Digitally programmable delays for use in a beamforming IC receiver module

by

Stephanie L. Krass

B.S., Kansas State University, 2022

B.S., Kansas State University, 2022

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Mike Wiegers Department of Electrical and Computer Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Approved by:

Major Professor
Don M. Gruenbacher

Copyright

Notice: This manuscript has been authored using funds under the Honeywell Federal Manufacturing & Technologies under Contract No. DE-NA-0002839 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

Acknowledgments

The author would like to thank Dr. Don Gruenbacher for his time, expertise, and feedback. The author would also like to thank committee members Dr. Dwight Day, Dr. Xiaolong Guo, and Dr. Caterina Scoglio for their support and guidance throughout her education. Lastly, the author would like to thank Mr. Garrett S. Peterson for his time, feedback, and guidance throughout her education at Kansas State University.

Dedication

To my parents Micheal and Cynthia, and my sister Kirstin without their support and encouragement, this work would not have been possible.

Abstract

Phase alignment in beamforming systems is vital for high-performance operation. Phase alignment can be achieved through phase shifts in RF that are produced with the aid of a variable time reference to a frequency synthesizer. In addition, programmable delays are often used to vary the delay of a signal. Programmable delays for linear, high resolution, and wide delay range need to be designed as a single circuit on-chip solution for ICs. Otherwise, parasitic and other effects degrade the performance of the programmable delay. This thesis proposes two possible programmable delays produced using two different tool sets: 1) Cadence Virtuoso, and 2) the combination of Cadence Genus and Cadence Innovus. Additionally, the two proposed programmable delays were designed using different types of MOSFETs: body contact MOSFETs (bulk-MOSFETs) and floating body MOSFETs (SOI-MOSFETS). The floating body MOSFET design suffers from delay step timing stability due to history effect. Programmable delays are characterized by LSB resolution, delay range, differential non-linearity, and integral non-linearity. The programmable delay design using Cadence Virtuoso had an LSB resolution in simulation of 3.24 ps and a total delay range of 6.873 ns with 2119 steps. The other programmable delay design had a simulated LSB resolution of 7.54 ps and a total delay range of 7.72 ns with 1023 steps.

Table of Contents

List of Figures ix

List of Tables xii

Listings xii

List of Abbreviations xiv

1 Introduction 1

1.1 Motivation 1

1.2 Prior Work 5

1.3 Main Contributions 6

1.4 Thesis Organization 6

2 Delay Elements 7

2.1 Conventional Delay Building Blocks 11

2.1.1 Inverter 11

2.1.2 Tristate Inverter 12

2.1.3 Buffer 13

2.1.4 Transmission Gate 13

2.1.5 Multiplexor 14

2.2 Conventional Delay Line Topologies Overview 14

2.3 Delay Variance Techniques 16

2.3.1 Supply Modulation 17

2.3.2 Variable Load 17

2.3.3 Controlling Transistors 18

2.4 Programmable Delay Elements 19
2.4.1 Shunt Capacitor Techniques 21
2.4.2 Current Starvation Techniques 22
2.4.3 Variable Resistor Techniques 23
2.4.4 Comparisons of Delay Techniques 24

3 Proposed ICV2 Programmable Delay 25
3.1 Block Diagrams of Proposed ICV2 Programmable Delay 25
3.2 Building Blocks 26
3.2.1 Transmission Gate 27
3.2.2 Inverter 29
3.2.3 Buffers 30
3.2.4 Multiplexor 31
3.3 Proposed ICV2 Programmable Delay 32
3.4 Simulation Results 33
3.4.1 Linearity Simulation 33
3.4.2 Monte Carlo Simulation 35
3.4.3 Temperature Simulation 37
3.4.4 Discussion 38

4 Proposed Programmable Delays
ICV3 & ICV4 40
4.1 Block Diagram of Proposed Programmable Delays
ICV3 & ICV4 40
4.2 Design Process 41
4.2.1 Design Elements 42
4.2.2 Code 42
4.3 Proposed Programmable Delays
ICV3 & ICV4 44
4.4 Simulation Results 47

4.4.1 Linearity Simulation 47
4.4.2 Monte Carlo Simulation 49
4.4.3 Temperature Simulation 50
4.4.4 Repeated Commands Simulation 51
4.4.5 Discussion 53

5 Conclusion 55
5.1 Results 56
5.2 Future Work 56

References 57

A ICV3 Verilog Code for Generating the Structured Netlist 61

B Genus Command Log Output File for ICV4 65

C Genus Synthesized Verilog Output for ICV4 66

D Innovus Command Log for ICV4 68

E Python to Generate SDP for a Specific Buffer 70

F Full Output Text File Generated by Python for the 128 delay step buffer 72

G SDP File used for ICV4 in Innovus 77

List of Figures

1.1 Two simple radars . . . 2
1.2 Example of a patch phased array antenna . . . 2
1.3 Phase alignment caused by incremental phase delays for a TX antenna . . . 3
1.4 Reference Phase Synchronization of 32 antenna elements, 1 GHz BW, 5/8 lambda spacing [5] . . . 4

2.1 Body Contact MOSFETs . . . 7
2.2 Body Contact MOSFETs input and output operation . . . 9
2.3 Delay and Transition Timing . . . 10
2.5 Inverter Logic Gate . . . 12
2.7 Tri-State Inverter Logic Gate . . . 13
2.9 Buffer Logic Gate . . . 13
2.11 Transmission Logic Gate . . . 14
2.12 2 to 1 MUX . . . 15
2.13 Tapped Inverter Chain Delay Line . . . 16
2.14 Tapped Buffer Chain Delay Line . . . 16
2.15 Single Output Inverter Matrix . . . 16
2.16 Simulation Results for Modulating V_{DD} . . . 17
2.17 Buffer with a Variable Load . . . 18
2.18 Simulation Results of Using a Variable Load with a Buffer . . . 18
2.19 Buffer with Controlling Transistors . . . 19
2.20 Simulation Results of Using Controlling Transistors on a Buffer . . . 19
2.21 Digital Controlled Delay Line Transfer Function [9] . . . 20
2.22 Shunt Capacitor Variable Delay . . . 22

2.23 Current Starved Variable Delay 22
2.24 Variable Resistor Delay 23

3.1 ICV2 Programmable Delay Block Diagram 26
3.2 ICV2 Programmable Delay Primary Block Diagram 26
3.3 ICV2 Programmable Delay Tuning Block Diagram 27
3.4 Implemented Transmission Gate 27
3.5 TG Schematic Simulations 28
3.6 TG Extracted Simulations 28
3.7 Inverter 29
3.8 Inverter Simulations 29
3.9 Buffer 30
3.10 Buffer Simulations 30
3.11 MUX 31
3.12 MUX Simulations 31
3.13 ICV2 Programmable Delay Layout 32
3.14 ICV2 Programmable Delay Thermometer Code Simulation 34
3.15 ICV2 Programmable Delay DNL without Tuning 34
3.16 ICV2 Programmable Delay DNL with tuning 34
3.17 ICV2 Programmable Delay INL 35
3.18 Proposed ICV2 PD Monte Carlo Sim 36
3.19 Proposed ICV2 PD Temperature Extracted Simulations 37
3.20 Proposed ICV2 PD Schematic and Extracted Temperature Sim Comparisons 37

4.1 ICV3 Programmable Delay Block Diagram 41
4.2 ICV4 Programmable Delay Block Diagram 41
4.3 ICV3 Programmable Delay Layout 44
4.4 ICV4 Programmable Delay Layout using SDP 47
4.5 ICV3 and ICV4 Programmable Delay Thermometer Code Simulation 48
4.6 ICV3 Programmable Delay DNL 48

4.7 ICV4 Programmable Delay DNL 48
4.8 ICV3 Programmable Delay INL 49
4.9 ICV4 Programmable Delay INL 49
4.10 Proposed ICV3 & ICV4 Programmable Delay Monte Carlo 50
4.11 Proposed ICV3 & ICV4 Programmable Delay Temperature Simulations . . . 51
4.12 ICV4 Programmable Delay Repeated Simulation 52

List of Tables

2.1 MOSFET Parameters and Definitions 8
2.2 Logic operation of MOSFETs as switches 8
2.3 Delay and Transition Timing Definitions 10
2.4 Truth Table 12
2.5 Truth Table 13
2.6 Truth Table 13
2.7 Truth Table 14
2.8 2 to 1 MUX Truth Tables 15
2.9 Selected Controlling Transistor Results 19
2.10 Comparison of Current Programmable Delay Lines 24

3.1 Monte Carlo Primary Delay Results 36
3.2 Monte Carlo Tuning Delay Results 36
3.3 Primary Delay over Temperature Results 38
3.4 Tuning Delay over Temperature Results 38
3.5 ICV2 PD Results Summary 39

4.1 Comparison of ICV3 Monte Carlo Sims 50
4.2 Comparison of ICV4 Monte Carlo Sims 50
4.3 Comparison of ICV3 Temperature Sims 51
4.4 Comparison of ICV4 Temperature Sims 51
4.5 Comparison of ICV4 Delays Through Gates over consecutive runs 53
4.6 ICV3 & ICV4 Results Summary 53

5.1 Comparison of Current Programmable Delay Lines 56

Listings

4.1 Partial Verilog Code of Creating the Structure Netlist 43
4.2 Text File used as Input for Python Script 45
4.3 Python Script for Creating a Structure Data Path 45
4.4 Partial Output File Generated by Python 46

List of Abbreviations

Abbreviation	Meaning
RF	Radio frequency
DLL	Delay-locked loop
PLL	Phase-locked loop
DCO	Digitally controlled oscillator
Radar	Radio detection and ranging
EM	Electromagnetic
TX	Transmit/transmitted/transmission
RX	Receiving/receive
SNR	Signal to noise ratio
BW	Bandwidth
IC	Integrated circuit
CMOS	Complementary metal-oxide semiconductor
MOSFET	Metal-oxide silicon field effect transistor
NFET	n-channel field effect transistor
PFET	p-channel field effect transistor
FET	Field effect transistor
INV	Inverter
High Z	High impedance
X	Don't care logic (can be logic 1 or logic 0)
TG	Transmission gate
MUX	Multiplexor
DCDL	Digitally controlled delay line
DNL	Differential non-linearity
INL	Integral non-linearity
FOM	Figure of merit
LSB	Least significant bit
MSB	Most significant bit
SCI	Shunt capacitor inverter
CSI	Current starved inverter
PD	Programmable delay
ICV2	Refers to the 2^{nd} IC fabricated in GF45RFSOI at Kansas State University
ICV3	Refers to the 3^{rd} IC fabricated in GF45RFSOI at Kansas State University
ICV4	Refers to the future 4^{th} IC fabricated at Kansas State University
SOI	Silicon-on insulator
SDP	Structured data path

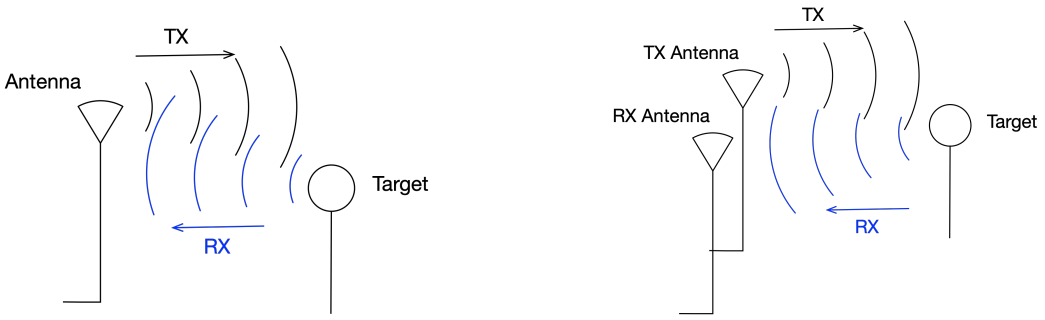
Chapter 1

Introduction

This work is focused on the design and execution of digitally controlled programmable delays. A total of three different delay designs were developed during the course of this thesis. The programmable delays discussed in this work were designed with the intent of use in a phased array radar module to synchronize synthesizer references, ultimately leading to a phase shift in the radio frequency (RF) wave. However, programmable delays are used in many other applications, including digital delay-locked loops (DLLs), phase-locked loops (PLLs), digitally controlled oscillators (DCOs), as well as microprocessor and memory circuits [1].

1.1 Motivation

Radio detection and ranging (RADAR) systems have been used since the early 20th century. They use RF or microwave frequency electromagnetic (EM) waves to detect objects, map terrain, and weather patterns. The EM waves are transmitted (TX) from an antenna, reflected off a target object then returned to the receiving (RX) antenna. Two examples of simple radar systems can be seen in Figure 1.1. The blue waves in Figure 1.1 indicate the EM wave reflected off the target. Figure 1.1a is a monostatic radar system where the TX and RX antennas are combined into a single antenna (or are in the same location). Figure 1.1b



(a) Single antenna radar (b) Two antenna radar

Figure 1.1: Two simple radars

is a simple bistatic radar system with a dedicated TX antenna and a dedicated RX antenna that are not in the same location.

Antennas can be combined to form a single-phased array antenna. This is commonly done with pyramidal horn, patch (microstrip), spiral, and slotted waveguide array antennas [2]. These antennas are either linear or two-dimensional. Also, the antenna elements are typically uniformly spaced [3,4]. Figure 1.2 depicts an example of a 4 x 4 patch phased array antenna. The phased array can act as a single antenna when all the elements are phase aligned in reference to each other. The fact that the individual antennas only need to be phased aligned with each other means that the array as a whole can stay in a stationary position while the phase of the components are changed in unison, which leads to effectively steering the direction of the radar. The implementation of this may differ depending on the antennas and control mechanisms used.

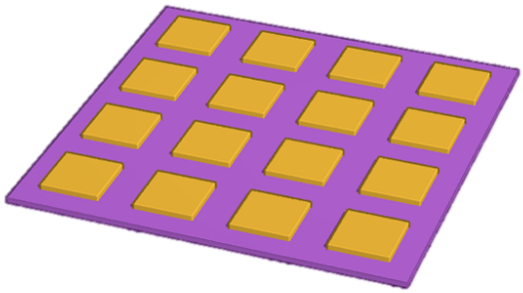


Figure 1.2: Example of a patch phased array antenna

Phased array antennas are an integral component of a beamforming system. Generally speaking, beamforming radars require no moving mechanical components to steer the radar in a desired direction since the antennas are electronically steered [3]. This is a desirable feature of beamforming systems as it allows for automatic channel operation, which leads to high data rates, and an increased signal-to-noise ratio (SNR) [3]. Although, these all rely on the individual antenna elements being phased aligned. Figure 1.3 shows how introducing an incremental phase shift to each subsequent element in the linear phased array can cause steering of the overall antenna beam. In this example of beam steering, the time delay is represented by τ and $\tau_0 > \tau_1 > \tau_2 > \tau_3 > \tau_4 > \tau_5 > \tau_6 > \tau_7$ in order to generate the wavefront and primary lobe depicted for a TX antenna.

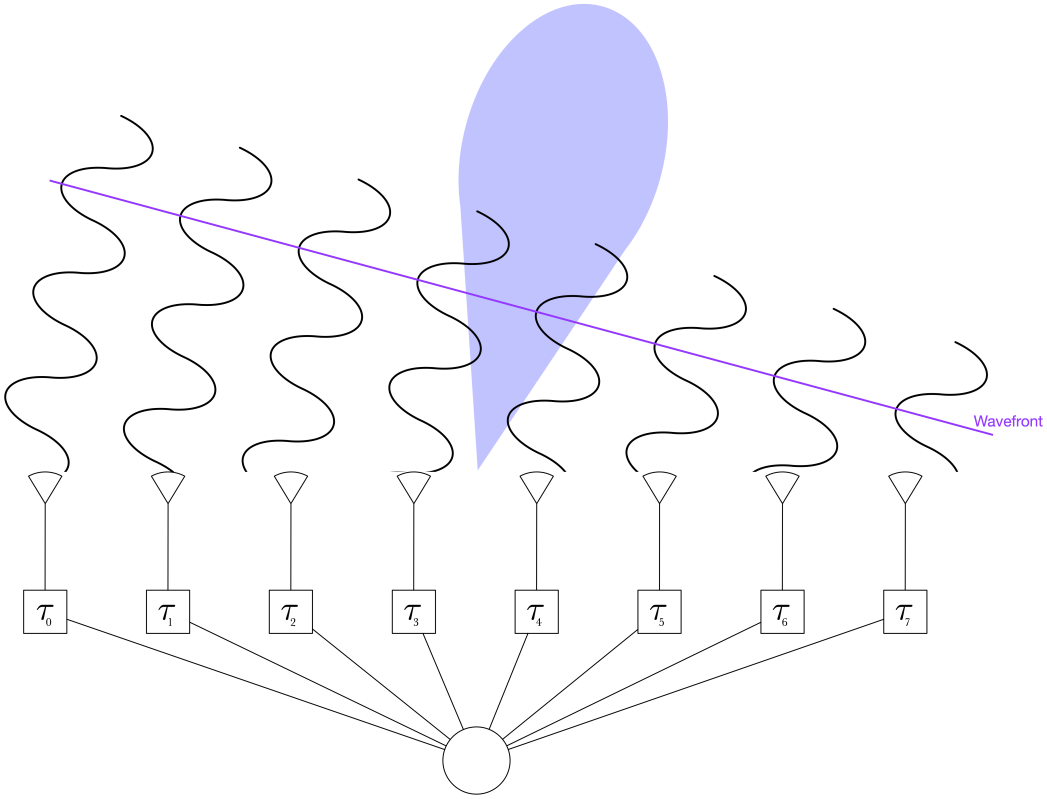
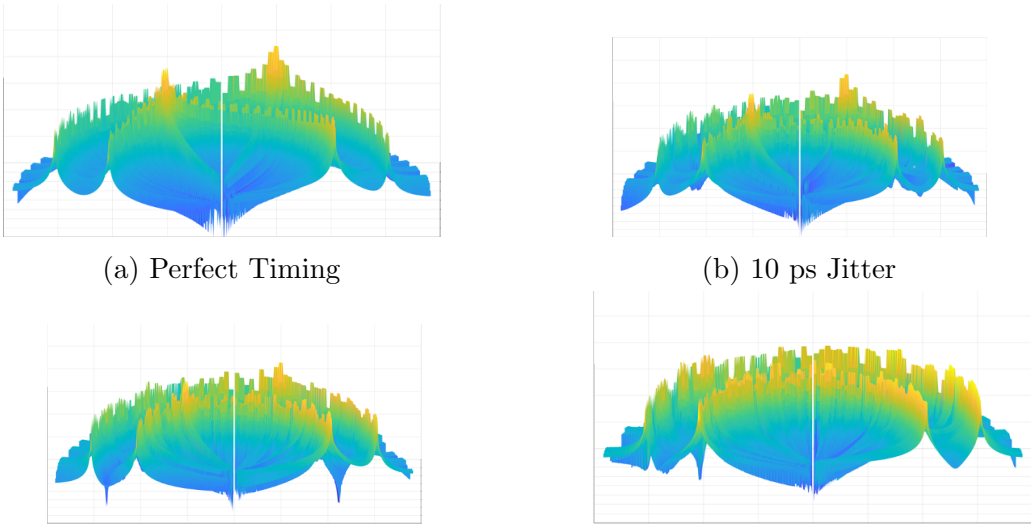


Figure 1.3: Phase alignment caused by incremental phase delays for a TX antenna

Phase alignment is frequently achieved through phase shifts. Phase shifts can be done with ferrite or diode phase shifters and by changing the timing of the reference to a frequency synthesizer, ultimately leading to the phase shift in RF. The latter method is frequently

seen in mixed signal beamforming and relies on reference phase synchronization for proper operation. Reference phase synchronization can be accomplished with a programmable delay element. This type of delay element can introduce a variable delay allowing for more than one configuration of the phased array antenna.

