing standard cell library. Chapter 7 provides a final summary of the work throughout the thesis project and summarizes avenues of potential future work where dozens of projects or theses could build upon the base line principles established here.

CHAPTER 2

AN OVERVIEW OF MCML

2.1 – Ideal MCML Gate

Like any theoretical foundation, one begins with the ideal case to understand the operation of a device on a general level. An MCML gate contains three main blocks: the current source, an NMOS pull-down network, and load resistances (shown in Figure 2.1).

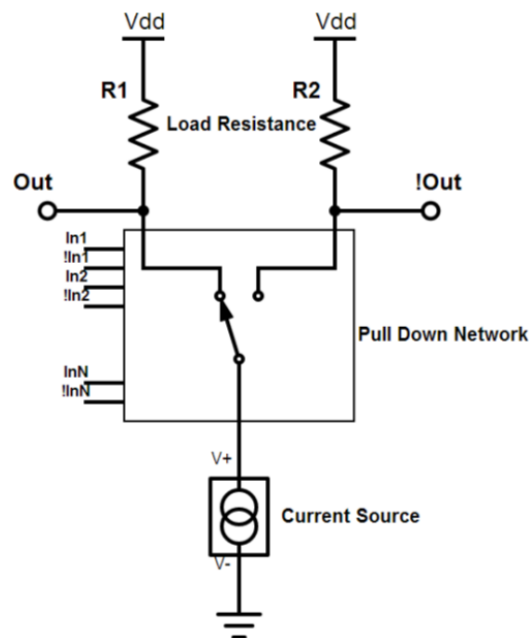


Figure 2.1: General Layout of an Ideal MCML Gate

For MCML, the inputs are differential, meaning one must provide the logical input and its inverse to the system. The pull-down network implements the Boolean function of the gate and is composed of NMOS transistors. This network steers more current to one side or the other of the two output branches, causing one side to have a lower voltage than the other, based on the logical function performed. The current source for an ideal gate will be viewed as ideal, though in practice, this source is implemented with an NMOS

transistor operating in the saturation region. This source provides a constant bias current, I_B . The output voltage swing is defined as the difference between the two output levels. Under ideal circumstances, this voltage swing V_S is given by Eq. 2.1. Note that since current is present in both paths of the device, each resistance should be tuned to achieve an equal voltage swing on either side of the device. This is especially important with non-symmetric layouts as examined later with the NAND gate. The factor of two comes from the differential nature of MCML where if one branch increases by a voltage, the other path decreases by that same voltage. The voltage swing for a *single* path through the circuit would be the same equation without the factor of two.

$$V_S = 2(V_{dd} - I_{Branch} * R_{Branch}) \quad (2.1)$$

This swing voltage is usually much smaller than V_{dd} , in the order of hundreds of millivolts, and is set exclusively by the load resistance and the biasing current source. The load resistances can be implemented as passive or active devices. On a chip, passive resistors require large amounts of real estate and depending on the resistance value, can occupy the bulk of the chip area. As a result, these load resistors are typically implemented as active devices in the form of a PMOS transistor operating in the linear region, allowing for a voltage controlled resistance.

For digital systems, delay calculations are important and can show the maximum frequency that a digital circuit can operate. For an ideal gate, MCML or otherwise, we can assume an RC time constant delay. If N identical MCML gates are connected in a chain, then the delay is realized in Eq. 2.2.

$$D_{MCML} = N * R * C \quad (2.2)$$

By using Ohm's Law where ΔV is V_s , we can define this delay as the following (Eq. 2.3):

$$D_{MCML} = N \frac{C\Delta V}{I} \quad (2.3)$$

Following standard power calculations for N identical gates, the power dissipated of a chain of identical MCML gates can be defined as follows in Eq. 2.4:

$$P_{MCML} = N * I * V_{dd} \quad (2.4)$$

Combining the power and delay values, the power-delay product can be calculated (Eq. 2.5):

$$\begin{aligned} PD_{MCML} &= P_{MCML} * D_{MCML} = (N * I * V_{dd}) * (N \frac{C\Delta V}{I}) \\ PD_{MCML} &= \frac{N^2 * C * V_{dd} * \Delta V}{I} \end{aligned} \quad (2.5)$$

Similarly, the energy-delay can be computed by taking the product of the power-delay and the delay, resulting in Eq. 2.6.

$$\begin{aligned} ED_{MCML} &= PD_{MCML} * D_{MCML} = (N^2 * C * V_{dd} * \Delta V) * (N \frac{C\Delta V}{I}) \\ ED_{MCML} &= \frac{N^3 * C^2 * \Delta V^2 * V_{dd}}{I} \end{aligned} \quad (2.6)$$

Note that the energy-delay for an MCML gate is proportional to the square of the output swing voltage. This makes designs with low voltage swings within hardware limits desirable for high frequency systems and lower power dissipation. For comparative purposes, equivalent CMOS equations are shown from Eq. 2.7 – Eq. 2.10, as derived by Mizuno *et al.* [2]:

$$D_{CMOS} = \frac{N * C * V_{dd}}{\frac{k}{2} * (V_{dd} - V_t)^\alpha} \quad (2.7)$$

$$P_{CMOS} = \frac{N * C * V_{dd}^2}{D_{CMOS}} \quad (2.8)$$

$$PD_{CMOS} = N * C * V_{dd}^2 \quad (2.9)$$

$$ED_{CMOS} = 2N^2 * \frac{C^2}{k} * \frac{V_{dd}^2}{(V_{dd}-V_t)^\alpha} \quad (2.10)$$

The parameters α and k are transistor sizing and process properties. V_t denotes the threshold voltage of the MOS transistors used in the CMOS circuit. As J. Musicer notes, MCML circuits do not have a theoretical energy-delay minimum, whereas their CMOS counterparts do [3]. An engineer can “arbitrarily reduce the energy-delay product by increasing the current for a given C , V_{dd} , and voltage swing [3].” This means that through optimization, MCML devices can achieve superior performance over CMOS logic with higher frequency of operation and less power dissipation at high frequency.

These equations are based around the fact of having N chained gates in a system. With logically chained devices in this manner, the length of the chain, or logical depth, affects the overall performance of a circuit. The power-delay and energy-delay products for MCML contain an extra N factor when compared to their CMOS counterparts due to delay dependent power consumption of CMOS devices. For large logical depth, MCML has poor delay products due to the static power consumption of an MCML gate, even when not switching [3]. Therefore, for MCML layouts, it is advantageous for the designer to minimize the logical depth if possible. For low frequencies, CMOS circuits dissipate less power than equivalent MCML circuits but for high switching frequency circuits and high performance applications, with shallow logic depth, MCML circuits vastly outperform the equivalent CMOS circuits [3]. This is due to the fact that CMOS dissipates power during every switching operation which makes power consumption a function of how often the circuit switches but MCML power remains constant regardless of operating frequency because of constant current flow. Therefore, as frequency

increases, a CMOS device will dissipate increasing amounts of power and eventually more than an MCML counterpart.

For mixed signal applications, the constant current in MCML devices is desirable as more constants in a large system will help limit the noise on sensitive analog circuitry through removal of ground and supply noise through no change in current during switching operations. CMOS circuits on the other hand, are dramatically affected by changes in current and a dI/dt can cause unwanted outcomes. For the ideal MCML case, these current changes are effectively zero. However with any realistic device, there are non-idealities in the form of small leakage currents. J. Musicer's MCML simulations show these current changes below 5%, making them relatively negligible [3]. The layout of MCML circuitry possesses inherent common mode rejection due to the differential input setup, making these devices highly resistant to power supply noise. This attribute makes MCML a favorable candidate for mixed signal applications. Since MCML circuits are differential, they are resistant to common mode noise which provides more reliable logical transitions and keeps the circuit robust in operation.

For standard cell applications, fan-out proves to be the major concern with MCML implementation. Since fan-in delays are exponential in nature they can prove to be the most taxing to performance of a logical network. Fan-in is handled by limiting the total number of inputs and/or cascading gates together. Fan-out concerns include having enough drive strength to power all the gates in a logical network. To account for fan-out, standard cells of 1x, 2x, and 4x current drives are developed for MCML to account for assorted fan-out needs. These principles are important to keep in mind and are detailed in later sections and chapters.

2.2 – Basic MCML Gates

2.2.1 – MCML Inverter/Buffer

An inverter is the simplest gate one can implement with MCML. Note that an MCML buffer shares an identical layout to an MCML inverter due to the differential nature of MCML gates. To change between gate operations, one only needs to switch the output or input. The transistor level schematic for an inverter or buffer is shown in Figure 2.2.

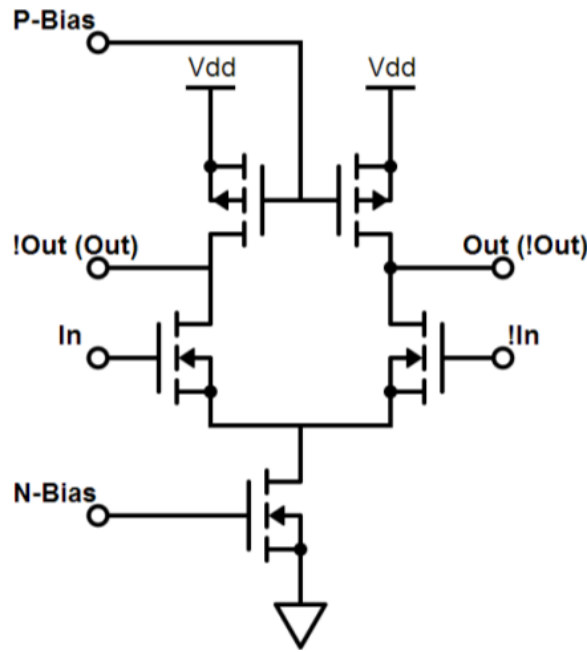


Figure 2.2: MCML Inverter/Buffer Schematic

Note that the schematic shows the buffer output pins as default. Flipping the output pins (shown in parentheses) constitutes inverter operation. For this gate, the pull-down network is implemented by the NMOS input transistors controlled with a single, differential input. Both load resistances are implemented with fixed gate voltage PMOS transistors that operate in the linear region to model passive resistor operation. Active loads can provide higher gain and faster transitions but the control of resistance is less consistent and makes the voltage swings more sensitive to variation. As mentioned with

the ideal gate, a fixed voltage NMOS transistor operating in the saturation region acts as the current source for the device. Note that creating a *better* current source can be achieved through creating a biasing network to automatically set the gate voltage of the tail current source. This can result in lower noise operation but requires more transistors and may require a higher V_{dd} . The gates designed for this thesis utilize a simple current mirror to set the gate voltage during simulation and achieve desired results but more elaborate designs for current setting are possible.

Current flowing through both paths of the device is favorable for the low-noise circuits aimed for in this thesis as with careful tuning of swing, the transistors never change regions of operation and as a result do not affect ground and supply with additional switching noise. With current flowing through both paths constantly and a swing much smaller than V_{dd} , the current steering creates the necessary output voltage for logical operation but keeps the transistors in consistent regions of operation. This means that there will be minimal noise when inputs to an MCML device switch and therefore should behave well in sensitive applications. Though many configurations can act as a current source depending on the application, MCML gates usually use a single NMOS transistor as the current source both for simplicity and to minimize the area footprint of a necessary component of the gate. However, a single biasing network that sets the gate voltage of this transistor allows for consistent and reliable operation and this thesis utilizes a current mirror to set the N-Bias voltage for the MCML gates. The PMOS load transistors are typically preferred to remain as minimally sized as possible, especially for digital applications, as increasing the width of these devices will increase the capacitance: a major issues that would cause problems given that the PMOS devices

should perform as close to passive resistors as possible. The MCML gates developed for this thesis contain the same voltage swings on input and output which is achieved through linear operation of the PMOS loads. If the load transistors operate in saturation, the voltage swings fluctuate and the circuit is less resistant to variation.

2.2.2 – MCML AND/NAND/OR/NOR Gates

The next level of complexity for basic logic gates are AND, NAND, OR, & NOR gates. Digital systems typically use NAND and NOR gates since any logic function can be implemented by those gates. However, the differential properties of MCML allow for any of these logic functions to be implemented with the same circuit topology. The presence of both polarities for inputs and outputs allow for this versatility in logical configuration. The only differences in implementing each function are the configuration of the inputs and the output branch utilized. Figure 2.3 depicts the general layout for each of these four gates with generalized pin labeling.

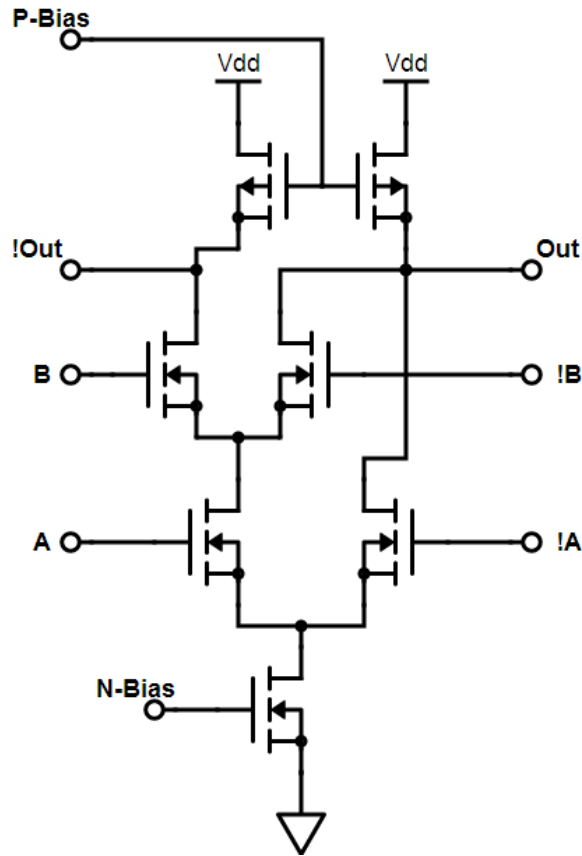


Figure 2.3: MCML AND/NAND/OR/NOR Schematic

The generalized pins show an AND gate configuration where $A \& B = \text{Out}$. Flipping the output pins achieve the NAND pin configuration for the gate where $A \& B = \overline{\text{Out}}$. The device achieves OR gate operation with the output pin configuration depicted and the polarities of the A pins flipped yielding the $A + B = \text{Out}$. With flipped A pins, the output pins could then be reversed to achieve the NOR gate operation with $\overline{A + B} = \text{Out}$. This versatility allows for simple logic design when one can use the same layout for many logic functions. However, this does rely on MCML's nature of having both polarities of each input signal present to the device. With that in place, logical configuration simply becomes a wiring setup or input/output pin selection setup.

2.2.3 – MCML XOR Gate

The XOR gate is a more complicated logic gate that serves as a benchmark in digital electronics due to its unique logical function and more component heavy layout. This results in more transistors in an XOR gate than with other basic logic functions. The layouts for a 2-input and 3-input XOR gate are shown in Figure 2.4.

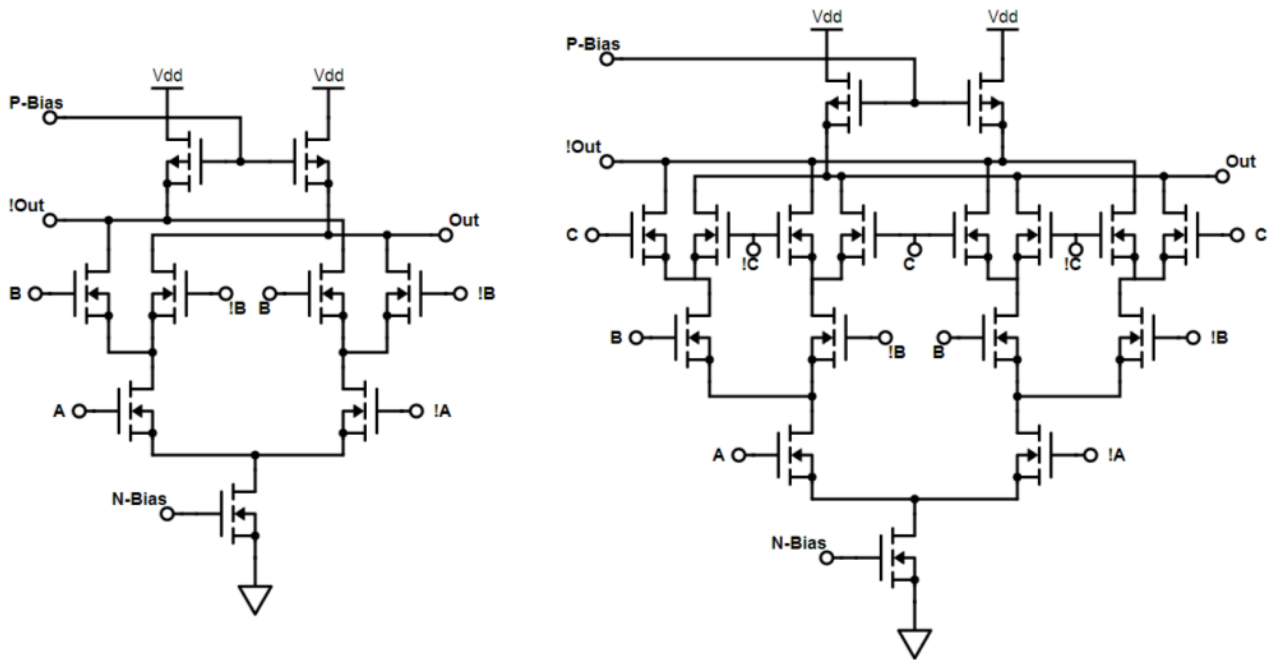


Figure 2.4: MCML 2-Input XOR Gate & 3 Input XOR Gate Schematics

Note that the MCML schematic for an N-input XOR device is more compact than the equivalent CMOS designs and as a result, should perform better and are more suited towards high speed digital applications. However, given the doubling of pull-down components per input added, logical depth should be minimized and XOR gates with many inputs should likely not be implemented both for area concerns and increased device complexity, especially when designing on the IC level. However, with a simpler pattern than the CMOS XOR gate, MCML XOR gates of any number of inputs are quicker to design and test. Note that XOR gates can be simplified (especially CMOS)

through implementation in AND & OR gates. The associative nature of logic gates allow for construction in this manner and enable serializing multiple input gates. However, examination into these types of gates is not within the scope of this thesis.

2.2.4 – MCML MUX

Digital systems also need other components aside from logic gates to perform their functions. The multiplexer (MUX) is a fundamental circuit block in digital applications. The device contains a select signal that chooses one of an assortment of input branches to forward along to a single output branch. This data transmission capability is crucial in digital systems and requires examination within the scope of MCML. For simplicity, a 2:1 MUX is constructed on a schematic level and displayed in Figure 2.5.

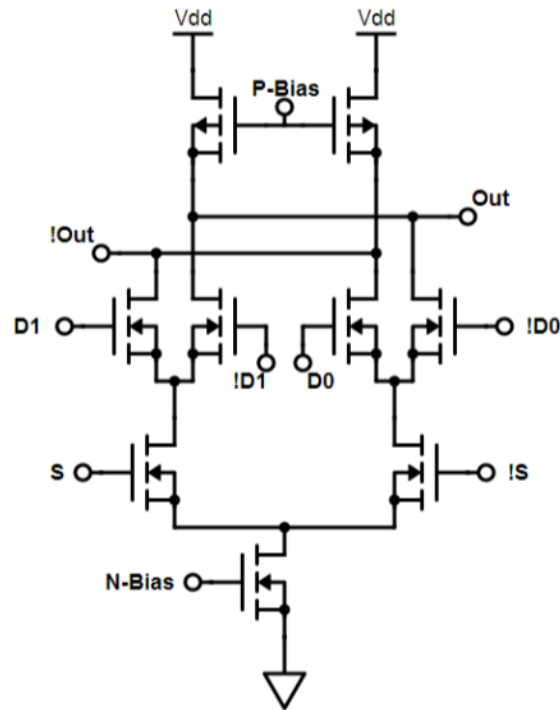


Figure 2.5: MCML 2:1 MUX Schematic

The differential nature of MCML proves advantageous again as a MUX of any dimension would follow the same pattern where one polarity of each input is tied to one output,

while the remaining pins are tied to the other. This configuration also allows for intuitive output pinning regardless of the circuit size.

Note that MCML can also implement a DEMUX function as well, though due to the static power consumption of MCML devices at low frequencies, these functions are typically implemented in some form of CMOS. Due to the advantages of using CMOS devices alongside MCML devices in these situations, CMOS to MCML and MCML to CMOS converters can be designed simply for applications where utilizing multiple configurations proves advantageous for a system. These converters are covered in detail in Section 2.3.

2.2.5 – MCML D-Latch

The D-Latch is a fundamental memory device and serves as the primary storage element for sequential circuits and for processor register memory. The basic structure of a D-Latch in MCML is shown in Figure 2.6. Like the XOR gate, the MCML D-Latch possesses a smaller layout than the equivalent CMOS implementation. Also note that the D-Latch schematic can be tweaked to a D Flip Flop with the pins tuned to a master and slave configuration.

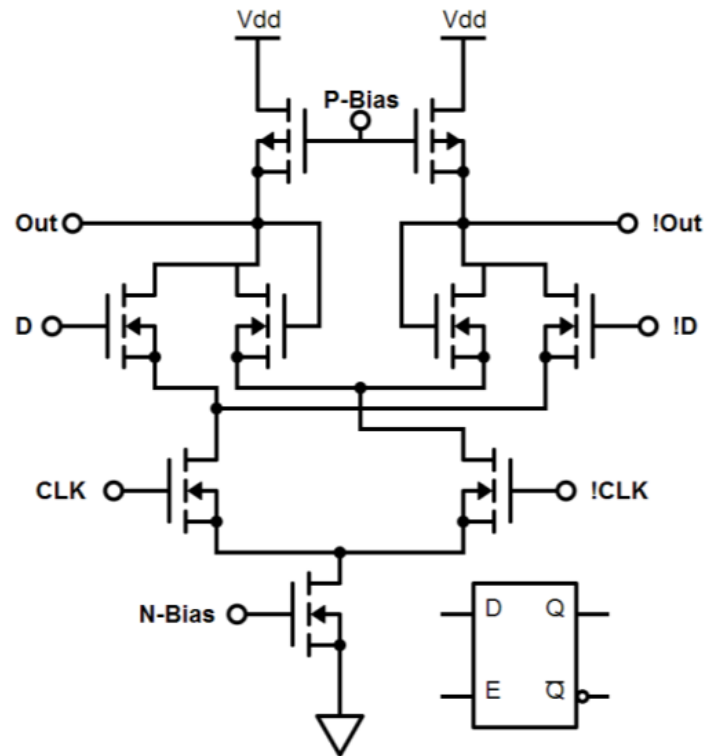


Figure 2.6: MCML D-Latch Schematic

2.2.6– Arbitrary Logic Functions

The block level of MCML gates remains the same, regardless of the logic function being implemented. Each gate will contain two pull-up load devices, a current source, and the logical pull-down network. This means that MCML can realize any logic function based on the configuration of the NMOS pull-down network. The pull-down network structure for MCML is identical to that of emitter coupled logic (ECL). Though ECL gates use BJTs, the logical design process remains the same. However, ECL serves more as a textbook implementation for high-speed digital circuits but is not widely used in modern electronics largely due to high power consumption.

To examine the design process for an arbitrary logical function, let's consider a simple 3-input truth table (Table 2.1) and create an MCML gate for the final result.

Table 2.1: Arbitrary 3-Input Truth Table

A	B	C	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Here, the final equation can be simplified to the following: $OUT = \overline{A}\overline{B}\overline{C} + AB + BC$. For design and implementation, consider the fully laid out binary decision tree shown in Figure 2.7. Note that with the differential configuration of MCML, the un-simplified result proves easiest to implement the full logic function.

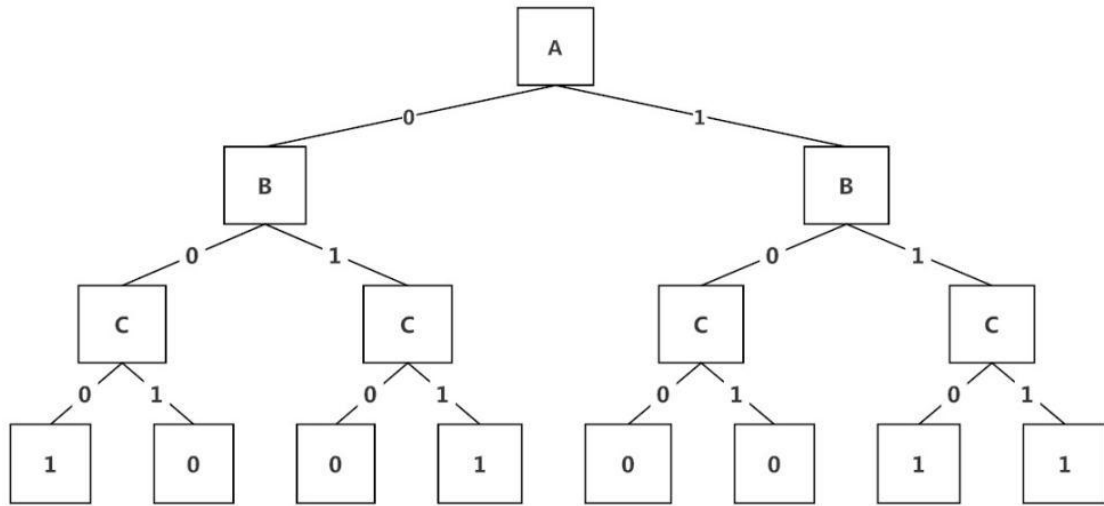


Figure 2.7: Binary Decision Tree for Table 2.1

The tiered structure presented creates a logical pyramid for the pull-down network. In this form, each variable block (A, B, or C) is replaced with an NMOS transistor with the final result, looking identical to the original decision tree. Figure 2.8 depicts the final MCML layout for this example.

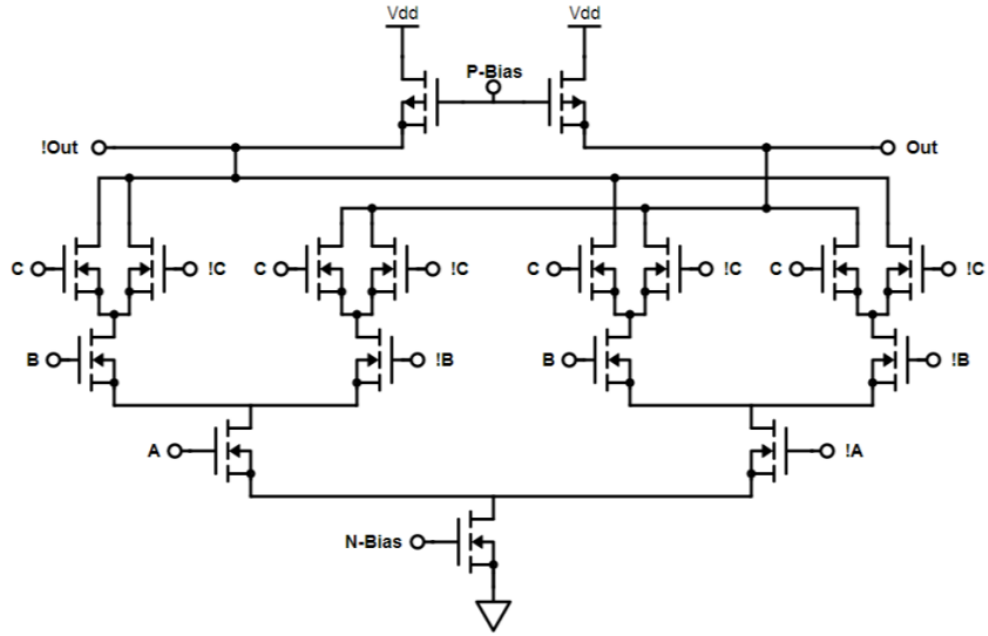


Figure 2.8: MCML Gate for Table 2.1

Note that the final pull-down network structure emulates the structure of the decision tree (inverted in this layout). With the decision tree and final schematic realized, one sees the process for implementing an MCML device from an arbitrary logic function. Note that upon inspection, the circuit can be reduced to the simplified equation by eliminating the transistors that do not provide any logical action to the circuit. Specifically, this includes the two pairs of C and \bar{C} transistors that have no effect on the logical outcome of that circuit path.

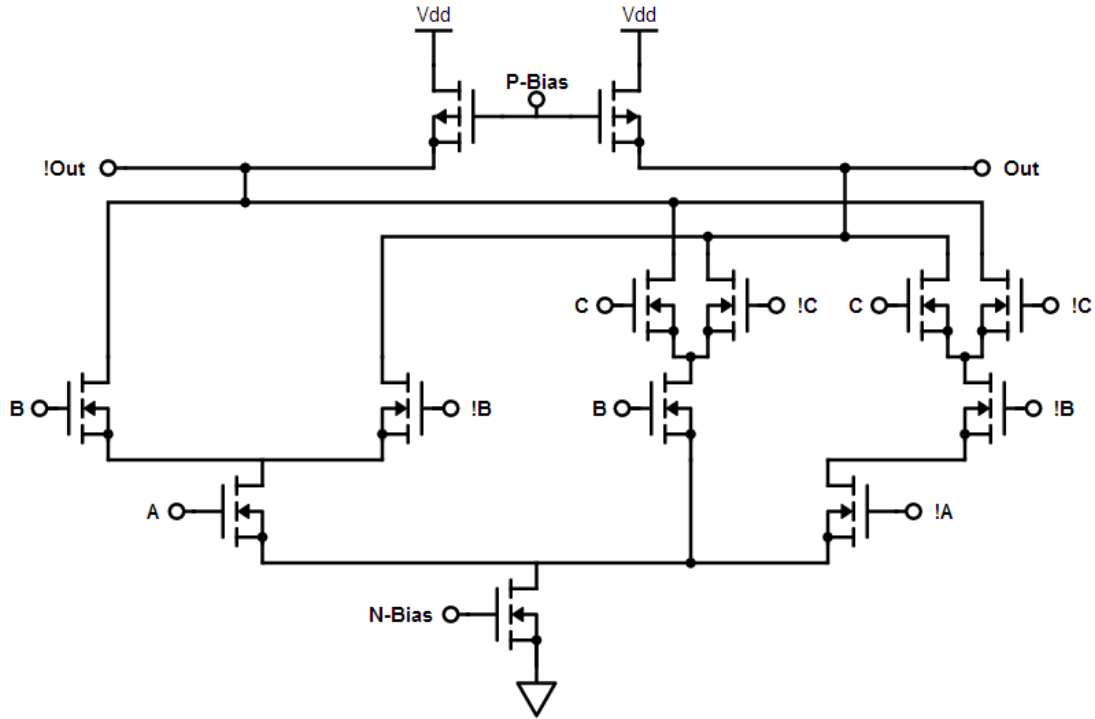


Figure 2.9: Simplified MCML gate for Table 2.1

This final result depicts the maximum simplification for the logic function where four transistors are removed and the logic path on another branch is simplified. Note that both circuits satisfy the logical operation specified by the truth table and subsequent decision tree, however for overall simplicity and minimization of area, an engineer should opt for the reduced solution in order to reduce area and improve speed by reducing overall capacitance of the logical network.

2.3 – MCML to CMOS & CMOS to MCML Conversion

Given the prominent use of CMOS circuitry throughout academia and the industry, one should not expect full transitions to MCML layouts and some applications are better suited to CMOS implantation than MCML. However, the net result becomes possible use of both logic families in a single circuit. As a result, one should examine the conversion circuitry required for interaction between the logical families.

One can implement CMOS to MCML conversion with a CMOS inverter and an MCML inverter. This takes the single CMOS output and creates the differential MCML input. This relies on MCML's ability to operate correctly with larger than required differential inputs [3]. MCML to CMOS conversion is trickier as one needs to take the differential MCML output and create a single signal for the CMOS logic. This is accomplished with a differential to single ended amplifier and a CMOS inverter. Complications arise in this conversion due to the fact that MCML tends to have a much smaller swing than CMOS and may not provide a large enough swing to switch the CMOS inverter. This constitutes a need for a differential pair with a larger swing to provide the CMOS components with the necessary signal. These thoughts should be kept in mind when utilizing both logic families to ensure that the circuit behaves as intended. Figure 2.10 displays both converters with appropriately labeled inputs/outputs.

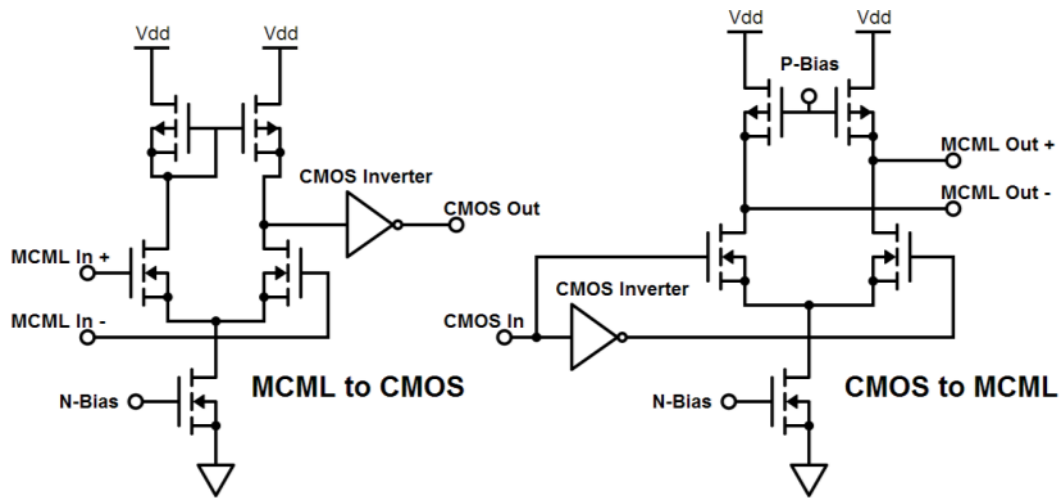


Figure 2.10: MCML to CMOS & CMOS to MCML Converters

The amplifier in the MCML to CMOS converter requires enough gain to achieve an output voltage swing capable of switching the inverter. With an input of 200mV, the amplifier only needs a gain of 10 to satisfy all points of operation [3]. The cells

developed for this library possess a high output and may not provide the swing necessary to create the CMOS transition. If conversion is necessary in this situation, a differential pair with larger voltage swing can replace the CMOS inverter in the MCML to CMOS conversion.