Figure 1.4 shows four plots from a MATLAB simulation showing the impact of reference phase synchronization between 32 elements on the ability to make out two separate targets. Figure 1.4a shows the results of scanning for the two separate targets at two different distances with perfect timing. Figure 1.4b shows the result for the two targets with 10 ps of jitter, Figure 1.4c is what happens with 20 ps of jitter, and Figure 1.4d is the result of having 50 ps of jitter. The difference between Figure 1.4a and Figure 1.4b is not very noticeable, and the two targets' locations can still be easily distinguished. However, this is not the case for Figure 1.4c and Figure 1.4d. The plots indicate that the elements in the phased array antenna need to be reference phase synchronized to 10 pS or less otherwise, the radar will suffer from a decreased ability to resolve the two targets.



(a) Perfect Timing

(b) 10 ps Jitter

(c) 20 ps Jitter

(d) 50 ps Jitter

Figure 1.4: Reference Phase Synchronization of 32 antenna elements, 1 GHz BW, 5/8 lambda spacing [5]

1.2 Prior Work

Delay lines introduce a delay in time to a signal. This time delay can be static or dynamic based on the intended application. A static time delay is a constant time delay added to a circuit. On the other hand, a dynamic time delay can be changed while the circuit is in operation. There are numerous methodologies to accomplish time delays. The methods can be categorized based on the technology used, optical or electronic [6].

One of the oldest methods for altering the amount of time it takes for a signal to reach its destination, the output, was to mechanically switch in a longer or shorter line length. This method has been implemented in both optical and electronic circuit-based delay lines. Optical variable delay lines have two main techniques used in integrated devices to achieve slow-light on-chip, using coupled-resonator optical waveguides or photonic crystal waveguides. In addition, optical delay lines can be cascaded to increase the delay range without sacrificing the high-resolution and linear increments associated with them [7]. Optical delay lines are used in optical beamformers and are incompatible with the 45 nm integrated circuit(IC) process used for this research. For completeness, optical delay line technology is mentioned; however, the focus of this Master's thesis is on electronic technologies.

Traditional CMOS delay lines face two main challenges jitter performance and achieving linear high-resolution delay steps for wide delay ranges [6]. The jitter performance of CMOS delay lines falls in the range of several picoseconds [8, 9]. Research is being done into sub-picosecond jitter performance on CMOS ICs as they are cheaper and easier to integrate into a system than optical delay lines [6]. There is a link between sub-picosecond jitter performance and the smaller nanometer processes, although strategic design also plays a significant role. Processes that have been reported to produce sub-picosecond jitter are 14 nm, 28 nm, 65 nm, and 0.13 μm [10–13]. The ability to have sub-picosecond jitter performance in a larger process like the 0.13 μm comes at the cost of increased control complexity. The design proposed by Abdulrazzaq et al. [13] requires 32 control bits and 3 separate delay stages.

A challenge with CMOS delay lines for large delay ranges and linear high-resolution delay steps is related to parasitic capacitance leading to non-linear delay steps. It also impacts the

ability to connect multiple identical CMOS delay lines in series and maintain the original linearity and resolution of the delay steps. So, a single delay line is ideal to avoid issues with linearity and resolution of delay steps for a wide delay range [6]. The designs documented in this thesis are centered around producing high-resolution linear delay steps for a wide delay range implementing at least 10 bits to control the programmable delay. More details on previous work on variable delay line topologies will be covered in Chapter 2.

1.3 Main Contributions

The work documented in this Master's thesis made 2 main contributions to the field of variable delay elements.

1. A tunable wide range, linear, 3.4ps resolution, 6.8ns delay range, inverter chain based digitally controlled delay line

2. A wide range, linear, digitally synthesized, 7.5ps resolution, 7.7ns delay range digitally controlled delay line

1.4 Thesis Organization

This thesis is divided into five chapters. The first chapter has covered the motivation behind designing delay lines, an overview of prior work, the main contributions attributed to this work, and the organization of the remainder of the thesis. Chapter two will cover delay elements starting with conventional building blocks and topologies and then moving on to current digitally controlled programmable delay topologies. The third chapter covers the fully custom programmable delay's conception and simulated results. Chapter four investigates the semi-custom programmable delay, its iterations, design process, and simulated results. Lastly, chapter five will compare the two programmable delay designs, limitations, possible improvements, and future plans.

Chapter 2

Delay Elements

CMOS transistors are commonly used in analog and digital circuits. Their use in such circuits is extensively covered in many works. The information about CMOS transistors as related to this work will be covered. However, more in-depth details can be found in *CMOS: Circuit Design, Layout, and Simulation* [15], *CMOS VLSI Design: A Circuits and Systems Perspective* [16],and *Microelectronic Circuits* [17] among numerous other sources.

Delay elements are commonly created from CMOS transistors shown in Figure 2.1. The gate electrode, commonly called the gate, is denoted by G. The source and drain regions, usually referred to simply as the source or drain, are marked by S and D, respectively. The substrate terminal typically called the body, is labeled with B. Table 2.1 is included for clarity on MOSFET parameters that will be mentioned later in this chapter.

(a) NFET (b) PFET

Figure 2.1: Body Contact MOSFETs

NFETs and PFETs can be operated as switches based on the voltage applied to their gate terminal. For an NFET, if the gate voltage is the logic equivalent of 0 (ground), the

Table 2.1: MOSFET Parameters and Definitions

Name	Symbol	Value/Equation
Vacuum Dielectric Constant	ϵ_0	8.85 aF/μm
SiO_2 Silicon Dioxide Dielectric Constant	ϵ_{ox}	$3.9\epsilon_0$
Gate Oxide Thickness	t_{ox}	Determined by process
Oxide Capacitance per area	C'_{ox}	ϵ_{ox}/t_{ox}
Oxide Capacitance	C_{ox}	$C'_{ox}WL$
Saturation Drift Velocity of an Electron	v_{sat}	
Electric Field	E	
Electron Mobility	μ_n or μ_p	v_{sat}/E
On or drive Current	I_{on}	$v_{sat}C'_{ox}\left(V_{GS} - V_{THN} - V_{DS,\ sat}\right)$
Process Transconductance	$k_n{}'$ or $k_p{}'$	$\mu_n C_{ox}$ or $\mu_p C_{ox}$

Table 2.2: Logic operation of MOSFETs as switches

Gate Logic Level	NFET	PFET
0	Off	On
1	On	Off

NFET is "off", and if the gate voltage is the equivalent of logic 1, the NFET is "on". The opposite is true for the PFET; see Table 2.2. The current that flows from the drain terminal (i_D) in NFETs and PFETs when they are on is described by

$$i_D = \frac{1}{2} k_{n/p}{}' \left(\frac{W}{L}\right) {V_{OV}}^2 \tag{2.1}$$

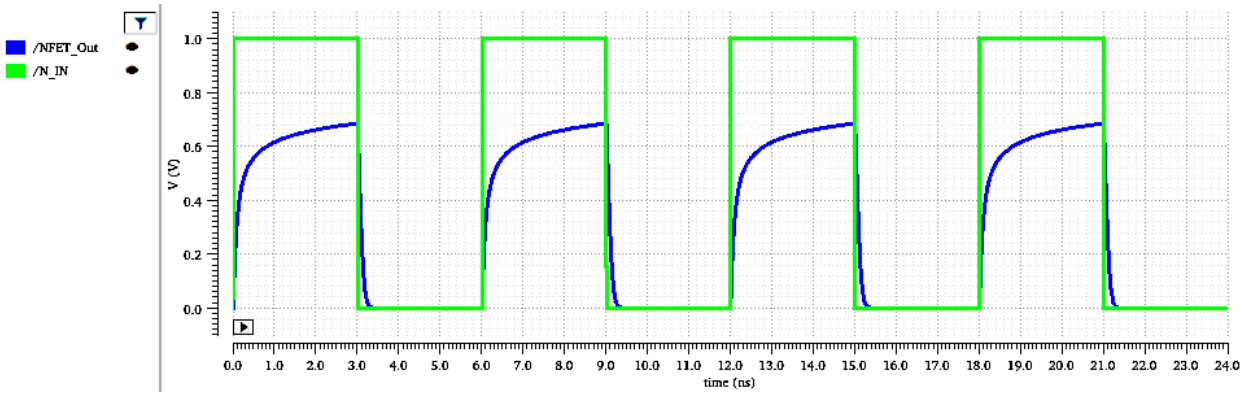
where $k_{n/p}{}'$ is the process transconductance parameter where n denotes an NFET and p would denote a PFET, W is the width of the transistor, L is the length of the transistor, and V_{OV} is the overdrive voltage. $k_{n/p}{}'$ is the product of the electron mobility ($\mu_{n/p}$) for the NFET (or PFET) and the oxide capacitance (C'_{ox}). The overdrive voltage for an NFET is

$$V_{OV} = V_{GS} - V_{THN} \tag{2.2}$$

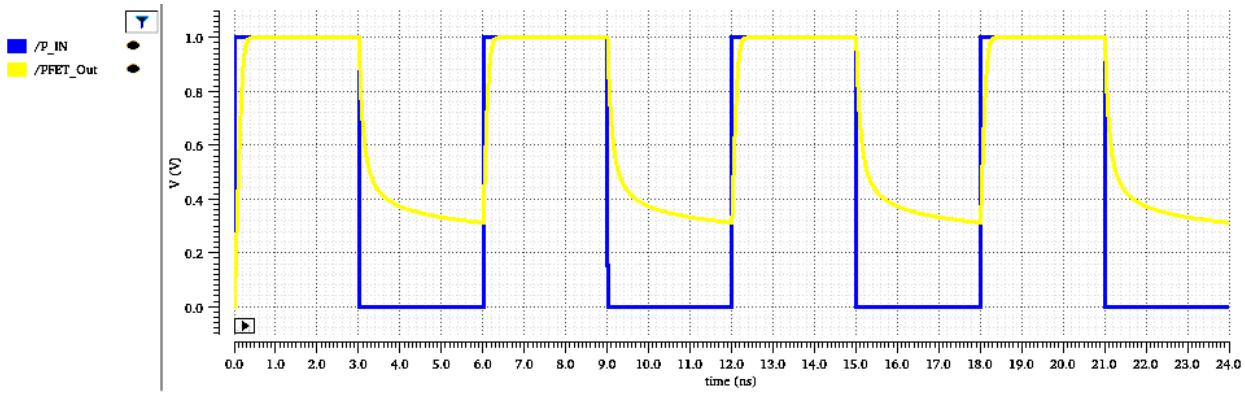
where V_{GS} is the gate to source voltage and V_{THN} is the NFET's threshold voltage. The overdrive voltage for a PFET is

$$V_{OV} = V_{SG} - |V_{THP}| \tag{2.3}$$

where V_{SG} is the source to gate voltage and V_{THP} is the PFET's threshold voltage.



(a) NFET "On" operation



(b) PFET "On" operation

Figure 2.2: Body Contact MOSFETs input and output operation

This complementary operation of the NFET and PFET is also seen in the inability of the NFET to pull a signal all the way up to the supply voltage (V_{DD}) and the inability of the PFET to fully pass the ground signal (V_{SS}). The NFET device can pass 0 (or the ground signal) well but is not able to pass the full 1 or supply voltage signal as the voltage from gate to source (V_{GS}) needs to surpass the innate threshold voltage of the NFET (V_{THN}) first. This means that the maximum voltage an NFET can pass is $V_{DD} - V_{THN}$ when $V_{GS} = V_{DD}$ see Figure 2.2a. On the other hand, a PFET can pass V_{DD} but is unable to pass ground (V_{SS}) as the lowest voltage it can pass as an output is its threshold voltage (V_{THP}), see Figure 2.2b. This is partially explained by the fact that usually the body for the NFET is connected to V_{SS}, while the body of the PFET is connected to V_{DD} to maintain the cutoff condition for the substrate-to channel junctions as a result of body effect.

A combination of NFETs and PFETs is desirable when making logic gates, as the NFET will pull down the signal to V_{SS} and the PFET will pull up the signal to V_{DD}. The time it takes for a signal to transition from the low voltage, V_{SS}, to the high voltage, V_{DD} is called the rise time, t_r for an input signal. The transition from V_{DD} to V_{SS} is called the fall time t_f. The rise time for an output signal is usually referred to as t_{LH} (time it takes for the voltage to go from low to high, Equation 2.6), and the fall time is referred to as t_{HL} (time it takes for the voltage to go from high to low, Equation 2.7) [17]. The transition times depend on the resistance between the source and drain R_n or R_p for the NFET and PFET, respectively Equation 2.4, and C_{total}. C_{total} is the total capacitance seen by the device, which includes the capacitance of the load, C_L, in addition to the gate capacitance, C_{ox}, see Equation 2.5.

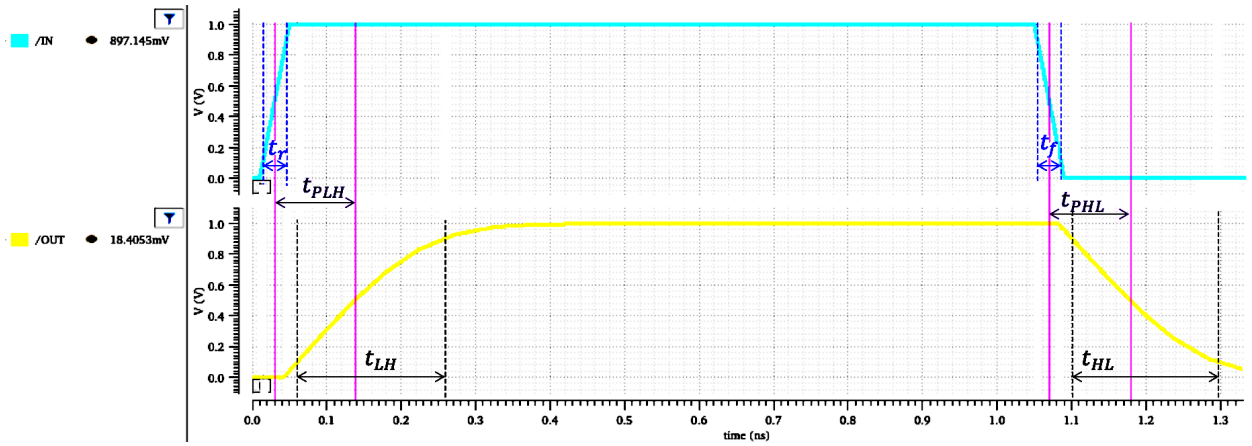


Figure 2.3: Delay and Transition Timing

Table 2.3: Delay and Transition Timing Definitions

Timing Characteristic	Edge	Formula
t_r	Rising	$t_{(Vin\ 90\%\ of\ VDD)} - t_{(Vin\ 10\%\ of\ VDD)}$
t_f	Falling	$t_{(Vin\ 10\%\ of\ VDD)} - t_{(Vin\ 90\%\ of\ VDD)}$
t_{LH}	Rising	$t_{(Vout\ 90\%of\ VDD)} - t_{(Vout\ 10\%\ of\ VDD)}$
t_{HL}	Falling	$t_{(Vout\ 90\%of\ VDD)} - t_{(Vout\ 10\%\ of\ VDD)}$
t_{PLH}	Rising	$t_{(Vout\ 50\%of\ VDD)} - t_{(Vin\ 50\%\ of\ VDD)}$
t_{PHL}	Falling	$t_{(Vout\ 50\%of\ VDD)} - t_{(Vin\ 50\%\ of\ VDD)}$

$$R_{n/p} = \frac{V_{DD}}{\frac{k_{n/p}'}{2}\frac{W}{L} \cdot \left(V_{DD} - V_{TH(N/P)}\right)^2} = \frac{V_{DD}}{I_{on,\ n/p} \cdot W} \tag{2.4}$$

$$C_{total} = C_L + C_{ox} \tag{2.5}$$

$$t_{LH} \approx 2.2 \cdot R_p \cdot C_{total} \tag{2.6}$$

$$t_{HL} \approx 2.2 \cdot R_n \cdot C_{total} \tag{2.7}$$

The timing transitions t_r, t_f, t_{LH}, and t_{HL} are all measured using the 90% voltage level and the 10% voltage level of the respective signal along with the corresponding time at which that voltage level was achieved and calculating the difference. The delay of a signal from the input to the output is commonly called the propagation delay. The propagation delay is also characterized by the transition from high to low (t_{PHL} Equation 2.8) and from low to high (t_{PLH} Equation 2.9) [17]. They are measured by taking the difference in time between the input signal reaching the 50% voltage level and the output signal reaching the 50% voltage level. Figure 2.3 is visual representation of t_r, t_f, t_{LH}, t_{HL}, t_{PHL}, and t_{PLH}.

$$t_{PHL} \approx 0.7 \cdot R_n \cdot C_{total} \tag{2.8}$$

$$t_{PLH} \approx 0.7 \cdot R_p \cdot C_{total} \tag{2.9}$$

2.1 Conventional Delay Building Blocks

MOSFETs are commonly used to create digital logic gates, often used for generating time delays in a circuit. Some of the most frequently used logic gates for adding propagation delays to a signal are inverters, buffers, and transmission gates. Tri-state inverters and multiplexors are other common logic gates used to control the passage (or not) of specific signals through a circuit.

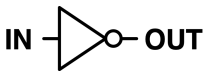
2.1.1 Inverter

Inverters can be made using MOSFETs, and they are used to invert an input signal from high to low or low to high, as shown in Table 2.4. Figure 2.5 shows the general symbol used to

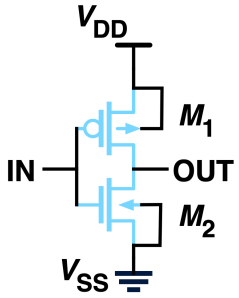
represent an inverter and the schematic for an inverter made from body contact MOSFETs. The goal in designing an inverter is generally to have t_{PLH} and t_{PHL} to be approximately the same value, meaning that the inverter will induce an equal amount of delay regardless of if it is given a high or low signal. This is accomplished by sizing the NFET and PFET in a way that makes $R_n \approx R_p$ while maintaining the value of C_{ox} (see Table 2.1).

$$C_{ox} = C'_{ox} W L \tag{2.10}$$

Also, referring back to Equation 2.4, it can be seen that the resistance between the source and drain terminal is inversely proportional to the width, $R_{n/p} \propto W$.



(a) Symbol



(b) CMOS Schematic

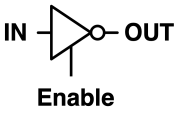
Input	Output
0	1
1	0

Table 2.4: Truth Table

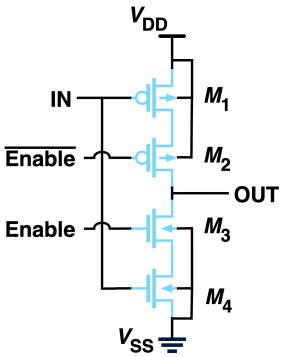
Figure 2.5: Inverter Logic Gate

2.1.2 Tristate Inverter

Tri-state inverters are very similar to inverters discussed in the previous section, as seen by the similarity in the symbol used for a tri-state inverter in Figure 2.6a and the one used for a simple inverter. A tri-state inverter is also used to invert a signal. However, it requires an enable signal for its output. If the Enable signal is 1, the device will act as a regular inverter. If the Enable signal is 0 regardless of the input being 1 or 0 (denoted by X in Table 2.5), the output will be at high impedance (also called High Z). A tri-state inverter can be made from two NFETs and two PFETs, as seen in Figure 2.6b. In Figure 2.6b, the PFET M_1 and NFET M_4 are used to invert the signal and ideally would have $R_n \approx R_p$.



(a) Symbol



(b) CMOS Schematic

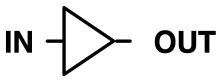
Enable	Input	Output
0	X	High Z
1	0	1
1	1	0

Table 2.5: Truth Table

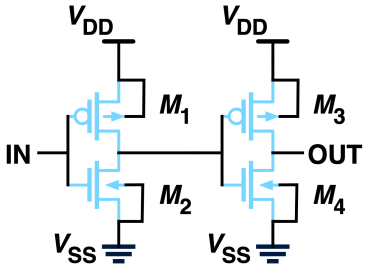
Figure 2.7: Tri-State Inverter Logic Gate

2.1.3 Buffer

Buffers are used to add a time delay to a signal without changing its polarity. There are many buffer design options, including inverters, transmission gates, and flip-flops. Buffers like inverters will ideally have an equivalent t_{PLH} and t_{PHL}. One way to ensure this is to use identical inverters to create the buffer. Looking at Figure 2.8b this would mean that $M_1 = M_3$ and $M_2 = M_4$ and the R_p(of M_1 and M_3) = R_n(of M_2 and M_4).



(a) Symbol



(b) CMOS Schematic

Input	Output
0	0
1	1

Table 2.6: Truth Table

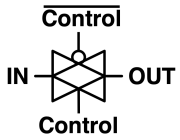
Figure 2.9: Buffer Logic Gate

2.1.4 Transmission Gate

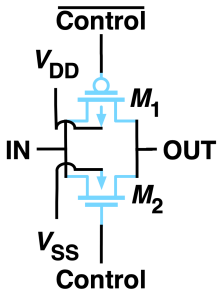
Transmission gates (TG) are made using a PFET and an NFET, as shown in Figure 2.11. They are used to simply pass a signal when the Control signal is a logic 1 or have the output be at high impedance when the Control is logic 0. As with the logic element discussed

previously, ideally, the TG has equivalent t_{PLH} and t_{PHL}. The capacitance seen by the input or total capacitance of a TG is given in Equation 2.11.

$$C_{total} = C_L + \frac{C_{ox}}{2} \tag{2.11}$$



(a) Symbol



(b) CMOS Schematic

Control	Input	Output
0	0	High Z
0	1	High Z
1	0	0
1	1	1

Table 2.7: Truth Table

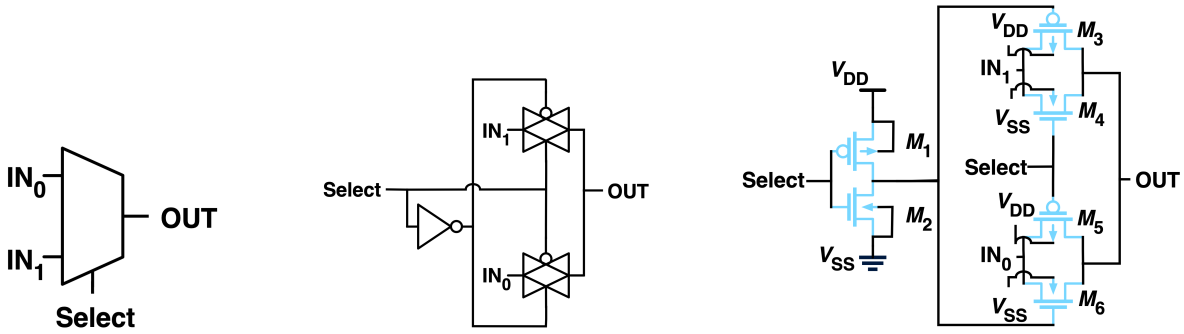
Figure 2.11: Transmission Logic Gate

2.1.5 Multiplexor

A multiplexor (MUX) with 2 inputs and 1 output can be made using two transmission gates and an inverter see Figure 2.12b. The FETs for the inverter would ideally be sized so that t_{PLH} and t_{PHL} are equivalent. The FETs for the transmission gates should also be sized for similar t_{PLH} and t_{PHL}. Although, the t_{PLH} and t_{PHL} for the inverter and transmission gates can be different. Ideally, the propagation delays for the two TGs will be the same for the inputs IN_0 and IN_1. A MUX uses a Select signal to determine which input to pass to the output. Logic operation for a 2 input MUX is shown in Table 2.8a, which can be simplified to Table 2.8b.

2.2 Conventional Delay Line Topologies Overview

CMOS delay lines can be either a tapped delay line or a single output delay line. A tapped delay line is constructed by connecting N-identical delay elements in series. The outputs of all the delay elements are then passed to a MUX, which is used to select the



(a) Symbol (b) Logic Gate Schematic (c) CMOS Schematic

Figure 2.12: 2 to 1 MUX

IN_0	IN_1	Select	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(a) Truth Table

Select	Output
0	IN_0
1	IN_1

(b) Simplified Truth Table

Table 2.8: 2 to 1 MUX Truth Tables

desired delay. Figure 2.13 is an example of a tapped inverter chain delay line. It is designed using N inverters where an inverter is paired with an adjacent inverter to form a buffer, as seen by the blue box in Figure 2.13. The delay of the buffer is equal to the propagation delay of the two inverters, which is indicated in Figure 2.13 by red text for the non-inverted output. Another example of a tapped delay line is shown in Figure 2.14, where the outputs are tapped after a pair of inverters that act as a buffer. The difference between the tapped delay lines in Figure 2.13 and Figure 2.14 is that the latter has only non-inverted outputs.

A single output delay line has only one output. An example of a single output delay line implementation is shown in Figure 2.15. In this type of implementation, the delay is generated by including or excluding gate delays based on the controlling signals D0, D1, D2, and D3.

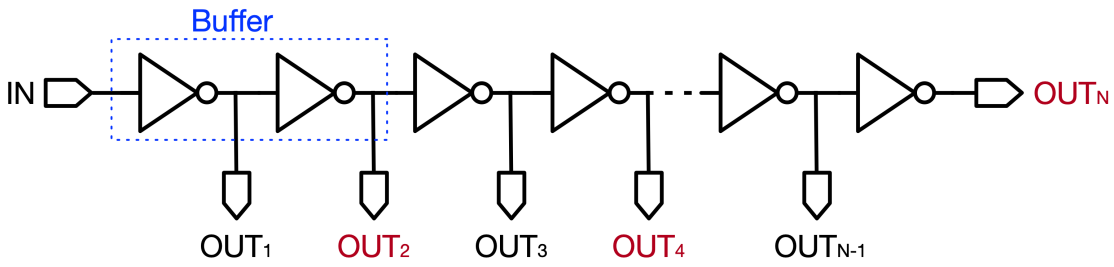


Figure 2.13: Tapped Inverter Chain Delay Line

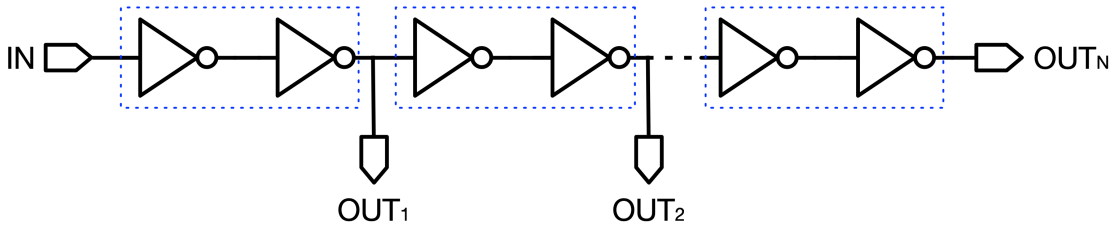


Figure 2.14: Tapped Buffer Chain Delay Line

2.3 Delay Variance Techniques

The overall delay through a circuit can be changed by altering the t_{PLH} and t_{PHL} of a single element or by switching in different delay paths. The delay paths could be chains of logic gates or a physically longer connection line. The delay of an element can be achieved through various techniques, including supply voltage modulation, a variable load, or controlling transistors [14].

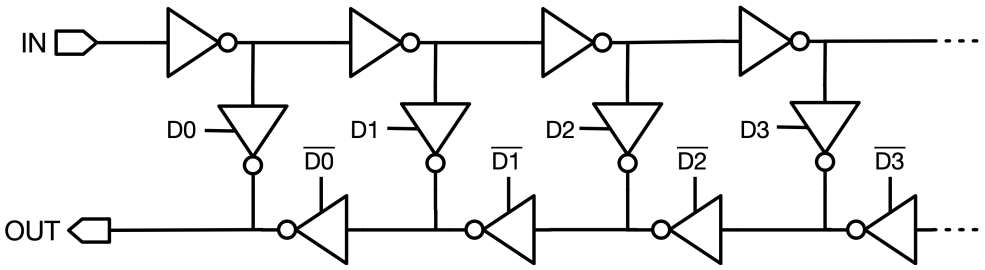


Figure 2.15: Single Output Inverter Matrix

2.3.1 Supply Modulation

Supply voltage modulation is the least desirable and involves changing the voltage level of the supply voltage V_{DD} (or the ground voltage V_{SS}) to induce a faster or slower delay. Increasing the supply voltage allows more current flow, resulting in a shorter delay. Decreasing the supply voltage leads to a longer delay due to less current flow. Figure 2.16 shows the results of varying the voltage supply to a buffer from 0.7 V to 1.5 V. The figure definitively shows that as the supply voltage V_{DD} is increased from 0.7 V to 1.5 V at 0.1V intervals t_{PLH} for the delay element is shorter in reference to half the maximum voltage of the input signal (500 mV). The timescale on the x-axis is in nanoseconds. It is harder to see that the same is true for t_{PHL} as the t_{PHL} for the various supply voltages do not vary as significantly since the pull-down network (NFET) is mainly impacted by changes to V_{SS}. One drawback to this method is that it is highly dependent on the output load's capacitance, which could be a problem when integrating into a larger circuit.

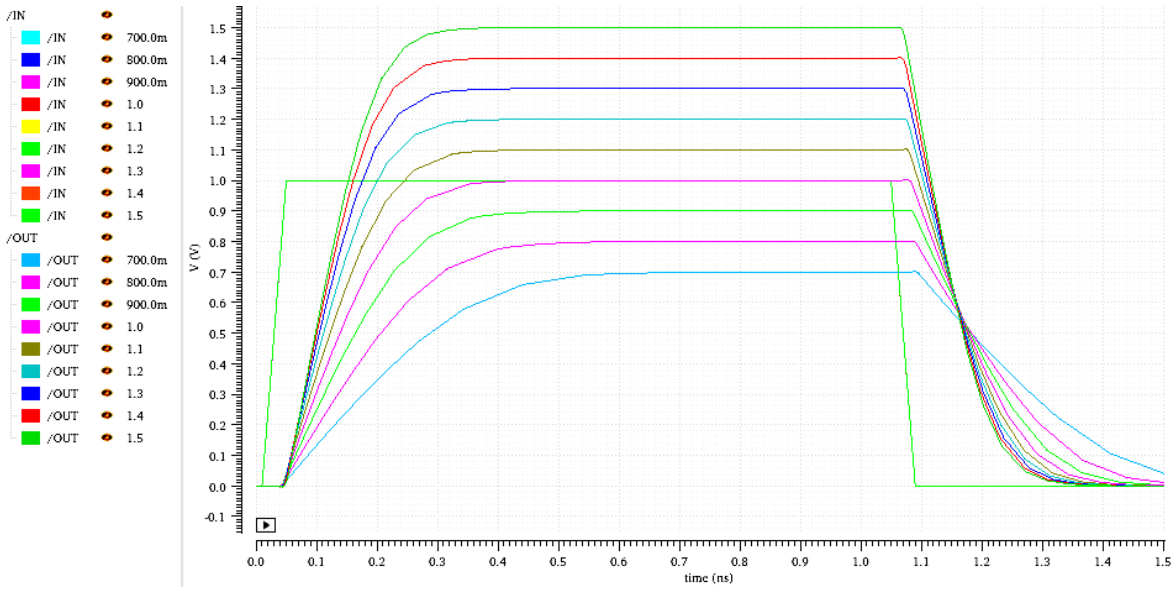


Figure 2.16: Simulation Results for Modulating V_{DD}

2.3.2 Variable Load

The delay of a buffer can be changed using a variable load capacitor (or shunt capacitor). Capacitors can be created using transistors; see Figure 2.17. The transistor M_1 will act as a variable resistor, and the transistor M_2 will act as a capacitor [14]. Changing the voltage V_{cntrl} changes the length of delay through the buffer by controlling charging/discharging current through the gate of M_2. A downside to this method is that there is a limited range of delay that can be generated. Figure 2.18 shows the impact of varying V_{cntrl} (denoted VC). The timescale on the x-axis is in nanoseconds. As V_{cntrl} is increased from 0 V to 1 V in steps of 0.25 V, the t_{PLH} can be seen to increase in duration.

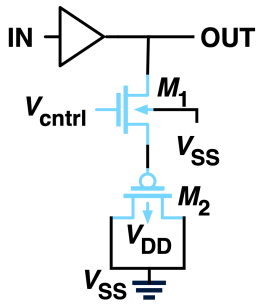


Figure 2.17: Buffer with a Variable Load

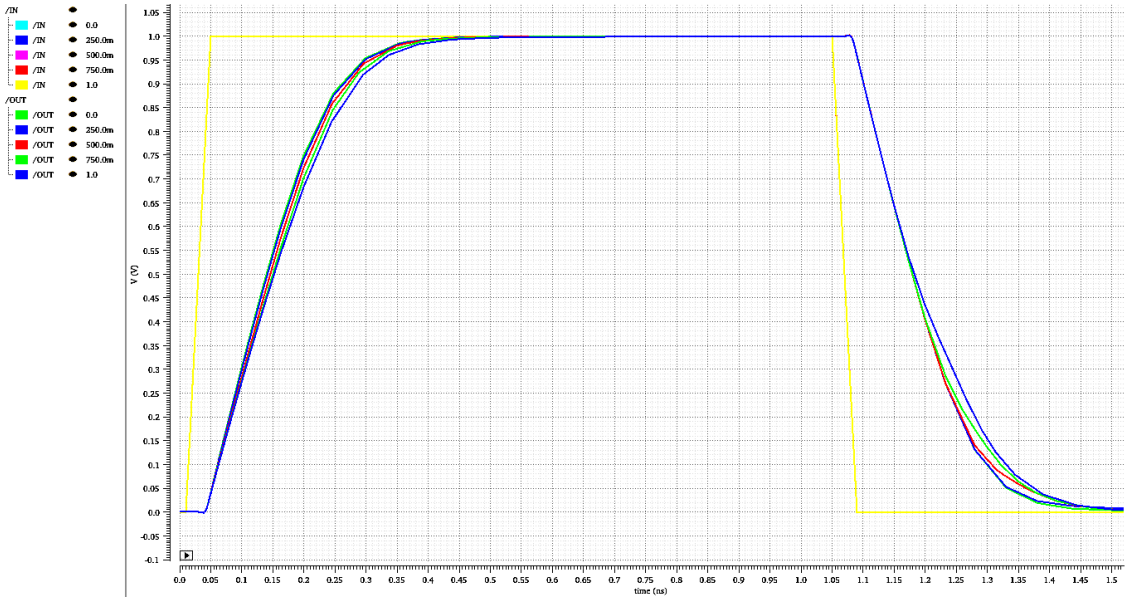


Figure 2.18: Simulation Results of Using a Variable Load with a Buffer

2.3.3 Controlling Transistors

Controlling the delay of an element can also be accomplished by adding controlling transistors(Figure 2.19). This method is commonly referred to as current starving. In Figure 2.19, the transistors M_5, M_6, M_7, and M_8 are added as control transistors for a

buffer that is created from two inverters. The controlling transistors are used to manipulate the effective drive resistance (R_{eq}) of the inverters, which alters the length of the delay. Lowering V_n and increasing V_p increases R_{eq}, subsequently increasing the length of the delay. The downside to this method, like the variable load technique, is that it can produce a limited delay range. This can be seen from the results listed in Table 2.9 and the simulation shown in Figure 2.20. The timescale of the x-axis in Figure 2.20 is in nanoseconds.

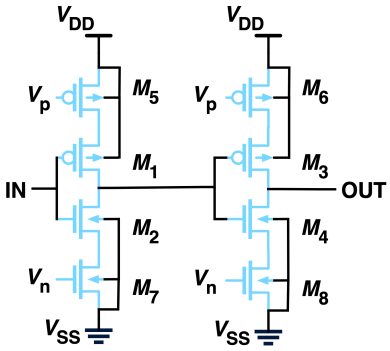


Figure 2.19: Buffer with Controlling Transistors

Table 2.9: Selected Controlling Transistor Results

Vn	Vp	Delay
0.5 V	0 V	208.300 ps
0.5 V	0.25 V	269.121 ps
0.5 V	0.5 V	766.313 ps
0.75 V	0 V	186.018 ps
0.75 V	0.25 V	249.152 ps
0.75 V	0.5 V	748.476 ps
1 V	0 V	183.940 ps
1 V	0.25 V	247.069 ps
1 V	0.5 V	746.373 ps

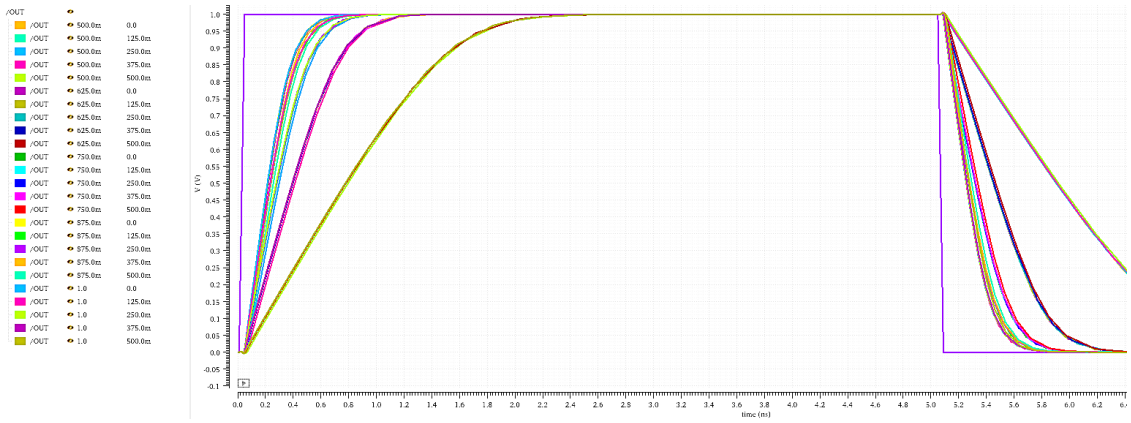


Figure 2.20: Simulation Results of Using Controlling Transistors on a Buffer

2.4 Programmable Delay Elements

CMOS variable delays are generally controlled by an analog signal, digital control bits that directly indicate the desired delay or a combination of both. Figure 2.21 depicts the

transfer function of a digitally-controlled delay line (DCDL). The x-axis is the decimal equivalent of the digital control bits for the desired delay. The y-axis is the value of the time delay of the output, T_d. The ideal time delay, T_d, can be calculated using Equation 2.12 for a linear variable delay element.

$$T_d = D_{min} + N_0 \times d_r \tag{2.12}$$

where d_r is the delay resolution, D_{min} is the minimum possible delay (corresponds to control bits equalling 0), and N_0 is the digital setting value which can be $0 \leq N_0 < N$ with N being the total number of control bits.