Standard cell libraries typically convert an HDL deck of logic to automatically generate the gates in a circuit. This generation utilizes only one topology (CMOS, MCML, etc.) and would prove challenging to implement mixtures with nothing but logical HDL input. For the purposes of this standard cell library, only MCML devices are generated and used for quiet and mixed signal applications. Conversion circuitry is not necessary for the applications presented in this thesis but the theory should be kept in mind when designing with MCML in a mixed logic family application. Diligence with cell naming in the HDL structural view allows for the standard cell layout tool to select the correct cell even with multiple logic families in use.

CHAPTER 3

STANDARD CELL LIBRARY BASICS

This chapter aims to detail the fundamental theory regarding standard cell libraries and the SKILL language (the scripting used for implementing the library in Cadence ® integrated circuit front to back (icfb) software suite). Understanding both the components of a standard cell library and the means of implementing one greatly enhances the ability for one to understand the MCML standard cell implementation and layout optimization theory that are detailed in later chapters. These theoretical concepts are utilized during the design process for the MCML standard cells presented in this thesis.

3.1 – Standard Cell Library Overview

This section examines the basics of standard cell libraries and discusses the goals for creating one. The section also investigates the conventional design techniques for standard cells and compares that process with a newer approach, with the goal of creating cells in a quick and efficient manner. Given the lack of a standard cell compiler [8], the more rapid approach that will be introduced was not feasible for this project but serves as a potential area for future research and projects.

A standard cell library is a compilation of digital devices that meet and follow certain parameters of electronic operation and physical properties such as height and position of inputs and outputs, drive strengths and topologies. The goal of a standard cell library is to provide a level of design abstraction and rapid design process where a designer can utilize simple gates and blocks. This is accomplished through high level

HDL design of a system where the software tools take the code and synthesize the design on a gate level in silicon. This allows for rapid transition from high level design to gates and implementation. For complex designs, this abstraction is necessary to dramatically reduce the time required to generate a circuit layout, or a more complex block layout. To achieve this level of simplicity, the abstract view for a cell contains the size and I/O pin positions which allow for the software tools to handle routing gates together and allow the designer to not have to deal with the layouts for each gate and transistor. Figure 3.1 shows the paths one can take to design a circuit based on customization level. Standard cells focus on the left portion of the flowchart and provide the highest level of abstraction to the designer due to the use of standardized gates. Depending on more specialized applications, more customization of components and parts is required to achieve desired performance metrics.

For a designer utilizing standard cells, a high level HDL design is written in behavioral fashion. The Cadence tools then synthesize the HDL to create a new Verilog file containing the *structural* or gate configuration of the design. Next, the tools use standard cells for each gate and perform routing based on the structural configuration. With the preliminary design created, Virtuoso is then used for error checking and verification that the synthesized layout follows layout design rules and matches the schematic (behavioral) configuration originally created by the designer. Once all validation is complete, files are generated to send to a fabricator for IC creation.

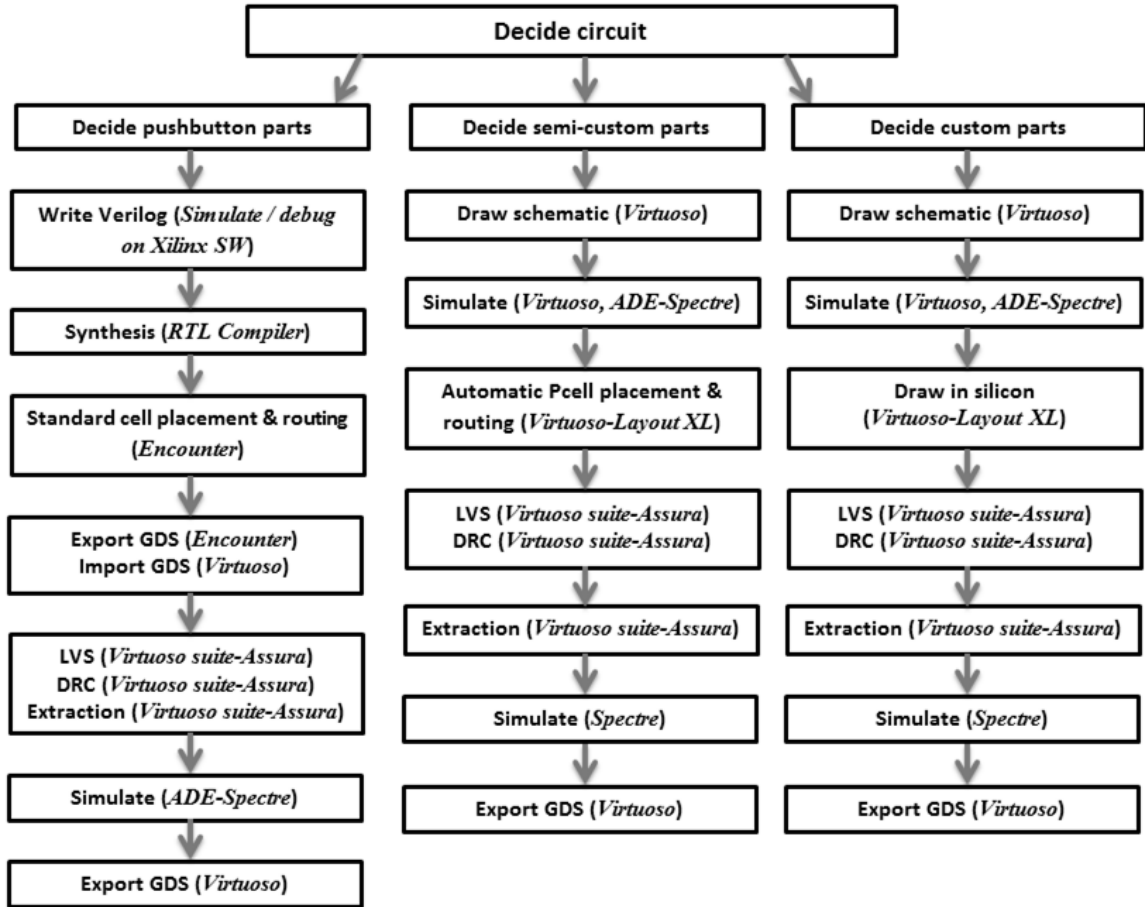


Figure 3.1: Flowchart of Various Design Paths Using Cadence Tools [10]

Unlike a customized grouped layout, where specific layout blocks are manually selected together for copy paste use, a standard cell contains fixed dimensions where each cell has the same height or a multiple of that height. The software tools place standard cells onto a grid layout and then perform routing between each gate. For a chip with many gates, power and ground connections (rails) are routed evenly throughout an entire IC region. This guarantees that power and return signals do not have to travel extremely far depending on where a component is located on a chip. Placing standard cells that have uniform height on a fixed grid ensures that the power rails of the cells line up with the grid the cells are placed on for rapid development. The width of standard cells can vary due the different sizes of blocks gate to gate. As long as the height remains fixed, then the

rail connections remain intact. Standard cells also contain multiple layout sizes that have different current drive strengths. This allows for the Cadence tools to select cells correctly based on the structural Verilog code and ensure that each gate in the circuit can be driven effectively. Grouped layouts or custom symbols created from grouped layouts cannot be used as standard cells unless implemented into the library itself while adhering to the necessary sizing guidelines that allow standard cells to interface with each other. This shows how standard cells provide efficient means of going from high level abstract design to gate level layout in silicon. Standard cells primarily reduce development time and as a result, shorten time to market of products and systems that utilize this design method. However, the tradeoff come in device performance as large standard cells designs may be less optimal than partially or fully customized designs specific to an application.

3.1.1 – Components of a Standard Cell

1) Circuit Layout

Each standard cell includes a silicon level layout for a gate or block. This includes full metal layer detailing on the silicon level for the schematic of that gate. Figure 3.2 presents an unconnected layout of an MCML inverter created in Cadence virtuoso with the Layout XL tool. The unconnected layout should prove similar in view and easy to conceptualize with the schematic in Figure 2.2 in mind, recreated in Figure 3.2 for convenience. Note that the formal standard cells presented in this document are created by hand as all standard cells must be and then used in the automated development process. The following layouts were created with 180nm IBM technology.

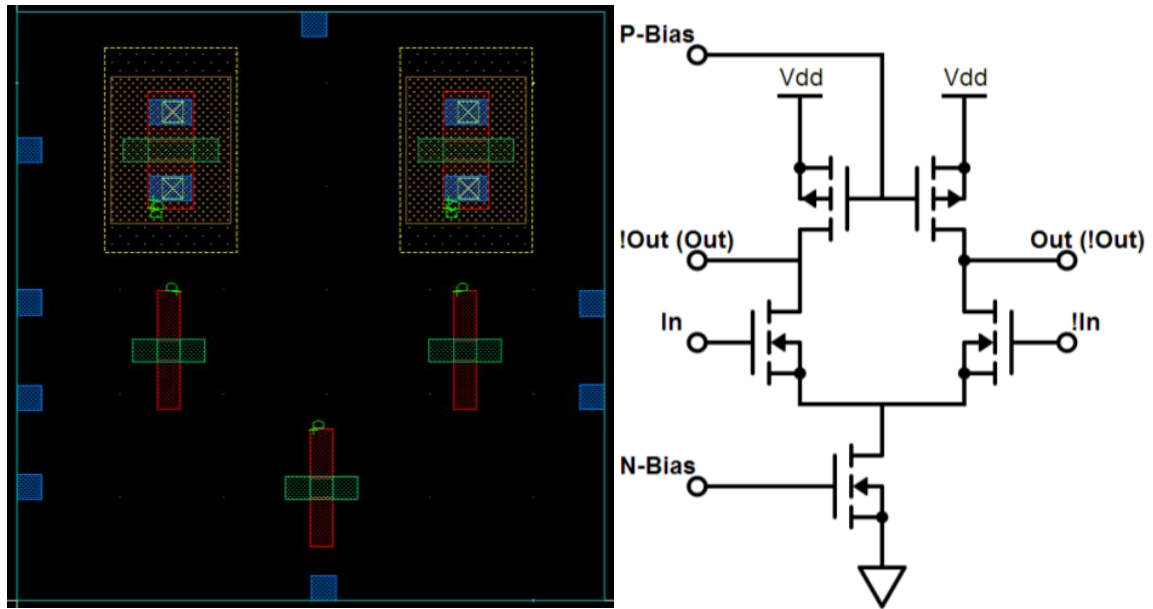


Figure 3.2: Unconnected MCML Inverter Layout & Copy of Figure 2.2 Schematic as Reference

A partially automated and hand-drawn layout for the connected inverter is shown in Figure 3.3. Note that much of the layout, including the placement of vias and most metal layers was performed by hand and all design rule checking (DRC) errors were removed to constitute a “valid” layout. The following represents the first iteration of a simple layout that contains no physical errors. This non-optimal layout took multiple hours to create with the bulk of that time devoted to fixing DRC errors. The DRC errors check to ensure that a chip can be fabricated on the equipment of the fabricator, MOSIS in this case, as well as guaranteeing that the physics of the traces and transistors are valid. Checks made on the traces and transistors include checks such as making sure that wires are not too narrow or too close and that regions of the transistor do not couple.. Having no DRC errors does not ensure that a circuit will function when fabricated, only that the circuit will physically have no errors. Other Cadence tools such as the LVS (layout verses schematic) check the layout to ensure it complies with the schematic devices and

connections. Other checks involve simulations to make sure the layout functionally works as well as performs at the speed the user demands.

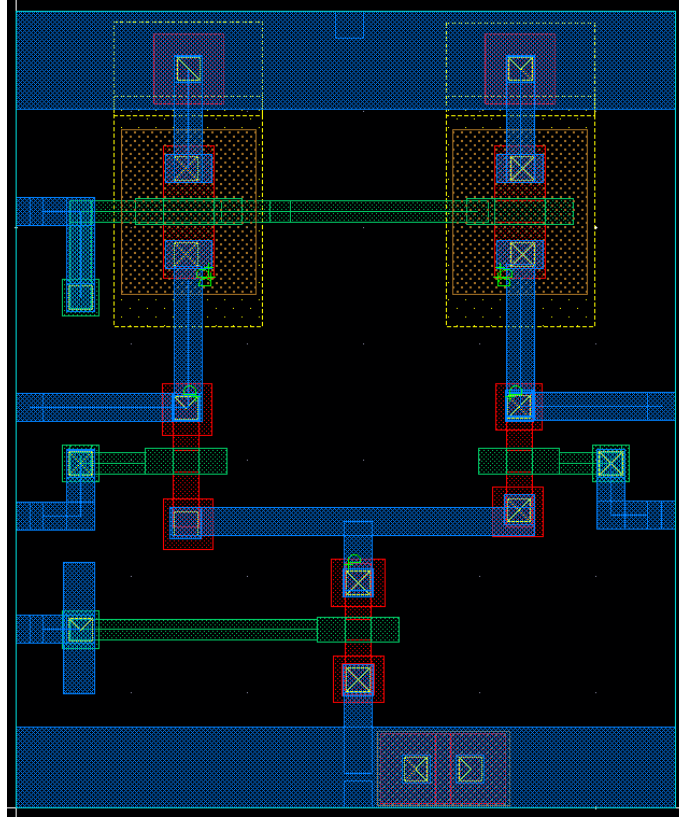


Figure 3.3: Connected Layout for an MCML Inverter with no DRC Errors

Note that there are some inefficient aspects in this layout especially with respect to the metal to gate layer contacts. For example, the left PMOS and tail transistor contain extra fingers of metal layers that do not provide direct connection. This is due to the fact that the DRC rules require a minimum trace area for a particular trace layer. These fingers provide the trace with overall net area to satisfy that rule. Given different layout dimensions or another iteration of routing development, these fingers could be removed. However, with this layout in the same form as the circuit schematic, the detail presented is much easier to understand. In attempts to simplify the design process for this layout, auto route was used to generate as much of the connection as possible to minimize

manual layout. Auto routing does not create an optimal result but attempts to perform a “good enough” job that balances functionality and performance while maintaining rapid development time through automation. The algorithm for auto routing looks for unconnected traces and connection points and attempts to generate paths between each component terminal and device. Due to the sizing of the transistors, not each connection was generated automatically. Since auto routing did not provide a suitable layout for this design, a manual approach was chosen for this layout. This MCML inverter utilizes 7rf technology and was created in attempts to tape out a chip in the winter of 2014. The standard cells developed for this thesis utilize the On Semiconductor C5 (SCMOS) technology. The standard cells developed for the SCMOS library contain a stricter layout that meets the requirements and height dimensions necessary for library implementation. Inverters in both CMOS and MCML topologies were designed with the goal of comparing their differing performance with regards to noise and operating frequency.

2) Software Simulation Models

Once a layout is constructed, one must simulate the device and this requires computer simulation models. These are the types of files that an engineer links to components in a Spectre schematic simulation. Schematic simulation allows proof of functionality but not an accurate check of speed or IR drop affects. Schematics do not include capacitance and resistance of traces. Traces are treated as ideal (no capacitance or resistance and no drop in voltage from end to end. Also, when one end has a new voltage on it, the other end changes instantaneously). Capacitance in the transistors is approximated so some frequency related behavior can be observed even in a schematic simulation. To get an accurate simulation of an actual laid out circuit, extraction needs to be carried out on the

finalized layout of the circuit. Now that wire lengths and distances between devices is known, coupling and resistive drops can be accurately simulated. Extraction adds the parasitic capacitances and resistances to the netlist and that netlist can then be simulated just as the netlist of the schematic was simulated but just with more accuracy. Schematic simulations utilize model files for components but assume that wires are ideal and cannot account for coupling or other intrinsic properties of a device on silicon. Since standard cells are layout blocks, the models for a standard cell include parasitic values and properties based on the physical layout of the device. For an accurate simulation, the model takes wire length/width, metal and layer proximities, and parasitic capacitances into account to allow for software based simulation to explore how a device will perform on the silicon level before fabrication of an IC.

3) Electrical Properties

The physical properties of each component and layer in silicon control the capabilities of a specific cell and determine performance metrics like power dissipation and propagation delay. The Encounter tool performs the extraction that generates realistic simulation files that take these properties into account. These values allow for accurate, realistic software simulation based on the physical layout for a cell and help the standard cell tool select the appropriate size (current drive) for a gate in a given position in the circuit. Note that each cell should be characterized under worst case conditions to provide the most realistic benchmarking on the limits and delays within a circuit. This is done through simulation at the corners. This means that for PMOS and NMOS devices, they are simulated with best case (B) and worst case (W) metrics within the fabrication spec. Each extreme combination of performance for the NMOS and PMOS transistors are performed in the

following manner: BB, BW, WB, WW. This creates four corners of performance that characterize the operating range for the device based on potential fluctuations during chip fabrication. As long as the circuit performs as desired anywhere within the operating range created by the corners, then the fabricated chip should work as the designer intends; regardless of any fluctuations in the fabrication process.

3.1.2 – Traditional Standard Cell Creation Process

The traditional standard cell design process includes creating the listed components for a standard cell exclusively by hand. This means that each file and necessary view is created by the standard cell designer. Starting from a blank slate, standard cells must have three cell views. These include a transistor level schematic, behavior Verilog view, and silicon level layout with no DRC errors. High quality libraries contain timing information with the Verilog files as well. Since standard cells require fixed heights to match with power rails, sizing information can be found in a definitions file for the technology being used. This sizing is usually based around the I/O pad sizing which provides the physical connection capabilities of a chip with the outside world. Next, an extracted view of the cell is made which includes the parasitic properties of the device on silicon. This allows the Spectre tool to perform realistic simulations as to how the device will perform once fabricated. This usually includes corner simulations to create an operating range for the device. Then an abstract view derived from the layout is created within Cadence that tells the tools where to place and route the I/O port locations and the dimensions of the standard cell. This file also includes any “keep out” locations that constitute regions where no routing should occur. Similarly to the abstract view, a library exchange format (LRF) file is used by Encounter within Cadence and also contains technology information

along with the standard design information contained in the abstract view. A Verilog interface file provides the inputs and outputs of each cell and works together with the structural Verilog file that is created by the RTL compiler. Finally, the Liberty (.lib) format file is the synopsis file used by Cadence and is utilized in the RTL compiler. This file describes the I/O interface, logical function, latency of the device, and capacitances on the pins, specifically C_{in} and C_{out} . This file can be generated during the Spectre simulation of the cell. These files provide the complete design for a standard cell and all the necessary data and characteristics of the device for simulation and fabrication on both the transistor level and silicon level.

Standard cells must also contain a specific geometry to allow for grid placing of the devices that line up with power and ground rails and allow the Cadence tools to route more easily between gates. The geometric requirements for a standard cell are as follows:

- Each cell must be the same height (or a multiple of)
 - Allows for uniform grid placement on power and ground rails.
- PMOS on the top of the cell and NMOS on the bottom of the cell (or vice versa) for every standard cell.
 - Ensures that cells snap onto the power and ground rails correctly.
- Ensure that the PMOS n-well goes to the edges so that it is continuous across a standard cell row.
 - Ensures equivalent power connections gate to gate between V_{dd} and the PMOS transistors.
- V_{dd} and GND rails must be the same width on each cell.
 - Makes sure each cell snaps onto the same fixed grid and that each power rail is uniform across the entire IC and each cell.
- V_{dd} and GND rails must extend to the edge of each cell.
 - Enables a continuous connection across a row on the grid between cells/gates.

The standard cells designed for this thesis utilize each of these mentioned geometric rules during layout design to fulfill the requirements for valid standard cell layouts.

Since a common library can contain upwards of 200 elements, the creation of an entire library could take multiple years. Though libraries are acceptable only containing Inverters, NAND gates, and a flip-flop, all with various sizes; that serves as a starting point where further cells can be added. Inverters and NAND gates are capable of performing any logical function though for complicated systems, having other cells in the library can dramatically reduce the number of gates necessary to implement some logic. Given the rapid evolution of technology, by the time a standard cell library is completed via these means, the technology utilized may already be obsolete [8]. The manual approach was selected for this thesis with inverters and NAND gates as the focus of the design. Though standard cell libraries contain a flip-flop as well for minimum requirements, one can be constructed using NAND gates so effective functionality remains with the cells developed. Despite the manual design for each cell, the automated process is worth mentioning as it serves as a more reliable and rapid development process especially for large standard cell libraries.

3.1.3 – Automated Standard Cell Creation Process

A. Martinez, S. Dholakia, and S. Bush created a fully automated compilation technique for creating a standard cell library in fractions of the time compared to traditional hand done processes [8]. The automated approach helps illuminate the range (manual to auto) of possible paths for standard cell development. The requirements for a standard cell: automated or otherwise, remain the same. Therefore, full automation requires that the same components be included in a cell including full layout, simulation modeling and results, and optionally, a full documentation page for the cell. The compiler approach presented in [8] demonstrates a process where five individuals were able to create 280

standard cells over the course of nine months. This would be impossible by hand and only a small fraction of cells would be created with more designers working on the library over an equivalent amount of time.

The compiler approach presented includes a step by step process detailing the formation of a standard cell. The first step is generation which involves layout creation and includes creating larger designs by working with common building blocks for a digital device like: latches, flip-flops, and counters [8]. The compiler excels at creating efficient layouts for smaller gates and devices that do not include IO pads. The lack of common building blocks for such a device makes the compiler ineffective. However, for any logical component or gate, the compiler rapidly generates the layout for a particular gate size and provides the necessary documentation and files required for library use of that cell.

Next, the compiler performs full behavioral simulation for each standard cell and the complete timing and performance metrics are generated automatically [8]. Cadence tools can achieve similar operation through script usage that performs the discrete steps necessary to simulate and output performance data. Note that place and route tools only require the cell size and I/O placement metrics so that gates can be routed together. However, ensuring that the layout performs as expected within the block involves simulation and thus, the automated process performs the necessary characterization to ensure that the cells behave as desired. The models used for software simulation are generated independently from the layout models but are compared together to ensure the logical integrity and timing of the device. As mentioned previously, the simulations are performed under worst case conditions and Martinez *et al.* linearized the delay outputs

and placed them into a high level parameter that could be used by a model compiler. The worst case power dissipation as a function of frequency was computed for each standard cell with a software program [8].

With this data, [8] then moves towards model generation the behavior for each cell is detailed in a primitive model (PMD). The PMDs show example building blocks one can use as standard cells where each primitive model can be a component in a larger system. The primitive models each contain a primitive type, connectors, and timing parameters. The list of primitives used includes [8]:

- Boolean Gates (AND, OR, NAND, NOR, XOR, XNOR)
- Inverters/Buffers (including transceivers)
- Flip-Flops (D, T, sc, MUX, JK, w/scan, tri-state, etc.)
- Boolean Expressions (multiple inputs, outputs, and equations)
- Muxes
- Decoders
- I/O Pads

With functional models, the analysis moves to a characterization step. This step proves to be the most time consuming during the generation of a standard cell library using these automated techniques. The Cadence Spectre simulation engine generates the necessary files required for characterization. The automated models in [8] are extracted and implemented as SPECTRE simulation files where characterization is expedited by the executive decision to use a uniform output load of 1pF [8]. Since many devices are comprised of common blocks, the blocks can be characterized individually and their respective delays summed to generate the result for an entire cell. Note that complicated cells utilize common pieces, but do not combine to create a large standard cell that is comprised of smaller standard cells. Static timing analysis and the use of corners, as

detailed earlier in the chapter, allow for delay estimation for a digital network of standard cells and investigation of operational corners allows for a range of operation to be established based on possible variation during the IC fabrication process.

Finally with all necessary components of a standard cell, the automated compiler creates documentation for the device. Having a datasheet for a standard cell is more uncommon than not due to the added time in manually creating documentation and the fact that log files generated throughout a cell's development can provide the same information. However, the automated compiler presents the information in an easily readable fashion without forcing a designer to track down necessary files. The Cadence tools do not include standard cell documentation or a means of automatically generating such a file. Though datasheets are useful for standard cells, one would need to create them manually when utilizing a non-automated process. Each data sheet created for [8]'s standard cells contains combinations of text, delay equations, and supporting graphics. These components are blended together automatically and include a functional description of the cell, schematic icons, layout outlines and signal locations, a logical truth table, timing waveforms, propagation delay equations, and propagation delay tables [8]. Figure 3.4 shows an example of a data sheet created by [8] for their standard cells using this approach.

DFCTNB

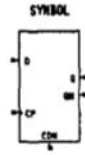
BUFFERED D FLIP-FLOP WITH CLEAR

DESCRIPTION

DFCTNB is an HCMOS positive-edge-triggered D flip-flop with an active LOW clear, CDN. Clock input CP and outputs, Q and QN, are buffered. Data present at D during the positive edge of CP is transferred to the Q and QN outputs.

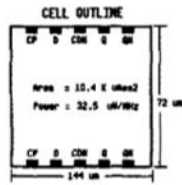
Gate Equivalents: 8

Cell Types: Internal



FUNCTION TABLE

INPUTS			OUTPUTS	
CDN	CP	D	Q	QN
L	X	X	L	H
H	↓	L	L	H
H	↓	H	H	L
H	L	X	Q	QN
H	H	X	Q	QN



PIN DESCRIPTION

Name	Type	Capacitance (pF)	Description
CP	Input	0.07	Clock Input
D	Input	0.04	Data Input
CDN	Input	0.07	CLEAR Input (active LOW)
Q	Output	0.07	Data Output
QN	Output	0.08	Complementary Data Output

DFCTNB

BUFFERED D FLIP-FLOP WITH CLEAR

AC CHARACTERISTICS

CONDITIONS: VDD = 4.5 V, worst-case process
TEMP = 70 deg C ambient, 85 deg C junction
Cld = load capacitance in pF

PERFORMANCE EQUATIONS

tQ, CP→Q	$6.4 + 0.2 + 3.3 \cdot Cld$
tQN, CP→QN	$4.5 + 0.3 + 3.3 \cdot Cld$
tCQ, CDN→Q	$1.8 + 0.2 + 3.3 \cdot Cld$
tCQN, CDN→QN	$5.1 + 0.3 + 3.3 \cdot Cld$
fMAX	58 MHz
tWH, CP	6.7 ns
tS, D→CP	4.8 ns
tH, CP→D	3.0 ns
tREL, CDN→CP	4.8 ns
tHC, CP→CDN	0.0 ns

PROPAGATION DELAYS (ns) FOR SAMPLE LOADS

	Cld = 0.1	Cld = 0.2	Cld = 0.5	Cld = 1.0
tQ	7.0	7.3	8.3	10.0
tQN	5.1	5.5	6.5	8.1
tCQ	2.4	2.7	3.7	5.4
tCQN	5.7	6.1	7.1	8.7

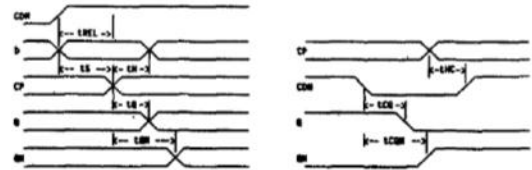


Figure 3.4: Automatically created data sheet for D flip flop using compilation [8]

The data sheet presents itself in an organized manner and details the important features of the standard cell. Since human text input is kept at an absolute minimum, the likelihood for entry errors diminishes as well. The automation also creates a uniform formatting that should present itself the same way regardless of the standard cell utilized. This means that regardless of which cell is selected from the library, the corresponding data between cells will be detailed in the same document location and should make properties easy to find when working within a single library [8]. Such documentation is not within the scope of this thesis but is important to keep in mind, especially when automated techniques can generate these forms during standard cell compilation.

Creating a single library of standard cells that operate with one another in any configuration is an essential goal for the gates designed in this thesis. The cells are tuned to accept the same level and swing of input signals and output an output signal of fixed swing that matches with different cell types and the different cell sizes. A complete standard library includes a verified schematic, layout, and simulation data. Documentation is an added feature that can allow performance metrics of a standard cell to be more readily available but is not necessary for completion of a library. The Cadence tools require size and I/O placement for placing and routing a large device with many cells. Each standard cell should be individually simulated and verified at both the schematic and layout level to ensure that the gates will function as intended in implemented designs.

3.1.4 – Created MCML Standard Cell Library for this Thesis

For the library created in this thesis, elements of the traditional design are implemented for creation of Inverter and NAND gate MCML standard cells. With no full automation capabilities for generating standard cells, manual layouts that fit to a fixed grid were developed using SCMOS technology. Since an existing SCMOS standard cell library was not available, standardized dimensions from a standard cell tutorial [9] are used to size the grid layout needed for the MCML gates. Inverters and NAND gates with 1x, 2x, and 4x current drive capability were developed and simulated for implementation into the standard cell library already present on the Cadence system with the SCMOS process. The design work for the MCML standard cells is entirely manual to ensure uniformity between the cells and guarantee that each cell fits on the same grid height.

CHAPTER 4

MCML CELL OPTIMIZATION AND TRANSISTOR SIZING

4.1 – Challenges in the Optimization of Standard Cell Libraries

When one implements a system on the silicon level, perfection is unattainable. There is intrinsic variability that occurs across a chip causing small changes in operation possibly away from the desired result. Coping with this property poses a significant challenge in the semiconductor industry. Here, I examine current literature regarding variability tolerance for a CMOS standard cell library where the goal is to create high-speed and low-power circuits by optimizing transistor dimensions. The atomic properties that occur on the silicon / IC level occur regardless of process (CMOS or MCML) and an engineer or designer should keep them in mind.

Device variability occurs in two domains: namely the spatial domain and the temporal domain. Spatial variability occurs when the device shape differs from the intended design, including uneven doping, non-uniformity in layer thickness, and polycrystalline surfaces. This variability is possible at all levels: “over the lifetime of the fabrication system, across a wafer of chips, between cells within a very large scale integration (VLSI) chip and between individual devices within that cell [7].” Temporal variability includes the effects of electromigration where smaller grains of a crystalline structure begin to move and separate, gate-oxide breakdown, and the distribution of negative-bias temperature instability. Temporal variabilities can be estimated, resulting in an approximate lifetime for a particular device. Common intrinsic properties of a device

will vary during fabrication due to atomic level variations in devices that are identical in schematic, layout, construction, and environment. These properties include [7]:

- **Random Dopant Fluctuations:** The unavoidable variations caused by the number and position of dopant atoms within the silicon lattice. Even with precisely controlled implant and annealing, these variations will cause fluctuation in device threshold voltage, subthreshold slope, and drive current.
- **Line Edge Roughness (LER):** The horizontal plane deviation of a fabricated feature boundary from its ideal form. This is caused by mask imperfections, photo-resist and etching process, and the stochastic nature of discrete molecules throughout the photo-resist layer.
- **Surface Roughness:** The vertical plane deviation of the actual surface compared to its ideal form. Surface layer (oxide layer) dimension changes can cause parasitic capacitance variation and in turn modification of device threshold voltage.
- **Poly-Silicon Grain Boundary Variability:** The variation due to the random arrangement of grains within the gate material. Implanted ions can penetrate through poly-silicon and insulator layers and into the devices. This creates localized stochastic variation.

To model these effects for a particular transistor, one must generate a large number of current-voltage (I-V) curves for the device and use the resulting data to calibrate the parameters of a model library. Walker *et al.* investigate the optimization of CMOS devices using a genetic algorithm [7]. To avoid gratuitously long computational times, the algorithm optimizes within percent tolerances of the parameters. Utilizing an intelligent algorithm and selected fitness objectives, [7] was able to achieve better delay and power results compared to standard design techniques for a Buffer, NAND, OR, and XOR gate from a commercial CMOS standard cell library. They also noted that larger cells benefit more from the optimization techniques. Their research helps pave the way into possible future work where one can create variability tolerant standard cells, whether using CMOS or MCML topologies.

4.2 – General Optimization Techniques

Depending on the application, circuits can be optimized around any number of parameters, including combinations of size, speed, power, noise, time-to-market, and robustness. For general optimization techniques, the approaches are detailed in a broad scope then compared to how one could use such an optimization technique for MCML devices. The goal here is to illuminate methods for circuit optimization and provide some tools for improving designs whether working on MCML, CMOS, or other logic family. Each optimization method's influence on the designs of the MCML gates of this thesis is explored as well. Further research and projects could be performed strictly in the realm of optimization while focusing on one or a combination of techniques.

4.2.1 – Area

Area optimization involves minimizing the area or footprint that a circuit takes up or occupies on a chip. The idea being, that a smaller chip is cheaper to produce, package, and ship to a consumer than a larger chip that performs the same function which increases overall yield. Minimizing chip area requires the designer to keep principles in mind to allow for correct circuit operation on a smaller package or can simplify a logical network by reducing the number of components before moving to creating a layout. MCML gates occupy a larger area than equivalent CMOS gates because of the increased number of transistors. To shrink the area, higher aspect ratios can be used but high and low aspect ratios provide routing challenges and require transistors to be spaced apart. This metric of spacing is known as 'congestion.' A higher congestion percentage means that the devices in a circuit are more densely placed in layout. An aspect ratio at either extreme; high or low, requires lower congestion to allow for successful routing.

However, this in turn tends to increase the overall footprint of the circuit. Balancing these two properties are crucial to optimizing or minimizing the area of a circuit on a chip.

Before transitioning from schematic to layout, or even before schematic creation, large logic networks should be examined to see if simplification is possible. The Cadence tools provide automated gate reduction and optimization in the synthesis step where HDL code is converted into standard cell gates. For manual optimization however, Boolean algebra principles allow for logical reduction of a network while keeping an equivalent logical function. Having fewer gates to create in layout can naturally reduce the footprint of a circuit without any complicated layout modifications. However, with a lower number of gates, especially buffers, the width of the gates must be larger if they need to drive a larger load. The RTL compiler performs this simplification when using standard cells to create a design from logical code. For simpler designs and understanding, manual component reduction can be quicker than performing the entire standard cell procedure. Figure 4.1 shows a simple simplification that can occur where the total number of transistors in a circuit is reduced manually through observation and application of reduction rules where paths of both logical polarities do not change the output of the circuit. For those situations, the transistors that have no effect on logical output can be removed.