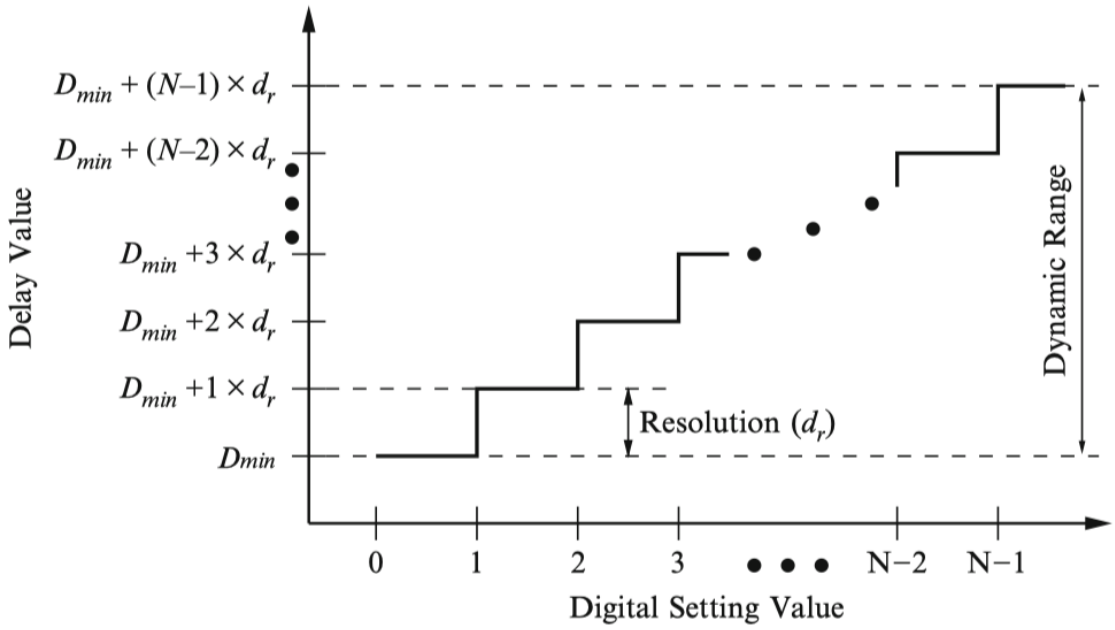


Figure 2.21: Digital Controlled Delay Line Transfer Function [9]

Programmable delay elements are characterized by more than simply their smallest delay resolution and overall dynamic range. They are also characterized by differential non-linearity (DNL), integral non-linearity (INL), and figure of merit (FOM). DNL is the deviation of the nth delay from its expected value. INL is the difference between the ideal delay length and the actual delay for a specific input code. The DNL and INL are frequently reported using the worst-case value divided by the LSB step. This singular value for the DNL and INL is referred to as the normalized DNL or normalized INL. The FOM is a figure that can be used to compare delay elements. Equation 2.13 is used for calculating DNL, n

is the current control in decimal, and $n+1$ is the following consecutive decimal control.

$$DNL(n) = \frac{V_{out}(n+1) - V_{out}(n)}{\text{ideal smallest delay step size}} - 1 \tag{2.13}$$

INL is found using

$$INL(n) = |V_{out,ideal}(n) - V_{out,actual}(n)| = \sum_{n=1}^{N} DNL(n) \tag{2.14}$$

where n is the current control step in decimal, and N is the total number of steps that could also be considered FOM. FOM is the delay range divided by the delay step; see Equation 2.15.

$$FOM = \frac{\text{Delay Range}}{\text{Delay Step}} \tag{2.15}$$

2.4.1 Shunt Capacitor Techniques

Shunt capacitor inverter (SCI) techniques leverage the fact that changing the shunt capacitance on the output of an inverter alters the propagation delay through it. Figure 2.22 is an example of a delay element that uses a shunt capacitor array to control the delay through the first inverter. The output of the first inverter is connected to MOSFETs that act as capacitors by NFET switches. The switches are controlled by the signals D_0 to D_N, which vary the load capacitance and alter the t_{PHL} and t_{PLH} of the inverter. Then, the signal is again inverted back to its original polarity when passing through the output inverter. Shunt capacitors or varactors are still being researched and combined with other delay techniques to improve performance [18–20]. One downside to SCI techniques is that they are often limited by the delay range that can be produced and is susceptible to process variation [6].

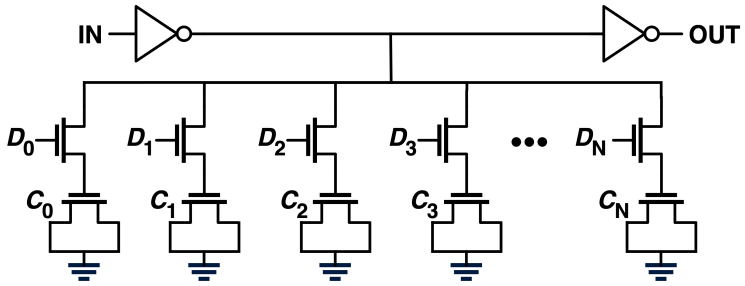


Figure 2.22: Shunt Capacitor Variable Delay

2.4.2 Current Starvation Techniques

Current starved inverter (CSI) techniques are often used with two inverters to create a buffer with digitally controlled variable delay. Similarly to the shunt capacitor technique in the previous section, current starving also focuses on altering the t_{PHL} and t_{PLH} through the first inverter, see M_{12} and M_{13} in Figure 2.23. The second inverter M_{14} and M_{15} are simply used to return the signal to its original polarity. In the figure, M_1 to M_5 are used to set the current supplied to the current mirrors (M_7 to M_{11}). M_6 is simply used to set the minimum amount of current supplied correlating to the longest possible delay [21]. The change in the amount of delay comes from turning on or off the PFETs with the signals D_0 to D_4. Current starving techniques have been one of the more popular methods for creating a pro-

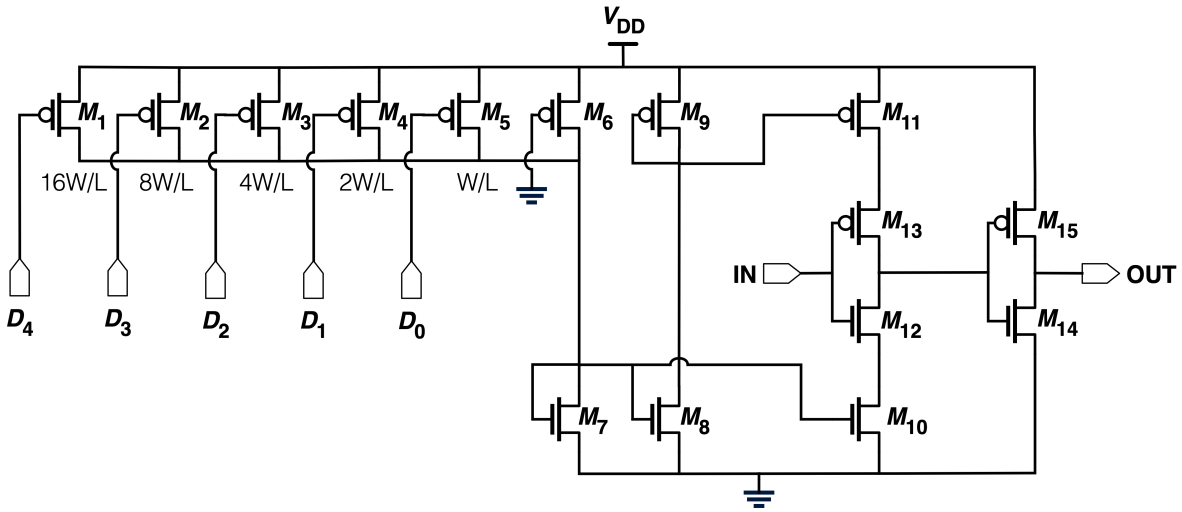


Figure 2.23: Current Starved Variable Delay

grammable delay and have been implemented in conjunction with other techniques in recent

years to improve performance [20–24]. Although, CSI techniques have similar downsides to SCI, including susceptibility to process variation, limited delay range, and reduced linearity performance over the delay range.

2.4.3 Variable Resistor Techniques

A variable resistor array like the two previous techniques discussed is used the control the t_{PHL} and t_{PLH} through the first inverter, see M_1 and M_2 in Figure 2.24. A PFET stack can be attached to the source terminal of M_2, and an NFET stack can be attached to the source terminal of M_1 as shown in Figure 2.24. Turning off transistors in the variable resistor array increases the effective resistance, which in turn increases the time delay. Turning the resistors on has the opposite effect [25]. Then, the second inverter M_3 and M_3 are used to invert the signal back to match the polarity of the input signal. Although the idea behind

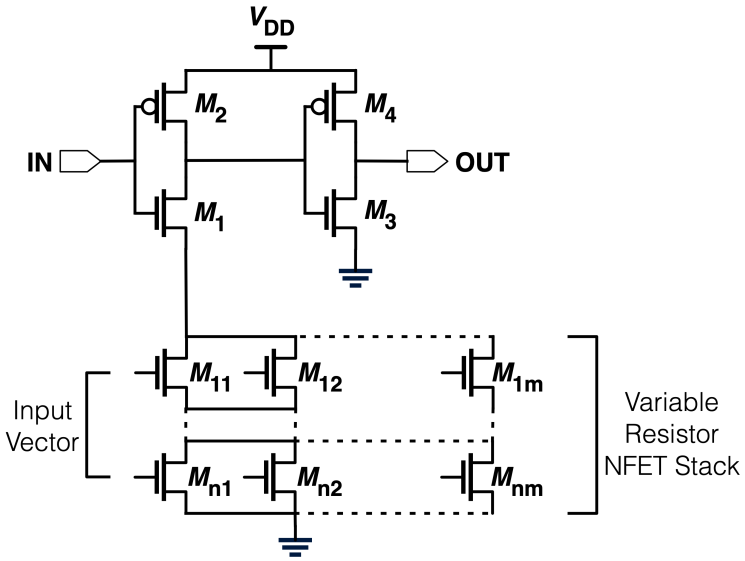


Figure 2.24: Variable Resistor Delay

this method appears simple, the parasitic capacitance changes as transistors are turned on and off, causing difficulties with achieving linear delay steps. The impact of the parasitic capacitance is a difficult challenge, and as such, resistor arrays have not had much traction in recent years [25]. However, implementing other components that induce a variable resistance, like a memristor, is showing some promise [26].

2.4.4 Comparisons of Delay Techniques

The three main digital programmable delay techniques that have maintained traction over the years have been inverter chains, shunt capacitor inverters, and current-starved inverters. SCI delay elements outperform other techniques used in the same-size process in terms of delay resolution. In addition, CSI delay elements generally take up less area than their counterparts in the same process. While cascaded inverters usually have a better delay range, linearity, and robustness to process variation than SCI and CSI [6]. Table 2.10 shows data given by publications on programmable/variable delay designs. N/A is used to mean not available in Table 2.10.

Table 2.10: Comparison of Current Programmable Delay Lines

Type	Process (nm)	Res. (ps)	Range	FOM	Area	DNL (norm.)	INL (norm.)
SCI [18]	350	1.43	40 ps	27.972	850 μm^2	0.37	0.19
SCI & Body Bias [19]	65	0.004	30 ps	7500	0.514 mm^2	N/A	N/A
SCI & CSI [20]	130	340	50 ns	147.05	2.25 mm^2	N/A	N/A
CSI [21]	350	40	400 ps	10	450 μm^2	N/A	N/A
CSI [22]	180	2	320 ps	160	5000 μm^2	13.61	36.15
DLL [24]	350	5	8.6 ns	1720	N/A	N/A	N/A
Resistor Array [25]	180	1	50 ps	50	N/A	1.37	1.04
Cascaded Inverters [27]	350	5	300 ps	60	0.36 mm^2	N/A	N/A

Chapter 3

Proposed ICV2 Programmable Delay

The previous chapter discussed various topologies and techniques for designing programmable delay elements. The proposed ICV2 programmable delay element was designed for delay range, linearity, and robustness to process variation in IC fabrication. This chapter covers the functions and features of the different blocks used in this proposed programmable delay element. The results included in this chapter are based on extensive simulations as the fabricated IC with this proposed design was recently wire-bonded, and test structures are currently being designed. The proposed ICV2 programmable delay element is simulated in Cadence Virtuoso GF45RFSOI 45nm technology.

3.1 Block Diagrams of Proposed ICV2 Programmable Delay

The high-level block diagram of the proposed programmable delay is shown in Figure 3.1. It consists of two main blocks, the primary delay circuit shown in Figure 3.2 and the tuning delay circuit shown in Figure 3.3. The purpose of the primary delay circuit is to create time delays to the input signal in a linear fashion. It is composed of buffer chains and MUXs used to switch between the path with the delay buffer and the path with no delay using a select signal SN with N, the specific controlling bit of a select register. In Figure 3.2 and

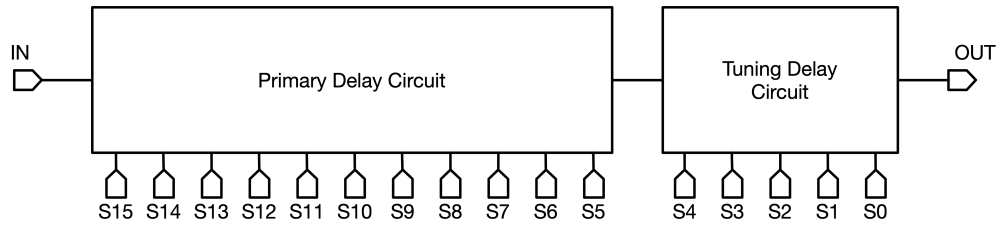


Figure 3.1: ICV2 Programmable Delay Block Diagram

Figure 3.3, the number indicated in the buffer symbol is the number of delay steps the buffer is equivalent to.

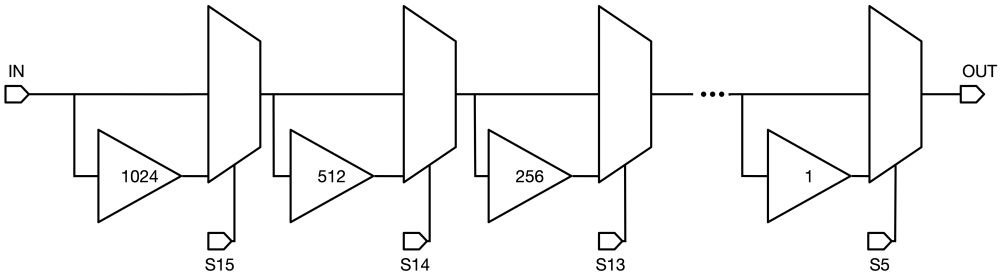


Figure 3.2: ICV2 Programmable Delay Primary Block Diagram

Initial simulations showed that the performance of the primary delay circuit alone would not produce the desired performance, so the tuning delay circuit was added. The goal of adding the tuning delay circuit was to increase the linearity of the delay line and provide a method to calibrate the programmable delay in cases of process variation and temperature changes. The tuning circuit is similar to the primary delay circuit. However, it differs in that the buffer chains are only one or two delay steps, while the buffer chains in the main delay element start at the equivalent of 1024 delay steps in the first buffer and is halved to 512 delays steps in the buffer chain after the MUX following the 1024 delay step buffer chain. This pattern of halving continues until the single delay step buffer chain section is reached.

3.2 Building Blocks

The higher-level elements shown in the block diagrams of the previous section were made from smaller elements. The two most basic building blocks were the transmission gate and

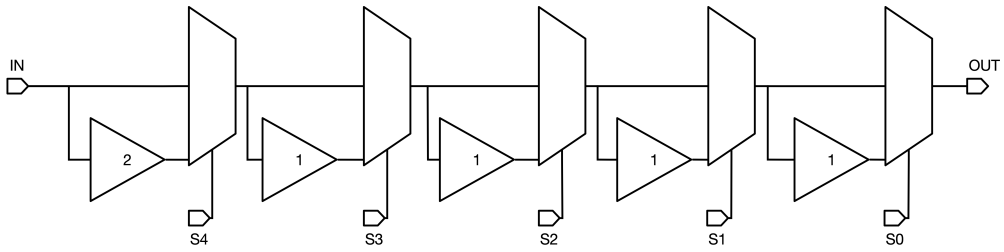
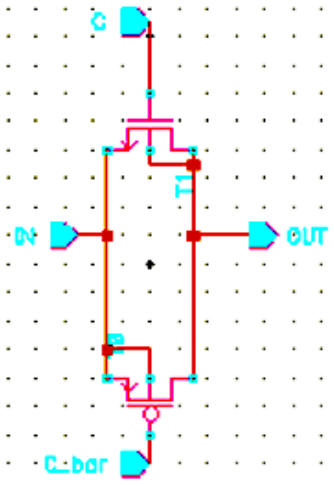


Figure 3.3: ICV2 Programmable Delay Tuning Block Diagram

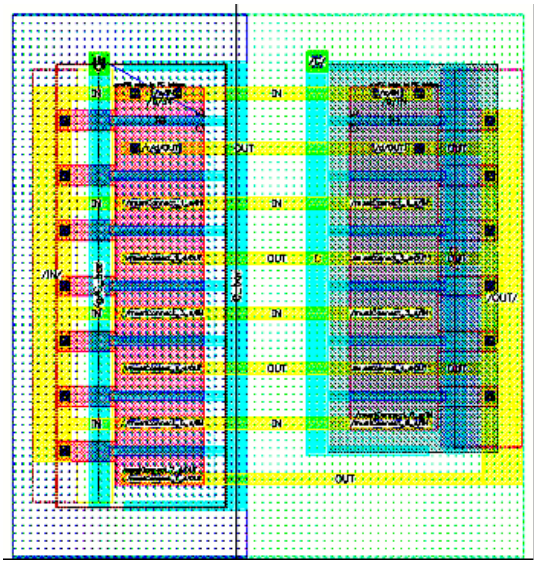
inverter, which were eventually used to create the buffers and MUXs.

3.2.1 Transmission Gate

The schematic and layout of the transmission gate are shown in Figure 3.4. The TG was created using body contact FETs. However, in the course of making various test building blocks, it was not initially caught that the FETs used were essentially converted to non-body contact FETs. This mishap does not significantly impact the functioning of the transmission gate, as seen by the simulations in Figure 3.5 and Figure 3.6. A transmission gate was used as the single delay step buffer chain.



(a) Transmission Gate Schematic

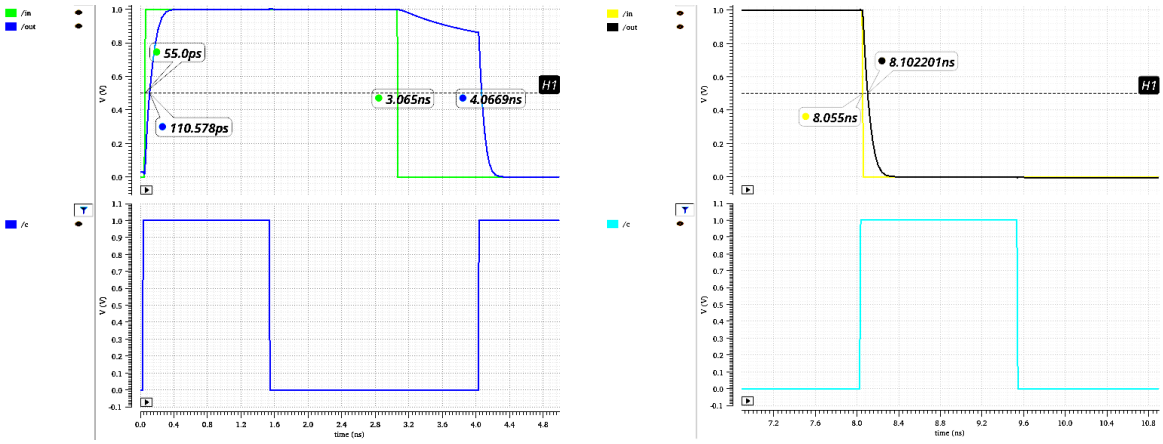


(b) Transmission Gate Layout

Figure 3.4: Implemented Transmission Gate

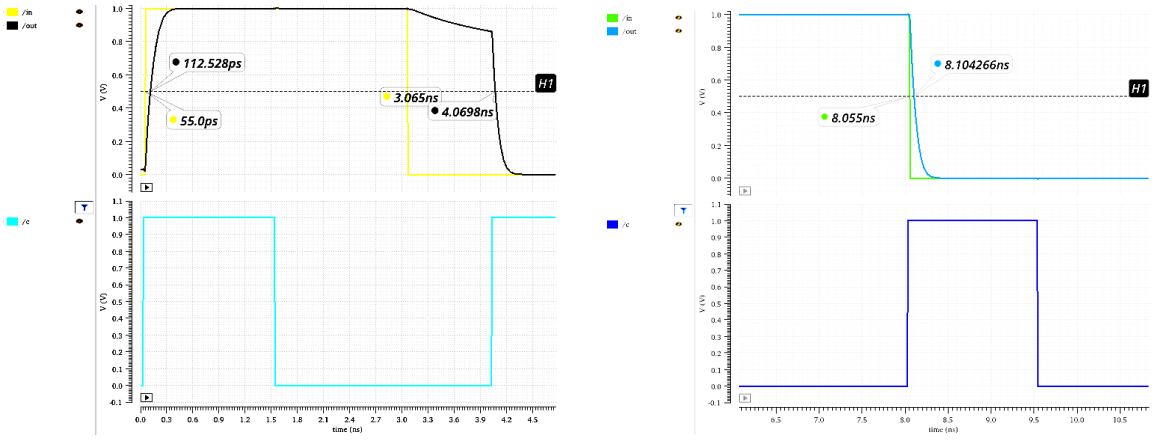
Figure 3.5 shows the simulation using only schematic, while Figure 3.6 depicts the sim-

ulations after layout and RC extraction was done on the design. Comparing Figure 3.5 and Figure 3.6 to Table 2.7, when C is high, then the TG passes the input signal to the output; otherwise, the output is at high-z as expected. There are minor differences in the t_{PLH} and t_{PHL} between the two figures, which is expected when going from simulating a schematic to the extracted layout, which includes the parasitic resistances and capacitances. For the schematic simulation $t_{PLH} = 55.578\ ps$ and $t_{PHL} = 47.201\ ps$. For the extracted simulation $t_{PLH} = 57.528\ ps$ and $t_{PHL} = 49.266\ ps$, with a difference of $8.262\ ps$.



(a) TG Schematic Simulation L → H (b) TG Schematic Simulation H → L

Figure 3.5: TG Schematic Simulations

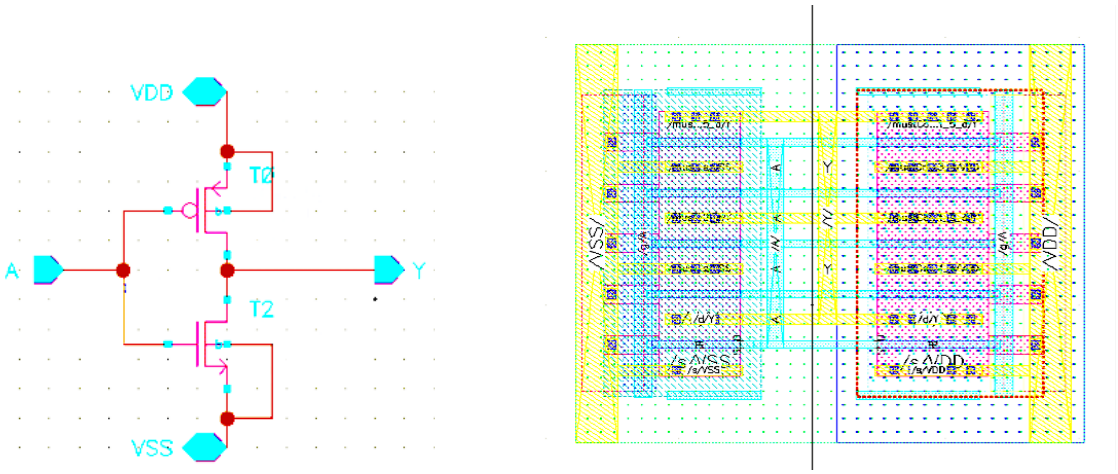


(a) TG Extracted Simulation L → H (b) TG Extracted Simulation H → L

Figure 3.6: TG Extracted Simulations

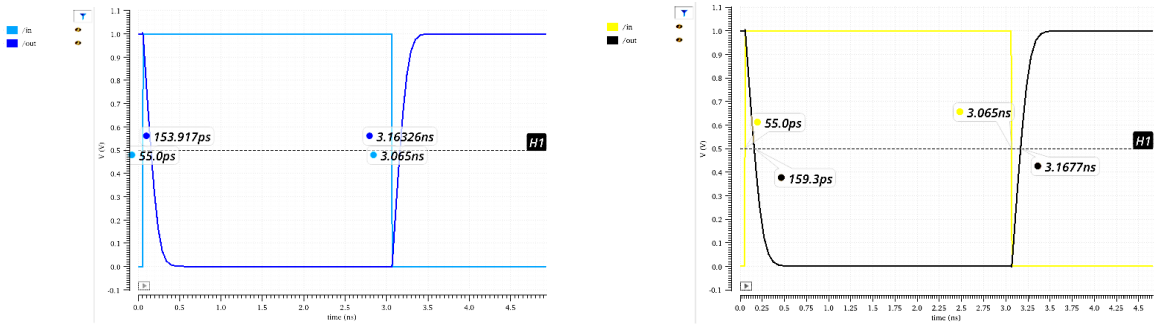
3.2.2 Inverter

The schematic and layout of the inverter are shown in Figure 3.7. The inverter was designed using a body contact NFET and PFET that were sized for $R_n \approx R_p$ in order to have nearly symmetric t_{PLH} and t_{PHL} during the schematic simulations. For the schematic simulation $t_{PLH} = 98.92\ ps$ and $t_{PHL} = 98.26\ ps$. For the extracted simulation $t_{PLH} = 98.93\ ps$ and $t_{PHL} = 102.7\ ps$, with a difference of 3.77 ps.



(a) Inverter Schematic (b) Inverter Layout

Figure 3.7: Inverter

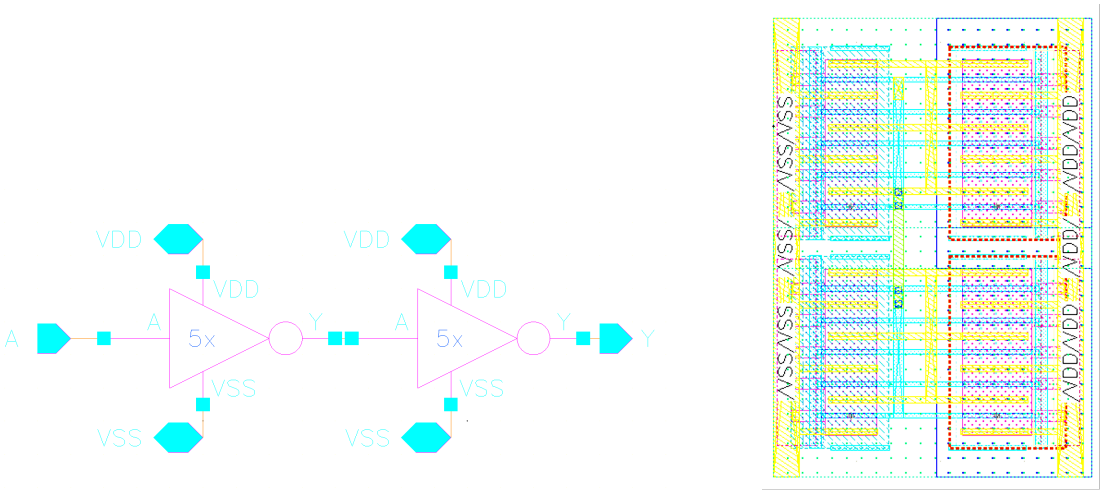


(a) Inverter Schematic Simulation (b) Inverter Extracted Simulation

Figure 3.8: Inverter Simulations

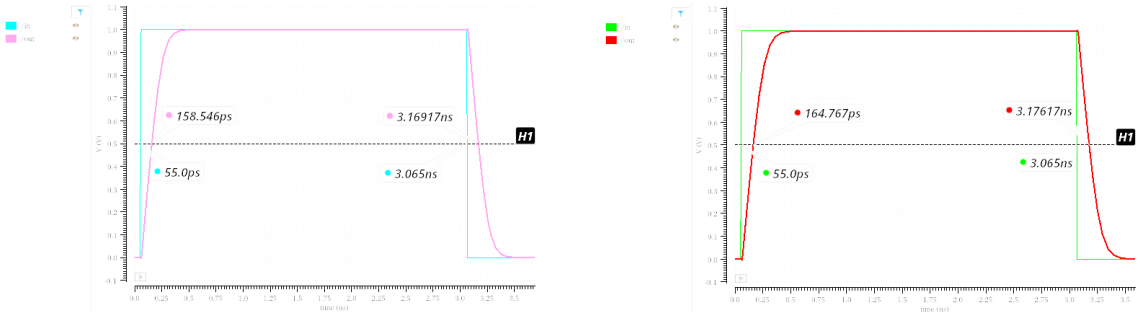
3.2.3 Buffers

The schematic and layout of the two delay step buffer are shown in Figure 3.9. The buffer was designed using two of the inverter described in the previous section. For the schematic simulation $t_{PLH} = 103.546\ ps$ and $t_{PHL} = 104.17\ ps$. For the extracted simulation $t_{PLH} = 109.767\ ps$ and $t_{PHL} = 111.17\ ps$, with a difference of $1.403\ ps$. This is an improvement over the difference of t_{PLH} and t_{PHL} seen for the extracted simulation of the inverter.



(a) Buffer Schematic (b) Buffer Layout

Figure 3.9: Buffer

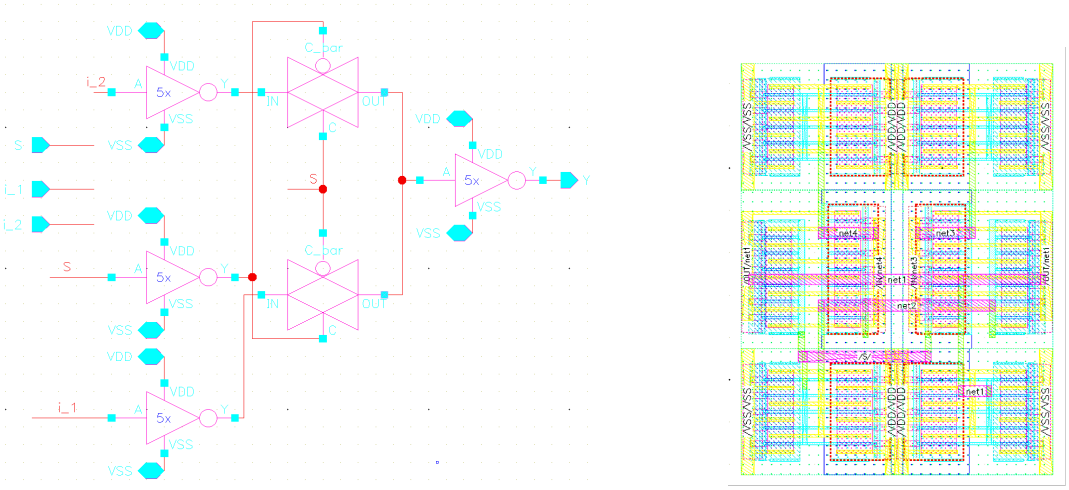


(a) Buffer Schematic Simulation (b) Buffer Extracted Simulation

Figure 3.10: Buffer Simulations

3.2.4 Multiplexor

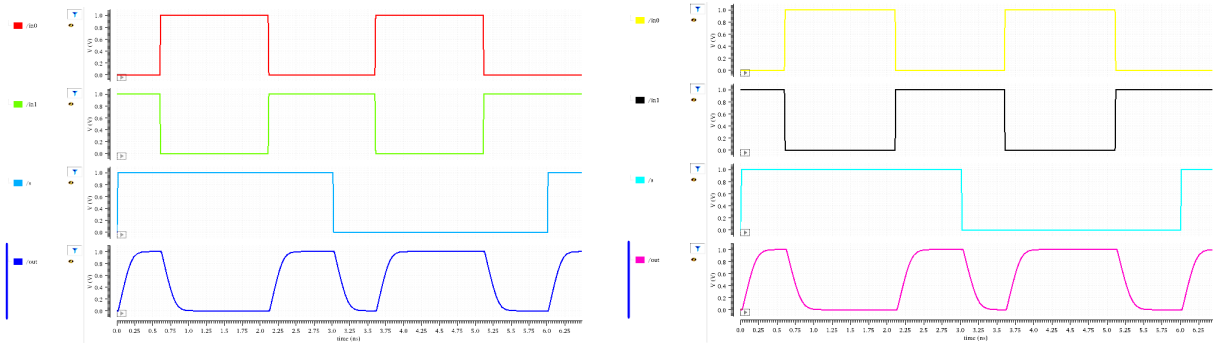
The schematic and layout of the MUX are shown in Figure 3.11. The MUX was designed using two TGs, and four inverters. One inverter was necessary for use in inverting the path select signal for proper operation of the MUX. The inverters on i_1, i_2, and the output were added so that the capacitive load from one element in the circuit to the next was similar. Comparing the simulations of the designed MUX to Table 2.8b. It is seen that the MUX proposed in this section operates normally.



(a) MUX Schematic

(b) MUX Layout

Figure 3.11: MUX



(a) MUX Schematic Simulation

(b) MUX Extracted Simulation

Figure 3.12: MUX Simulations

3.3 Proposed ICV2 Programmable Delay

The final layout of the proposed ICV2 programmable delay is shown in Figure 3.13. The layout phase of the ICV2 programmable delay involved a few attempts at the layout of the entire delay element. This was because the individual buffers and the MUXs need to be as close together as possible to reduce the amount of parasitic effects in the connecting traces. Additionally, the connections for the path select signals needed to be accessible in an ordered manner so that integration into larger on-chip circuits could be streamlined. The path select signals are on the right-hand side of the figure, and the input and output signals are on the left.

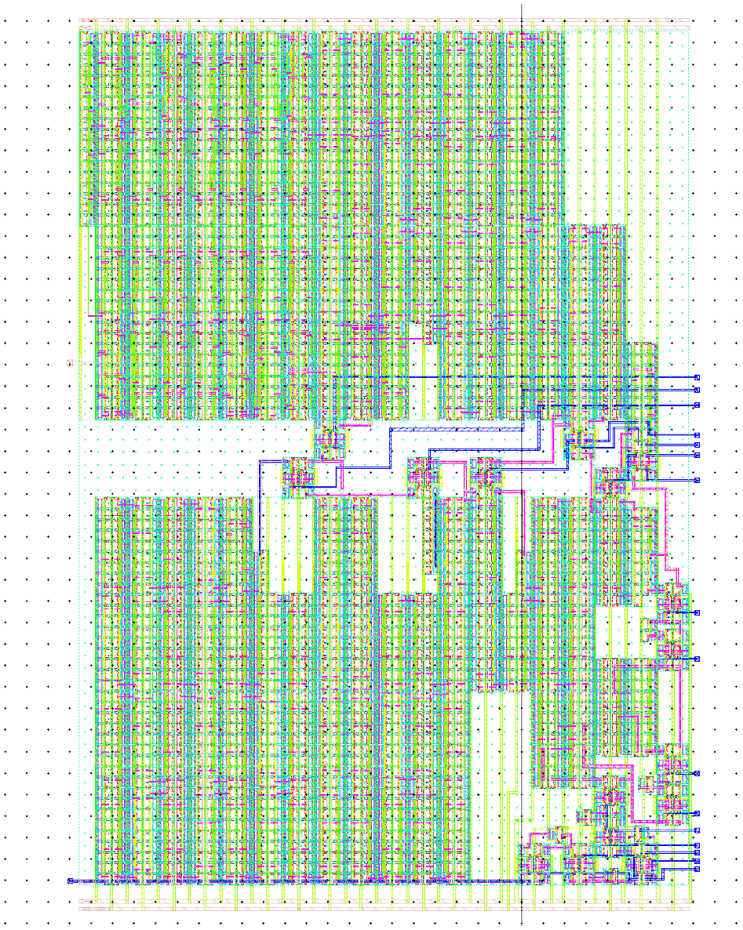


Figure 3.13: ICV2 Programmable Delay Layout

3.4 Simulation Results

After the layout of the proposed ICV2 programmable delay was finished, it was extracted to obtain the parasitic resistances and capacitances to use in simulation. The results included in this section are simulations of the programmable delay before using the tuning circuit and then with the tuning circuit. The tuning for the proposed ICV2 PD was done by sorting the primary delay circuit output delays from smallest to largest. Where needed, the tuning bits were implemented to achieve as uniform as possible LSB step sizes between the delays from the primary delay circuit. The control for the tuned simulation is in reference to a control value that would map to a select register bit combination in a lookup table. The lookup table would contain the select register combinations in increasing order.

3.4.1 Linearity Simulation

The supply voltage used was 1V, and the input signal logic high was also 1V. A thermometer code simulation steps through all of the possible control code combinations. In this case, that means select register bits S15 to S5, the primary delay circuit control bits. The tuning delay circuit select register bits S4 to S0 were simulated apart from the primary delay circuit bits to save time during simulations. Adding the tune gate delay(s) to the selected primary circuit delay was approximately additive in simulation tests. The tuning bits could have also been used to slightly extend the overall range of the proposed delay element.