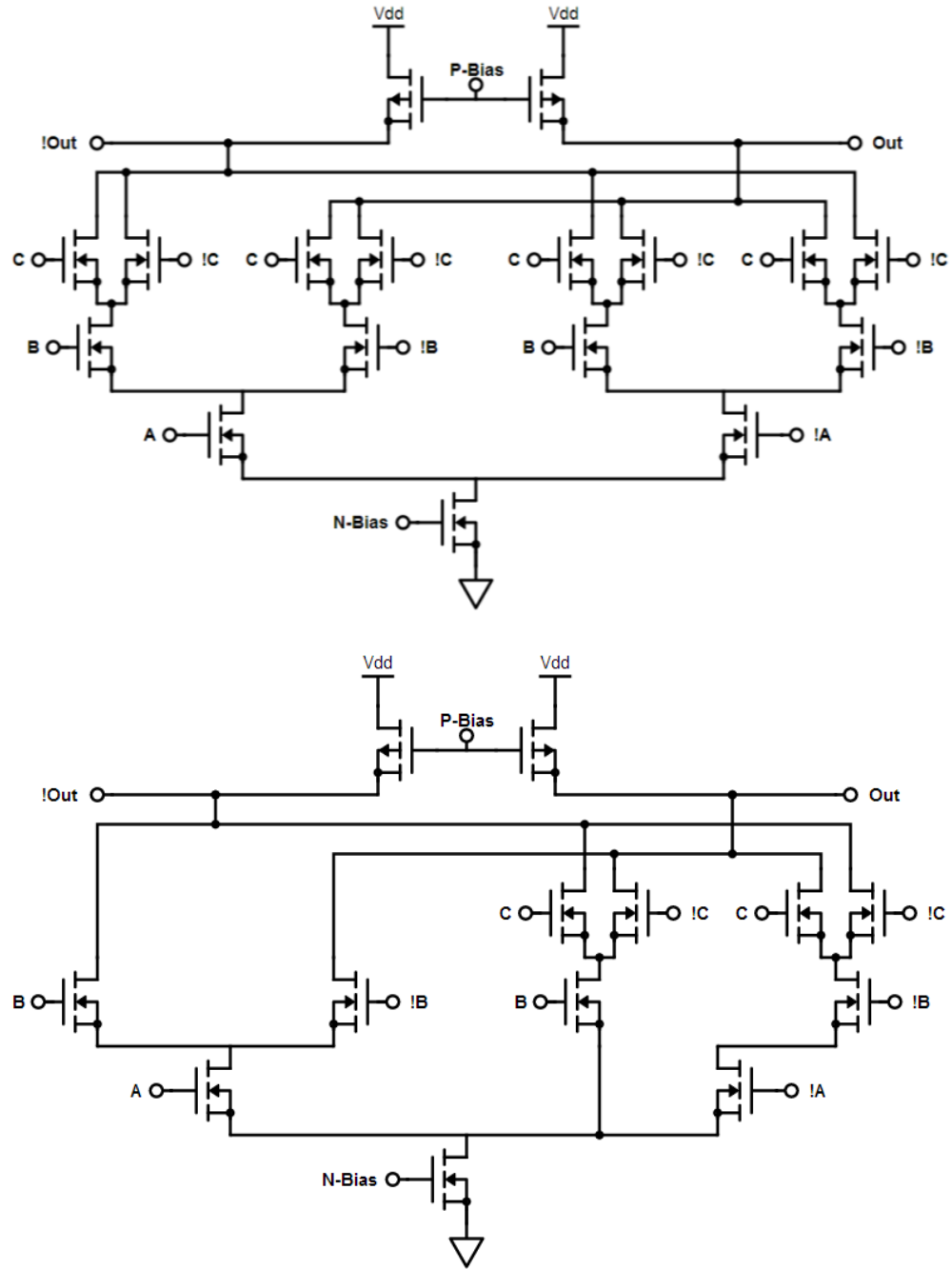


Figure 4.1: Reducing Component Count through Elimination of Logically Redundant Transistors that do not Affect the Final Output of the Circuit

This creates substantially less components that are required to generate a functional layout. Even before techniques are employed to reduce the overall footprint of a layout,

circuits should be checked for possible simplifications as that could drastically reduce the effort required to optimize and allow for more ideal results. However, note that reducing the total number of gates too much could result in an inability to drive a large load. For standard cell usage, the RTL compiler will ensure that all gates in a circuit can be driven. If a logical network must drive a large capacitive load, a longer chain of gates could prove advantageous in preserving correct circuit operation. The RTL compiler will automatically add gates onto a path if necessary to achieve necessary drive strength. For standard cell development, each gate can be optimized manually to minimize the width of a cell (fixed height). The standard cell compiler handles optimization in routing between individual cells but each cell template created for this thesis is manually laid out and tuned to allow for compatibility with other cells of any drive strength.

4.2.2 – Power

Power optimization involves taking a chip and minimizing the amount of power the device dissipates, while still ensuring correct operation and performance. Optimizing the power consumption of a design does not dramatically affect the layout of a circuit in general. Reducing the power a circuit dissipates involves reducing the voltages and/or currents in the network while preserving circuit function. Modifying transistor dimensions and changing transistor bias points are two possible means of reducing net power consumption. Changing the bias point of a transistor affects the overall current flow through the device. Since the MCML standard cells developed for this thesis are biased with a fixed current source at the tail of each device. 1x, 2x, and 4x current strength devices were developed and as a result of each increased current drive, the power through each device increases by that same factor. Since the bias points for these

transistors in this thesis are designed for specific current strengths and operating voltage swings, power optimization was not heavily influencing the design choices here but the principles of power tuning should still be discussed. This allows for more context surrounding the operational theory of the standard cells and illuminate avenues for further development or cell optimization.

Resistance in a FET can be modified by either increasing transistor width or decreasing transistor length. Increasing the width of a transistor allows current to more easily flow through the device and in increasing the cross sectional area of the transistor, the resistance decreases following this equation for resistance in silicon: $R = \rho \frac{l}{A} = \rho \frac{l}{W * T}$. Increasing the width or thickness of a transistor has the same effect of allowing more space for current to flow and accordingly, less resistance. Through the same equation, decreasing the length of a transistor achieves the same effect in reducing the resistance of the device. If current must travel a farther distance to traverse through a device (increased length), then the voltage drop across the device is higher due to the net resistance increase. Since MCML cells have constant overall current, the power dissipation remains constant as it is dependent on V_{dd} and I_{bias} . Tweaking the dimensions of a transistor has limits however. Changing the dimensions too drastically can result in incorrect transistor operation or different biasing properties. Transistor regions of operation can be affected and with MCML this can change the voltage swing of a device or even overall device current if the tail transistor is biased incorrectly.

Transistor current bias points affect how much power the device dissipates. Changing the gate voltages on a FET can influence the current that travels through the device and in turn, the power dissipated. Bias point changes can dramatically influence

the total power used by a system since bias points directly affect the power and delay characteristics of the device. For digital electronics and MCML, the delay in a logical network can be of paramount importance depending on the system. To ensure correct system operation, the dimensions of the components in the device should be examined and tweaked when attempting such optimization. Power reduction is an incredibly broad and deep area of study where techniques are constantly being investigated and tested for reducing the net power of a circuit. At the same time, transistor density increases which drives power and heat up and thermal dissipation also becomes a concern. Due to the nature of the topic, additional projects or theses could be performed on power optimization alone. The goals of the MCML devices designed in this thesis include low noise and constant power dissipation. Therefore, power minimization of individual cells is not explored, though the standard cell compiler can provide optimizations on this front for large systems with many gates.

4.3 – MCML Sizing in This Project

The goal of the MCML gates developed for this project is implementation into a standard cell library that performs with low noise and constant power dissipation. This requires that cells snap onto a grid of fixed width and length and that each transistor meets specified sizing factors. As a result, the optimizations discussed in this chapter are not directly applied to create an optimal standalone result. The principles and theory discussed through the optimization sections are kept in mind during the development of standard cell layouts but full effort towards optimizing any category would prove infeasible in keeping with the requirements of the standard cell library. Therefore, the cells developed during this thesis are meant to be logically functional and able to with the

grid sizes dictated by the standard cell rules. Since noiseless operation is the primary focus of the MCML standard cells, specific optimization in power or area are not applied however, the area of each cell is specifically designed to provide easier grid implementation of cells and ensure maximum compatibility between each gate and gate size. The MCML standard cells are implemented into the same library as the SCMOS standard cells on the Cadence side and conform to those sizing requirements such that they can be implemented and tested on an equivalent grid without substantial redesign.

CHAPTER 5

MCML IMPLEMENTATION OF A STANDARD CELL LIBRARY

5.1 – Overview

This chapter details the development of MCML cells for the standard cell library. Since any logical operation can be performed with inverters and NAND gates which serve as a minimum requirement for developing cells that provide a full range of functionality. Standard cells typically include different variants for current capabilities and following that model, devices with 1x, 2x, and 4x drive strengths are created for both the inverters and NAND gates. These strengths are a metric of how much capacitance a gate can drive at a particular frequency. Since capacitance and subsequent current drive is proportional to transistor width, these drive strength variations in capacitance are created by doubling and quadrupling the widths and current draws of each gate. The standard cells for this thesis were laid out by hand to ensure conformity to height and width constraints. The constraint in this case involves creating layouts that will fit into an existing SCMOS standard cell library where the created MCML cells match the dimensions of existing cells. This includes ensuring that transistor dimensions and cell dimensions could snap onto an identical “grid” as with the SCMOS standard cells. This mainly involves ensuring that active transistor regions are matched and that the rails of each cell are the same size and line up precisely so that cells can be tiled regardless of which cell is being used and allowing the MCML cells to fit seamlessly into the library.

First, schematics for each cell had to be developed and simulated to ensure correct logical operation and current compliance with each desired step in strength. For the

SCMOS C5 technology, the minimum transistor length in this project is 600nm, minimum width is 1.5 μ m, and the increment for length or width sizing is also 300nm. The 1x current models for both devices are generated with as minimally sized devices as possible that achieve desired low noise results and high frequency performance and then scaled up for larger current drive devices.

5.2 – MCML Inverter Schematic

The MCML inverter follows the schematic shown in Figure 5.1 (recreated from Figure 2.2 for convenience). Note that depending on the arbitrary output pin configuration, the MCML inverter can also serve as a buffer.

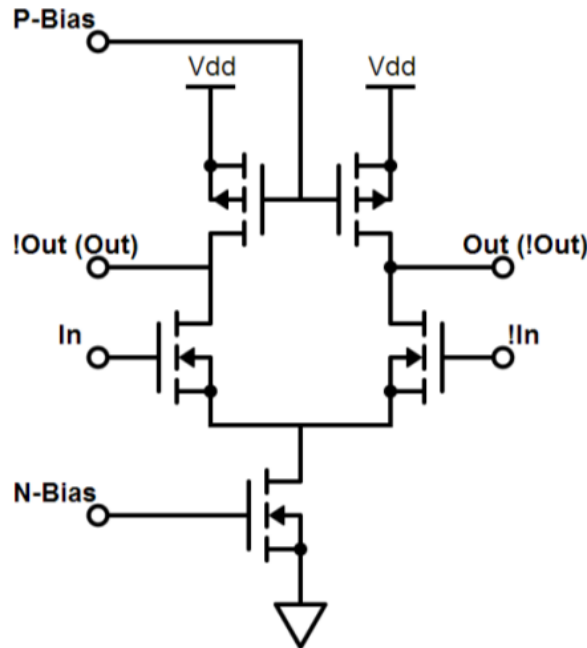


Figure 5.1: Recreated MCML Inverter/Buffer Schematic

The standard “1x” current variation of the inverter utilizes minimally sized devices and the symmetrical nature of each path of operation for both inverter or buffer allow for the device to perform equally well in both modes of operation and allow for issues fixed in

one situation to positively influence the same error on the other side. For correct operation, the biasing voltage for the load resistances was kept at 0V to ensure linear operation of the load transistors.

5.2.1 – 1x Inverter Schematic

The first iteration of the inverter utilizes the smallest transistor widths out of the inverter iterations. The current for MCML typically ranges from 10 μ A to 100 μ A [3]. To fit this range and operate within the bounds of AMI06/C5N Technology, the base line 1x inverter current is set to 20 μ A. Following previously discussed MCML setup, the inverter is assembled as shown in Figure 5.1. The nmos4 transistor symbols and the pmos4 transistor symbols from the NCSU_Analog_Parts library are used for the creation of the schematic. The V_{dd} and GND rails are also from the NCSU_Analog_Parts library such that every component falls within the same fabrication process. Note that there are varieties of other supply and ground rails in other libraries that appear similar in layer composition but will not operate correctly unless every component is from the same technology library. For schematic purposes of any NCSU technology, the transistors and components in NCSU_Analog_Parts will correctly serve the desired need. Model files for simulation and layout generation that utilize a specific technology, like SCMOSC5, are defined and linked later during simulation and layout. Figure 5.2 shows the baseline MCML inverter with variables indicated for transistor sizing. These variables are then set during simulation to allow for maximum design flexibility and editing efficiency. A current mirror is used as control circuitry to set the current of the tail current source that acts as the biasing current for the MCML gate. This circuitry works by forcing current through the N4 transistor and to compensate, the device acquires a V_{GS} to match that

current draw. This voltage is tied to the tail transistor of the MCML gate and since both devices are the same size, the same current ($20\mu\text{A}$ set point) is drawn through the MCML gate.

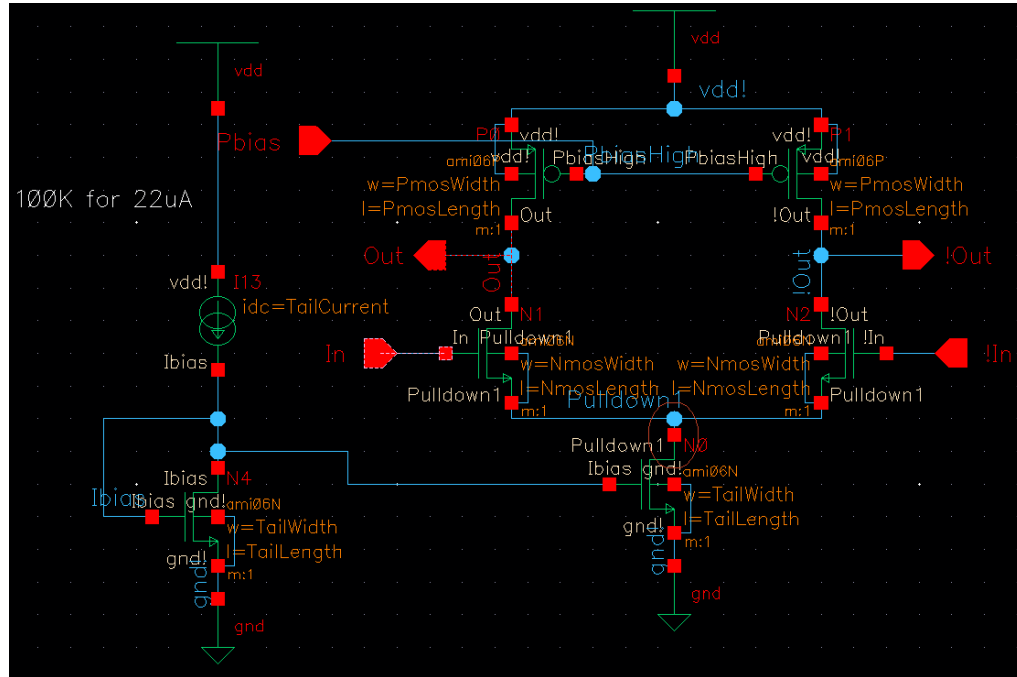


Figure 5.2: MCML Inverter Schematic with Transistor Sizing Variables

For more robust and readable simulations and layout error checking, note that each node is labeled even if already handled by a pin. This guarantees that nodes are not automatically assigned names that become difficult to track for larger circuits with many connections. This also ensures that the schematics and layouts are more readable and understandable when moving into other design modules within Cadence. The pin configuration shown in Figure 5.2 shows inverter operation. As before, swapping the output pins would allow readability as a buffer, but this is an aesthetic change; the circuit performs both functions in this state depending on how it is examined and probed.

Simulating the schematic takes place in Cadence through the ADE L module. The Analog Design Environment (ADE) allows for selection of model files, setting of test

signals, and selection of desired outputs to be examined, plotted, and/or saved. More details on setting up the simulation settings and screenshots along those steps can be found in Appendix A. The simulated inverter results are shown in Figure 5.3. The input signal consists of a sine wave that oscillates between 1.8V and 2.6V at a 2kHz frequency and is represented by the pink trace with the device output shown as the red trace. The input values were examined and chosen to achieve consistent logical operation and achieve circuit stability where each period, whether first or in steady state, operates the same way. These values also perform to these standards for all the MCML devices and sizes developed over the course of the thesis.

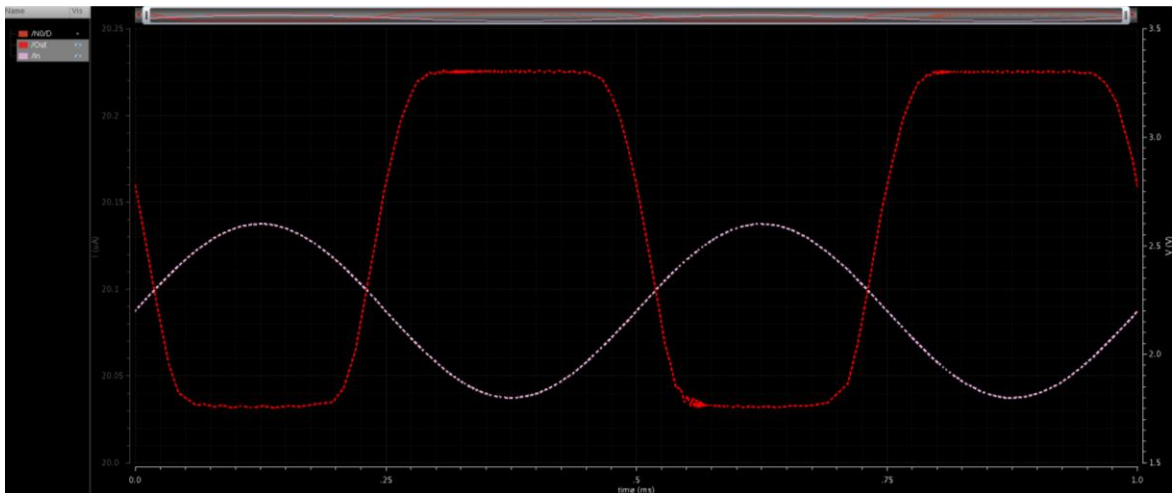


Figure 5.3: 1x Inverter Simulated Results Showing Correct Logical Operation

Here the inverter operation shows a voltage swing of about 1.5V which reaches the input pulse value of 3.3V. The logical low in this current operation are shown as 1.8V, the same value as the input minimum. Though 1.5V is a larger swing for MCML, the primary goal of these devices is low noise operation and the set points and ranges used are designed with noiseless operation in mind. Including the output current for the 1x inverter shows a final average current value of 20.13μA. For an MCML gate, the power

consumed follows this equation: $P = V_{dd} * I_{bias}$. With the 3.3V Vdd, this nets an average power of 66.43 μ W. Eliminating the small current noise would net the effective same power dissipation without the fluctuation that occurs. The associated simulation output for this result is shown in Figure 5.4.

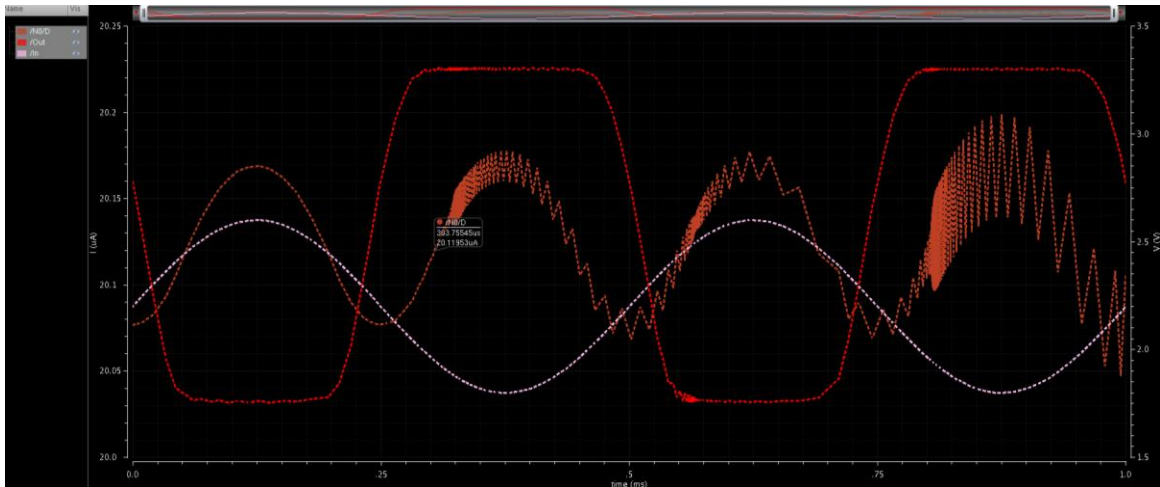


Figure 5.4: 1x Inverter Simulation with Current through the Tail Transistor

For MCML gates, it is okay to allow the voltage to rail at Vdd, but the tradeoff with this operation is the potential source of noise this introduces. Since the goal for these cells is to provide quiet circuitry for mixed signal applications, any source of noise must be examined and kept as minimal as possible. The current also begins to ‘spike’ in a small way after a couple cycles. These spikes range in the 10-50nA range however, and have negligible effect on the circuit performance. Despite these oscillations, the average current value remains steady. Further development of control circuitry can likely reduce this abnormality however the net operation remains correct and consistent. Current averaging occurs through software integration of the current signal in the Cadence simulator and over simulation timeframes that include many periods of signal operation. Further simulation and parameter captures for the 1x Inverter can be examined in

Appendix A. Control circuitry for the Pbias node was also experimented with for the purposes of keeping the load resistances in linear operation and able to make minor corrections based on operation. Once the current control was implemented however, correct operation was achieved using a 0V Pbias. Further details regarding the control circuitry development can be found in Appendix A along with additional info surrounding the inverter design. The full list of input parameters set during simulation for the 1x inverter is shown in Table 5.1.

Table 5.1: 1x Inverter Parameter Settings for Simulation

Parameter Name	Parameter Value
NmosWidth	2.7 μ m
NmosLength	600nm
PmosWidth	1.8 μ m
PmosLength	3.9 μ m
TailWidth	30 μ m
TailLength	9 μ m
TailCurrent	20 μ A
Signal Name	Signal Value
In	2.2V Offset -0.4V Amplitude Sine Wave
!In	2.2V Offset 0.4V Amplitude Sine Wave
Vdd	3.3V
Pbias	0V
Output Swing	1.5V

5.2.2 – 2x Inverter Schematic

The 2x drive inverter configuration aims to provide identical logical function with the difference of providing 2x the current and subsequent output drive. To set the doubled current for the device, the current mirror biasing network is set to 40 μ A and the tail transistor width is doubled. To compensate for the doubled current, the widths of all the transistors in the device are doubled as well. This ensures that the voltage drops through each transistor remain consistent through drive strength iterations. For the PMOS

however, length is adjusted as well to achieve voltage drops comparable to the rest of the variations in drive strength gates. As discussed in Chapter 2, the load resistors effectively perform as linear devices based on the bias current. The biasing is left constant and the resistance is halved by doubling the width while leaving the biasing and other signals the same. The other transistor widths are doubled as well due to the symmetric nature of the device and to allow for the equivalent ease in driving double current. This is a useful result for the inverter as the increased current configurations can be implemented with no changing of the input signals. Small incremental changes were made in sizing after doubling to achieve similar output functionality as the 1x inverter. This allows for interchangeability and the versatility expected with standard cells. The 2x inverter utilizes the same schematic as shown in Figure 5.2 with different parameters and settings for the variables. The doubled inverter meets the incremental requirements specified by the standard cell library constraints. The fact that a doubling in width linearly provides double the current supports the discussed theory and allows for greater ease of use and implementation of the inverter cells into the standard cell library. Simulating the double current inverter shows identical operation to the 1x design. Figure 5.5 shows the 2x inverter logical performance.

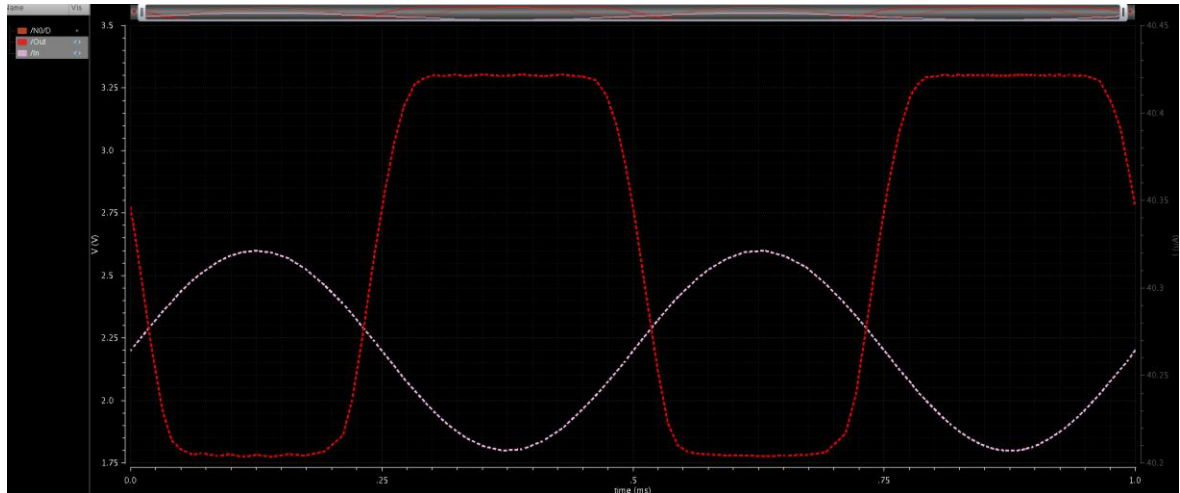


Figure 5.5: 2x Inverter Simulated Results Showing Correct Logical Operation

Note that the swings for each logical high and low value are the same as with the 1x current model. The peak output voltage rails at 3.3V and extends to 1.78V, netting a 1.52V swing. As mentioned previously, this is a larger swing for MCML but is consistent across each of the gates designed. This swing matches the 1x inverter results and the consistency with the input minimums allow for inverters in series to perform the same logical operations with the same voltage swings. The main difference in simulated performance between the 2x inverter and 1x inverter is the doubling of current. Figure 5.6 shows the current plot superimposed with the rest of the data.

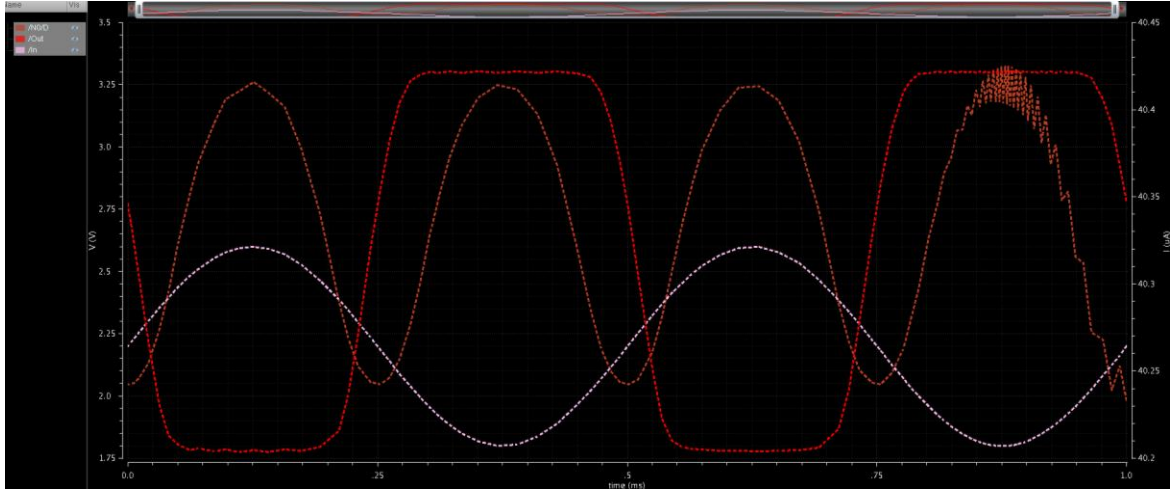


Figure 5.6: 2x Inverter Simulation with Current through the Tail Transistor

The marked current in the graph shows nominal current operation at $40.34\mu\text{A}$ that oscillate on the 200nA range with the input signal and small noise values that do not exceed 10nA . The spikes present with the baseline case are have a larger magnitude than that of the noise or spikes in the 2x case. This can be attributed to the increased transistor widths making the circuit resistant to noise that low in magnitude. The noise spikes only affect the current while the voltage remains constant. These oscillations may be attributed to simulator error or the behavior of ideal rails during simulation. These spikes tend to appear and disappear over three to five periods in steady state operation. The average current of $40.34\mu\text{A}$ is a 2.004 factor increase between the 1x case and the 2x result. The associated power consumption for the 2x current case computes to $133.12\mu\text{W}$. This consistency of current doubling allows for interfacing between these MCML gates and enables use in the standard cell library. As with the previous inverter, further analysis and details from the 2x inverter design can be found in Appendix A. Table 5.2 shows the final parameter list utilized for the 2x inverter simulation. Note that the widths of each transistor are doubled from the 1x inverter settings but the PMOS load transistors

required some minor incremental changes and length increase to achieve the desired voltage swing that matches where the lowest output value matches the lowest input value. The fact that the same input signals provided for the 1x inverter create the same result on the 2x inverter allow for cells to be exchanged without system redesign and make the standard cells more robust and interchangeable.

Table 5.2: 2x Inverter Parameter Settings for Simulation

Parameter Name	Parameter Value
NmosWidth	5.4 μ m
NmosLength	600nm
PmosWidth	3.9 μ m
PmosLength	5.1 μ m
TailWidth	60 μ m
TailLength	9 μ m
TailCurrent	40 μ A
Signal Name	Signal Value
In	2.2V Offset -0.4V Amplitude Sine Wave
!In	2.2V Offset 0.4V Amplitude Sine Wave
Vdd	3.3V
Pbias	0V
Output Swing	1.52V

5.2.3 – 4x Inverter Schematic

Following the same design practices as before, the widths of the transistors are doubled again to achieve the quadrupled current for the 4x MCML inverter design. With the doubling methodology performing effectively for the 2x case, the same approach proved viable with the 4x as well. This also allows for the same functionality and compatibility as the other cells in utilizing the same signals for operation and providing the same output voltage swings and logical operation. The process proved effective once again and as with the 2x inverter, the W/L PMOS loads had to be tuned slightly to achieve the desired voltage swing. Figure 5.2 shows the final schematic iteration for this standard cell library

at the 4x current specification where the parameters are updated again for the increased result. The same operation holds again for the quadrupled current case and the voltage swing remains steady at 1.5V to match the previously designed inverters. The maximum and minimum voltages for both input and output remain equivalent to the 1x and 2x inverters. Figure 5.7 shows this result with identical logical operation.

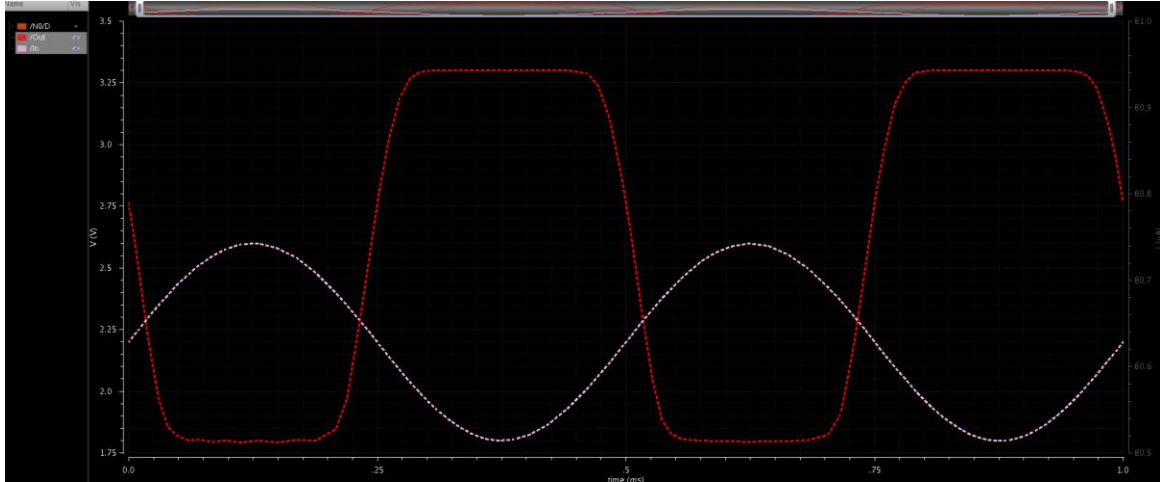


Figure 5.7: 4x Inverter Simulated Results Showing Correct Logical Operation

The output voltage again keeps the same 3.3V to 1.8V swing where the input and output minimums have the same value. This ensures operation that stays consistent when the MCML gates are chained together. The increased width again has a positive impact on the noise within the current signal. The transistors are wide enough to where noise is not noticeable in the simulation envelope. To verify another correct doubling of current, Figure 5.8 shows the simulated result with current plotted. Here, the current stays steady and smooth with no noise interfering. The only current fluctuation present occurs in line with the input signal to the system and the fluctuations yield about a 325nA variation in current throughout the duration of a signal pulse which constitutes $\sim 0.5\%$ in current variation over the course of a period of operation. This oscillation can be attributed to the

idealities in simulation of the rails and gate chain simulations shown later exhibit near constant operation.

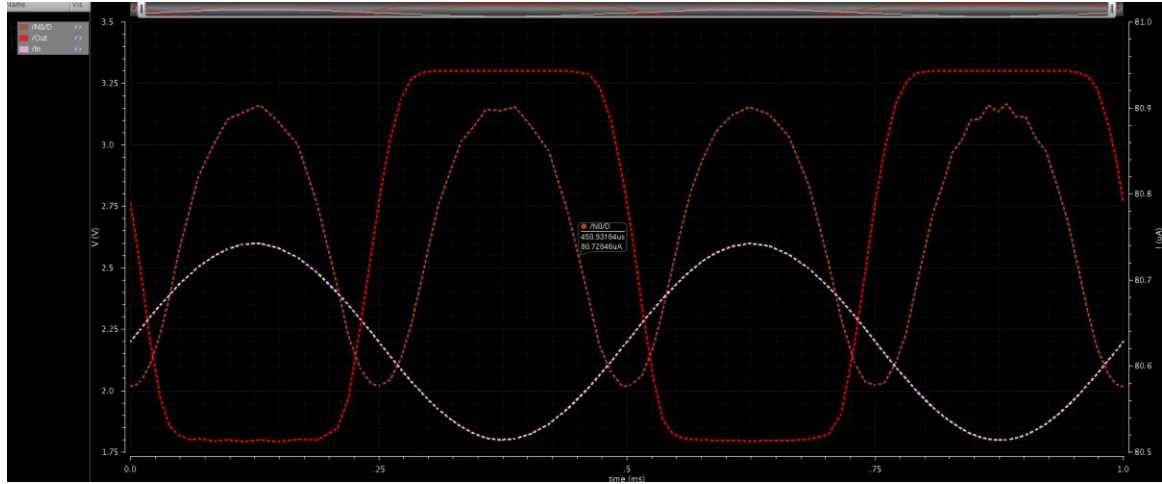


Figure 5.8: 4x Inverter Simulation with Current through the Tail Transistor

The average current for the 4x case comes out to $80.76\mu\text{A}$. This is a 2.002 factor of current increase from the 2x case and an overall 4.012 factor more current than the base 1x case. The power dissipated by the 4x inverter is $266.51\mu\text{W}$. The smooth operation, especially in the larger cells, shows the smooth operation achievable with MCML gates and the consistent control possible during operation. These results show that the inverter cells are suitable for standard cell operation and the fact that the currents were so closely doubled without any change to the input signals speaks favorably for the compatibility and interchangeable nature of these inverter cells. The minimal variation in current also supports the goal of creating quiet standard cells where logical operation of the device does not add noise to a system. Table 5.3 shows the final parameter configuration for the 4x inverter simulation. The trends from the 1x to 2x stage hold up well with regards to the widths of the transistors. The widths for the NMOS and tail transistors all double while the length for those components remains constant. The width of the PMOS loads is doubled and then modified slightly to increase the voltage drop and match up the output

minimum with the input minimum. Increasing W/L for the PMOS transistors decreases the voltage drop across the device and decreasing W/L through either width narrowing or length increasing raises the voltage drop. Small incremental tunings of the W/L values for the load transistors achieve the desired voltage swing.

Table 5.3: 4x Inverter Parameter Settings for Simulation

Parameter Name	Parameter Value
NmosWidth	10.8 μ m
NmosLength	600nm
PmosWidth	3.9 μ m
PmosLength	5.1 μ m
TailWidth	60 μ m
TailLength	9 μ m
TailCurrent	40 μ A
Signal Name	Signal Value
In	2.2V Offset -0.4V Amplitude Sine Wave
!In	2.2V Offset 0.4V Amplitude Sine Wave
Vdd	3.3V
Pbias	0V
Output Swing	1.5V

5.2.4 – MCML Inverter Schematic Remarks

The voltage swing attained during simulation falls into the desired range for these MCML devices to achieve low noise operation. Unlike CMOS gates, MCML devices should never swing rail to rail since both current paths remain active during correct operation. The current between each path is shifted one way or the other depending on the input signal and logical function of the device. The fact that each path remains active means that there are no devices changing between conduction and non-conducting operation which puts less noise on the supply and ground rails than a logic family that changes from conducting to non-conducting during switching as is the case with CMOS. The MCML gates also dissipate the same power regardless of switching frequency.

Changing the pulse times of the input signals causes no voltage or current change. As a result, the power consumed effectively remains equivalent; an advantageous property for MCML devices especially at high switching frequencies. CMOS counterparts dissipate nearly all of their power during switching, so a higher switching frequency results in higher power consumption.

Other notable trends include the common performance in logical operation and switching time over the course of each MCML inverter iteration. The voltage swings and magnitude of the outputs remain consistent with the same input signals for each gate size and drive strength. The oscillation magnitude of the current increases proportionally with the overall current increase and the noise within the current fluctuations decreases with the higher current values and wider transistors. Restated, the overall current fluctuation from nominal, as a result of ideal rail behavior in simulation, remains at ~0.5% the average current value. Each inverter also performs with the same input signals where the pulses on the transistor gates are sine waves that range from 1.8V to 2.6V at a frequency of 2kHz. These sine waves allow for clear depiction of logical operation for simulation of a single gate. Chains of gates and an oscillator provide more realistic representation of performance in a system and are examined later in the chapter. V_{dd} is set to 3.3V, and the P_{bias} for the load transistors is kept at 0V to ensure linear operation of the load devices. The biasing for the current regulation is controlled by a current mirror that sets the gate voltage of the MCML tail transistor based on the current through forced through the mirror. The consistent performance across each MCML inverter shows the modular nature of the devices and speaks favorably to low noise performance and the ability to implement the designed gates into a library.

Finally, additional control circuitry could smooth out the small fluctuations present in the operation of the device. Only the biasing tail current has simple control circuitry. The Pbias value would benefit from control as well where the resistive nature of the loads could adjust to keep the voltages within a desired region. There are many means of implementing these types of control circuitry and determining which option is best depends on the application. In a larger system, control circuitry would be utilized to keep sensitive mixed signal applications calm in operation. This would yield the constant power operation expected from standard cells and MCML. To attain standard cells, the schematics were developed independently of control circuitry and such additions would be external to the cell itself. The results show that the inverter standard cells function as expected in terms of logical operation and current capabilities. Next steps include layout development for implementation into a full standard cell library and development of other library components. This library develops a NAND gate cell to allow for any logical function to be created with the cells designed.

5.3 – MCML NAND Gate Schematic

The MCML NAND gate follows the schematic shown in Figure 5.9, recreated from Figure 2.3 for convenience. Note that depending on the arbitrary output pin configuration, the gate can also serve as an AND, OR, or NOR gate. The pin configurations for the output explicitly shown in Figure 5.9 show the AND gate configuration.

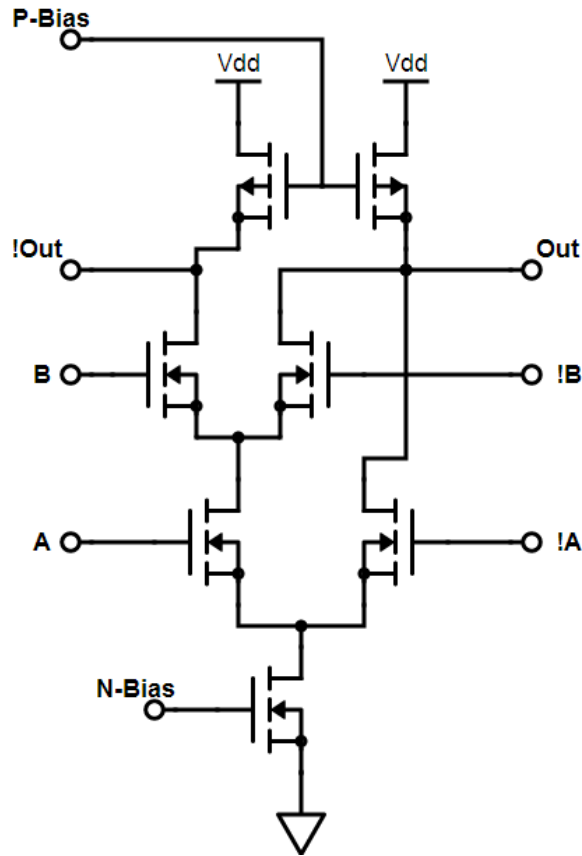
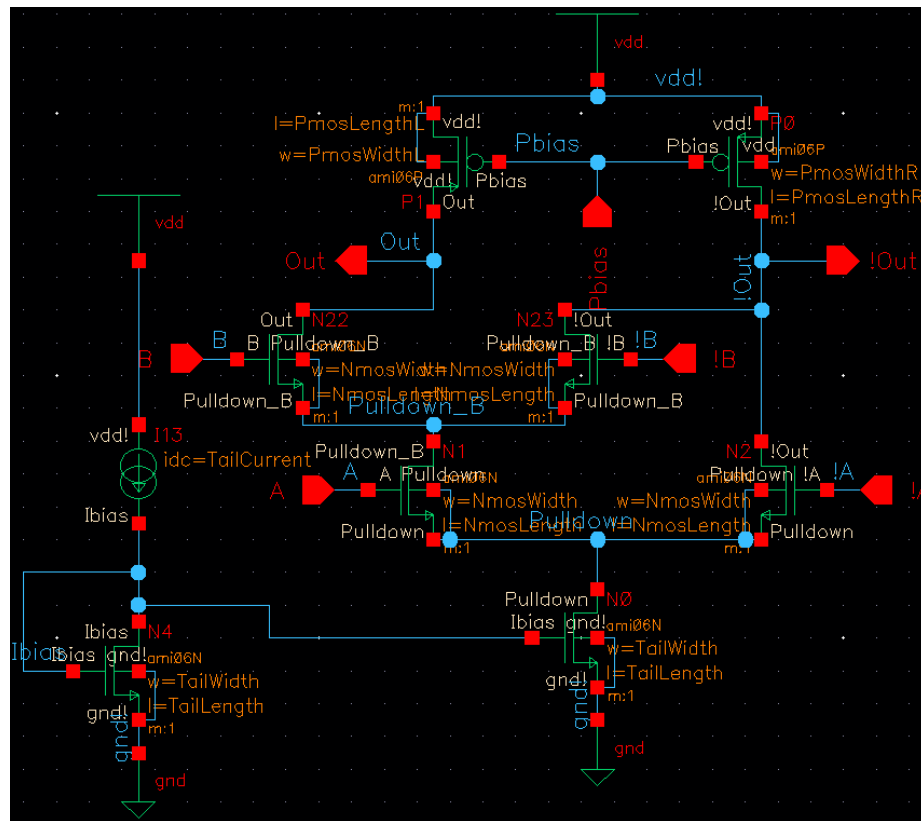


Figure 5.9: Recreated MCML AND/NAND/OR/NOR Schematic

Flipping the output pin labels causes the circuit to read as a NAND gate. In that light, the pin configurations used for the schematics detailed in this section utilize the general NAND gate configuration and are labeled as such. The standard “1x” current variation of the NAND gate utilizes minimally sized devices that still provide correct logical functionality. The non-symmetric nature of the MCML NAND gate creates additional challenges in the development of these cells and make biasing and transistor sizing trickier. To match up the voltage drops between the Out and $\overline{\text{Out}}$ lines, the PMOS load transistors were sized independently to allow for the necessary voltage drop across either path for low output. The current mirror used for controlling current set points for the inverter is present in the NAND as well.

5.3.1 – 1x NAND Gate Schematic

The first iteration of the NAND gate follows the same design principles as with the inverter and utilizes the minimally sized transistors with both the length and width set to 600nm. This is the minimum dimension allowed by the SCMOSC5 technology and serves as the baseline for minimum drive operation. The NAND gate is assembled as shown in Figure 5.9. The same nmos4 transistor symbols and pmos4 transistor symbols from the NCSU_Analog_Parts library are used for the creation of the NAND schematic. The same vdd and gnd rails are utilized as well. With the same Cadence procedures and steps as the inverter, the schematic for the general NAND gate is generated as shown in Figure 5.10.



Simulating the NAND gate follows the same process as the MCML inverter except with a two more input signals to supply. The same magnitudes for input signals were used for the NAND gate as with the inverter for maximum compatibility. The NAND gate however, uses square wave inputs to correctly show the logical transitions based on input with a resolution that sine wave inputs would not provide due to non-exact switching between logical positions. Figure 5.11 depicts the 1x NAND gate simulations with the logical operation of the NAND signals visible. Like the inverter, the PMOS load gate voltages are kept at 0V for linear operation.

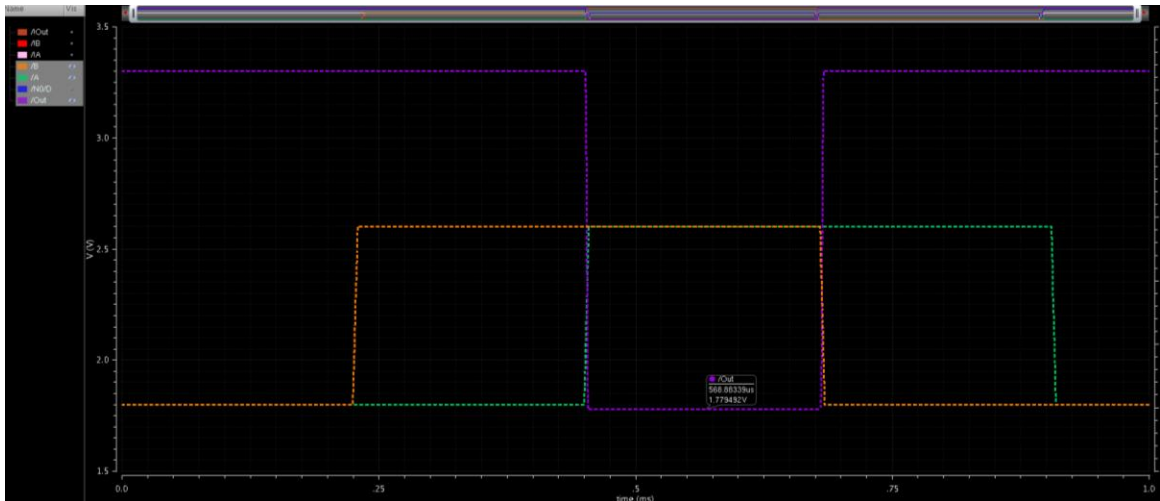


Figure 5.11: 1x NAND Gate Simulated Results Showing Logical Performance

The logical operation for the NAND path follows inputs of: 00, 01, 11, & 10. As the results show, the NAND gate performs as expected with the only low output swing given for the 11 input. The voltage for the NAND gate rails similarly to the inverter for logically high output at 3.3V and is designed to swing the logical low output to the minimum of the input signal. As with the inverter, the input ranges from 1.8V to 2.6V and the NAND output reaches a minimum of 1.78V, yielding a 1.52V output swing. The 1.8V to 2.6V input range again proves to provide stable starting operation for a single

MCML gate under simulation. Realistic gate to gate connections are explored later in the chapter. The same swinging ranges and input signals as the MCML inverter allow for interfacing of gate types with correct logical operation maintained. To examine the current capabilities of the base case, Figure 5.12 includes the output data with current plotted as well.

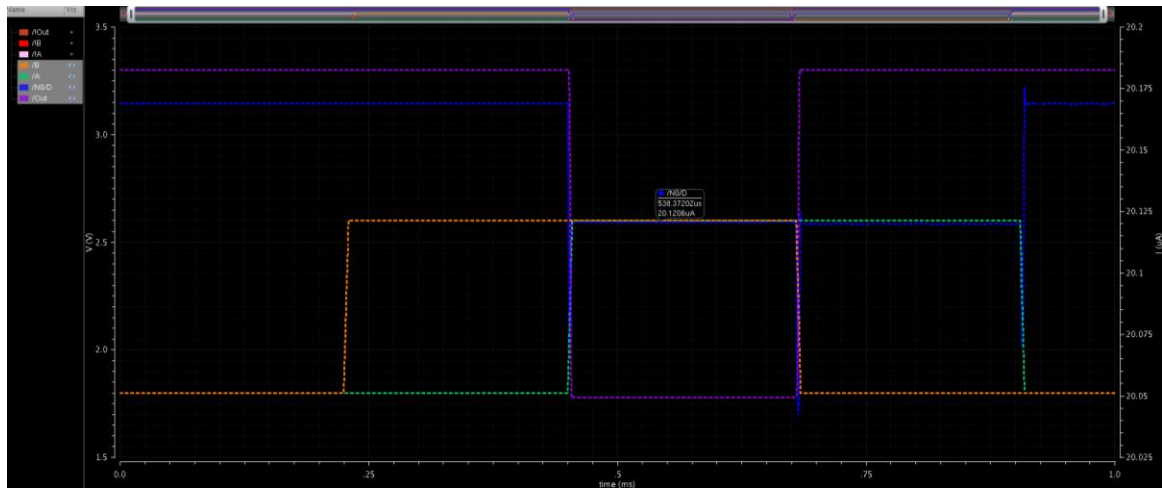


Figure 5.12: 1x NAND Gate Simulated Results with Biasing Current Visible

The blue trace represents the current for the NAND gate and this current follows the operation of the ‘A’ input meaning that when ‘A’ switches, the tail current source fluctuates due to the switching operation and current steering. The ‘A’ inputs control which main path is steered to the tail current path and the non-symmetric nature shows the associated fluctuation in current when the ‘A’ signal switches. This fluctuation remains on the order of about 100nA where the average current for the 1x NAND gate computes to 20.15 μ A. Similar to the inverters, this variation represents a 0.5% current fluctuation during the most severe switching operation. Despite these minor current fluctuations, the current stays balanced over logical sweeps and nominally performs at the 20 μ A intended current setting. Note that this current set point matches that of the 1x

MCML inverter and the subsequent drive strength iterations continue to follow those trends in consistency between NAND gates and inverters at each drive strength. This ensures compatibility between each of the cells developed and a ring oscillator is created to prove this point and is examined later on. The power dissipated for the 1x NAND gate averages 66.50 μ W with the same 3.3V Vdd that was used for each iteration of the MCML inverter. Following the same methodology as the inverter design, 2x and 4x current iterations for the NAND gate are developed as well for compatibility in the standard cell library. The principle of doubling transistor widths is employed again for current doubling and the PMOS load transistors are tweaked incrementally to maintain the desired voltage drops. Further analysis and captures from the 1x NAND gate development can be found in Appendix A along with inverted logic (AND) results that mirror the NAND functionality. Table 5.4 displays the final parameters utilized for the 1x NAND gate. Design variables match up with the labeled schematic variable naming in Figure 5.10.

Table 5.4: 1x NAND Gate Parameter Settings for Simulation

Parameter Name	Parameter Value
NmosWidth	2.7 μ m
NmosLength	600nm
PmosWidthL	2.7 μ m
PmosLengthL	6.6 μ m
PmosWidthR	2.1 μ m
PmosLengthR	4.8 μ m
TailWidth	30 μ m
TailLength	9 μ m
TailCurrent	20 μ A
Signal Name	Signal Value
In	2.2V Offset -0.4V Amplitude Pulse
!In	2.2V Offset 0.4V Amplitude Pulse
Vdd	3.3V
Pbias	0V

Output Swing	1.52V
--------------	-------

5.3.2 – 2x NAND Gate Schematic

The 2x current NAND gate follows the same principles as before. First, transistor widths are double to allow for approximate halving of resistance and subsequent doubling of current. The generalized MCML applications discussed in Chapter 2 show the simple nature of the equations behind MCML operation and even in the realistic cases simulated, the expected performance and nature of the devices holds true. The 2x NAND gate model utilizes the same general schematic as presented in Figure 5.10. Simulating the doubled current NAND gate follows the same guidelines as with the inverters. The signals are kept the same as the 1x NAND gate and all transistor sizes are kept to legal standard cell library increments. Since the load transistors change resistance when the width of each component is doubled, the PMOS load dimensions are tweaked to achieve the desired voltage drop across the loads and create the expected output voltage swing. Logical operation for the 2x current NAND gate follows the same performance trends as with the base case with nearly identical output voltage swings for the same input signal as the 1x case. Figure 5.13 shows the 2x NAND gate logical operation.

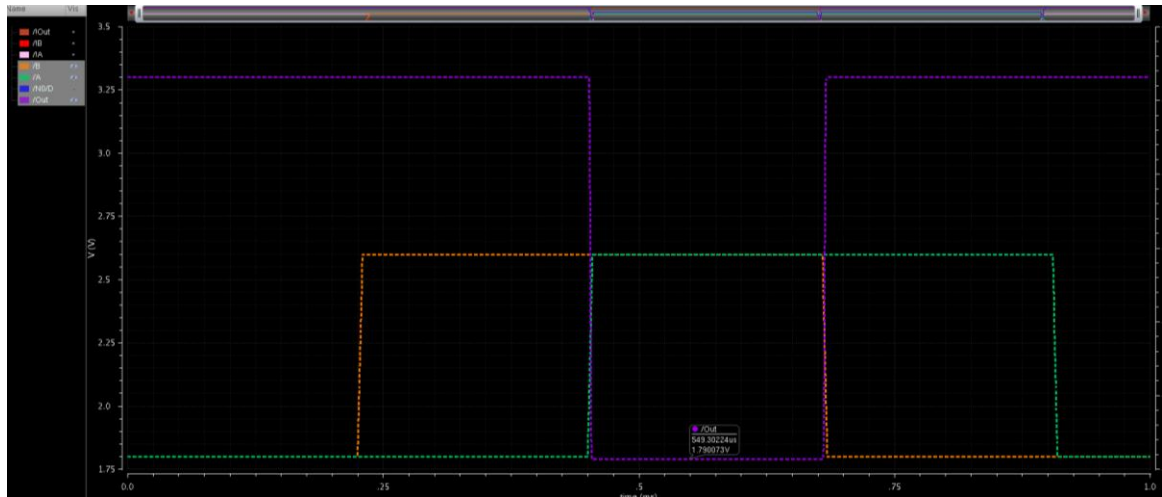


Figure 5.13: 2x NAND Gate Simulation Showing Logical Performance

With identical logical performance with the 1x drive case, the performance metric remaining is the current performance of the doubled NAND gate. Note that the voltage swing for the output ranges from the 3.3V rail down to 1.79V, totaling a 1.51V output swing. This matches with both the minimum of the input signal and the minimum of the previous NAND gate and inverters. This ensures that the 2x NAND gate is compatible with the other cells in the library and can be interfaced with other logical devices or gates of various drive strengths while preserving logical functionality. Figure 5.14 displays the current performance of the doubled NAND gate. Small tail current spikes occur during the transitions of the ‘A’ signal for the same physical reasons as examined in the 1x drive NAND case and the current remains within a 200nA region where only the switching spikes exceed any variation over 100nA from the nominal tail current during operation.

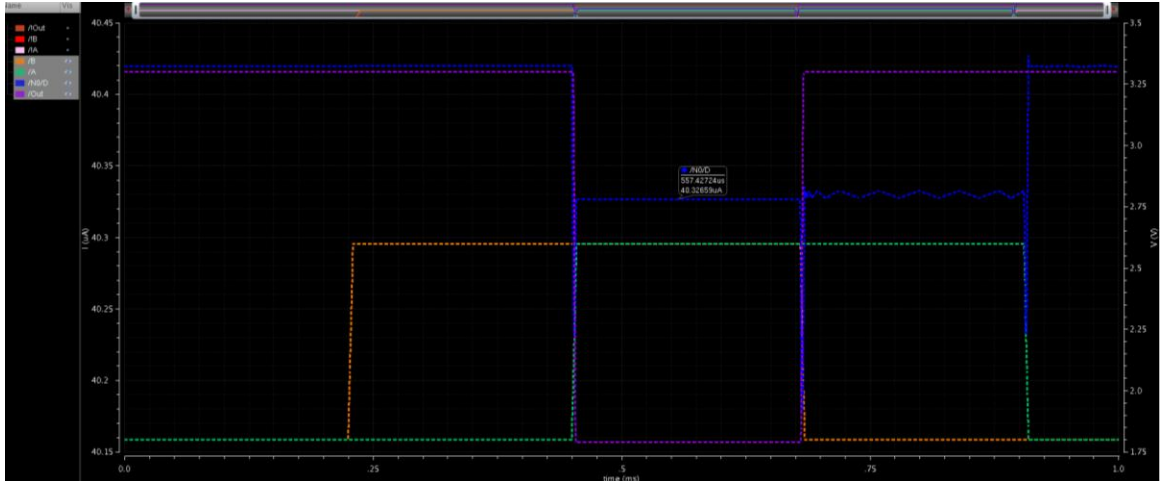


Figure 5.14: 2x NAND Gate Simulated Results with Current Performance Visible

The current matches the predicted trends and follows the same patterns as the 1x case. Note that the largest and most inconsistent tail current operations occur in the switching of signal ‘A’. The nonsymmetrical nature between the NAND gate paths explain this fluctuation with the net resistance on the line depending on the active path. Since ‘A’ dictates the overall current steering of the gate between the left and right paths, it makes sense that the current fluctuates. However, single gate simulations behave with idealized rails which also attribute the switching spikes. Connected gates explored later demonstrate even tighter performance in current which highlights the low noise nature of the designed MCML cells. Otherwise, the current remains steady and averages out to 40.38 μ A. This is a 2.004 factor increase over the base case NAND gate. The power dissipated by the 2x NAND gate totals 133.25 μ W. Further analysis of the NAND gates and development process including logical performance of the $\overline{\text{Out}}$ AND line can be found in Appendix A. The simulation parameters required for 2x NAND gate performance are detailed in Table 5.5. Note that each transistor width doubles and the

PMOS load transistors then show small incremental changes for increased performance in achieving the desired results.

Table 5.5: 2x NAND Gate Parameter Settings for Simulation

Parameter Name	Parameter Value
NmosWidth	5.4 μ m
NmosLength	600nm
PmosWidthL	3.9 μ m
PmosLengthL	5.1 μ m
PmosWidthR	4.5 μ m
PmosLengthR	6 μ m
TailWidth	60 μ m
TailLength	9 μ m
TailCurrent	40 μ A
Signal Name	Signal Value
In	2.2V Offset -0.4V Amplitude Pulse
!In	2.2V Offset 0.4V Amplitude Pulse
Vdd	3.3V
Pbias	0V
Output Swing	1.51V

5.3.3 – 4x NAND Gate Schematic

As with the MCML inverters and NAND gates so far, the same trends are expected to hold again during a subsequent doubling of transistor widths to achieve the 4x drive strength. The schematic again follows the variable dimension setup in Figure 5.10 where the transistor sizings are set in the simulation interface. Figure 5.15 depicts the logical performance of the 4x NAND gate. The voltage swings remain consistent with the previous 1x and 2x NAND gates as well as the inverter iterations. The voltage swing remains steady around the desired 1.5V where the output swing traverses from the 3.3V Vdd rail to 1.82V. This nets a 1.48V output swing where the minimum values of the output and input align. This again makes for compatibility and interchangeability with

other cells in the library and these criteria serve as a useful metric of performance for characterizing additional cells.

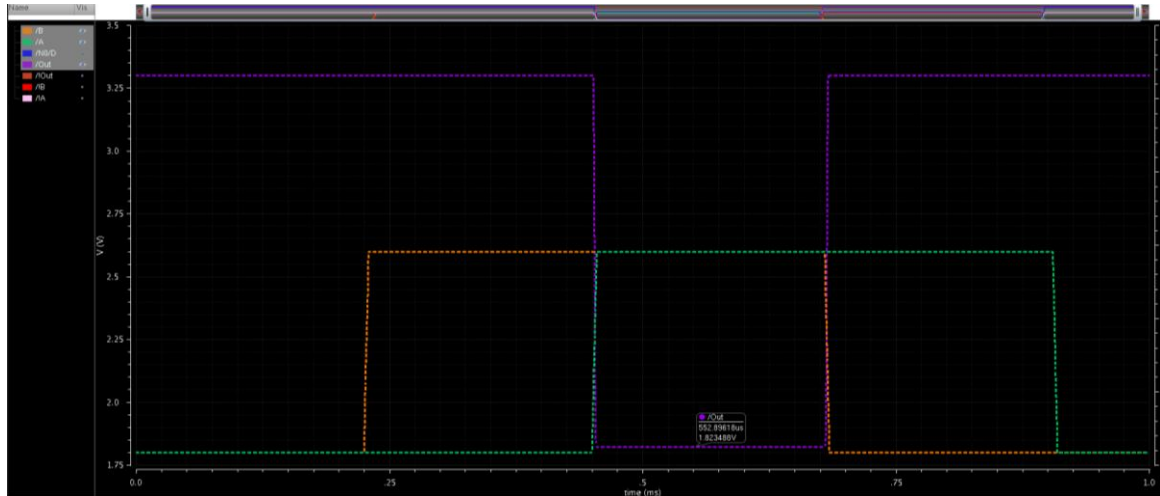


Figure 5.15: 4x NAND Gate Simulation Showing Logical Performance

With identical performance to the previous NAND gates including voltage swing and logical operation, the metric for classifying the successful design of the 4x gate lies with the current performance. Figure 5.16 shows the logical operation of the NAND gate with current plot in place. Note that the fluctuation trend from before follows where the tail current nominally lowers when ‘A’ switches into a logical high input. This occurs because signal ‘A’ controls the steering between the overall left and right branches of the gate. Again, these transitions account for a 0.5% variation off average current and can be attributed to the ideal nature of rails during schematic simulation.

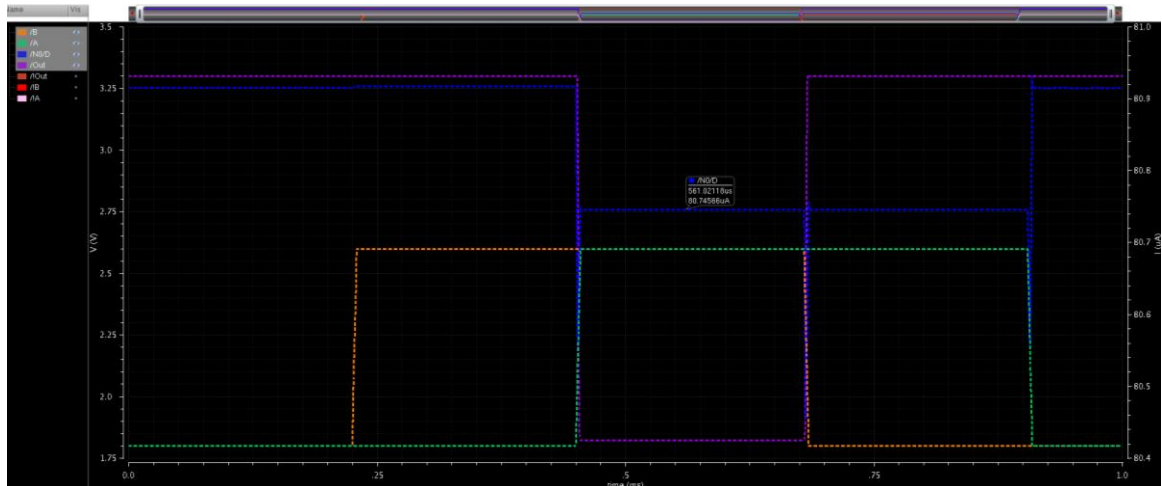


Figure 5.16: 4x NAND Gate Simulated Results with Current Performance Visible

The simulated current for the 4x NAND gate follows the same characteristic fluctuation as with the 2x and 1x inverters. As expected with the subsequent doubling of current, the variation in current performance doubles as well. The current spikes during ‘A’ switching peak at maximum of 350nA where the nominal operational variation in current sits at about 175nA. Interestingly this is not fully double the variation observed in the 2x case and as a result, one could speculate that a larger gate would be more tolerant to noise, but would likely perform more slowly. As with the previous gates, the variation peaks around 0.5% of the nominal current value and can be attributed to major current steering changes when ‘A’ logically switches and the ideal nature of Vdd and GND when simulating at the schematic level. Ring oscillator simulations and connected gate simulations later in the chapter display even more constant current operation and demonstrate the low noise capabilities of the MCML gates.

The current average for the 4x NAND gate comes out to 80.84μA. This is a 2.002 factor of current increase from the 2x NAND gate and an overall 4.012 increase from the

1x NAND gate. The net power dissipated by the 4x inverter is $266.77\mu\text{W}$. Table 5.6 shows the final 4x NAND gate simulation parameters.

Table 5.6: 4x NAND Gate Parameter Settings for Simulation

Parameter Name	Parameter Value
NmosWidth	10.8 μm
NmosLength	600nm
PmosWidthL	9.6 μm
PmosLengthL	6.9 μm
PmosWidthR	9.6 μm
PmosLengthR	6.9 μm
TailWidth	120 μm
TailLength	9 μm
TailCurrent	80 μA
Signal Name	Signal Value
In	2.2V Offset -0.4V Amplitude Pulse
!In	2.2V Offset 0.4V Amplitude Pulse
Vdd	3.3V
Pbias	0V
Output Swing	1.49V

Interestingly, the load transistors for the 4x NAND gate ended up achieving the desired swing with the same transistors dimensions. In each of the other NAND gate iterations, comparable delays between the output and its inverse required fairly different sizings for the load transistors. As with each schematic, further captures detailing operation are presented in Appendix A.

5.3.4 – MCML NAND Gate Schematic Remarks

The NAND gates performed largely as expected after the development of the MCML inverters. The same principles in transistor and gate biasing were applied and the NAND gate performed as desired. This is favorable for further MCML gate development as the

same principles for other logical functions and devices should allow for rapid development of more standard cells or gate sizings.