Figure 3.14 is the result of the thermometer code simulation, and contains two plots. The blue plot in Figure 3.14 is of the proposed ICV2 PD without any tuning. Specifically, the blue plot is of the output delays of the select signal in the exact order that would happen when using the MSBs of the controlling select register as a bit counter. The red plot is the tuned version of the proposed ICV2 PD. Figure 3.15 shows the DNL of the proposed ICV2 PD. The y-axis for the DNL plot is in reference to the LSB step size since calculating the DNL at a specific point takes the difference between a control step and the previous one and divides it by the LSB step size. The plot shows distinctive peaks for certain control sequences, which correlate to the control sequences in Figure 3.14 that showed a sharp decrease in the

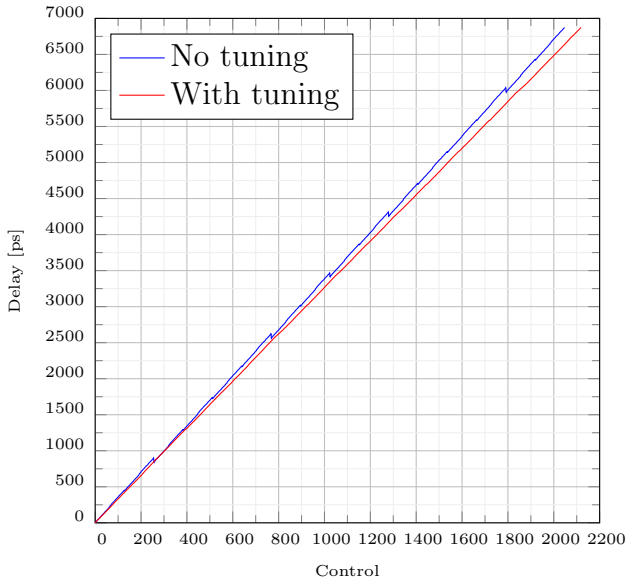


Figure 3.14: ICV2 Programmable Delay Thermometer Code Simulation

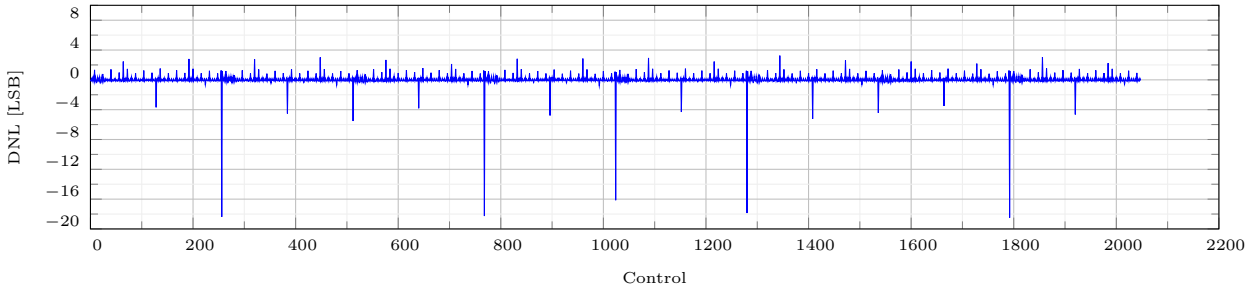


Figure 3.15: ICV2 Programmable Delay DNL without Tuning

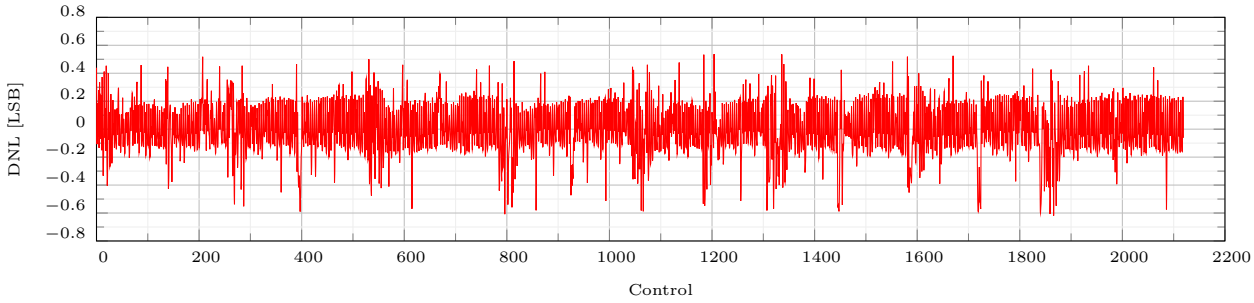


Figure 3.16: ICV2 Programmable Delay DNL with tuning

delay length instead of a steady increase in delay between steps. The worst case $|DNL|$ is approximately 18 LSB, which is far from ideal. Figure 3.16 shows the DNL of the proposed ICV2 PD when tuned. The tuned version has a worst-case $|DNL|$ of approximately 0.62 LSB, a significant improvement compared to the untuned version.

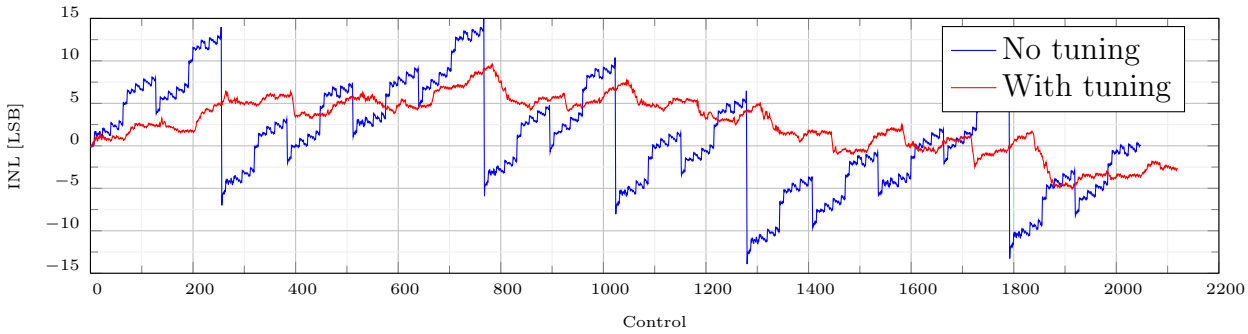


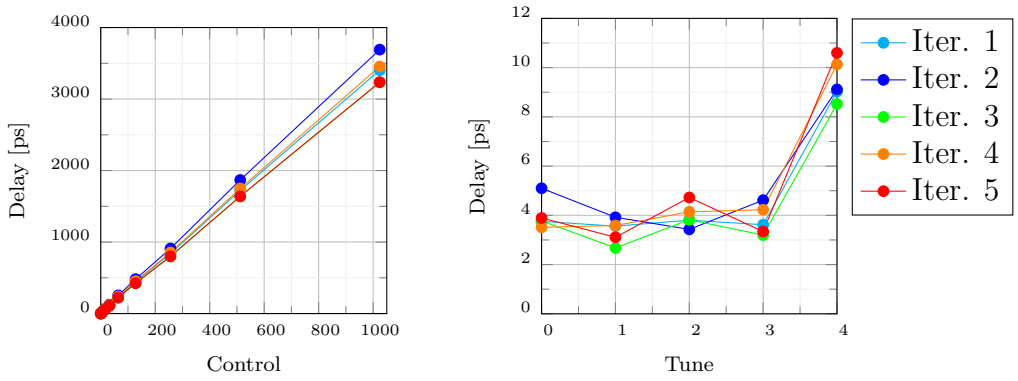
Figure 3.17: ICV2 Programmable Delay INL

Figure 3.17 shows the INL for both versions of the proposed ICV2 PD. The INL of both versions of the proposed ICV2 PD are not ideal. Ideally, the INL would fluctuate around and close to 0. The tuned version was tuned in an attempt to have LSB step sizes of at least 3.5 ps, which neglected to have step sizes slightly smaller, which is why for most of the plot, it is above 0. However, in our radar application, the INL performance is not much of a concern since the difference between the individual steps leads to a worst-case DNL of less than one LSB.

3.4.2 Monte Carlo Simulation

A Monte Carlo analysis is based on statistical distributions, and for each run, it calculates every process variation and mismatch parameter randomly according to a statistical model. The simulation allows for testing the process variation and mismatching between devices in a single chip based on previous production data. A Monte Carlo simulation analysis was conducted using the Cadence toolset. The simulation was conducted five times due to time and server disk space limitations. This simulation would have ideally been run closer to 1000 times (or another "a large number of times") so that a more reliable mean (μ) and standard deviation (σ) of the individual buffers could be obtained. Reliable is used to mean a more accurate and precise value.

Figure 3.18a and Figure 3.18b depict the different iterations of the Monte Carlo analysis in comparison to each other. Figure 3.18a shows the Monte Carlo of the primary delay circuit, and Figure 3.18b is of the tuning circuit. The exact values for each point plotted



(a) Primary Delay Circuit (b) Tuning Delay Circuit

Figure 3.18: Proposed ICV2 PD Monte Carlo Sim

Table 3.1: Monte Carlo Primary Delay Results

Control	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	μ	σ
0	0.0 ps	0.0 ps	0.0 ps	0.0 ps	0.0 ps	0.0 ps	0.0 ps
1	2.574 ps	3.558 ps	2.217 ps	2.857 ps	2.670 ps	2.775 ps	0.496 ps
2	8.281 ps	6.967 ps	6.432 ps	7.398 ps	5.855 ps	6.987 ps	0.927 ps
4	14.353 ps	15.059 ps	13.947 ps	14.363 ps	13.925 ps	14.329 ps	0.459 ps
8	31.793 ps	35.101 ps	30.499 ps	34.215 ps	31.378 ps	32.597 ps	1.964 ps
16	58.411 ps	65.227 ps	56.164 ps	60.431 ps	57.348 ps	59.516 ps	3.557 ps
32	112.349 ps	125.117 ps	108.526 ps	117.684 ps	110.577 ps	114.851 ps	6.670 ps
64	231.256 ps	254.095 ps	221.076 ps	237.996 ps	223.286 ps	233.542 ps	13.305 ps
128	444.584 ps	481.228 ps	423.155 ps	453.484 ps	424.145 ps	445.319 ps	23.960 ps
256	839.729 ps	909.825 ps	797.652 ps	854.398 ps	800.683 ps	840.457 ps	45.883 ps
512	1722.580 ps	1867.993 ps	1638.123 ps	1749.899 ps	1638.801 ps	1723.479 ps	94.920 ps
1024	3403.751 ps	3691.062 ps	3232.576 ps	3455.117 ps	3235.879 ps	3403.677 ps	188.857 ps

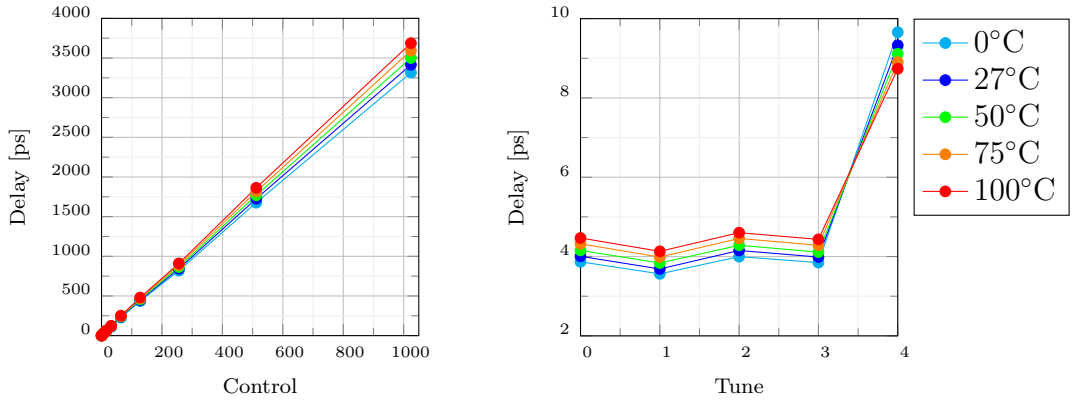
Table 3.2: Monte Carlo Tuning Delay Results

Tune	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	μ	σ
0	3.756 ps	5.099 ps	3.772 ps	3.512 ps	3.894 ps	4.007 ps	0.626 ps
1	3.562 ps	3.922 ps	2.677 ps	3.587 ps	3.109 ps	3.372 ps	0.484 ps
2	3.802 ps	3.430 ps	3.822 ps	4.142 ps	4.726 ps	3.984 ps	0.485 ps
3	3.620 ps	4.616 ps	3.196 ps	4.229 ps	3.342 ps	3.801 ps	0.604 ps
4	9.027 ps	9.116 ps	8.524 ps	10.131 ps	10.599 ps	9.479 ps	0.855 ps

are included in Table 3.1 and Table 3.2 for the primary and tuning circuits, respectively. In addition, the mean and standard deviation of the individual buffer gate delays are included in the final columns of Table 3.1 and Table 3.2 for the primary and tuning circuits, respectively.

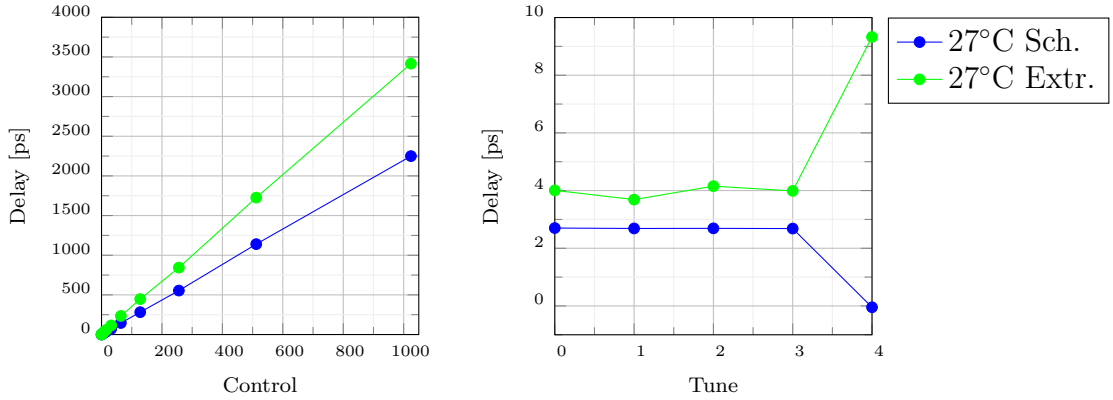
3.4.3 Temperature Simulation

The simulation was conducted with a sampling of temperatures 0°C, 27°C, 50°C, 75°C, and 100°C. Figure 3.19a and Figure 3.19b show the temperature simulation results for the primary delay circuit and the tuning circuit, respectively. Additionally, the exact values for each point plotted in Figure 3.19a and Figure 3.19b are included in Table 3.3 and Table 3.4 for the primary and tuning circuits respectively. The mean and standard deviation of the individual buffer gate delays over the sampled temperatures are included in the final columns of Table 3.3 and Table 3.4.



(a) Primary Delay Circuit (b) Tuning Delay Circuit

Figure 3.19: Proposed ICV2 PD Temperature Extracted Simulations



(a) Primary Delay Circuit (b) Tuning Delay Circuit

Figure 3.20: Proposed ICV2 PD Schematic and Extracted Temperature Sim Comparisons

Table 3.3: Primary Delay over Temperature Results

Control	0°C	27°C	50°C	75°C	100°C	μ	σ
0	0.0 ps	0.0 ps	0.0 ps	0.0 ps	0.0 ps	0.0 ps	0.0 ps
1	2.821 ps	2.968 ps	3.107 ps	3.233 ps	3.378 ps	3.101 ps	0.218 ps
2	6.788 ps	6.916 ps	7.058 ps	7.211 ps	7.349 ps	7.065 ps	0.224 ps
4	13.910 ps	14.274 ps	14.590 ps	14.926 ps	15.240 ps	14.588 ps	0.524 ps
8	31.791 ps	32.556 ps	33.180 ps	33.868 ps	34.523 ps	33.184 ps	1.072 ps
16	58.304 ps	59.838 ps	61.10 ps	62.514 ps	63.884 ps	61.128 ps	2.188 ps
32	112.004 ps	115.102 ps	117.740 ps	120.629 ps	123.532 ps	117.801 ps	4.521 ps
64	227.922 ps	234.304 ps	239.756 ps	245.732 ps	251.734 ps	239.890 ps	9.339 ps
128	434.995 ps	447.027 ps	457.363 ps	468.676 ps	480.055 ps	457.623 ps	17.676 ps
256	819.556 ps	843.961 ps	864.871 ps	887.737 ps	910.733 ps	865.371 ps	35.762 ps
512	1676.099 ps	1726.206 ps	1769.045 ps	1816.016 ps	1863.060 ps	1770.085 ps	73.338 ps
1024	3315.593 ps	3415.231 ps	3500.231 ps	3592.932 ps	3687.261 ps	3502.250 ps	145.662 ps

Table 3.4: Tuning Delay over Temperature Results

Tune	0°C	27°C	50°C	75°C	100°C	μ	σ
0	3.867 ps	4.006 ps	4.155 ps	4.320 ps	4.469 ps	4.163 ps	0.240 ps
1	3.564 ps	3.687 ps	3.837 ps	3.988 ps	4.131 ps	3.841 ps	0.227 ps
2	3.997 ps	4.153 ps	4.284 ps	4.457 ps	4.602 ps	4.299 ps	0.239 ps
3	3.848 ps	3.988 ps	4.112 ps	4.283 ps	4.433 ps	4.133 ps	0.232 ps
4	9.656 ps	9.327 ps	9.114 ps	8.903 ps	8.736 ps	9.147 ps	0.361 ps

The nominal temperature used throughout the simulation process was 27°C. The results of the schematic and extracted layout simulation were compared to show the impact of the parasitic effects during the design process; see Figure 3.20a and Figure 3.20b for the primary delay circuit and tuning circuit, respectively.

3.4.4 Discussion

The simulation results for the proposed ICV2 PD show that the performance significantly increases by implementing the tuning circuit and implementing a lookup table for the desired time delay. The overall linearity, DNL, and INL improved. The DNL improved to the point that it is less than one LSB for the worst case. The Monte Carlo Analysis and simulation over various temperatures showed that both negatively impacted the delay element overall. The impacts were most noticeable on the largest delay buffers, which is expected since larger buffers mean more MOSFETs, and the impacts seen for the smaller buffers compound for the larger ones. The plots in Figure 3.20a and Figure 3.20b at 27°C for the schematic

compared to the extracted layout exemplify the impact parasitic effects in the layout impact the overall functionality. The impacts of the parasitic effects can also partially explain the variance in results for the Monte Carlo analysis and the temperature simulation results. Table 3.5 summarizes the characteristics of the proposed ICV2 PD.

Table 3.5: ICV2 PD Results Summary

PD	Process (nm)	Res. (ps)	Range	FOM	Area	DNL (norm.)	INL (norm.)
ICV2	45	3.4	6.873 ns	2047	0.03 mm^2	18.504	13.889
ICV2 tuned	45	3.24	6.873 ns	2119	0.03 mm^2	0.619	9.64

This proposed PD contributes to the field of variable delay elements by being a tunable wide range, linear, 3.4ps resolution, 6.8ns delay range, inverter chain based digitally controlled delay line.

Chapter 4

Proposed Programmable Delays ICV3 & ICV4

In the previous chapter, the proposed programmable delay ICV2 was discussed. The proposed programmable delays ICV3 and ICV4 were modeled after it. However, these two delay elements were designed using a synthesized Verilog netlist. This chapter covers the functions and features of these proposed programmable delay elements. The results in this chapter are based on extensive simulations as the IC with the proposed programmable delay ICV3 is still in the fabrication process. The proposed programmable delay ICV4 is an improvement on its predecessor proposed ICV3 and will be taped out in the future. The ICV3 and ICV4 programmable delays are simulated in Cadence Virtuoso GF45RFSOI 45nm technology.

4.1 Block Diagram of Proposed Programmable Delays ICV3 & ICV4

The block diagram used to design the proposed ICV3 programmable delay is shown in Figure 4.1, and the block diagram for the proposed ICV4 programmable delay is shown in Figure 4.2. The proposed programmable delays are composed of buffer chains and MUXs

with the same selection and switching mechanism used for the proposed programmable delay element discussed in the previous chapter. In Figure 4.1 and Figure 4.2, the number indicated in the buffer symbol is the number of delay steps the buffer is supposed to be equivalent to.

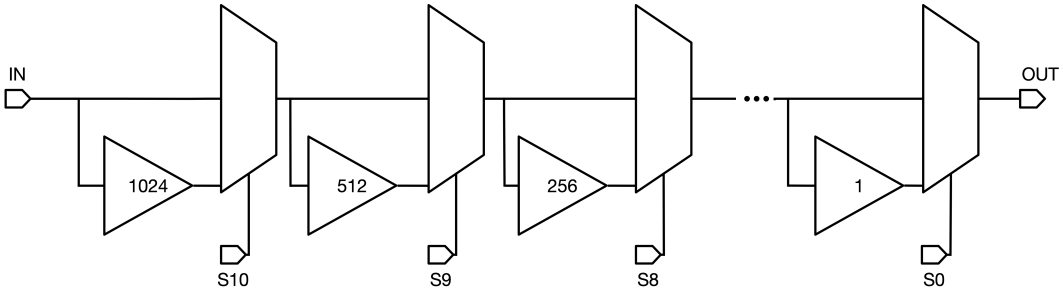


Figure 4.1: ICV3 Programmable Delay Block Diagram

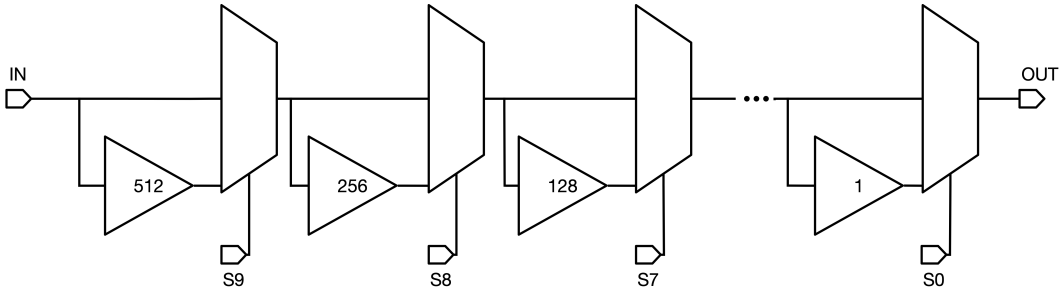


Figure 4.2: ICV4 Programmable Delay Block Diagram

4.2 Design Process

The proposed programmable delays discussed in this chapter vary from the proposed ICV2 PD in the tools used to create the PD. The proposed ICV3 and ICV4 PDs were created using digital design tools. Digital design tools are frequently implemented for designs using digital logic that do not generally require analog control mechanisms. The digital design tools used to create the PDs proposed in this chapter were Cadence Genus and Cadence Innovus.