Since the NAND gate is not symmetric, the PMOS transistors were sized independently to balance either the output voltage swing or the swing of its inverse. Interestingly, the sizings for the larger NAND gates utilize PMOS transistors that were much closer in size to one another. The 4x NAND gate specifically ended up achieving the desired voltage swing on both outputs with identical PMOS loads. Since the PMOS loads are usually intended to be kept at the smallest sizing possible, the increased transistor widths of the NMOS components eventually dominated the network due to higher W/L ratios and the PMOS transistors became less impactful in dictating the output response for a given dimension change.

The fact that the MCML NAND gates utilize the same signal input bounds as the inverter allow a common source to drive both gates without fear of losing logical operation. The output swings of the NAND gates also match those of the inverter and each gate developed swings from rail to the minimum input value. This constitutes the first step in examining gate interfacing. The ring oscillator developed takes this further and ensures that these gates maintain the same performance results in a system with many MCML gates connected together. This allows gates to be mixed, matched, and strung together while ensuring correct logical performance. Further cells or sizes developed for the MCML standard cell library should adhere to the same principles and attempt to function using the same input signals and be sized to provide the same output swing. This allows the library to be extended in the future by future projects and ensures compatibility for anyone deciding to use the library in design of a system.

Additional development info and captures for the MCML NAND gates can be found in Appendix A. With each schematic generated for any logical function and varying current capabilities, the layouts for each standard cell can then be designed with the confidence that the cells can interact favorably with each other and are compatible in terms of input and output signals.

5.4 – MCML Inverter Layout

The inverter layouts for the 1x, 2x, and 4x sizings are created by hand and maintain the sizing requirements for standard cells. This means that the height of each cell is fixed and the width of the cells can be variable. The tool that utilizes the standard cell library must know the size of the cell and the I/O positioning for placing each block into the grid. The fixed height allows for the cells to snap onto uniform V_{dd} and GND rails. Utilizing the height sizing of $36.15\mu\text{m}$ from another standard cell library [9] using the ami06 technology, the layouts for the MCML devices can be generated for each sizing of the device. The current biasing circuitry implemented for the schematic simulations are not included in each cell layout and are an external addition to the standard cell. Each cell contains the core logic of the device and utilizes the simulated transistor sizings from the previous sections.

5.4.1 – 1x Inverter Layout

For each layout, a pr boundary is created that contains the overall sizing of the cell layout. Each layer fits within this boundary and the height of each pr boundary represents the consistent height required for standard cells to be placed and connected in grid-like fashion. The 1x Inverter layout utilizes a pr boundary with height of $36.15\mu\text{m}$ and width of $47.7\mu\text{m}$. The width of the cell is arbitrary and flexible as long as the cell snaps into the

grid rails that are configured based around the height. The width of the cell is selected to create enough space between devices to allow for routing when creating the layout. Pcells are used for each of the transistors when moving from schematic to layout. These are parameterized cells that take the dimensions set in the schematic and create the layout for each transistor. From there, the transistors are positioned and routed to achieve the same operation as the simulated schematic. The rails for the inverter are created using 3 μ m long metal layers that create V_{dd} and GND. The V_{dd} rail also needs a 4.5 μ m n-well below the rail to meet the standardized requirements for cell development as in [9]. The PMOS transistor n-wells must also overlap the major n-well so each bulk region carries the same potential and the transistors function correctly. Figure 5.17 shows the block layout for the 1x inverter with visible Pcells and routed paths/rails. Figure 5.18 depicts the full completed layout for the 1x inverter where all layers are visible for the transistors and the necessary routing layers. Rulers are present around the cell on the top and left to indicate overall dimensions and ensure that the rails and heights meet the necessary requirements for the standard cell in the ami06 technology.

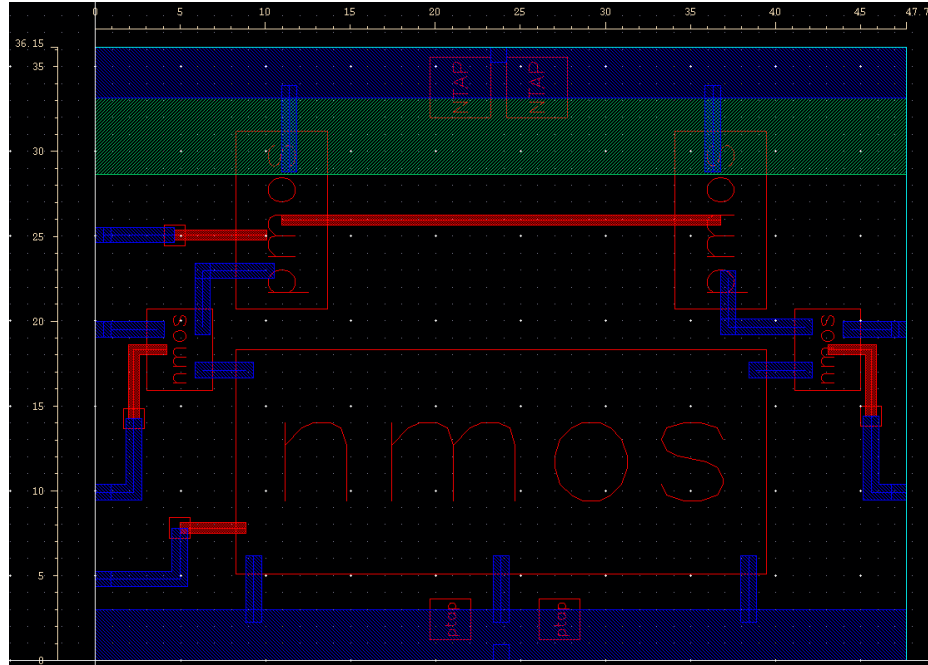


Figure 5.17: MCML 1x Inverter Block Layout with Pcells and Routing

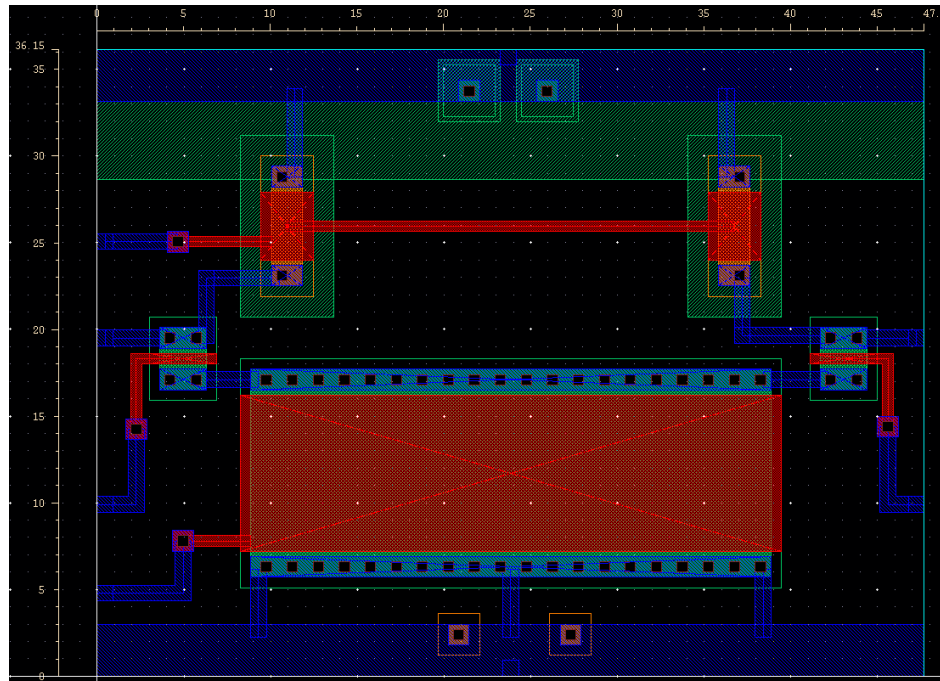


Figure 5.18: MCML 1x Inverter Full Routed Layout with All Layers and Pathing

Note that the Vdd rail requires a PTAP which is provided in the NCSU ami06 library.

This provides the necessary connections for the rails and the connection to the outside

world for powering. Similarly, the GND rail uses the NCSU NTAP which provides the

same functionality for the ground portion of the circuit. For standardized implementation and usability, the layout is kept as symmetrical as possible with similarly sized paths for each signal. Multiple pathing layers connect the tail current source to ground to ensure even current flow through the wide device.

5.4.2 – 2x Inverter Layout

The 2x MCML Inverter layout follows the same sizing design principles as the 1x inverter. The height of the cell remains fixed at $36.15\mu\text{m}$ while the width must increase to $90\mu\text{m}$ to accommodate the larger width of the tail current source. For this cell and for other layouts, the width sizing is attempted to be kept at the tail transistor width plus $30\mu\text{m}$ to allow for the other smaller devices. Following these ideas and the same rail positioning and sizing as the 1x inverter, the final layout for the 2x inverter standard cell is generated. Again, Pcells are used from the NCSU ami06 library for the transistors, vias, and rail taps. The general form of the layout keeps the same design as the 1x inverter with the larger components. Figure 5.19 shows the block layout with routed layers and vias. Each block represents a Pcell utilized to generate a small piece of the layout including transistors and rail taps. Figure 5.20 shows the final routed version of the 2x MCML inverter with all layers for each transistor and pathing available. Again note that there are rulers present on the layout for reference sizing in microns. Note that the sizing was successfully fit in the $90\mu\text{m}$ width ($60\mu\text{m}$ tail width + $30\mu\text{m}$).

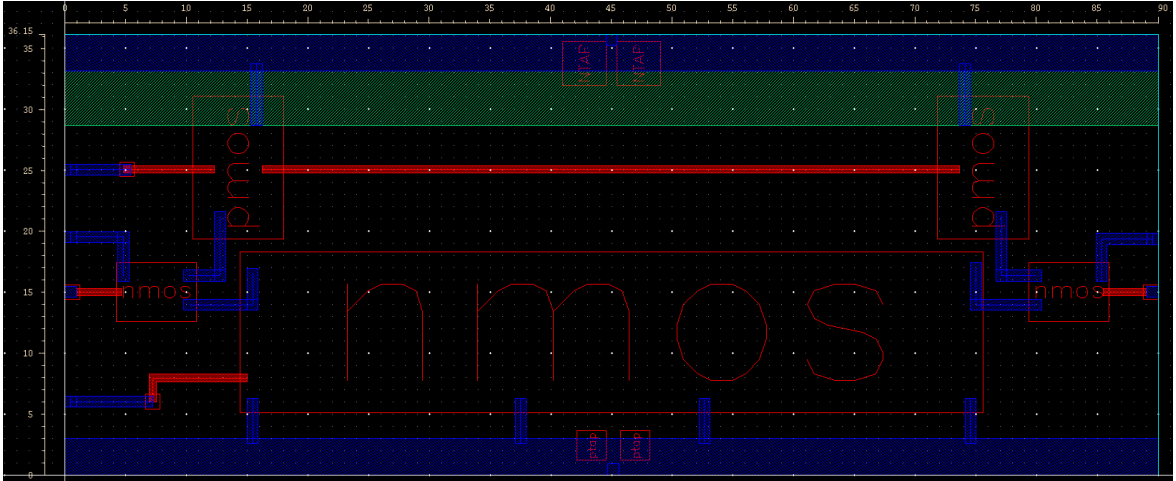


Figure 5.19: MCML 2x Inverter Block Layout with Pcells and Routing

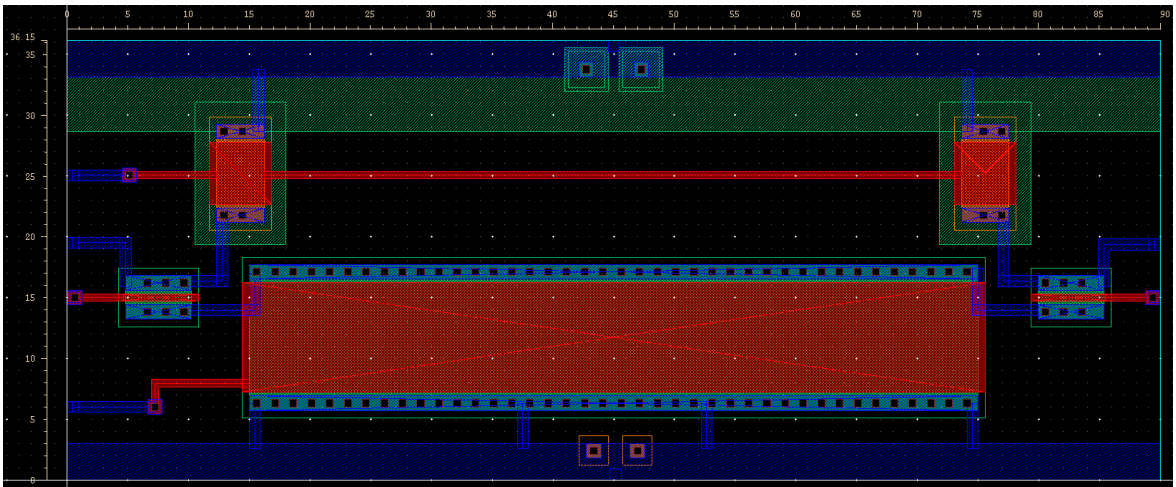


Figure 5.20: MCML 2x Inverter Full Routed Layout with All Layers and Pathing

As with the 1x inverter, the layout is kept symmetric and the tail transistor contains multiple ground routing paths for even current flow over the wider device.

5.4.3 – 4x Inverter Layout

The layout for the 4x MCML Inverter follows the same design methodology as the first two sizing iterations. Again, the height of the standard cell is fixed at $36.15\mu\text{m}$ and the rails are designed with the same sizing and height dimensions. The tail transistor of the 4x inverter is $120\mu\text{m}$ wide and following the same sizing decision as the 2x inverter, the

overall width for the cell is set to 150 μm . Even with the large layout of the 4x device, additional metal levels are not required due to the simple nature of the MCML inverter. As a result, the final layout for the 4x MCML appears similar to that of the smaller drive strength inverters. Figure 5.21 shows the block layout with Pcells and routing visible and Figure 5.22 shows the final full layout for the 4x inverter where all transistor layers, vias, and pathing are visible.

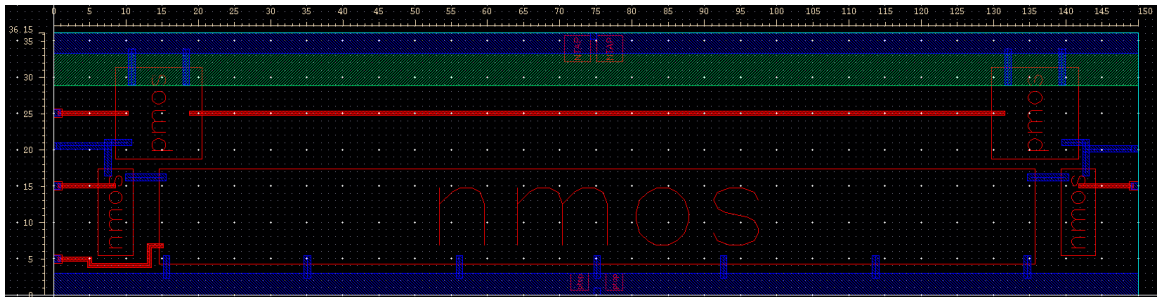


Figure 5.21: MCML 4x Inverter Block Layout with Pcells and Routing

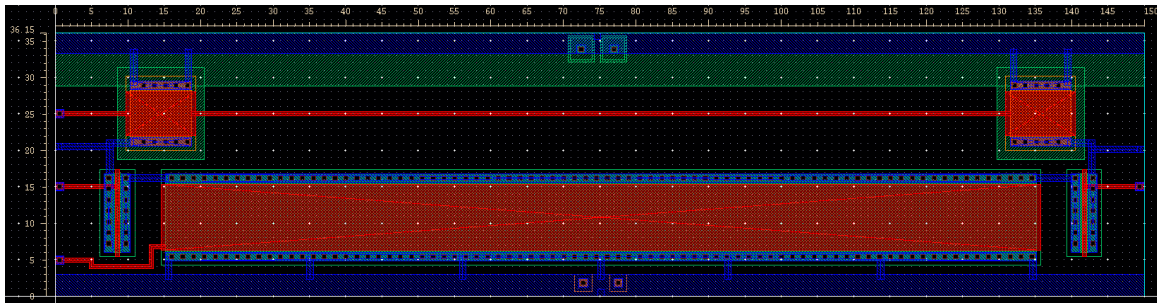


Figure 5.22: MCML 4x Inverter Full Routed Layout with All Layers and Pathing

The elongated nature of the NMOS transistors required a small path around one of the devices for biasing the tail current transistor. As with the previous iterations, additional paths from the tail transistor to ground are added for even current flow and the PMOS devices are connected to V_{dd} with two path lines for smoother current flow across the device and lower overall impedance from wiring to ensure that the devices perform like the schematic simulations.

Each inverter utilized a mix of auto and manual routing where necessary. The auto routing took care of non-via connections for the I/O pins where the final transistor connections and power routing all took place by hand following the same principles and guidelines. The same process steps and constraints are kept in mind when developing the NAND gate layouts as well.

5.5 – MCML NAND Gate Layout

The layouts for the MCML NAND Gate standard cells follow the same guidelines as the MCML inverter. The height remains fixed at $36.15\mu\text{m}$ and the width is variable and set to tail transistor width plus $30\mu\text{m}$ for similarity in sizing with the inverter cells. The more complex nature of the NAND gate in MCML requires the metal 2 layer to be utilized for some of the routing to meet the desired dimensions. The number of transistors in the NAND gate make the height of the standard cell the limiting factor and main challenge in routing. Using metal 2 allows for the layout to remain compact but adds fabrication complexity. As with the inverters, the NAND gate layouts are developed without the current biasing circuitry as in implantation, biasing circuitry would only be required every few cells and would add to overall complexity if implemented in every gate of a larger system.

5.5.1– 1x NAND Gate Layout

The 1x NAND gate uses a pr boundary for the layout of $36.15\mu\text{m}$ by $60\mu\text{m}$. This width fits the arbitrary specification of setting the standard cell width to tail width plus $30\mu\text{m}$. The process follows the same steps as the inverter layouts with a more complicated layout due to the added components necessary for a two input gate. The same Pcells from the ami06 NCSU library are used for the NMOS and PMOS transistors. The rail sizing

and dimensions match that of the inverter layouts so that the standard cells are compatible when implemented together or in a library. Figure 5.23 shows the block layout of the 1x NAND gate with routing and paths. Figure 5.24 shows the final layout with all paths and layers visible for each component. Due to the non-symmetric nature of the NAND gate, the layout cannot be kept exactly symmetric and a small metal2 layer is required to allow for all necessary transistor connections. The height begins to cause some challenges in layout already with the nature of the NAND gate and one long path had to be created to connect the ‘b’ input transistors together. The PMOS load transistors also reflect the width differences explored during schematic simulation. Extra power routing paths are added to allow for lower impedance and uniform current flow through the device. Note that each layout view contains visual rulers for scaling reference.

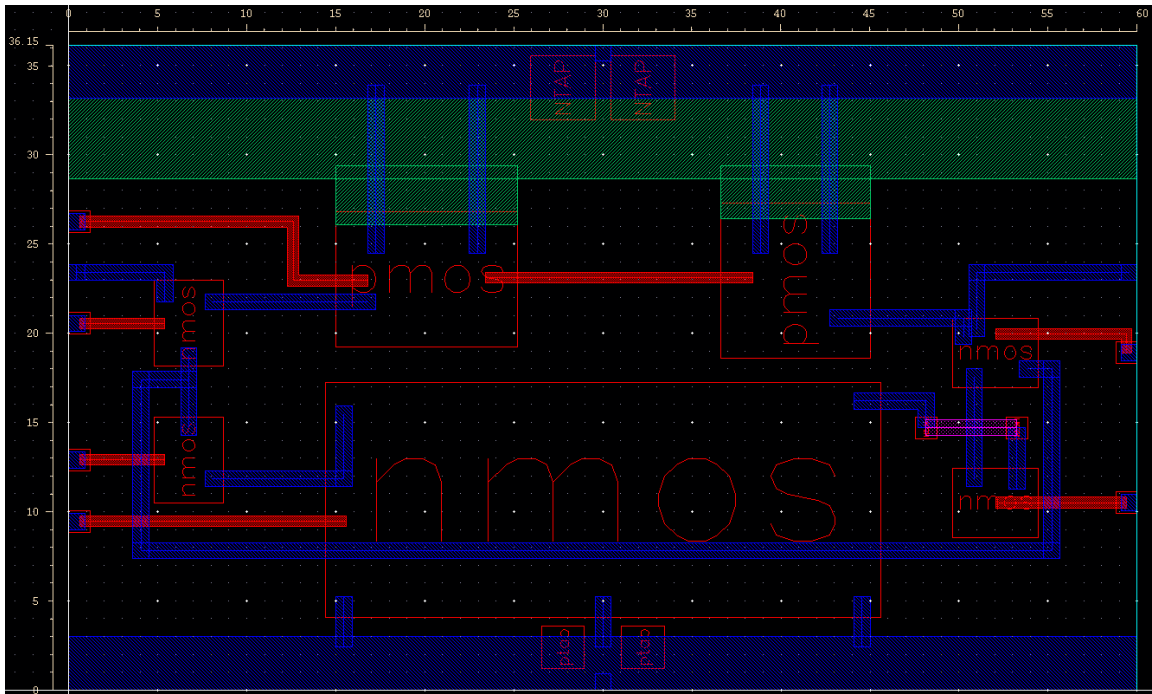


Figure 5.23: MCML 1x NAND Gate Block Layout with Pcells and Routing

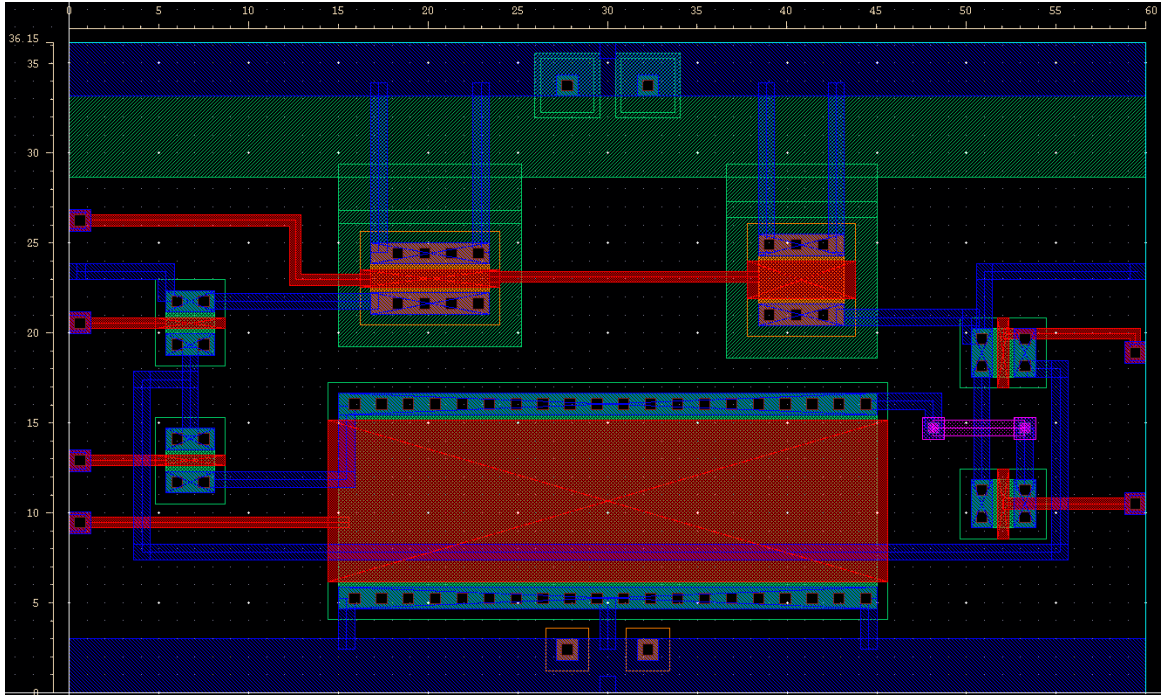


Figure 5.24: MCML 1x NAND Gate Full Routed Layout with All Layers & Pathing

5.5.2 – 2x NAND Gate Layout

The 2x NAND gate layout for the MCML standard cell library takes the same specifications into consideration during development. The width of the cell is set to 90 μ m to fit the increased tail transistor that has a width of 60 μ m. As the NAND gate cells get larger, more layout challenges come into play and slightly more metal2 is used as a result. With all rails and height dimensions the same as the 1x NAND gate and all inverters to meet standard cell requirements. Figure 5.25 shows the 2x NAND gate block layout with Pcells and pathing routes visible. Figure 5.26 depicts the final layout for the 2x NAND gate with all transistor layers and routing aspects visible. For this layout, metal2 is used to connect the two ‘b’ input transistors and the path stretches across much of the width of the layout. The rest of the pathing for the layout was able to be kept at the lower metal1 layer to keep the design simple.

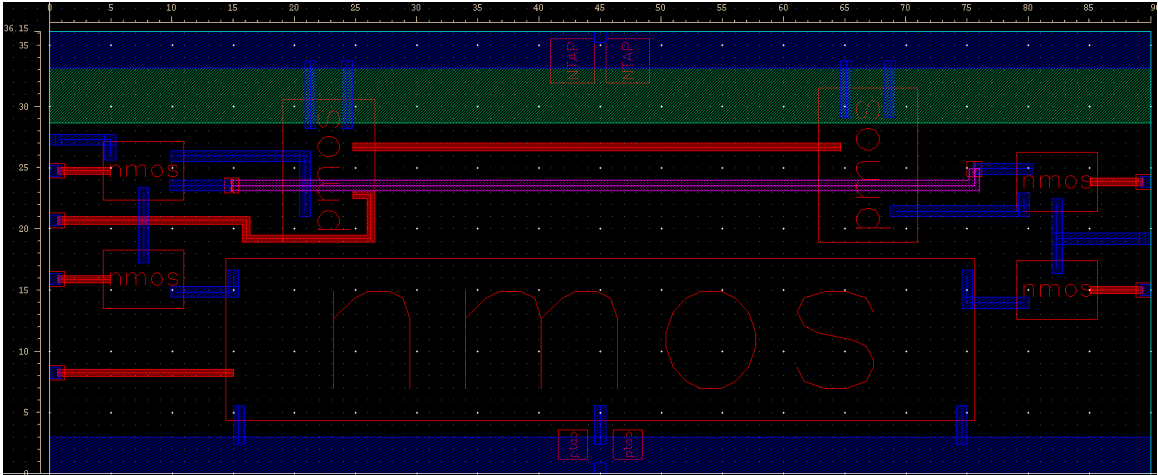


Figure 5.25: MCML 2x NAND Gate Block Layout with Pcells and Routing

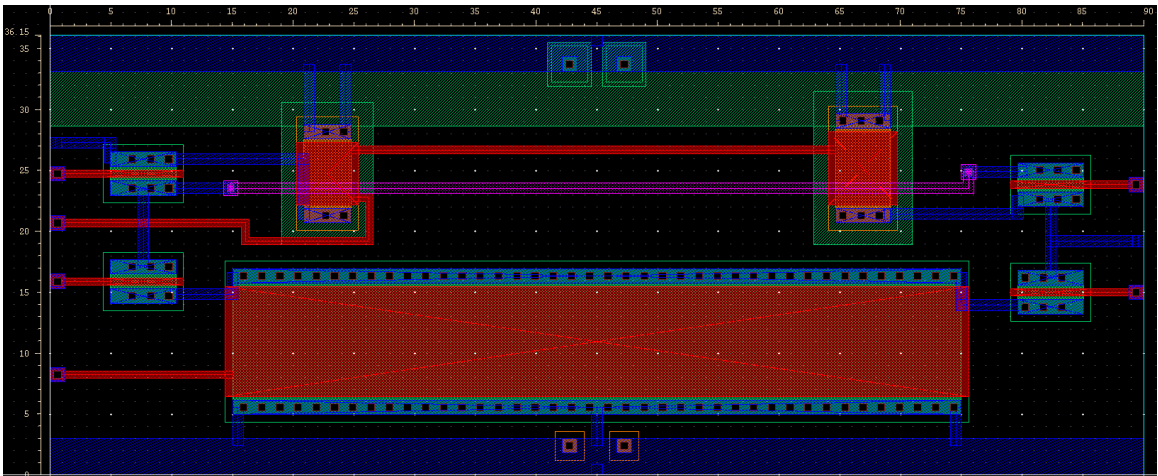


Figure 5.26: MCML 2x NAND Gate Full Routed Layout with All Layers & Pathing

The small NMOS transistors allow for comfortable spacing between the devices on this layout. The larger dimensions across all devices for the 4x case cause more encroachment when following the same sizing guidelines.

5.5.3 – 4x NAND Gate Layout

The MCML layout for the 4x NAND gate sets a cell width to 150μm to accommodate the larger 120μm tail current transistor. The added width for the other transistors cause the 30 micron cushion to create a denser layout and more metal2 is used to allow for the same fitting of overall dimensions and remain comparable to the 4x inverter layout. Figure 5.27

shows the block view with routing connections of the 4x NAND gate where each block represents a Pcell used for a transistor or via. Figure 5.28 shows the final layout for the MCML 4x NAND gate where all layers for transistors, pathing, and routing are shown.

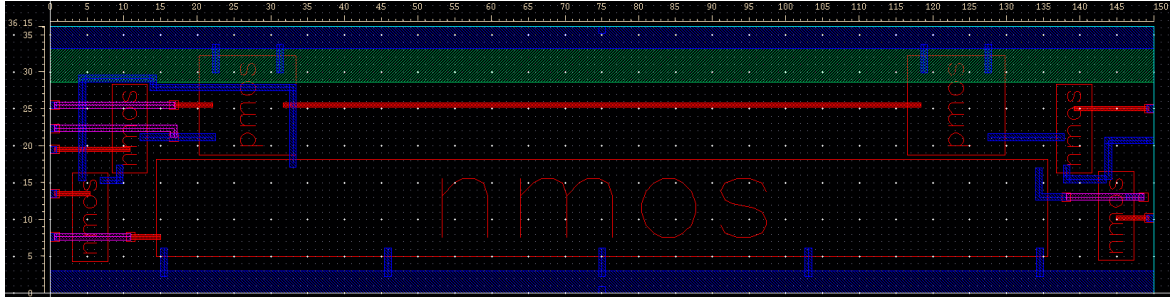


Figure 5.27: MCML 4x NAND Gate Block Layout with Pcells and Routing

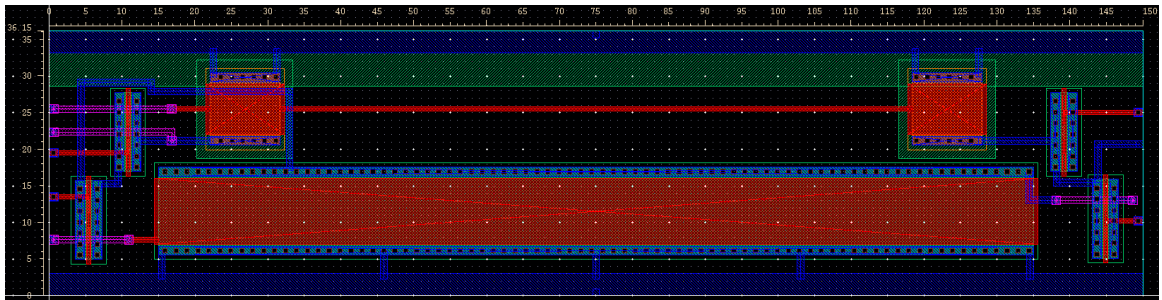


Figure 5.28: MCML 4x NAND Gate Full Routed Layout with All Layers & Pathing

As mentioned, the increased widths of all devices cause a denser final layout for the 4x NAND gate. As a result metal2 had to be used in more places to fit the same sizing dimensions. Clever positioning and via use allows for these layers to be kept minimal in size and maintain circuit functionality. As with the wider layouts developed so far, the wider transistors contain more power connections to achieve uniform current flow through the device and maintain the performance of the schematic level simulations.

5.6 – MCML Layout Remarks

The MCML layouts developed for the 1x, 2x, and 4x versions of the Inverter and NAND gate all meet the sizing criteria for standard cells. The height matches with other standard

cells that utilize the ami06 technology and are fixed at $36.15\mu\text{m}$. The width is variable and kept uniform between the inverters and NAND gates with the exception of the 1x sizing. Each cell is compatible with existing standard cells and fits onto a grid where those height guidelines are utilized. The layouts only need the metal1 layer for pathing with exception of small routes on the 2x and 4x NAND gates. This keeps the design simple and minimized difficulty in fabrication of these devices. Though optimization is not achievable with hand layouts, symmetry and wire impedance were kept in mind throughout the development of the standard cells to create layouts that perform like their schematic counterparts. With the completed layouts, the next step involves overwriting existing standard cell layouts such that the inverter and NAND gate cells utilize the MCML logic family without having to redesign, reprogram, and rescript, an entire library from the ground up. With the standard cell layouts of MCML complete, the remaining tests speak towards the low noise and high frequency performance capabilities of MCML and compare the results to equivalent CMOS devices.

Additional information regarding more tutorial based information to supplement existing works and the creation and settings of configuration files that helped develop the MCML standard cell layouts can be found in Appendix A. Additional information includes ensuring Pcell linking matches with the schematic device selection and the small schematic modifications necessary to allow for rapid transition from schematic to layout. Since the compilation tools for using standard cells only require the size of the cell and I/O positions, conformity to the library is ensured and the fixed height guarantees that the cells can snap onto the grid created when developing a large system with standard cells.

5.7 – MCML Ring Oscillator

This section examines MCML's suitability for quiet, fast digital circuitry. A ring oscillator is a common bench marker for this test as one can derive speed and noise performance from the oscillator operation. The circuits developed for this test utilize the 1x drive MCML inverter developed and detailed in Section 5.2. Before developing a large scale ring oscillator, tests are performed to ensure the validity of prior conclusions and examine a couple gates connected together.

5.7.1 – MCML Gate Chaining Proof of Concept

To verify the principles discussed throughout the chapter and ensure that the MCML gates can work together, a test of three 1x inverters in series is performed to verify consistent operation across each gate. The same current mirror control circuitry is used here and since each inverter will be of identical size, the biasing circuitry controls every tail current source in the inverter chain. The associated schematic for the three inverter chain is shown in Figure 5.29. Note that each inverter utilizes the same biasing network and parameterized variables. Each inverter is configured to be the same size during simulation and the 1x inverter sizings and biases are used.

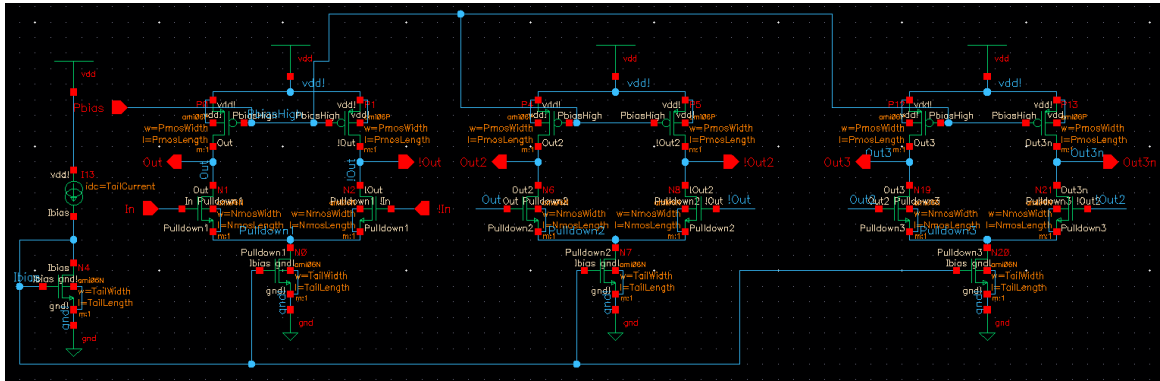


Figure 5.29: Three 1x MCML Inverters in Series for Concept Testing

This inverter network operates in series with no feedback into the first inverter to examine how the gates perform and swing when connected together. The desired voltage swings for these inverters should range from the 3.3V Vdd rail to around 1.8V, or the minimum value of the input. To show proof of concept before creating a ring, voltages at each stage should remain stable and within the established swing boundaries. Figure 5.30 shows the 3 stage open-ended inverter chain simulation. Note that since the gain is greater than one, the signal squares off as stages progress but values remain within the designed minimum bound.

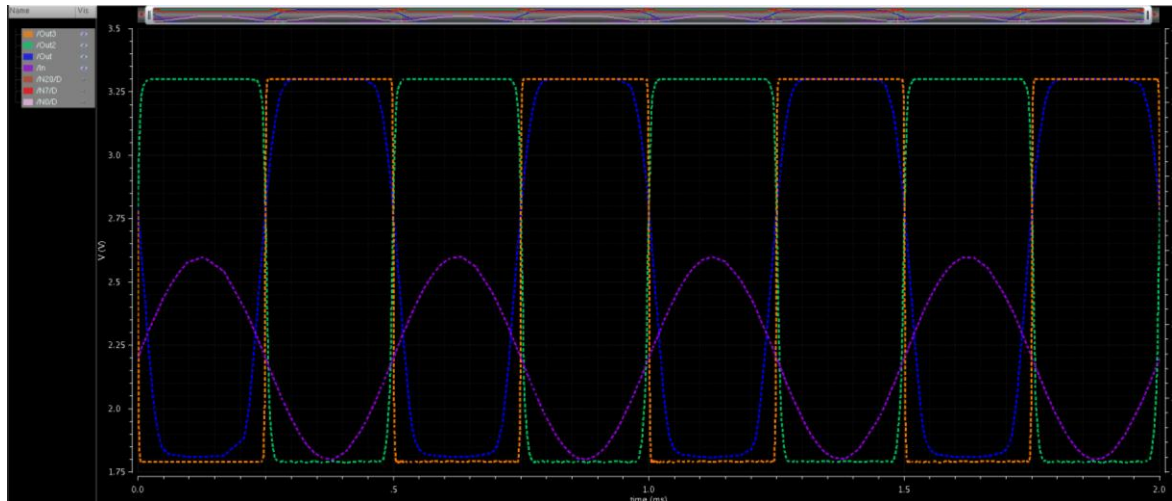


Figure 5.30: Open-Ended Three 1x MCML Inverter Chain Logical Test Results

The output of each stage ranges from 3.3V to 1.8V as desired and remains steady over multiple periods of operation. For examination of low noise operation, the stage currents are explored in Figure 5.31.

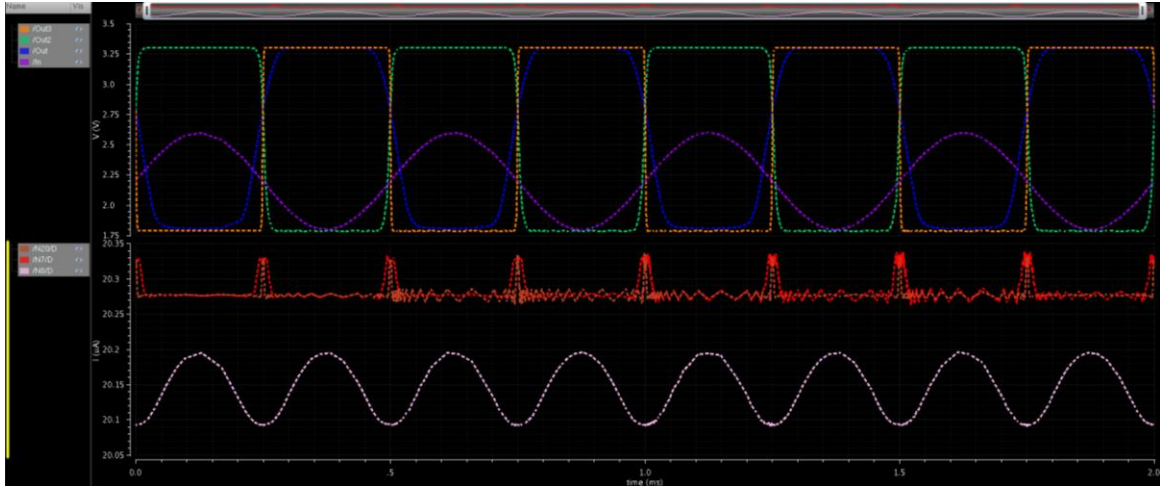


Figure 5.31: Open-Ended Three 1x MCML Inverter Chain with Current Data

The first stage current performs identically to that of the standalone inverter examined previously where the current follows the inverse of the input. Both the second and third stages increase nominally in current by about 200nA and remain steady within 20-25nA with 50nA spikes during switches between stages. The first stage current is still affected by the ideal nature of simulated rails. The second and third stages show realistic results where their inputs are provided from another gate. The fact that both stages keep virtually identical currents with one another and remain constant within 10s of nA demonstrates the noiseless nature of MCML and serves as proof of concept for developing a larger chain of gates and shows the compatibility of the MCML cells designed.

With successful results in an open ended chain of MCML inverters, the next proof of concept stage involves placing the same three inverters in a loop that act as a mini ring oscillator and examining the stability of the device operation. Symbols are created for the inverter for ease of development and clean schematic design. The same current biasing system used for each gate so far, is used for the oscillator as well. Initial conditions are also set to the current max and min of the inputs used so far. Figure 5.32 shows the

symbolic three stage ring oscillator with initial condition settings visible on the starting node. Note that the initial conditions are set to the minimum (1.8V) and maximum (2.6V) inputs used in single gate simulation.

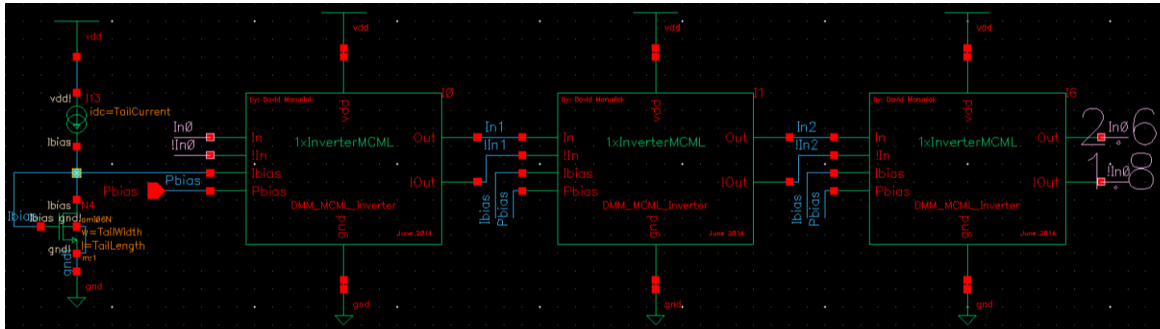


Figure 5.32: Three Stage MCML Ring Oscillator Concept Test

The same sizing parameters used for the 1x inverter are used for each stage of the ring oscillator test. The simulated results for the small ring oscillator aim to show perpetual stability in operation where steady state operation remains in control and constant in voltage swing. Figure 5.33 shows the simulated results for the small oscillator. Note that the voltage swing is only 0.8V due to the short nature of the chain. The system does not have enough time to swing fully with a three gate chain. However, the stability between each stage during operation shows that a ring oscillator is feasible with the inverters developed. The fact that each stage remains consistent in period and voltage swing demonstrates that a larger ring oscillator with many stages can be designed with the same MCML inverters and perform with stability.

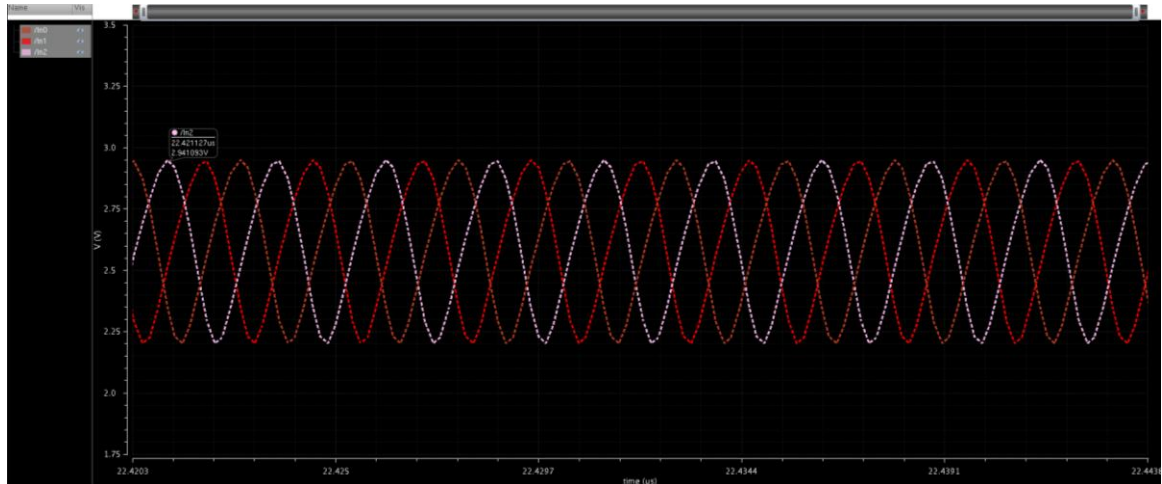


Figure 5.33: Three Stage MCML Ring Oscillator Output Results

The 0.8V swing compared to the observed 1.5V swing so far is a non-issue. The short chain limits the time the inverter has to fully swing and for a longer chain the expectation is that of a more square result with a higher voltage swing, comparable to that of the single gate simulations. The stability in a connected loop though shows proof of concept for the ring oscillator using the MCML inverters designed here.

5.7.2 – MCML Ring Oscillator Design and Test

With chained operation verified and tested, a ring oscillator with many stages can be developed with the expectation of steady state stability. The goal of the ring oscillator is to examine speed capabilities of MCML and compare with CMOS results of a fabricated ring oscillator in the SCMOSC5 technology. Mosis created a 31 stage ring oscillator using CMOS inverters and tested the fabricated result. Since the dimensions of these gates are unknown and the MCML oscillator remains in a simulation state, the comparison is not ideal but allows for light to be shed on the effectiveness of the MCML design. For better matching with the Mosis CMOS results, the ring oscillator designed in

MCML is also kept to 31 stages. Figure 5.34 shows the block diagram view of the ring oscillator.

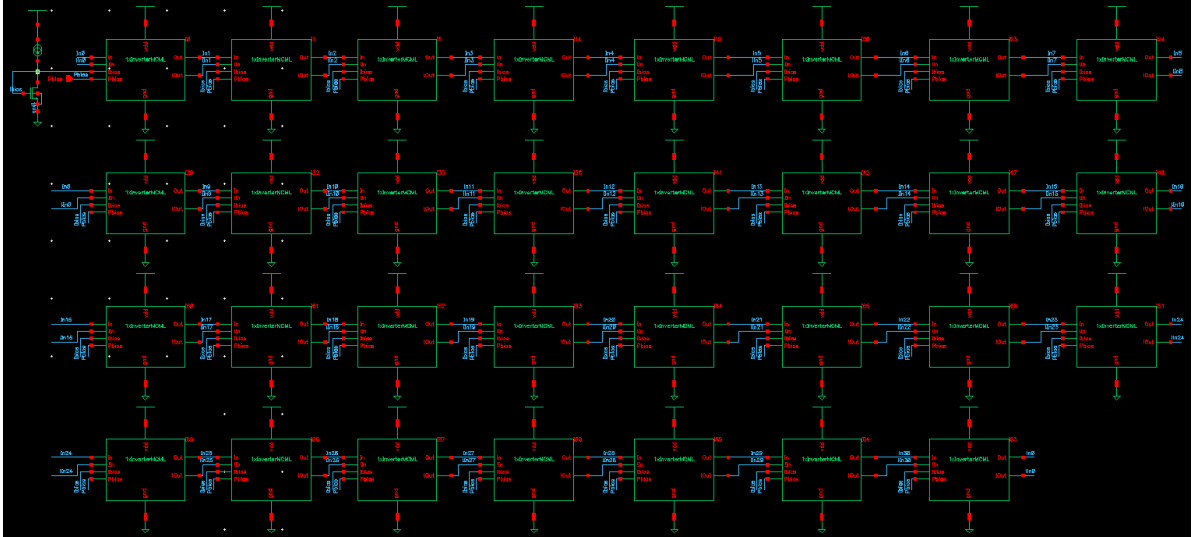


Figure 5.34: 31-Stage Ring Oscillator Block Diagram Schematic

A symbol for the MCML inverter was created to allow for stamping down of inverters for the 31-stage oscillator, allowing for more rapid design. The nodes for each block are labeled such that wires don't need to be drawn across the length of the schematic. Initial conditions are set at the origin node identical to the three stage oscillator test. Figure 5.35 shows results inverted and non-inverted output during steady state oscillation of the 31-stage device. The labels on each plot indicated the time at which the peaks take place. Taking a full period of the signal allows for calculation of the operational frequency of the oscillator where the frequency is determined by how long it takes the oscillator to return to its original value.

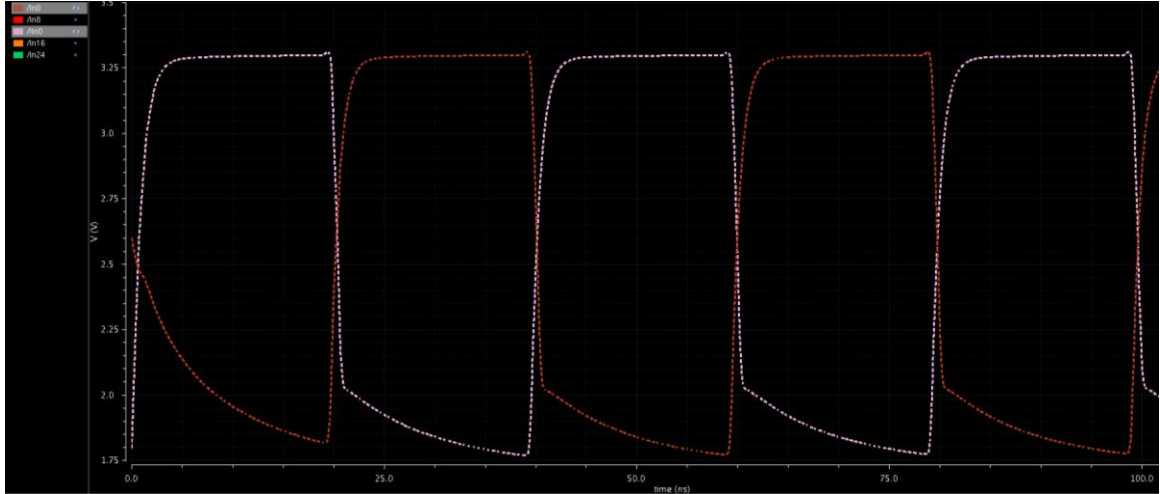


Figure 5.35: Steady State Peaks of the 31-Stage Ring Oscillator

Since the ring oscillator contains an odd number of gates, one cycle of signal through the system inverts the input and the next cycle returns it to its original value. This means that for a 31-stage ring oscillator, the signal must travel through 62 gates to return to its original value and complete one period of operation. The sizings for each inverter follow the 1x parameters detailed in Table 5.1. The peaks occur every 39.5ns which over the course of 62 gates, indicates that it takes about 0.637ns per gate for the signal to transfer. This yields a total operating speed of 1.57GHz. The Mosis simulation for the CMOS 31-stage oscillator yielded an operation at 150.35MHz. This shows the MCML device operating approximately 10 times faster than its CMOS counterpart. This speaks favorably to the high speed and high frequency capabilities of MCML devices over CMOS even despite the fact that the MCML remains in simulated state compared to the fabricated CMOS device. The constant current trends that were explored on the 3-stage proof of concept example hold for the 31-stage ring oscillator as well which show the quiet operation of MCML devices. These results speak towards two of the major advantages of MCML over CMOS in select applications. The results from the chapter are

summarized in Section 5.8 to place all relevant data in the same place with a more readable format.

5.8 – MCML Results Summary

This section tabulates the numerical results from Chapter 5 into a readable table for overall viewing ease and analysis. Results include comparisons in performance between the inverters and NAND gates for all three sizes developed over the course of the thesis. Table 5.7 and Table 5.8 show the MCML inverter and MCML NAND gate results ranging from power consumption and output swing voltages.

Table 5.7: Inverter Results Summary Detailing Simulated Operation

Gate Size	V_{out_min}	V_{out_max}	V_{swing}	Current	Power
1x Inv	1.80 V	3.3 V	1.50 V	20.13 μ A	66.43 μ W
2x Inv	1.78 V	3.3 V	1.52 V	40.34 μ A	133.12 μ W
4x Inv	1.80 V	3.3 V	1.50 V	80.76 μ A	266.51 μ W

Table 5.8: NAND Gate Results Summary Detailing Simulated Operation

Gate Size	V_{out_min}	V_{out_max}	V_{swing}	Current	Power
1x NAND	1.78 V	3.3 V	1.52 V	20.15 μ A	66.50 μ W
2x NAND	1.79 V	3.3 V	1.51 V	40.38 μ A	133.25 μ W
4x NAND	1.82 V	3.3 V	1.48 V	80.84 μ A	266.77 μ W

These results summarize the consistent operation detailed in the previous sections of the thesis. The swings of each device and device size remain consistent and the current doubling of each device size is nearly perfect. The consistency in swings between the inverter and NAND gate show the compatibility between the cells in the library and the ability for all gates to function together and provide necessary signals to each other regardless of sizing requirements. Further testing shows MCML's constant power dissipation at any frequency. Frequency is swept in decades across six orders of magnitude and the current and power dissipation of the devices are compared. Table 5.9

shows the results of the frequency sweep for the inverter and Table 5.10 shows the results of the same frequency sweep range for the MCML NAND gate. The tests for both gates take place using the 1x baseline sizings of each device.

Table 5.9: Frequency Sweep for the 1x MCML Inverter

Frequency	Current	Power	V_{swing}
1 kHz	20.13 μ A	66.43 μ W	1.55 V
10 kHz	20.13 μ A	66.43 μ W	1.55 V
100 kHz	20.13 μ A	66.43 μ W	1.55 V
1 MHz	20.13 μ A	66.43 μ W	1.55 V
10 MHz	20.12 μ A	66.40 μ W	1.58 V
100 MHz	20.13 μ A	66.43 μ W	1.70 V*
1 GHz	20.28 μ A	66.92 μ W	0.37 V*

Table 5.10: Frequency Sweep for the 1x MCML NAND Gate

Frequency	Current	Power	V_{swing}
1 kHz	20.14 μ A	66.46 μ W	1.52 V
10 kHz	20.14 μ A	66.46 μ W	1.52 V
100 kHz	20.14 μ A	66.46 μ W	1.52 V
1 MHz	20.14 μ A	66.46 μ W	1.52 V
10 MHz	20.15 μ A	66.50 μ W	1.51 V
100 MHz	20.18 μ A	66.59 μ W	0.70 V*
1 GHz	20.57 μ A	67.88 μ W	0.10 V*

The ideal current sources and rails involved in the simulation of a single gate cause the irregular voltage swings on the high frequency operation. These data points are indicated with a ‘*’ to indicate this effect. However, even with that irregular operation and logical function degraded, the power still remains consistent due to constant current across six orders of magnitude in frequency and in realistic settings this trend only improves.

The results from Section 5.7 show the capability of MCML to perform at a higher frequency than equivalent CMOS circuitry using the same technology. Granted the results of fabrication versus simulation will allow for fluctuation but not nearly on an order of magnitude that would impact the conclusion from the results. The results show

that the major goals of MCML operation are achieved with quiet operation and constant power across all frequencies. This makes MCML well suited for mixed signal applications. The comparison in operation for high frequency devices shows MCML over CMOS in the ring oscillator test making MCML a better candidate for high frequency operation. Through the open ended inverter chain and ring oscillator designs, the noiseless nature of MCML is validated for these designs through the constant current operation between gates and at any switching frequency. The standard cell layouts meet requirements for the standard cell library and meet compatibility needs with the development tools. This accomplishes the goals outlined for the thesis and provides the necessary fundamental knowledge to build upon or utilize MCML or the library for a designer's needs.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 – Summary

MOS Current Mode Logic provides an alternative to conventional CMOS technology for mixed signal applications due to the quiet nature of the circuit's operation. The results show that the MCML inverters and NAND gates of current variations 1x, 2x, and 4x perform as expected from MCML theory and the constant power dissipation across all frequencies demonstrates the quiet nature of MCML and the standard cells developed. During connected gate simulations, current remains constant within ~20nA during steady state switching operation which demonstrates the low noise nature of the MCML cells designed for this thesis. A 31-stage ring oscillator test demonstrates the superior frequency performance MCML provides over an equivalent CMOS system. The layouts developed over the course of the thesis meet the fixed height requirements of standard cells and as a result, are compatible with existing libraries that use a grid size of 36.15 μ m. The standard cells developed and the MCML theory discussed throughout the thesis create a fundamental knowledge platform that allows for dozens of theses or projects to build upon or extend the work developed here. The end result is to make MCML more accessible for designers and engineers, specifically in an academic environment. For ease of use, the topics covered are broad and the results are well explored to ensure that subsequent work can rapidly recreate results or utilize the fundamentals to explore their own applications in MCML.

6.2 – Future Work

Chapter 4 discussed optimization techniques in general and the thoughts associated with each type of optimization technique. Projects and theses can explore any one of these techniques with relation to MCML or the standard cell library and perform development on creating specifically optimized cells. Another option involves developing optimization using multiple techniques at a shallower level and creating a metric of performance for the cells and determining which technique proves most effective in overall device performance.

Layout work has been pursued throughout the course of this thesis and with tight deadlines and I/O Pad issues; final tape out files could not be submitted. However, layouts for both 7rf technology and SCMOS technology have been developed and are in a position where either technology could be utilized for tape out. The 7rf technology layouts include an MCML and CMOS inverter built from minimally sized components and further development of those layouts and chip tape out can provide a useful comparison between MCML and CMOS devices. The SCMOS layouts developed are for the standard cell library and as a result are constrained on net dimensions and rail sizing. However, the layouts could be simulated, modified, and taped out for analysis for these cells as well. Other cells could also be developed and taped out using similar processes and the layouts from this thesis as a reference along the way.

The MCML current oscillations are presented and discussed in Chapter 6. Control circuitry additions or replacements could provide a smoother signal control with robustness to limit both the signal noise on the lower current devices and the steady current oscillation present in the simulations. Musicer presents some forms of control

circuitry that assist in controlling the MCML gate biases for smoother operation [3]. Control circuitry ensures accurate voltages for each desired signal and aids in controlling the overall voltage swing the gate would have as well. Control circuitry for the tail biasing current and potential Pbias control was also experimented with. Both avenues can be expanded upon and optimized depending on the application. Control circuitry can be developed and simulated for comparative results with the non-controlled MCML devices and various control methods analyzed to see which is most important for consistent operation.

Standard cell libraries are effectively limitless in the iterations that can be developed on a cell to cell basis and the sheer number of cells in a library can always be increased as well. More variations of current capabilities for existing cells and other logical cells should be developed in MCML to provide a more comprehensive library. Though the nature of MCML allows for the cells developed to provide for any logical function to be implemented, other standard devices should be included as well ranging from flip-flops to Muxes etc. Following the procedures and development techniques highlighted throughout this thesis should hopefully reduce the development time required for a standard cell and allow the library to continue to grow over the course of further projects or theses.

Chapter 3 discusses a couple applications of MCML for modern technical systems. This research aims to shed more light on the potential avenues where MCML device can thrive over CMOS or other logic family counterparts. That chapter shows the reader potential research applications involving MCML in developing a more specialized system for a narrow purpose. The MCML theory provided throughout this thesis serves

as a knowledge baseline where applications and research can build upon these ideas and explore specific uses suited for MCML devices.

Any of these future work possibilities can serve as a suitable project or thesis for development. Creating the base that allows for further research in any direction allows this thesis to serve as inspiration for ideas that build upon this foundation. Other ideas and specialized applications are available to MCML as well, but this highlights a few that are directly related to the work developed in this thesis and build upon and expand the base of knowledge and library created. In that light, dozens of projects or theses could be developed from the ideas presented and should be pursued to further knowledge in MCML and continue to shed light on possible applications of conventional approaches, including CMOS devices.

WORK CITED

- [1] B. Davari. *et al.* “CMOS Scaling for High Performance and Low Power – The Next Ten Years”. *Proceedings of the IEEE*, vol. 3, no. 4, pp. 595-606, April 1995.
- [2] M. Mizuno *et al.*, “A GHz MOS Adaptive Pipeline Technique Using MOS Current-Mode Logic”. *IEEE Journal of Solid-State Circuits*, vol. 31, no. 6, pp. 784-791, June 1996.
- [3] J. Musicer, “An Analysis of MOS Current Mode Logic for Low Power and High Performance Digital Logic,” M.S. thesis, Department of Electrical Engineering and Computer Science, U.C. Berkeley, Berkeley, CA, 2000.
- [4] A. Cevrero *et al.*, “Power-Gated MOS Current Mode Logic (PG-MCML): a Power Aware DPS-Resistant Standard Cell Library”. *DAC 2011*, pp. 1014-1019, June 5-10 2011.
- [5] F. Cannillo, C. Toumazou, T. Sverre Lande. “Nanopower Subthreshold MCML in Submicrometer CMOS Technology”. *IEEE Transactions on Circuits and Systems*, vol. 56, no. 8, pp. 1598-1611, August 2009.
- [6] M. Yamashina *et al.*, “A Low-Supply Voltage GHz MOS Integrated Circuit for Mobile Computing Systems”. *IEEE Symposium on Low Power Electronics*, pp.80-81, 1994.
- [7] J. Walker. *et al.* “Optimizing electronic standard cell libraries for variability tolerance through the nano-CMOS grid”. *Philosophical Transactions of the Royal Society*, pp. 3967-3981, 2010.
- [8] A. Martinez, S. Dholakia, S. Bush, “Compilation of Standard-Cell Libraries,” *IEEE Journal of Solid-State Circuits*, vol. sc-22, no. 2, pp. 190-197, April 1987.
- [9] Wayne State University, “Layout Using Virtuoso Layout XL” *ECE Department, Wayne State University*, 2010, Lab 3.
- [10] T. Smilkstein, EE 431 Course Materials, Fall 2012.

APPENDIX A

STANDARD CELL LIBRARY CADENCE SCHEMATICS & LAYOUTS

This Appendix details additional development details for the cells developed for the standard cell library. It includes additional steps and guidance for simulation and layout creation in Cadence and can serve as a general reference for the related Cadence modules as well. Some additional windows and steps not explicitly detailed are highlighted here to again create the base of knowledge necessary for development and provide users with a helpful reference that minimizes the amount of outside research need be required to perform similar development.

Additional simulation results and explicit parameterized setup for the simulations performed over the course of this thesis are also present to allow for results to be more readily recreated.

MCML Inverter Schematic Development

For simulation in ADE, the simulation must be set to Spectre. By default, this should be the configuration and not require any changes. The model files utilized for the NMOS and PMOS transistors in SCMOS C5 include the Spectre model files for the ami06 library. Setting the file paths for the model files occurs under the Setup>Model Files window. There may be files presently on the list that are used for 7rf technology and can be deleted. Including the paths from the ami06 library yields a final result as shown in Figure A.1.

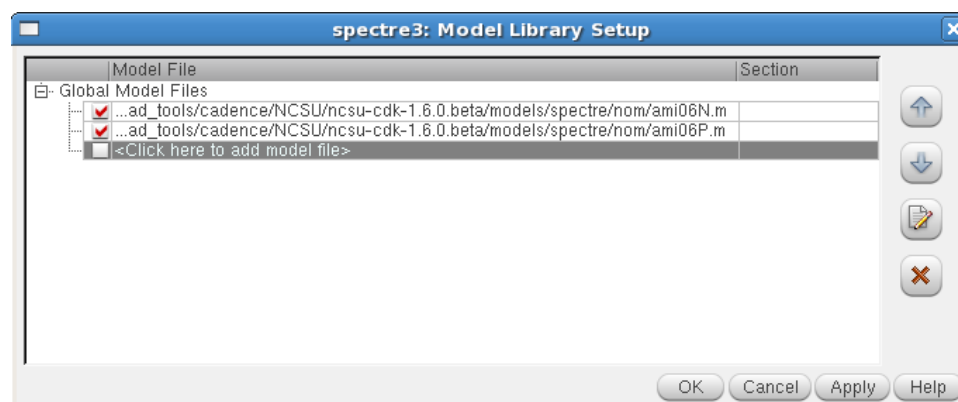


Figure A.1: Model File Configuration Window for SCMOS ami06 Transistors

Next, the stimuli for simulation should be configured to set up the circuit for operation. The stimuli window can be selected through setup or the icon can be clicked to bring up the stimuli. Each input or output pin created on the schematic appear as default disabled on the stimuli list and are configured individually and independent of one another. Select the desired pin to configure and use the on screen parameters to set up a signal. Note that the analogLib library must be included in the overall library hierarchy to have parameter values appear in the stimuli window. The configuration windows for each of the MCML inverter stimuli pins are shown in Figure A.2 and Figure A.3. Note that vdd appears under the ‘Global Sources’ radio button and all other input pins appear under the ‘Inputs’ radio button.