4.2.1 Design Elements

The building block elements used for the proposed ICV3 and ICV4 PDs come from ARM IP cells. These were chosen based on the equivalence of the propagation delay high to low and low to high of the buffer cells, along with the overall propagation delay through the buffer. Additionally, the chosen MUX used has equivalent input and output capacitances to those of the chosen buffer. A key difference between the proposed programmable delays discussed in this chapter and the proposed ICV2 PD from Chapter 3 is the MOSFETs used for ICV2 were body contact MOSFETs, while the MOSFETs used in the designs discussed in this chapter are floating body.

Silicon-On-Insulator (SOI) MOSFETs which are floating body MOSFETs, may exhibit the history effect [28]. The body is considered to be floating as it is contained between the front oxide and back oxide which is isolated. When electron-hole pairs are generated in this type of MOSFET, the electrons can be swept away instantaneously. However, the holes require a certain time interval to be discharged through the source contact due to the built-in potential at the source/channel junction leading to undesired effects that include the history effect [28]. This history effect can change data pulse widths by 5% to 15% for fixed data patterns [29].

4.2.2 Code

Once devices were chosen, Verilog code was written to import into Cadence Genus. Multiple pieces of code were written to realize the proposed ICV3 and ICV4 PDs. The first written was a Verilog structured netlist with the ARM IP cells directly instantiated. A partial portion of the Verilog is shown below in Listing 4.1, and the complete file can be found in Appendix A. MXT is a MUX, and BUFH is a buffer in the code. The MUX and buffer instances were named so it would be fast to tell if an unintended connection was made during compilation. The purpose of the module is to instantiate all the various MUX and buffer elements. The generate statement is used to connect multiple buffers in a chain to create larger buffers for longer delays. There was a generate statement for the buffer used

to implement 4, 8, 16, 32, 64, 128, 256, 512, and 1024 delay steps.

```
`timescale 1 ps / 1 ps
module pd_v1( input in, input [10:0] sel, output out);

wire w_gd2;
wire [3:0] w_gd4;
wire [7:0] w_gd8;
wire [15:0] w_gd16;
wire [31:0] w_gd32;
wire [64:0] w_gd64;
wire [127:0] w_gd128;
wire [255:0] w_gd256;
wire [511:0] w_gd512;
wire [1023:0] w_gd1024;
wire [10:0] muxi; // mux in
wire [9:0] muxo; //mux out
genvar i;

MXT2 MUX10(.A(muxo[9]),.B(muxi[10]),.S0(sel[0]),.Y(out));//after gd1
BUFH gd1_0(.A(muxo[9]),.Y(muxi[10]));
MXT2 MUX9 (.A(muxo[8]),.B(muxi[9]),.S0(sel[1]),.Y(muxo[9]));//after gd2
BUFH gd2_0(.A(muxo[8]),.Y(w_gd2));
BUFH gd2_1(.A(w_gd2),.Y(muxi[9]));
MXT2 MUX8 (.A(muxo[7]),.B(muxi[8]),.S0(sel[2]),.Y(muxo[8]));//after gd4
generate  //gd4
  for(i = 0; i < 4; i = i + 1)
    begin
      if(i == 0)
        BUFH gd4_i(.A(muxo[7]),.Y(w_gd4[i]));
      else if (i < 3)
        BUFH gd4_i(.A(w_gd4[i-1]),.Y(w_gd4[i]));
      else
        BUFH gd4_i(.A(w_gd4[i-1]),.Y(muxi[8]));
    end
endgenerate
```

Listing 4.1: Partial Verilog Code of Creating the Structure Netlist

The Verilog code was compiled using Cadence Genus. The commands used for compiling can be found in Appendix B, and the compiled output file can be found in Appendix C for ICV4 PD. Then, the output Verilog file from Genus was imported into Innovus. The commands used in Innovus for ICV4 can be found in Appendix D. Once in Innovus, the layout could be generated. However, since the design did not include a clock, there was no way to use timing constraints to optimize the layout. In the following section, see the layout for the proposed PD ICV3 Figure 4.3 for an example of a non-optimal layout.

4.3 Proposed Programmable Delays ICV3 & ICV4

The proposed ICV3 and ICV4 programmable delays used nearly the same Verilog structured netlist file for processing in Cadence Genus and loading into Cadence Innovus. The main difference is that the proposed ICV4 has gate delay buffers that go from 1 delay step to 512 delay steps instead of 1024 delay steps. The proposed ICV3 PD was the result of simply loading the compiled Genus Verilog output file into Innovus. The layout from Innovus was then imported into Cadence Virtuoso and extracted for simulations. The layout of the proposed programmable delay ICV3 is shown in Figure 4.3. The initial results from

Figure 4.3: ICV3 Programmable Delay Layout

the proposed ICV3 PD were not exactly as expected see Section 4.4. This led to removing the 1024 LSB step delay buffer and exploring ways to make the layout more uniform, leading to the use of structured data paths. These were both done to improve the INL and DNL.

Structured data paths (SDP) can be used to optimize layouts when other methods cannot be implemented. An SDP is simply a way to tell Innovus exactly how to place device instances in a multi-dimensional matrix format by specifying the order elements should be included. A larger buffer instance/buffer chain can be created using an SDP so that

the instances of a buffer are placed next to each other in a rectangular format. This is accomplished by specifying a column to act at the outer East-West boundaries, and then rows can be specified, listing the individual instances in the desired order.

The structured data path file used for the proposed ICV4 PD was created using a text file and a Python script. An example of the text file used for the 128 delay step buffer is included in Listing 4.2. It was created with Excel initially, then saved as a tab-separated text file.

```
3  2  1  0  127  126  125  124
4  33   34   63   64   93   94   123
5  32   35   62   65   92   95   122
6  31   36   61   66   91   96   121
7  30   37   60   67   90   97   120
8  29   38   59   68   89   98   119
9  28   39   58   69   88   99   118
10   27   40   57   70   87   100  117
11   26   41   56   71   86   101  116
12   25   42   55   72   85   102  115
13   24   43   54   73   84   103  114
14   23   44   53   74   83   104  113
15   22   45   52   75   82   105  112
16   21   46   51   76   81   106  111
17   20   47   50   77   80   107  110
18   19   48   49   78   79   108  109
```

Listing 4.2: Text File used as Input for Python Script

The text file was read by the Python script and processed to add the additional text needed for a structured data path file. The script could have been adapted to read multiple files at once. The script is included below in Listing 4.3.

```
import csv
import sys

file = 'gd128.txt'
delay = '128'
name = file.split('.txt')
rows = []
with open(file, 'r') as f:
    while(True):
        line = f.readline()
        if not line:
            break
        elements = line.strip().split("\t")
        row = []
        for item in elements:
```

```
            row.append(item)
        rows.append(row)
print(rows)
text = 'datapath pd{}\n{{\n\torigin 0.0 0.0\n'.format(delay)
t2 = '\tcolumn gd{}\n\t{{\n\t\tjustifyBy SW'.format(delay)
count = 1
temp = ''

for j in range(0,len(rows)):
    temp += '\n\t\trow gd{}_{}\n\t\t{{\n\t\t\tjustifyBy SW'.format(delay,j
    +1)
    for i in range(0,len(rows[j])):
        temp += '\n\t\t\tinst gd{}_{}'.format(delay,rows[j][i])
    temp += '\n\t\t}'

temp += '\n\t}\n}'
t3 = text + t2 + temp

newFile = 'sdp_{}.txt'.format(delay)

with open(newFile,'w') as f:
    f.write(t3)
```

Listing 4.3: Python Script for Creating a Structure Data Path

The partial output file created by the Python script in Listing 4.3 can be seen in Listing 4.4, and the full-text file generated can be seen in Appendix F. In addition, the .sdp file used for all the buffers can be found in Appendix G.

```
datapath pd128
{
  origin 0.0 0.0
  column gd128
  {
    justifyBy SW
    row gd128_1
    {
      justifyBy SW
      inst gd128_3
      inst gd128_2
      inst gd128_1
      inst gd128_0
      inst gd128_127
      inst gd128_126
      inst gd128_125
      inst gd128_124
    }
```

Listing 4.4: Partial Output File Generated by Python

The proposed ICV4 programmable delay's layout is shown in Figure 4.4. Comparing the

layouts of the proposed PDs, there was a significant improvement in the organization of the placed buffers in the ICV4 version over the ICV3 version. The organization and reducing the overall range of the delay allowed for the layout size to decrease by 67%. Additionally, the simulation results improved, which will be discussed in the next section.

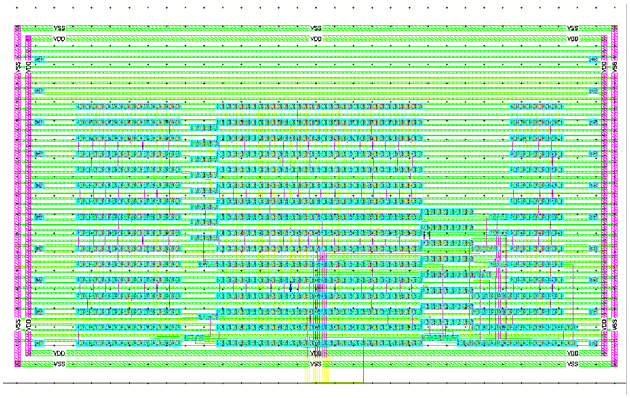


Figure 4.4: ICV4 Programmable Delay Layout using SDP

4.4 Simulation Results

After the layout of the proposed ICV3 programmable delay and proposed ICV4 programmable delay were finished using Cadence Innovus, the designs were extracted to obtain the parasitic resistances and capacitances to use in simulation with Cadence Virtuoso.

4.4.1 Linearity Simulation

The proposed ICV3 and ICV4 programmable delay elements are simulated in Cadence Virtuoso GF45RFSOI 45nm technology. The supply voltage used was 1V, and the input signal logic high was also 1V. The linearity simulations were conducted by stepping through all possible, select register bit combinations. Figure 4.5 is the result of the thermometer code simulation, and it contains two plots. The blue plot in Figure 4.5 is of the proposed ICV3 PD. The red plot is of the proposed ICV4 PD. This plot does not show a significant difference between the two proposed delays. The plots of the DNL and INL for the proposed

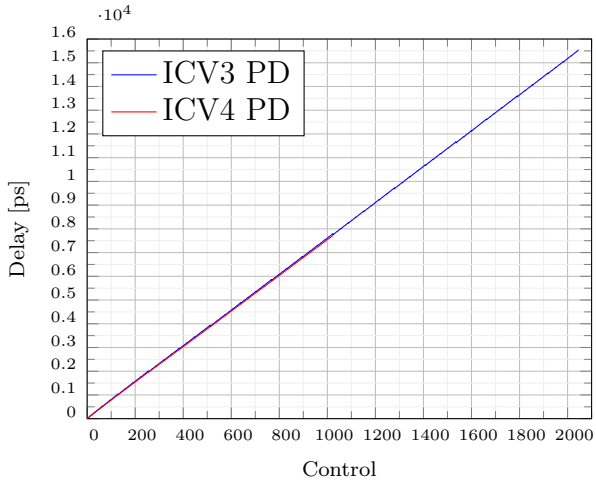


Figure 4.5: ICV3 and ICV4 Programmable Delay Thermometer Code Simulation

delays are where the performance differences are noticeable. Figure 4.6 shows the DNL for the proposed ICV3 PD, and Figure 4.7 shows the DNL for the proposed ICV4 PD.

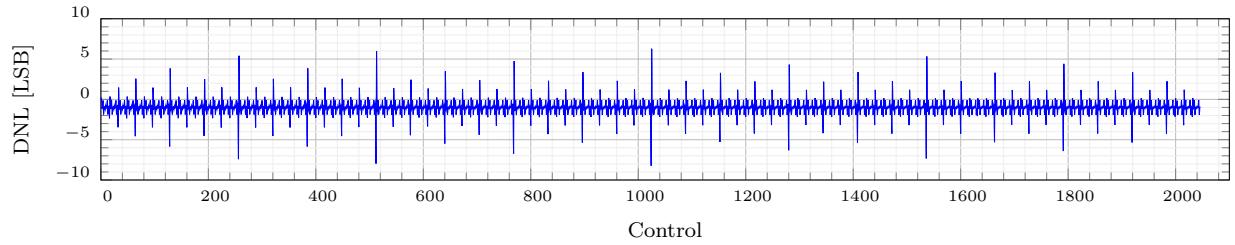


Figure 4.6: ICV3 Programmable Delay DNL

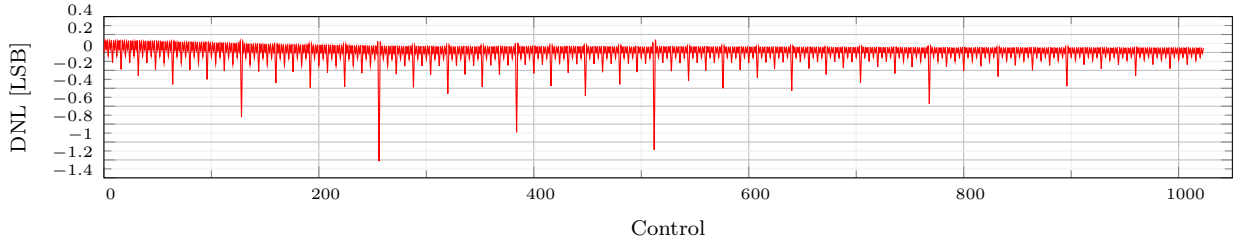


Figure 4.7: ICV4 Programmable Delay DNL

It is important to notice the scale of the y-axis for the figures. Figure 4.6 has a y-axis that goes from -10 to 10, while Figure 4.7 has a y-axis that only goes from -1.4 to 0.4. This is a significant difference in the DNL for the designs. This difference in scale indicates the ICV4 is significantly more linear. This is seen not only from the difference in the scale of the y-axis

but also by the frequency of the steps with significantly different DNL measurements. Since the cells used are the same between ICV3 and ICV4, the difference in DNL performance shows a benefit gained by implementing structure data paths in the design of ICV4.

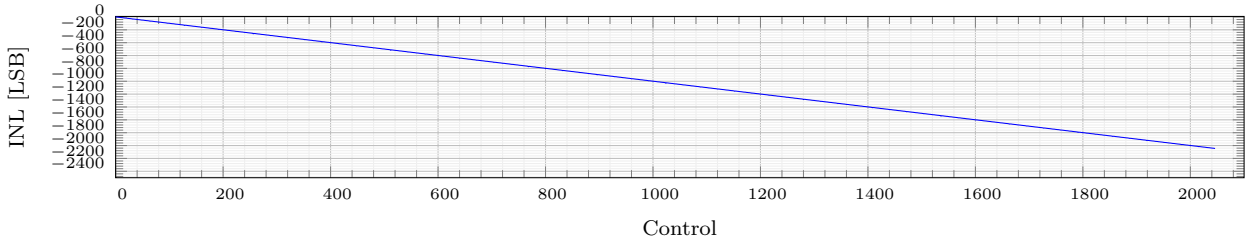


Figure 4.8: ICV3 Programmable Delay INL

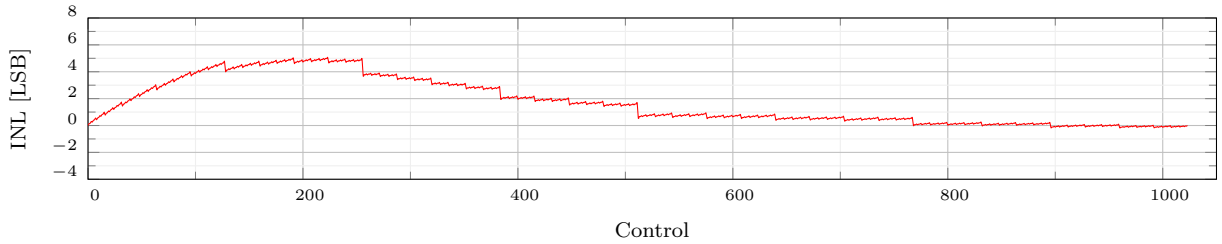
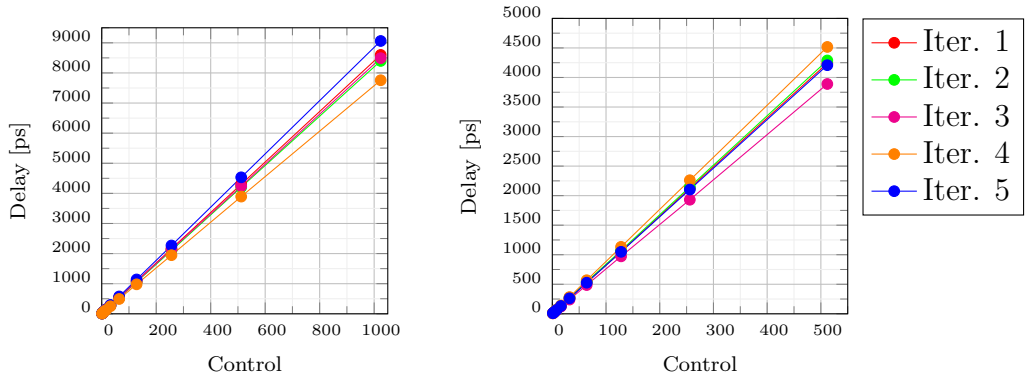


Figure 4.9: ICV4 Programmable Delay INL

Figure 4.8 shows the INL for the proposed ICV3 PD, and Figure 4.9 shows the INL for the proposed ICV4 PD. The INL, like the DNL for the two designs, shows a significant difference, with ICV4 being an improvement over the ICV3 design.

4.4.2 Monte Carlo Simulation

A Monte Carlo simulation analysis was conducted using the Cadence toolset for the two proposed PDs. This allowed for testing the process variation and mismatch between devices using a statistical model that changed parameter values for each iteration. The simulations were conducted five times due to time and server disk space limitations. Figure 4.10a and Figure 4.10a depict the different iterations of the Monte Carlo analysis compared to each other for the ICV3 design and ICV4 design, respectively. The exact values for each point plotted along with the mean and standard deviation over the iterations are included in Table 4.1 and Table 4.2 for the ICV3 design and ICV4 design, respectively.