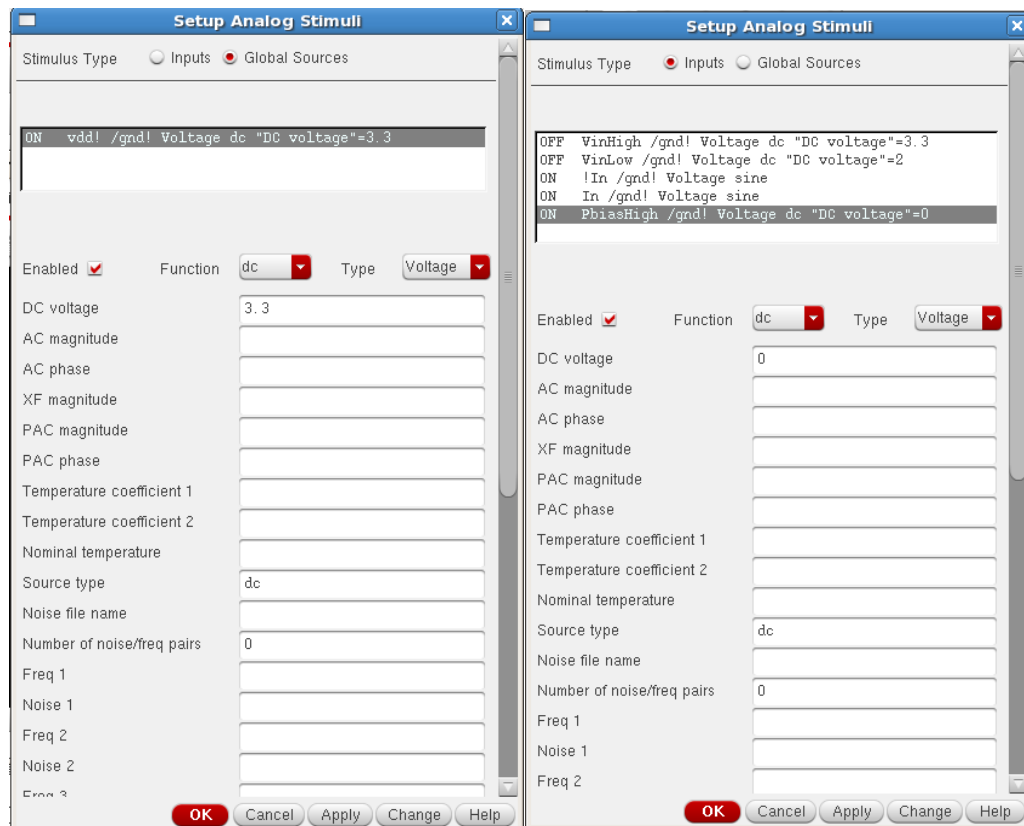


Figure A.2: Stimuli Windows for vdd and Pbias – Both Use DC Voltage Functions

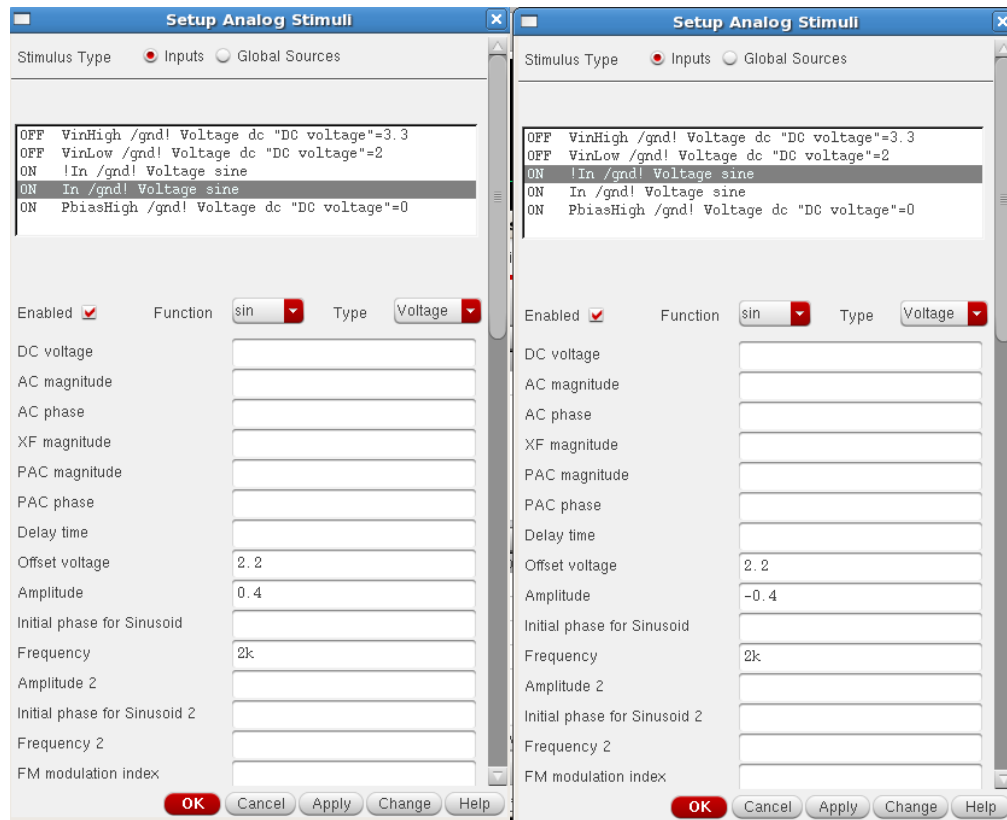



Figure A.3: Stimuli Windows for the In & !In Signals – The Signals are Complements

Once the stimuli are configured, the simulation conditions must be set. The  button at the top of the right hand side of the window allows for the final simulation setup step before running. For the inverter simulations, a transient simulation is used and set to 1ms to allow for a handful of switching operations to occur. The simulation could be run at this stage but Cadence needs to know what signals to plot so selecting the Outputs>To Be Plotted>Select on Schematic option brings the schematic to the front window where any mouse clicks will select a node for probing. Clicking on wires or the I/O pins selects a node for voltage probing. Selecting a terminal of a device (drain of a transistor) or the device itself will allow current measurement for current flowing into the selected terminal or current measurement for each terminal on the device respectively. Nodes and terminals that are selected become illuminated on the schematic to verify correct selection. A

sample highlighted schematic for the inverter is shown in Figure A.4 with associated pin list also depicting highlights. Once all desired terminals are selected, pressing escape will toggle off the selection of nodes.

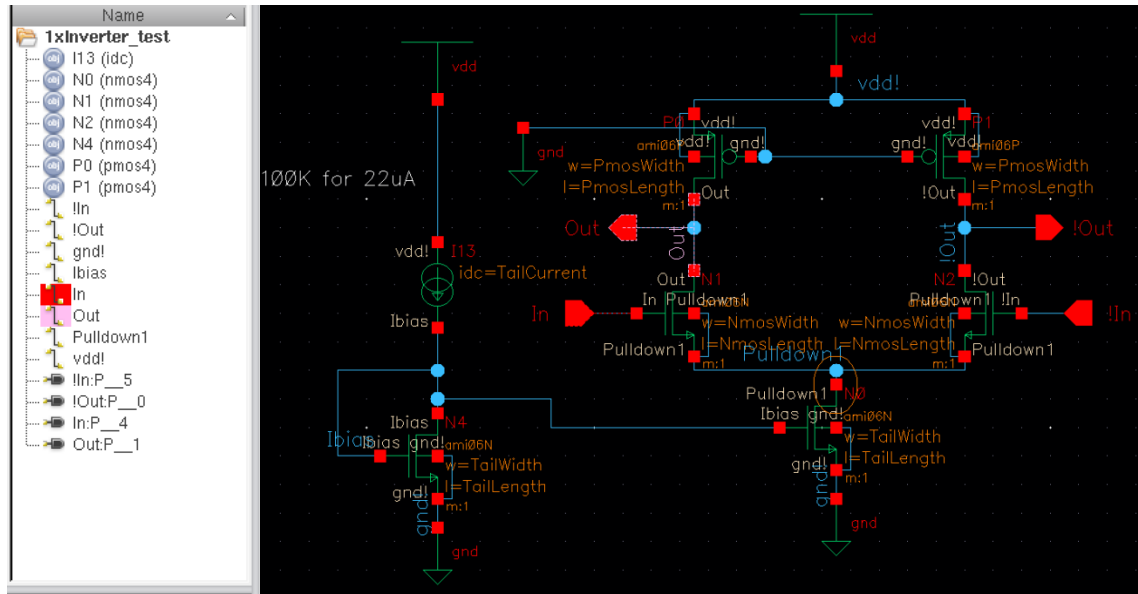



Figure A.4: Highlighted MCML Inverter Schematic with Highlighted Pin List

Now the simulation can be run by hitting the  button on either the ADE L window or the added toolbar in the schematic view. Figure A.5 shows the final setup window in ADE L before running the simulation.

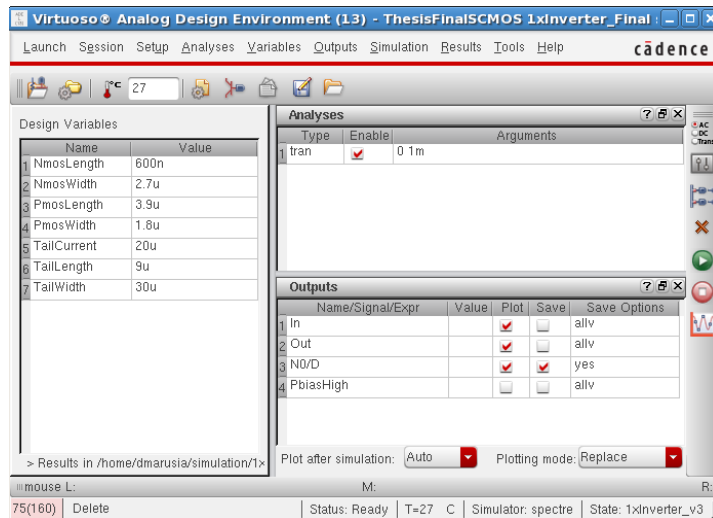


Figure A.5: Final ADE L Window Before Running Simulation

A successful simulation run opens up a plotting window automatically with all the signals selected in the plotting window. There is a color coded key on the right hand side of the simulation window that labels each signal and allows toggling on and off the view of a particular signal. Clicking the name of a signal bolds the signal in the graphical view and one can bold multiple signals using control or shift. Signals can be sent to separate plotting panes or windows through options present when the name of a signal is right clicked. Similar procedural steps are followed for setting up and developing the NAND gates as well.

Figure A.6 shows an additional operational graph of the 4x inverter with the inverted input and output signals present in the result as well. This shows the clean operation of the MCML inverter and the symmetric nature of the device ensures the mirrored performance on the positive and negative output results.

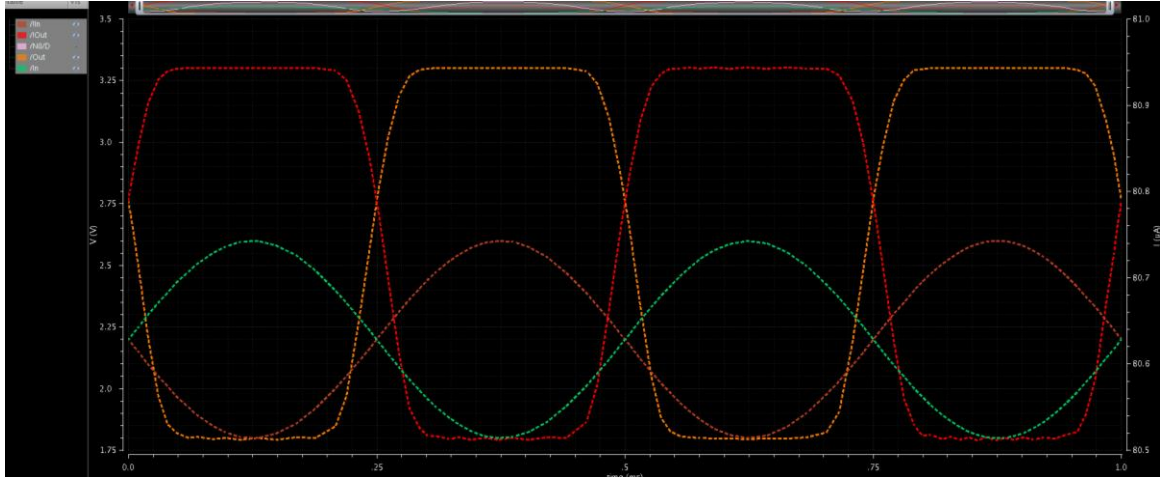


Figure A.6: 4x Inverter Full Logical Performance of Inverter/Buffer

MCML NAND Gate Schematic Development

The simulations for the NAND gate take place in ADE L like the inverters. The NAND gate is a two input device where both inputs must have their logical inverse provided to the system as well. Figure A.7, Figure A.8, and Figure A.9 show the stimuli windows for the NAND gate signals. The Vdd and Pbias signals are the same as for the inverter but the input logical signals are square wave pulses for clear logical operation.

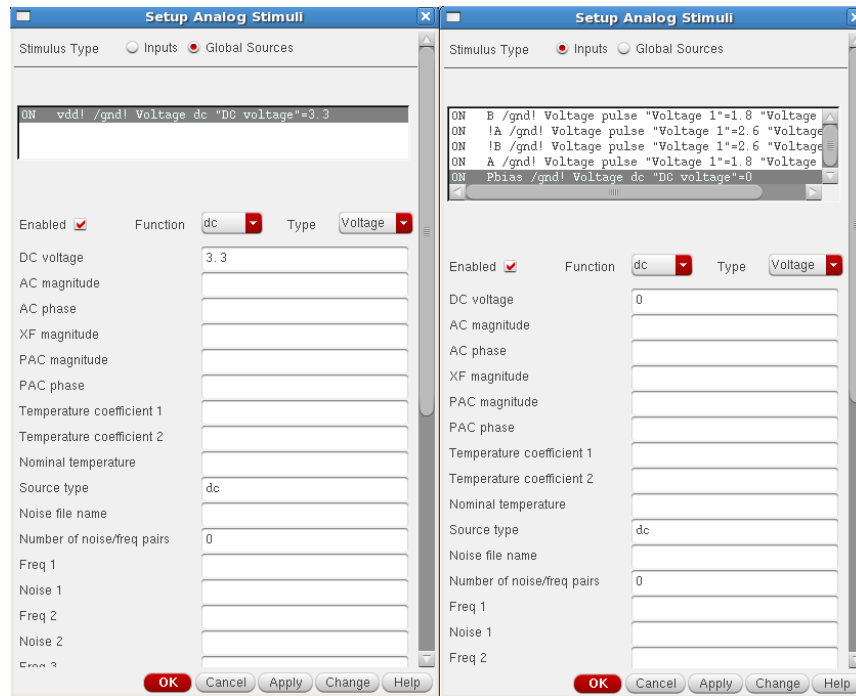


Figure A.7: Vdd and Pbias Stimuli Windows for the MCML NAND Gates

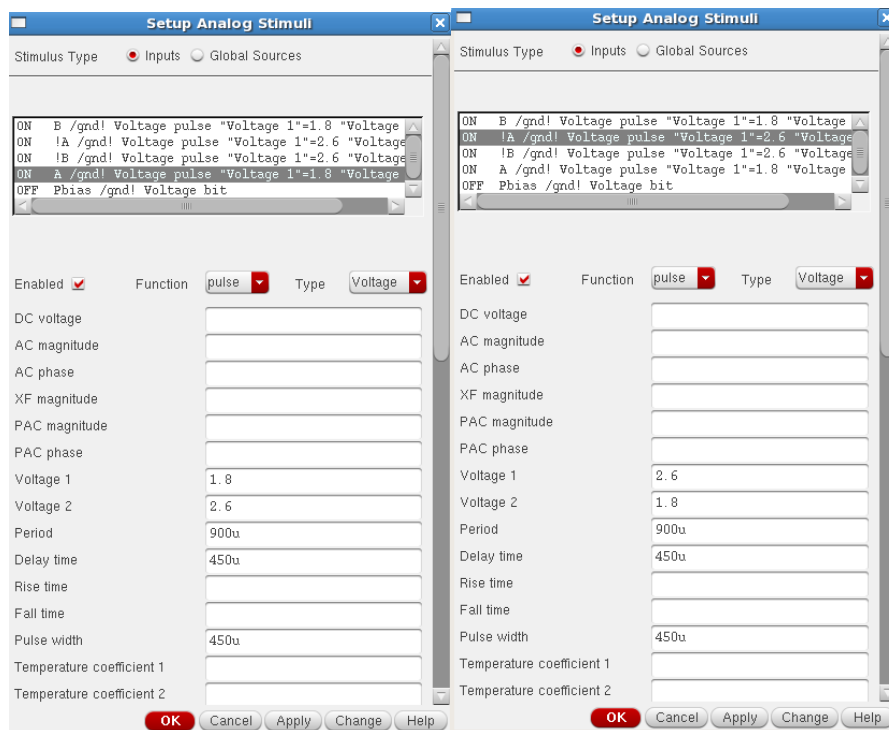


Figure A.8: A and !A Stimuli Windows for the MCML NAND Gates

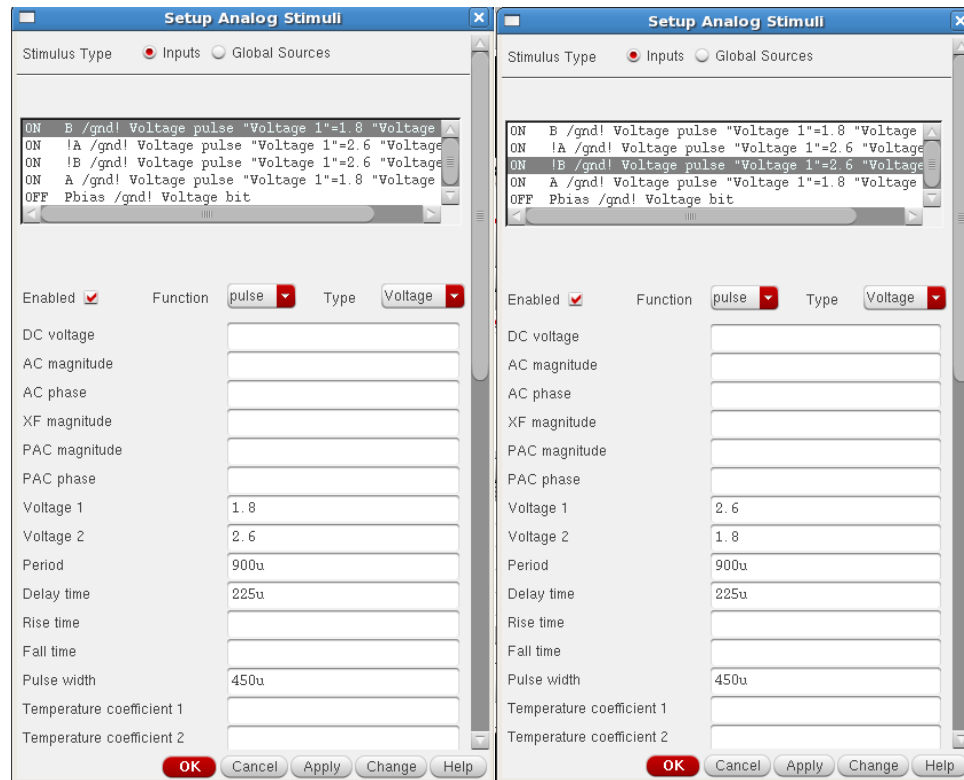


Figure A.9: B and !B Stimuli Windows for the MCML NAND Gates

The final parameter window is shown in Figure A.10. Note that populating the design variable list occurs by selecting Variables>Copy From Cellview. Values can then be entered in the value field for each variable before simulation.

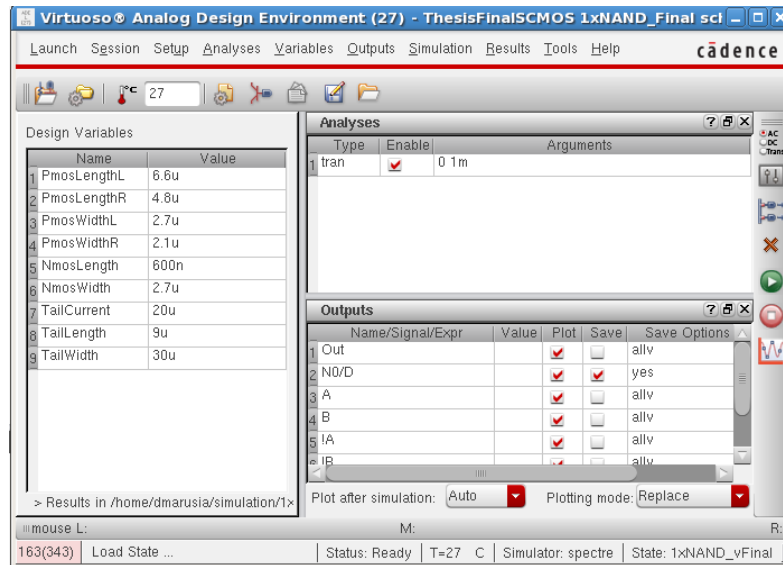


Figure A.10: Final Parameter Window in ADE L for NAND Gate Simulation

The fact that MCML devices output a signal and its inverse (and take in a signal and its inverse) allow for full logical performance to be examined. Figure A.11 shows a sample NAND gate that is highlighted for signal examination. The selected terminals and nodes provide full logical performance data.

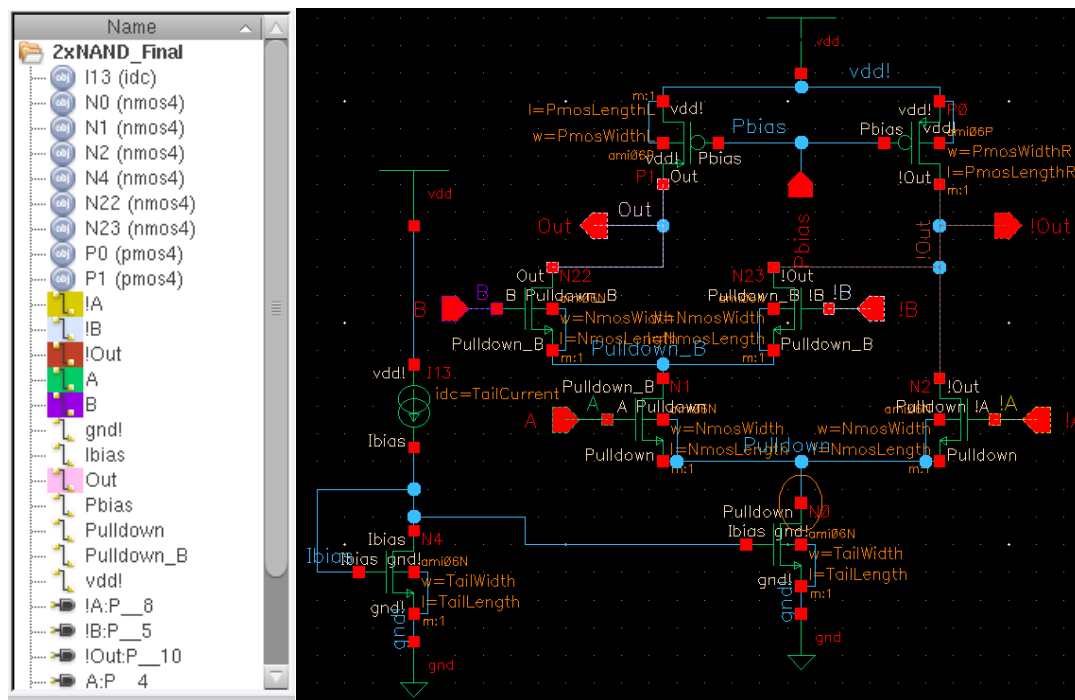


Figure A.11: Highlighted NAND Gate Schematic with Highlighted Pin List

Full logical performance ensures that the gates perform well for either output regardless of the focus of the gate (i.e. NAND operation). This allows for gates to be chained together where both outputs would be connected to the next gate in the sequence. In other words, the output and its inverse possess the same voltage swing and performance properties that allow interfacing between the MCML gates developed. Figure A.12, Figure A.13, and Figure A.14 show the full logical performance of the 1x, 2x, and 4x NAND gates respectively. Note that each gate possesses the same output swing for the same input signal. This again speaks to the potential for interfacing gates together and ensuring correct logical performance through a network of MCML devices.

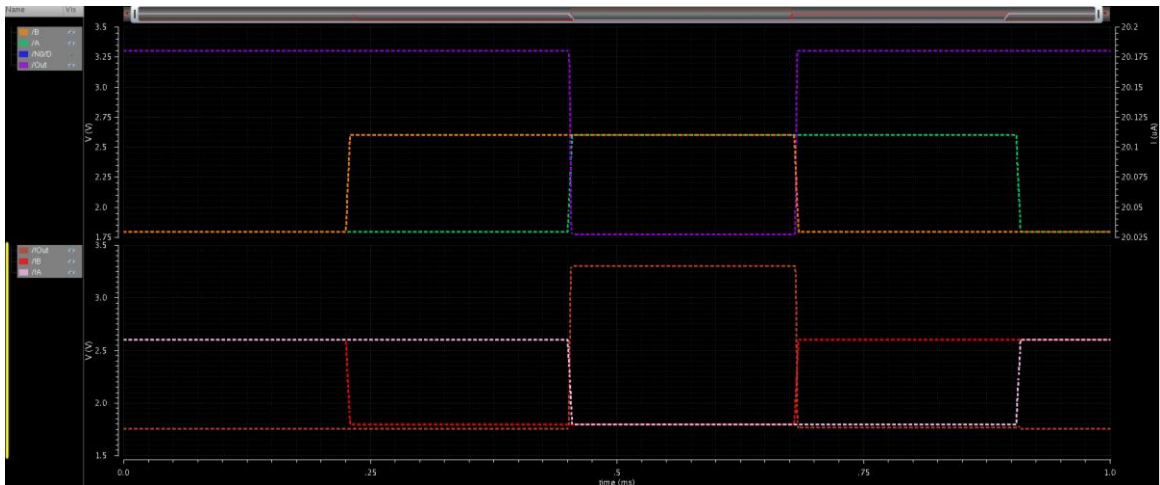


Figure A.12: 1x MCML NAND Gate Full Logical Performance

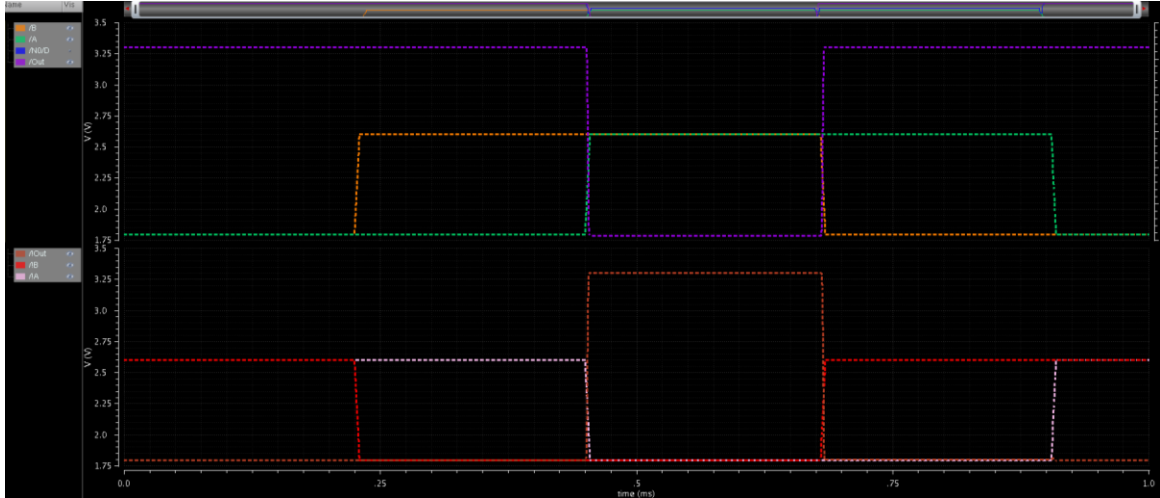


Figure A.13: 2x MCML NAND Gate Full Logical Performance

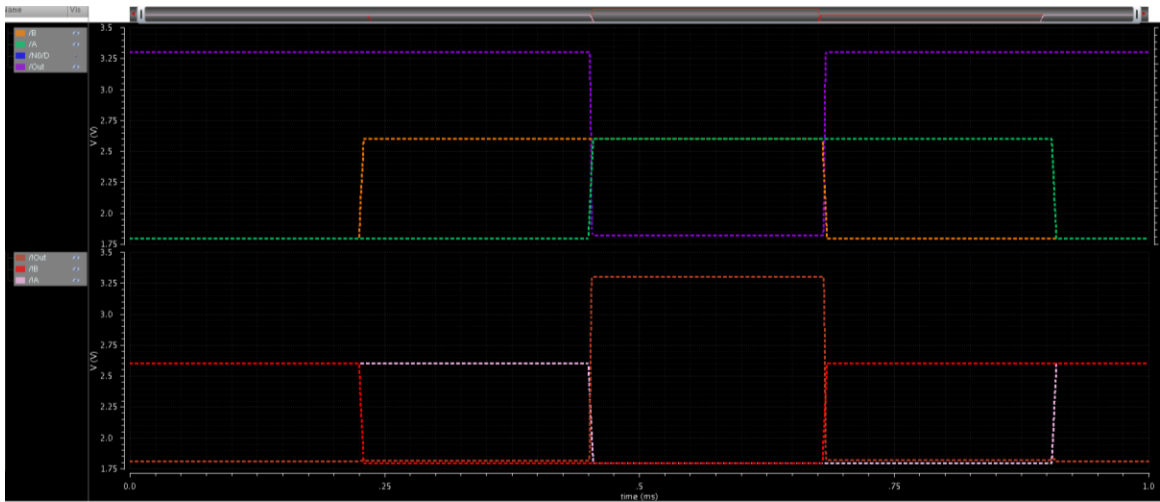


Figure A.14: 4x MCML NAND Gate Full Logical Performance

MCML Control Circuitry Development

Basic control circuitry for the MCML gates was also explored when trying to achieve correct simulations. The simple current biasing network detailed in Chapter 5 automatically biases the tail current transistor voltage based on the reference current. This allows for accurate current setting through the MCML gate. Since possible future work in the area includes exploration into control circuitry, these building blocks could prove useful for beginning study or development in the area.

The other control circuitry method explored and eventually disregarded was biasing circuitry for the Pbias signal. The idea was to make a network that would set the gate voltages for the PMOS load transistors automatically and could shift these resistances if an output voltage deviated above or below the desired value. The current biasing ended up providing enough stability for these not to be implemented and in fact, PMOS gate voltages were kept at 0V for the cells developed in this thesis. But precise systems or larger applications may benefit from biasing circuitry for the loads. Figure A.15 details the early attempt at a control circuit for the PMOS loads.

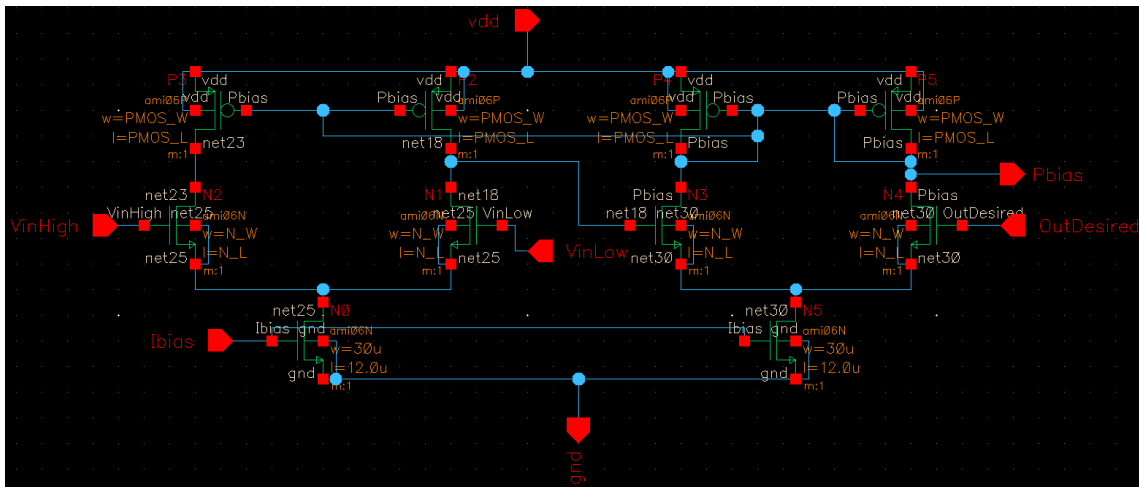


Figure A.15: Dual Differential Pbias Control Circuitry Attempt

This circuit was meant to contain two sets of differential paths that would shift based on the desired input voltages. The transistor dimensions of the control circuitry were meant to match that of the actual circuitry for better matching between the control and system. Here, the circuit is meant to lower the Pbias value if the voltage drop across the load transistors is too high and raise the bias voltage if the drop is not high enough. The biasing would be set on the control circuitry and output the biasing result to the rest of the system. Figure A.16 shows a symbolic representation of the control circuitry for schematic placement.

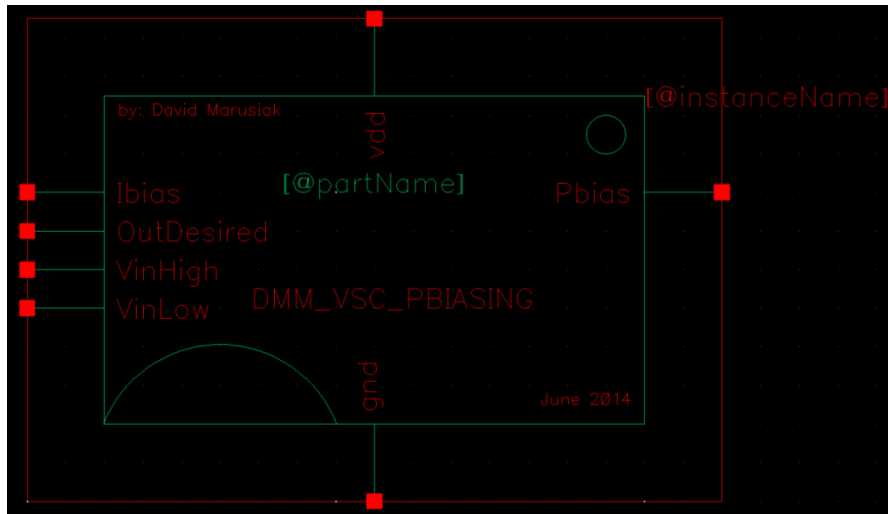


Figure A.16: MCML Pbias Control Circuitry Symbol

In practice, the control circuitry was setting the Pbias to around 1.1V which was close to the desired value at the time. However, once current control was implemented and the range of input signals was modified, there was no longer a need for the biasing circuitry for the load transistors. However, benefits can still be achieved for MCML devices through exploration of control circuitry. There are countless methods of implementing control circuitry to achieve desired signal control or stability. A characterization and comparison of different control circuitry types could serve as a useful project and reference for electronics development in this area.

MCML Layout Development

The layouts for the MCML standard cells involve taking the schematic form of the circuit and creating the layout for the device on the silicon level. For the standard cells developed in this thesis, the schematics needed modification before moving to layout to allow for the automated tools to perform some of the work. First, the biasing circuitry was removed and a pin was generated for the Ibias signal to come from external sources outside the cell. Next, the variable width names used for quickly making simulation

changes had to be set to defined numbers so that the Pcells know what dimensions to make the transistors. This results in a sample schematic like that shown in Figure A.17 for the MCML NAND gate. Note that the dimensions for the transistors are from the 1x NAND gate parameters in Table 5.4 but are explicitly defined on the schematic level.

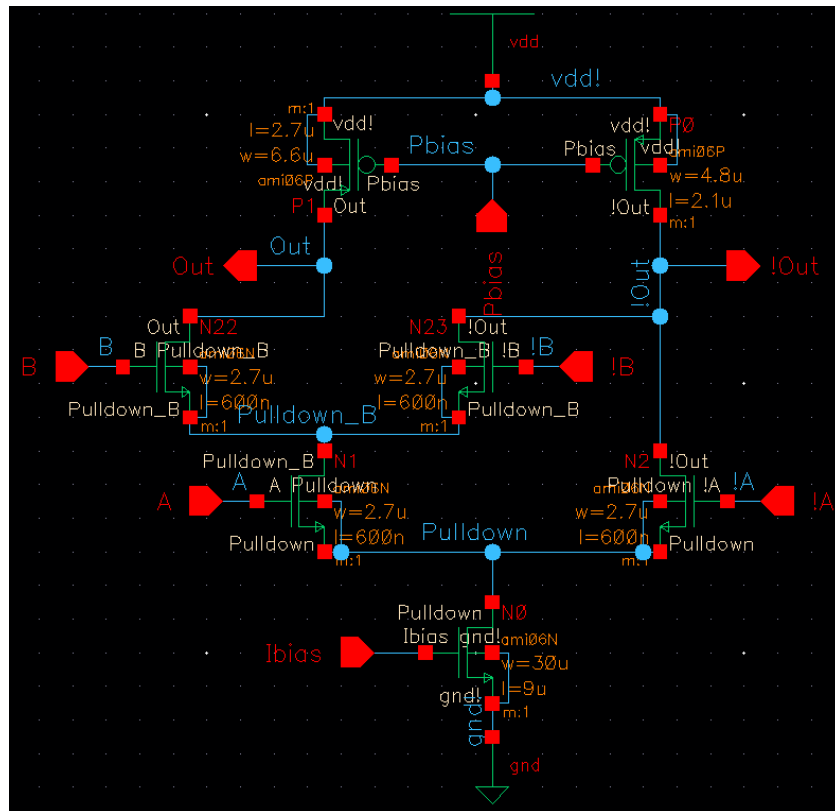


Figure A.17: 1x MCML NAND Gate Schematic Ready for Layout

When the Layout XL view is opened from schematic on a circuit with no existing layout, Cadence prompts for a physical configuration. Select ‘Create New’ for both options and advance through the naming window where default values should be fine. The resulting window comes up with blank regions for the transistor device fields. The ‘physical library’ should be selected in the blank field and the ‘nmos’ and ‘pmos’ views should be chosen for the ‘physical cell’ field. Since the layout is the only physical view available for those cells, the layout should automatically populate into the final field and the line

should turn blue indicating successful pairing between schematic symbol and physical layout. The resulting windows for a sample size of the MCML inverter and MCML NAND gate are shown in Figure A.18 and Figure A.19 respectively. Once the settings match the desired specifications, ensure to save the file and close it. Now the schematic is ready to be turned into a silicon level layout.

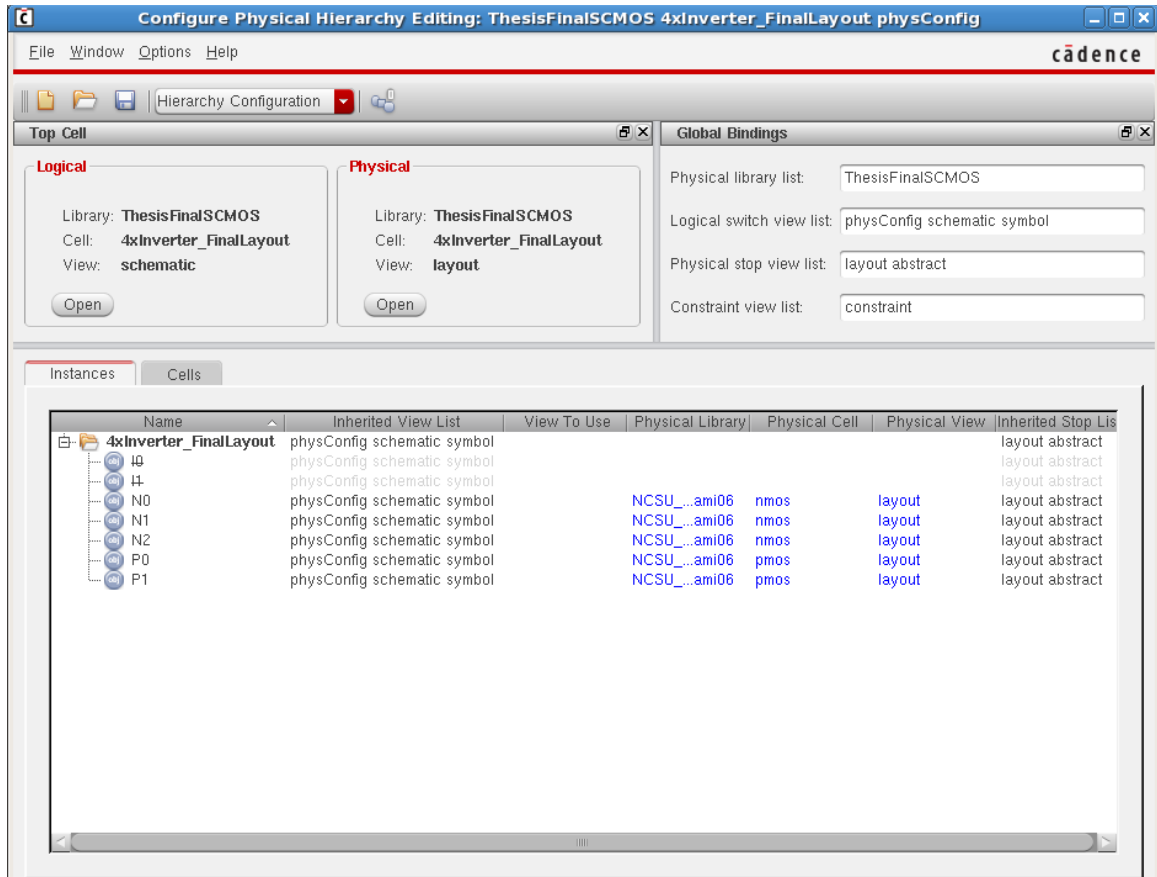


Figure A.18: Physical Configuration Window for an MCML Inverter

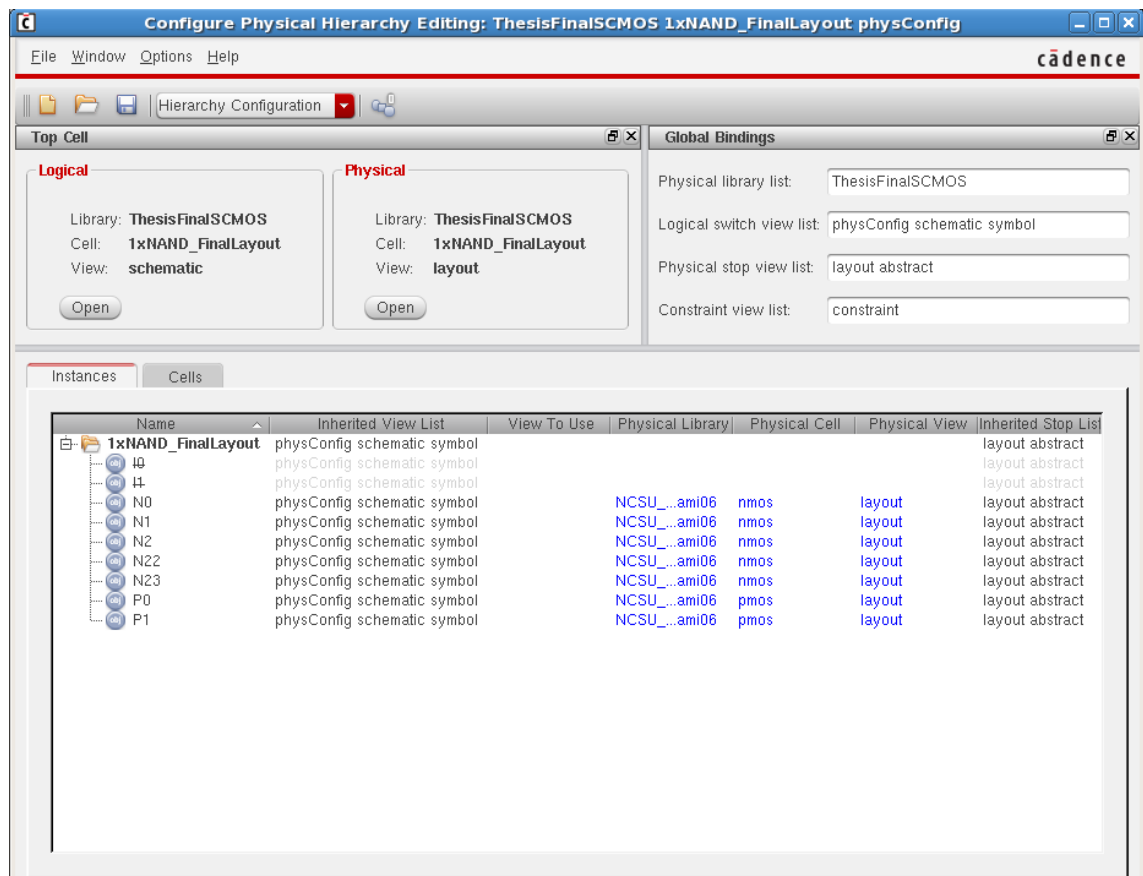


Figure A.19: Physical Configuration Window for an MCML NAND Gate

First the transistors must be generated on the layout side using Pcells. This is done by selecting Connectivity>Generate>All From Source. This places all I/O pins and transistors into the layout view window. The alternative method involves layout by hand using the same names of each transistor. The devices can be moved into the place and route (pr) boundary which act as the dimensional limits for a cell and close to intended regions by using Connectivity>Generate>Place as in Schematic. From there, manual modifications to placement can be performed to create the intended layout. The next step is routing everything together. Cadence has a built in auto route function but it usually cannot perform all the routing that needs to be done on the circuit. Major layers for power like V_{dd} and GND are created using the rectangle tool creation tool ('r') and selecting a


layer. Connections between I/O pins are done by using the path creation tool ('p') and selecting the desired connection layer. Paths can contain joints and right angle turns if desired. Clicking once after making a line path can allow for a change in direction and clicking a second time in the same location as the first (or double click) finishes the path creation. Some paths and connections cannot be made without vias however. Vias allow for transition between layers in the silicon. The vias are automatic layer combinations that perform this connection and can be accessed by using the via button  on the toolbar as shown in Figure A.20. Note that the toolbar section may need to be extended to show the button explicitly otherwise it can be accessed from the same toolbar section using the '>>' icon.



Figure A.20: Cadence Virtuoso Toolbar with 'Create Via' Button Indicated

To route devices together, perform the auto routing step by selecting Route>Automatically Route and then selecting 'Okay' on the subsequent window. The default options on that window should be fine for automatic routing. Finally, use the path tool and vias to finalize any pathing remaining on the circuit. Once everything is connected, the final checks can be made to ensure correct operation with the schematic by using layout versus schematic (LVS) or error checking to ensure fabrication validity using Design Rule Checking (DRC).

For standard cells, the layouts are then meant to be implemented into a library where the Cadence tools can automatically use the customized layout for use in large scale integration. This means that a single layout created for a gate can be used repeatedly

for a system containing any number of gates. To illustrate this point, the working 7rf technology library was used to generate a 16x16 multiplier which consists of a few hundred gates. Using standard cells, the final design can be generated quickly to create a product like what is shown in Figure A.21.

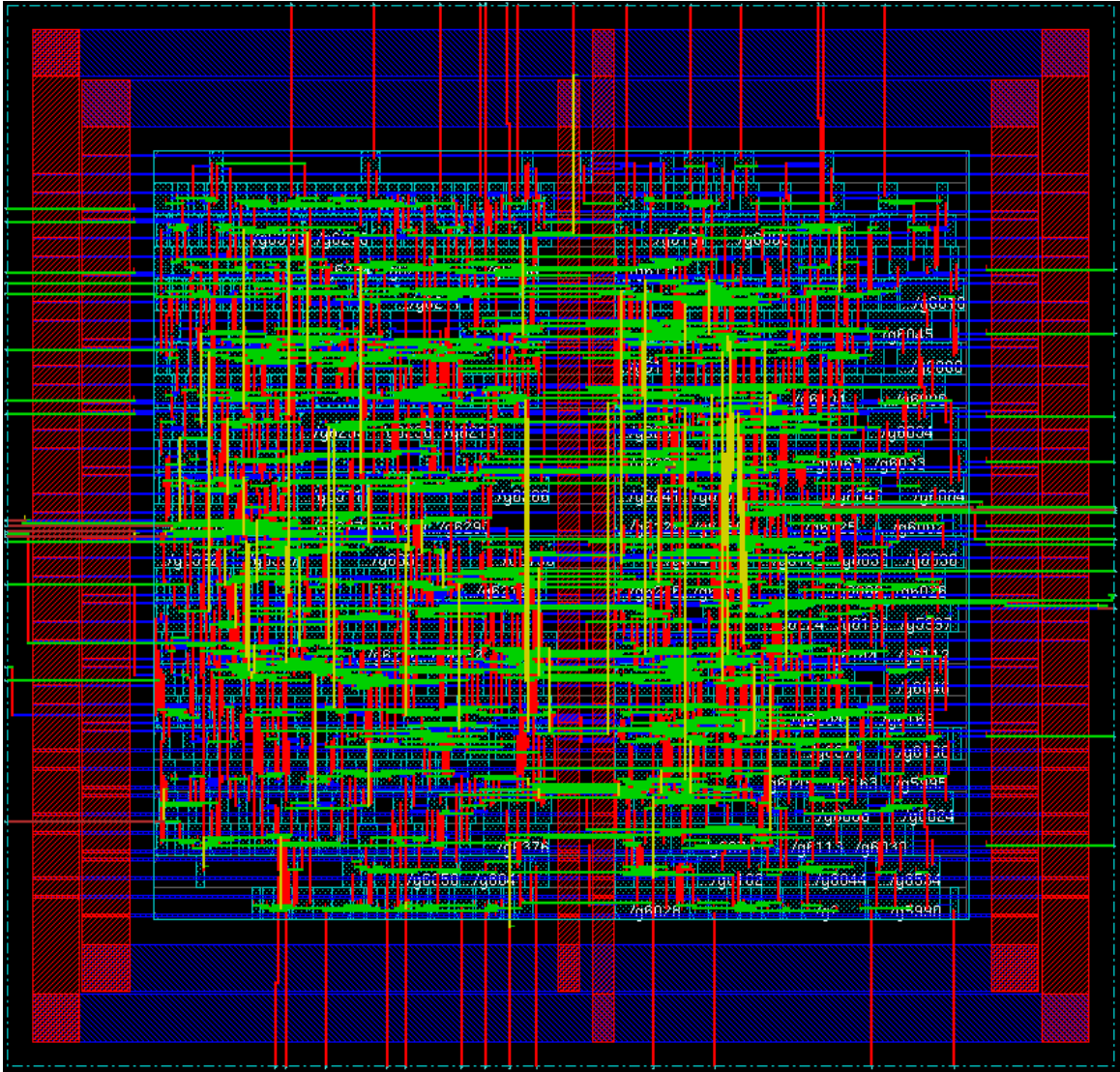


Figure A.21: 16x16 Multiplier Layout Generated Using 7rf Standard Cells

The MCML Standard Cell layouts created in this thesis aim to replace the layouts in the On Semiconductor ami06 technology so that similar outcomes can be created using MCML for specialized applications in VLSI.

APPENDIX B

CADENCE SETUP & SKILL SCRIPTS USED FOR DEVELOPMENT

Overview

This appendix shows some of the scripts and modifications required to run Cadence from a personal directory without interfering with common resources. These procedures and scripts should prove useful for any follow-up projects or theses that utilize Cadence and would benefit from the ability to have Cadence files and scripts in an editable, personal directory. Running Cadence in this manner allows for creation of new files, editing of existing files, and should any destructive files actions occur inadvertently, the common directories serve as a functioning backup so no net damage is done. This is the most ideal way to run Cadence for development of this nature without allowing accounts to have write permissions in community directories. Note that irrelevant commented portions of scripts are removed for readability and commentary for important parts of each script is provided when necessary and useful. The NCSU directory should be copied over into your personal directory before editing scripts and to allow for that directory to be fully editable. The changes described in the scripts look for various files from the NCSU folder in the personal directory and should be copied over as the first step.

Cadence Setup Scripts

- **cad-ncsu2.sh** – File called to run the Cadence virtuoso tools. This is the file called from your local directory to run the tools. The echo lines here are modified to show that the correct file is being run. The output text displays in the terminal once the script is called. The environmental variable CADDMM is defined here which points to the local working directory for the Cadence tools. Note that CADSETUPS utilizes the personal directory as well and is called in other scripts as well. Note that this script calls the setup-ncsu2.sh file which in turn calls the setup-cadence2.sh file. These file paths should be set to the working directory of

the Cadence install (your personal directory). Also note that the default bash file may attempt to search for the common directory script first. This can be changed by adding the path to your local directory below the line with the default search path (the bash file searches bottom to top). Alternately, the tools can be started from the personal directory by calling the file manually with the full path (for me as /home/dmarusia/SKILLSCMOS/CadCommon/cad-ncsu2.sh) or the relative full path from the terminal (for me as ./CadCommon/cad-ncsu2.sh). Either command works equivalently but the latter requires the terminal to be cd'd into the personal local directory.

```
#!/bin/bash
#####
# Startup script for icfb with the cmrf7sf technologies
#####
export CADPUB=/usr/cad_tools/cadence
export CADDMM=/home/dmarusia/SKILLSCMOS
export CADSETUPS=$CADDMM/CadCommon
echo "Running cad-ncsu2.sh - FIXED (DMM CUSTOM)"

# Set up for the Cadence tools with NCSU technologies
source ${CADSETUPS}/setup-ncsu2.sh

# reminds the user where they are...
echo "Working directory is" $PWD

# We need the cdsLibMgr.il file in the startup directory!
# Link it if it's not here already...
if [ -e cdsLibMgr.il ]
then
    ln -s $CDK_DIR/cdssetup/cdsLibMgr.il .
    echo "cdsLibMgr.il LINKED USING MODIFIED cad-ncsu2.sh"
fi

# Also link to the system-wide .cdsinit file
if [ ! -e .cdsinit ] ; then
    ln -s $FIXED_LOCATION/cdsuser/.cdsinit .
    echo ".cdsinit LINKED USING MODIFIED cad-ncsu2.sh"
fi

# You may want the class version of cds.lib. You need a copy of
# this so you can modify it as you add your own libraries.
if [ $?LOCAL_CADSETUP ] ; then
    if [ ! -e cds.lib ] ; then
        cp $FIXED_LOCATION/samples/cds.lib .
        echo "cds.lib COPIED from FIXED path:
/usr/cad_tools/cadence/installs/IC615/tools.lnx86/dfII/samples"
    fi
    echo "cad-ncsu2.sh local setup done"
fi

# Fire up virtuoso, and pass any args through to the program
#icfb $argv ---- IC615 uses virtuoso
```

```
echo "Starting Virtuoso"
virtuoso -64only $argv
```

- **setup-ncsu2.sh** – File called when cad-ncsu2.sh is run. Performs NCSU file setups for Cadence. Here the ‘CADDMM’ variable sets a path to my personal working directory. An environmental variable of any name to the directory you will run Cadence from. Note that the default CADPUB and CADSETUPS paths remain intact since some of those locations should draw from configuration files from the common directory. CDK_PUB was renamed from CDK_DIR to more readably differentiate between the common directory and personal directory. The echo of the final script line has an added custom setup text to verify that the correct script is called when cad-ncsu2.sh is called and that echo line will appear in the terminal window shortly before Cadence boots up. Note that this script is for NCSU file configuration and ONSEMI C5 topology. The 7rf files would be different with included paths but the personal variables modified would be the same as here.

```
#####
# set the directories that have the basic setups
export CADPUB=/usr/cad_tools/cadence
export CADDMM=/home/dmarusia/SKILLSCMOS
export CADSETUPS=$CADDMM/CadCommon
#####
# set the location of the NCSU local directory
# May be source of problem
echo "Next line in setup-ncsu2.sh may be a problem (creating filepath vars)"
#export FIXED_LOCATION=${CADPUB}/installs/IC615/tools.lnx86/dfII
export FIXED_LOCATION=${CADPUB}/NCSU/ncsu-cdk-1.6.0.beta
export CDK_PUB=${FIXED_LOCATION}
#####

# source the general Cadence setup file...
source $CADSETUPS/setup-cadence2.sh

# Set CDK_DIR
export CDK_DIR=${CADDMM}/NCSU/ncsu-cdk-1.6.0.beta

# Set CDS_SITE so that Cadence can find the cdsLibMgr.il, .cdsinit,
# and .cdsenv in the CDK directory...
echo "Next lines in setup-ncsu2.sh may be a problem (CDS_SITE) Stuff"
# ORIGINAL
#export CDS_SITE=${CDK_DIR}/cdsuser
#export CDS_SITE=${CDK_DIR}/samples:${CDS_SITE}
export CDS_SITE=${CDK_PUB}/cdsuser
export CDS_SITE=${CDK_PUB}/samples:${CDS_SITE}

unset base_dir uname

#---- End of Cadence NCSU Setup -----
#echo "You are now set up to run Cadence with the cmrf7sf packages."
```

These environmental variables are declared in cad-ncsu2.sh but are re-listed here for readability and inspection without requiring more files open to see the paths.

CDK_DIR set to the custom editable file path here. This variable is grabbed by other scripts that then propagate through and include more files. This MUST be set to your directory for full operation.

```
echo "You are now set up to run Cadence with the SC MOS setup. (CUSTOM SETUP - DMM)"
### EOF
```

- **setup-cadence2.sh** – File utilized for Cadence setup and called by setup-ncsu2.sh. No modifications here though it is important to note that this script sets up Cadence prior to performing any NCSU file setup. Personal directory operation should not require interaction with this file.

```
#####
# Startup script for basic Cadence. This does NOT set up specific NCSU
# or AMI or TSMC things. For those, you need to set things up on your own...
#
echo Using the setup-cadence script from F2012 EE431
#####
# Set the base directory for the cadence software
export base_dir="/usr/cad_tools/cadence/installs"
#####

echo "Running setup-cadence2.sh"

# Set some configuration environment variables.
export CLS_CDSD_COMPATIBILITY_LOCKING=NO
export CDS_Netlisting_Mode=Analog
export SPECTRE_DEFAULTS=-E
export CDS_LOAD_ENV=CWDElseHome
export SKIP_CDS_DIALOG
set uname = "/bin/uname"

# Point to each of the installation directories for the tool suites
export CDS=$base_dir/IC615      # Basic Cadence (i.e. IC tools)
export IC=$base_dir/IC615      # IC tools (composer, virtuoso, etc.)
export ICC=$base_dir/ICC11241  # ICC (ccar) tools
#export SOC=$base_dir/SOC81b    # SOC Encounter place and route
#export SOC=$base_dir/EDI11    # SOC Encounter place and route
export EDI=$base_dir/EDI101    # EDI Encounter Digital Implementation System

#export RC=$base_dir/RC91      # RTL Compiler synthesis
#export RC=$base_dir/RC111    # RTL Compiler synthesis
export RC=$base_dir/RC101      # RTL Compiler synthesis
export MMSIM=$base_dir/MMSIM72base # Spectre analog simulation
export IUS=$base_dir/IUS82     # Verilog simulators
export ETS=$base_dir/ETS91     # Encounter Library Characterizer
export ASSURAHOME=$base_dir/ASSURA615 # ASSURA DRC

# Set some environment variables for licensing
export CDS_LIC_FILE=5280@eelicensing.ee.calpoly.edu

export CDS_INST_DIR=$IC
```

```

export CDS_LIC_TIMEOUT=30
export TERM=$term
export LANG=C

# update the shell's path to point to the tools
export PATH=$ETS/bin:${PATH}
export PATH=$RC/tools/bin:${PATH}
# Added to get RTL compiler to run:
export PATH=$RC/bin:${PATH}
export PATH=$EDI/tools.lnx86/fe/bin/64bit:${PATH}

# Added to get SOC Encounter to work.
# Next line didn't work so went back to try to solve ELF64 error
#export PATH=/root/cadence/installs/SOC81/tools.lnx86/fe/bin/64bit
# ELF64 error:
#export PATH=/root/cadence/installs/SOC81/tools.lnx86/bin
export PATH=$ASSURAHOME/tools/bin:${PATH}
export PATH=$ASSURAHOME/tools/assura/bin:${PATH}
export PATH=$ICC/tools/bin:${PATH}
export PATH=$ICC/tools/iccraft/bin:${PATH}
export PATH=$ICC/tools/dfII/bin:${PATH}
export PATH=$IUS/tools/bin:${PATH}
# Added to get Verilog to work.
export PATH=$IUS/tools/dfII/bin:${PATH}
export PATH=$IC/tools/dfII/bin:${PATH}
export PATH=$IC/tools/bin:${PATH}
export PATH=$MMSIM/tools.lnx86/bin:${PATH}
export PATH=$MMSIM/tools.lnx86/bin/64bit:${PATH}
export PATH=$MMSIM/tools.lnx86/dfII/bin:${PATH}
export MMSIM_PATH=$MMSIM/tools/bin
export MMSIMHOME=$MMSIM

#### MAN FILES NOT PLACED TODO
if [ $?MANPATH ]; then
    export MANPATH=$IC/share/man:${MANPATH}
    export MANPATH=$IC/tools/man:${MANPATH}
else
    export MANPATH=$IC/share/man:${MANPATH}
    export MANPATH=$IC/tools/man:${MANPATH}
fi

if [ $?LD_LIBRARY_PATH ]; then
    export LD_LIBRARY_PATH=$ASSURAHOME/tools/lib:${LD_LIBRARY_PATH}
    export LD_LIBRARY_PATH=$ASSURAHOME/tools/assura/lib:${LD_LIBRARY_PATH}
    export LD_LIBRARY_PATH=$IUS/tools/lib:${LD_LIBRARY_PATH}
    export LD_LIBRARY_PATH=$IC/tools/lib:${LD_LIBRARY_PATH}
else
    export LD_LIBRARY_PATH=$ASSURAHOME/tools/lib:${LD_LIBRARY_PATH}

```

```

export LD_LIBRARY_PATH=$ASSURAHOME/tools/assura/lib:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=$IUS/tools/lib:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=$IC/tools/lib:${LD_LIBRARY_PATH}
fi

```

```

echo Basic Cadence script finished
# You're now ready to execute the Cadence tools!
# For testing without NCSU files:
#virtuoso -64only $argv

```

Library Management Scripts

- **cdslib** – File utilized for including the paths to desired libraries within Cadence. The important parts here for SCMOS operation include the NCSU_Analog_Parts library, NCSU_Digital_Parts library, and the NCSU_TechLib_ami06 library. These contain the files used to develop SCMOS layouts and further projects building upon or taping out some of these designs must be sure to include these files. The Analog_Parts library contains the transistor symbols and simulation data for the development performed in this project, and the ami06 library contains the layouts for those transistors when moving to layout development. The configuration file should be set to use those layouts for the NCSU transistors to allow for automated cell creation.

```

# File Created by David Marusiak at Tue May 27 23:10:03 2014
# assisted by CdsLibEditor
# File Created by root at Mon Apr 28 18:22:27 2014
# assisted by CdsLibEditor
# File Created by root at Thu Dec 19 13:33:54 2012
DEFINE NCSU_Analog_Parts $CDK_DIR/lib/NCSU_Analog_Parts
DEFINE NCSU_Digital_Parts $CDK_DIR/lib/NCSU_Digital_Parts

#####
## Library added for layout cells for SCMOS transistors ##
#####
DEFINE NCSU_TechLib_ami06 $CDK_DIR/lib/NCSU_TechLib_ami06
DEFINE NCSU_TechLib_ami16 $CDK_DIR/lib/NCSU_TechLib_ami16
#####

###DEFINE analogLib $CDS/tools/dfII/etc/cdslib/artist/analogLib
DEFINE sbaLib $CDS/tools/dfII/etc/cdslib/artist/sbaLib
DEFINE basic $CDS/tools/dfII/etc/cdslib/basic DEFINE sample $CDS/tools/dfII/samples/cdslib/sample
DEFINE US_8ths $CDS/tools/dfII/etc/cdslib/sheets/US_8ths
DEFINE avTech /usr/cad_tools/cadence/installs/ASSURA615/tools/assura/etc/avtech/avTech
DEFINE analogLib /usr/cad_tools/cadence/installs/IC615/tools/dfII/etc/cdslib/artist/analogLib

# Below are personal libraries created for development or testing.
DEFINE ThesisFinal /home/dmarusia/SKILLSCMOS/ThesisFinal
DEFINE SKILL_Name /home/dmarusia/SKILLSCMOS/TestLibraries/SKILL_Name
DEFINE SKILL_NameTest /home/dmarusia/SKILLSCMOS/TestLibraries/SKILL_NameTest
UNDEFINE SKILL_NameTest

```

```

DEFINE SKILL_NameTest /home/dmarusia/SKILLSCMOS/TestLibraries/SKILL_NameTest
DEFINE Tinaz /home/dmarusia/SKILLSCMOS/Tinaz #Removed by ddDeleteObj: DEFINE SCMOslibs
/home/dmarusia/SKILLSCMOS/SCMOslibs
DEFINE SCMOslibsTanner /home/dmarusia/SKILLSCMOS/SCMOslibsTanner
DEFINE OSU_stdcells_ami05 /usr/cad_tools/cadence/OSU/cdb2oa/OSU_stdcells_ami05
DEFINE cdsDefTechLib /usr/cad_tools/cadence/installs/IC615/tools/dfl/etc/cdsDefTechLib

```

- **cdsinit** – Lisp (or SKILL) file for performing SKILL files setups. This file is integral in moving forward with getting SKILL up and running. The script adds the file paths that each contains loadskill.il files in each subdirectory. Those files then call the SKILL files related to their directory. Main important note with this file is that the first line gets the CDK_DIR environmental variable from the shell. This is a file path set previously in the scripts utilized to start Cadence. This variable grab and the previous local directory settings allow for the rest of the Cadence tools to load and spider through all the necessary files from the local directory with only the few environmental variable changes detailed.

```

;;          -*-Lisp-*-
;;
;; NCSU CDK Copyright (C) 2006 North Carolina State University
;;

(let
  ((LOCAL_CDK_DIR (getShellEnvVar "CDK_DIR")))
  (if LOCAL_CDK_DIR
    (if (not (boundp 'NCSU_CDK_LOADED))
      (let () ; CDK needs to be loaded, so load it.

        (setq NCSU_CDK_DIR LOCAL_CDK_DIR)
        (procedure (prependNCSUCDKInstallPath dir)
          (strcat NCSU_CDK_DIR "/" dir))

        (printf "Loading NCSU CDK 1.5.1 customizations...\n")
        (setq NCSU_newLayoutMenuLabels t)
        (putpropq (hiGetCIWindow) 96 "maxLayerPoolSize")
        (envSetVal "graphic" "drfPath" 'string
          (strcat NCSU_CDK_DIR "/cdssetup/display.drf"))

        (if (isFile (prependNCSUCDKInstallPath "cdssetup/cdsenv"))
          (envLoadVals
            ?envFile (prependNCSUCDKInstallPath "cdssetup/cdsenv")
            ?tool "ALL"))

        (if (isFile "~/cdsenv")
          (envLoadVals
            ?envFile "~/cdsenv"
            ?tool "ALL"))

```

```

(let
  ((configFileList (list ; "aaConfig.il"
                        ; "dmConfig.il"
                        ; "dciConfig.il"
                        ; "metConfig.il"
                        ; "sysConfig.il"
                        ; "uiConfig.il"
                        ; "leConfig.il"
                        ; "schConfig.il"
                        ; "streamIn.il"
                        ))
   (path (strcat ". ~ "
                 (prependNCSUCDKInstallPath "skill/config_files")))
   (saveSkillPath (getSkillPath))
   file )
  (setSkillPath path)
  (foreach file configFileList
    (if (isFile file)
      (loadi file)))
  (setSkillPath saveSkillPath))

(let
  ((bindKeyFileList (list
                     "common_bindkeys.il"
                     ))
   (path (strcat ". ~ "
                 (prependNCSUCDKInstallPath "cdssetup")))
   (saveSkillPath (getSkillPath))
   file )
  (setSkillPath path)
  (foreach file bindKeyFileList
    (if (isFile file)
      (loadi file)))
  (setSkillPath saveSkillPath))

(sstatus writeProtect nil)

(let ((skillPathElements
      (list "." "~"
            (prependNCSUCDKInstallPath "skill")
            (prependNCSUCDKInstallPath "skill/cdf")
            (prependNCSUCDKInstallPath "skill/menus")
            (prependNCSUCDKInstallPath "skill/menus/artist")
            (prependNCSUCDKInstallPath "skill/menus/ciw")
            (prependNCSUCDKInstallPath "skill/menus/virtuoso")
            (prependNCSUCDKInstallPath "skill/misc")
            (prependNCSUCDKInstallPath "skill/pcells"))

```

These lines spider through the local directory originally defined from the CDK_DIR variable and include all the necessary SKILL files for the Cadence tools.

Adding SKILL files into the network should be as simple as adding another install path line here and following the same loadSkill.il file formatting as each subdirectory does where each load files then handles any other SKILL files in that directory.


```

        (prependNCSUCDKInstallPath "techfile"))))
sPE)

(foreach sPE skillPathElements
  (setSkillPath (cons sPE (getSkillPath))))))

(if (not (boundp 'NCSU_skillAlreadyLoaded))
  (let ()
    (setq NCSU_skillAlreadyLoaded t)
    (printf "Loading NCSU SKILL routines...\n")
    (load (prependNCSUCDKInstallPath "skill/loadSkill.il"))))

(envSetVal "graphic" "drfPath" 'string
  (strcat NCSU_CDK_DIR "/cdssetup/display.drf"))

(setq lePlotTemplate
  (prependNCSUCDKInstallPath "cdssetup/layoutPlotTemplate"))
(setq schPlotTemplate
  (prependNCSUCDKInstallPath "cdssetup/schPlotTemplate" ))
(unless (getShellEnvVar "SKIP_CDSLIB_MANAGER")
  (ddsOpenLibManager))
(printf "Done loading NCSU_CDK customizations.\n")
)
(printf "NCSU CDK already loaded.\n")
)

; you get to this let if NCSU_CDK_DIR is nil
(let ()
  (printf "Environment variable CDK_DIR must be defined to use\n")
  (printf "the NCSU CDK. It is not defined in the calling environment\n")
  (printf "so the NCSU customizations will not be performed!\n"))))

```