(a) ICV3 PD Simulation (b) ICV4 PD Simulation

Figure 4.10: Proposed ICV3 & ICV4 Programmable Delay Monte Carlo

Table 4.1: Comparison of ICV3 Monte Carlo Sims

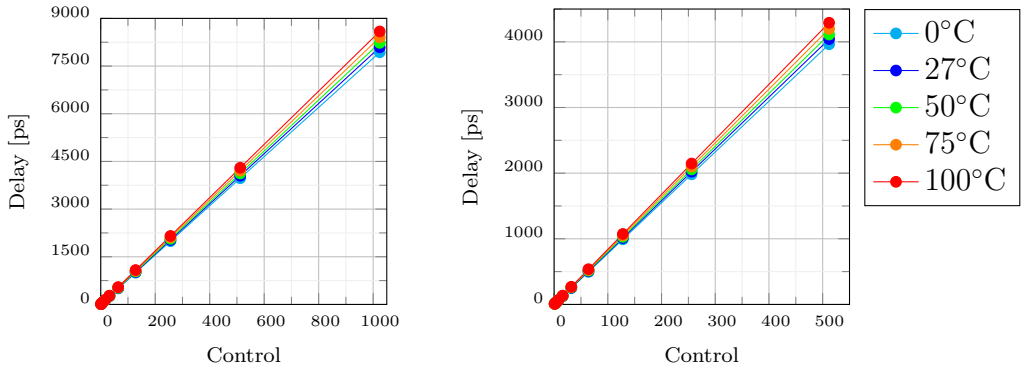
Control	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	μ	σ
1	8.261 ps	8.212 ps	8.384 ps	8.931 ps	9.573 ps	8.672 ps	0.580 ps
2	18.309 ps	16.550 ps	15.315 ps	16.838 ps	18.626 ps	17.128 ps	1.355 ps
4	43.325 ps	40.496 ps	38.688 ps	43.525 ps	46.514 ps	42.509 ps	3.016 ps
8	76.769 ps	73.70 ps	68.074 ps	73.695 ps	80.762 ps	74.60 ps	4.661 ps
16	142.886 ps	140.218 ps	128.005 ps	144.547 ps	150.007 ps	141.133 ps	8.165 ps
32	279.620 ps	278.158 ps	251.158 ps	275.430 ps	292.083 ps	275.290 ps	14.928 ps
64	542.945 ps	529.391 ps	492.268 ps	534.536 ps	575.846 ps	534.997 ps	29.955 ps
128	1082.766 ps	1057.235 ps	979.178 ps	1069.850 ps	1140.620 ps	1065.930 ps	58.049 ps
256	2161.356 ps	2102.636 ps	1947.150 ps	2137.223 ps	2268.40 ps	2123.353 ps	116.353 ps
512	4302.744 ps	4205.871 ps	3893.191 ps	4237.511 ps	4533.181 ps	4234.50 ps	229.776 ps
1024	8598.629 ps	8398.543 ps	7757.585 ps	8491.191 ps	9061.245 ps	8461.439 ps	468.732 ps

Table 4.2: Comparison of ICV4 Monte Carlo Sims

Control	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	μ	σ
1	9.375 ps	9.457 ps	7.966 ps	8.511 ps	10.235 ps	9.109 ps	0.884 ps
2	17.150 ps	18.183 ps	15.473 ps	18.835 ps	16.808 ps	17.290 ps	1.298 ps
4	34.520 ps	33.479 ps	31.189 ps	34.595 ps	32.147 ps	33.186 ps	1.493 ps
8	65.982 ps	64.685 ps	61.541 ps	70.094 ps	65.232 ps	65.507 ps	3.072 ps
16	130.916 ps	134.565 ps	122.310 ps	141.651 ps	132.192 ps	132.327 ps	6.969 ps
32	266.173 ps	270.988 ps	241.092 ps	283.082 ps	261.904 ps	264.648 ps	15.368 ps
64	528.965 ps	540.763 ps	486.775 ps	568.275 ps	526.553 ps	530.266 ps	29.418 ps
128	1053.624 ps	1070.872 ps	972.327 ps	1133.316 ps	1048.980 ps	1055.824 ps	57.565 ps
256	2124.211 ps	2147.879 ps	1930.711 ps	2260.944 ps	2102.836 ps	2113.316 ps	118.913 ps
512	4241.091 ps	4287.826 ps	3889.962 ps	4516.812 ps	4207.988 ps	4228.736 ps	224.627655

4.4.3 Temperature Simulation

The simulation was conducted with a sampling of temperatures 0°C, 27°C, 50°C, 75°C, and 100°C. Figure 4.11a and Figure 4.11b show the temperature simulation results for the ICV3 and ICV4 designs, respectively. Additionally, the exact values for each point plotted in Figure 4.11a and Figure 4.11b are included in Table 4.3 and Table 4.4 respectively, along with μ and σ for each of the test cases.



(a) ICV3 PD Simulation (b) ICV4 PD Simulation

Figure 4.11: Proposed ICV3 & ICV4 Programmable Delay Temperature Simulations

Table 4.3: Comparison of ICV3 Temperature Sims

Control	0°C	27°C	50°C	75°C	100°C	μ	σ
1	8.530 ps	8.444 ps	8.613 ps	8.706 ps	8.804 ps	8.619 ps	0.142 ps
2	16.328 ps	16.102 ps	16.542 ps	16.792 ps	17.056 ps	16.564 ps	0.375 ps
4	40.018 ps	39.328 ps	40.658 ps	41.413 ps	42.220 ps	40.727 ps	1.136 ps
8	71.596 ps	70.291 ps	72.80 ps	74.218 ps	75.764 ps	72.934 ps	2.148 ps
16	134.751 ps	132.227 ps	137.085 ps	139.837 ps	142.845 ps	137.349 ps	4.167 ps
32	261.073 ps	256.097 ps	265.646 ps	271.114 ps	276.994 ps	266.185 ps	8.205 ps
64	513.724 ps	503.899 ps	522.737 ps	533.599 ps	545.263 ps	523.844 ps	16.243 ps
128	1018.958 ps	999.553 ps	1036.992 ps	1058.397 ps	1081.638 ps	1039.108 ps	32.234 ps
256	2029.583 ps	1990.171 ps	2065.549 ps	2108.188 ps	2154.702 ps	2069.639 ps	64.533 ps
512	4049.973 ps	3970.975 ps	4121.374 ps	4206.836 ps	4299.782 ps	4129.788 ps	128.935 ps
1024	8089.745 ps	7933.533 ps	8230.316 ps	8401.671 ps	8589.357 ps	8248.925 ps	257.090 ps

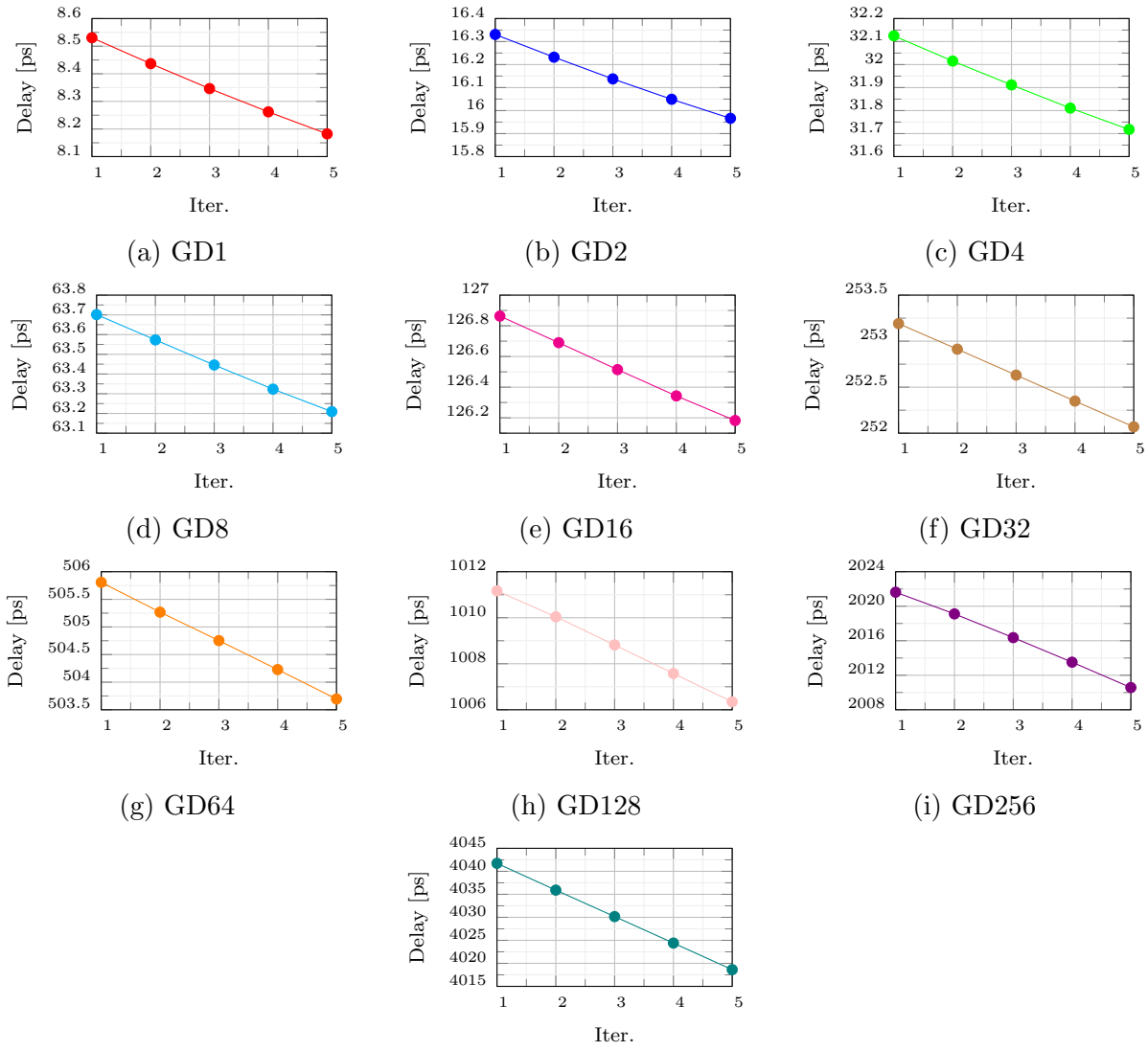
Table 4.4: Comparison of ICV4 Temperature Sims

Control	0°C	27°C	50°C	75°C	100°C	μ	σ
1	8.444 ps	8.531 ps	8.615 ps	8.707 ps	8.804 ps	8.620 ps	0.142 ps
2	16.101 ps	16.331 ps	16.542 ps	16.793 ps	17.057 ps	16.565 ps	0.376 ps
4	31.586 ps	32.125 ps	32.620 ps	33.204 ps	33.837 ps	32.674 ps	0.883 ps
8	62.554 ps	63.701 ps	64.762 ps	66.015 ps	67.377 ps	64.882 ps	1.893 ps
16	124.488 ps	126.864 ps	129.044 ps	131.656 ps	134.456 ps	129.302 ps	3.914 ps
32	248.389 ps	253.191 ps	257.602 ps	262.911 ps	268.611 ps	258.141 ps	7.941 ps
64	496.234 ps	505.807 ps	514.728 ps	525.377 ps	536.866 ps	515.803 ps	15.963 ps
128	991.481 ps	1011.168 ps	1029.094 ps	1050.381 ps	1073.515 ps	1031.128 ps	32.178 ps
256	1982.220 ps	2021.623 ps	2057.029 ps	2099.741 ps	2146.157 ps	2061.354 ps	64.274 ps
512	3963.324 ps	4041.726 ps	4111.745 ps	4197.414 ps	4291.374 ps	4121.117 ps	128.548 ps

4.4.4 Repeated Commands Simulation

This simulation was conducted to see if there was a difference in the delay duration for each delay element over five consecutive iterations during a single transient simulation. The simulation was only conducted on the ICV4 design due to its superior performance in the other previous simulations. The simulation was conducted by toggling one select bit on at a time for five consecutive iterations. Each iteration of turning on one select signal at a time

took approximately 160 ns, with the entire simulation running for approximately 800 ns. The results can be seen visually in Figure 4.12 and numerically in Table 4.5. In the figure and table, GD stands for gate delay, and the number following GD is the number of gate delays for that buffer delay element. The results show a decrease in the delay through each buffer for each consecutive iteration.



(a) GD1 (b) GD2 (c) GD4

(d) GD8 (e) GD16 (f) GD32

(g) GD64 (h) GD128 (i) GD256

(j) GD512

Figure 4.12: ICV4 Programmable Delay Repeated Simulation

Table 4.5: Comparison of ICV4 Delays Through Gates over consecutive runs

	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Δ(Iter. 1 – Iter.5)
GD1	8.53 ps	8.44 ps	8.35 ps	8.26 ps	8.18 ps	0.35 ps
GD2	16.33 ps	16.23 ps	16.14 ps	16.05 ps	15.97 ps	0.36 ps
GD4	32.12 ps	32.02 ps	31.91 ps	31.81 ps	31.72 ps	0.41 ps
GD8	63.70 ps	63.57 ps	63.45 ps	63.32 ps	63.21 ps	0.49 ps
GD16	126.86 ps	126.69 ps	126.51 ps	126.34 ps	126.18 ps	0.68 ps
GD32	253.19 ps	252.91 ps	252.63 ps	252.35 ps	252.07 ps	1.12 ps
GD64	505.81 ps	505.27 ps	504.75 ps	504.23 ps	503.70 ps	2.11 ps
GD128	1011.17 ps	1010.04 ps	1008.82 ps	1007.58 ps	1006.35 ps	4.82 ps
GD256	2021.63 ps	2019.11 ps	2016.36 ps	2013.51 ps	2010.57 ps	11.05 ps
GD512	4041.73 ps	4035.91 ps	4030.16 ps	4024.41 ps	4018.63 ps	23.10 ps

4.4.5 Discussion

The simulation results from the previous subsections showed that the proposed ICV4 PD outperformed the proposed ICV3 PD owing primarily to the structured data paths implemented. The DNL and INL were the characteristics that improved the most between the two different versions. The Monte Carlo Analysis on Control 512 also had an improvement in the standard deviation dropping from 229.776 ps to 224.678 ps, approximately a 2.2% reduction. The temperature corners simulations showed similar results, with the standard deviation of the Control 512 test being reduced by about 0.3% from 128.935 ps to 128.55 ps. The repeated commands simulation on the proposed ICV4 PD showed that over time as the programmable delay is on, the delay duration through the buffer chains acting as larger buffers decreases. This is seen by looking at the last column of Table 4.5. The decrease in delay duration is pretty significant considering it happened over 8×10^{-7} seconds and can be linked to the history effect discussed in Section 4.2.1. Table 4.6 summarizes the characteristics of the proposed ICV3 and ICV4 PDs.

Table 4.6: ICV3 & ICV4 Results Summary

PD	Process (nm)	Res. (ps)	Range	FOM	Area	DNL (norm.)	INL (norm.)
ICV3	45	7.59	15.990 ns	2047	0.0446 mm^2	8.229	2044.97
ICV4	45	7.54	7.72 ns	1023	0.0146 mm^2	1.214	5.057

Proposed ICV4 PD contributes to the field of variable delay elements as being a wide range, linear, digitally designed, 7.5ps resolution, 7.7ns delay range, inverter chain based

digitally controlled delay line.

Chapter 5

Conclusion

This work presents two functional designs for an inverter chain-based programmable delay: proposed ICV2 PD and ICV4 PD. The proposed ICV3 PD is technically functional as it will induce a delay for a signal. However, it is not linear and would not function properly for generating a usable reference for phase synchronization. The ICV2 PD obtains at least an 11-bit resolution, while ICV4 PD obtains a 10-bit resolution. The Monte Carlo and temperature corner analysis on the circuits show that the delays they produce will vary slightly and not maintain a consistent LSB step size over process and temperature variations. However, the proposed ICV2 PD will be able to compensate slightly and maintain linearity for a new LSB step size. The ICV4 proposed design does not have a tuning circuit, so it will not be able to compensate. The ICV4 design with the floating body MOSFETs also suffers from the history effect over time, leading to changes in the amount of time a signal will be delayed relative to the length of time the programmable delay element is powered on. However, this should not be an issue in our intended radar application since the propagation delay through all of the buffer elements occurs at a similar rate.

5.1 Results

Table 5.1 shows the comparison results for the proposed ICV2, ICV3, and ICV4 programmable delays with other variable delay elements previously shown in Table 2.10. N/A is used to mean not available in Table 5.1.

Table 5.1: Comparison of Current Programmable Delay Lines

Type	Process (nm)	Res. (ps)	Range	FOM	Area	DNL (norm.)	INL (norm.)
SCI [18]	350	1.43	40 ps	27.972	850 μm^2	0.37	0.19
SCI & Body Bias [19]	65	0.004	30 ps	7500	0.514 mm^2	N/A	N/A
SCI & CSI [20]	130	340	50 ns	147.05	2.25 mm^2	N/A	N/A
CSI [21]	350	40	400 ps	10	450 μm^2	N/A	N/A
CSI [22]	180	2	320 ps	160	5000 μm^2	13.61	36.15
DLL [24]	350	5	8.6 ns	1720	N/A	N/A	N/A
Resistor Array [25]	180	1	50 ps	50	N/A	1.37	1.04
Cascaded Inverters [27]	350	5	300 ps	60	0.36 mm^2	N/A	N/A
ICV2	45	3.4	6.873 ns	2047	0.03 mm^2	18.504	13.889
ICV2 tuned	45	3.24	6.873 ns	2119	0.03 mm^2	0.619	9.64
ICV3	45	7.59	15.990 ns	2047	0.0446 mm^2	8.229	2044.97
ICV4	45	7.54	7.72 ns	1023	0.0146 mm^2	1.214	5.057

5.2 Future Work

The simulation results obtained for the proposed programmable delays are promising. One challenge that will need to be addressed for testing the manufactured IC is designing a way to test and effectively measure the delay introduced to the output signal. Ideally, an oscilloscope with close to a 1 THz sampling rate would be used as it would sample fast enough for accurate measurements. Unfortunately, an oscilloscope with that high sampling rate is not feasible due to the immense cost of renting one. Possible alternatives would be designing a time-to-digital converter, using a sub-sampling oscilloscope, or using a PLL on an FPGA. A sub-sampling oscilloscope will be used for testing the PDs when the fabricated IC they are a part of is ready. The sub-sampling oscilloscope was chosen since one is available in the author's lab.

References

[1] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: Design and analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 871-878, Oct. 2003, doi: 10.1109/TVLSI.2003.810787.

[2] E. A. Wolff, *Microwave Engineering and Systems Applications*. New York, NY: J. Wiley, 1988, pp. 275-313.

[3] E. Brookner and P. J. Kahrilas, "Lecture 1 - Electronic Scanning Radar Systems (ESRS) Design & Architecture," in *Practical phased-array antenna systems*, Lexington, MA: Lex Book, 1997, pp. 1–2-1–3.

[4] D. Dogan and G. Gultepe, "A beamforming method enabling easy packaging of scalable architecture phased arrays," *2016 IEEE International Symposium on Phased Array Systems and Technology (PAST)*, Waltham, MA, USA, 2016, pp. 1-4, doi: 10.1109/ARRAY.2016.7832562.

[5] G.S. Peterson, private communication, Feb. 2021.

[6] B. I. Abdulrazzaq, I. Abdul Halin, S. Kawahito, R. M. Sidek, S. Shafie, and N. A. Yunus, "A review on high-resolution CMOS delay lines: Towards sub-picosecond Jitter Performance," *SpringerPlus*, vol. 5, no. 1, 2016, doi: 10.1186/s40064-016-2090-z.

[7] A. Melloni, *et al.*, "Tunable delay lines in silicon photonics: Coupled resonators and photonic crystals, a comparison," *IEEE Photonics Journal*, vol. 2, no. 2, pp. 181-194, Apr. 2010 , doi: 10.1109/JPHOT.2010.2044989.

[8] K. Klepacki, R. Szplet, and R. Pelka, "A 7.5 PS single-shot precision integrated time counter with segmented delay line," *Review of Scientific Instruments*, vol. 85, no. 3, 2014, doi: 10.1063/1.4868500.

[9] T. Xanthopoulos, "Digital delay locked techniques," in *Clocking in modern VLSI Systems*, Dordrecht: Springer, 2009, pp. 183-244, doi: 10.1007/978-1-4419-0261-0_6.

[10] S. Kim, B. Ham, M. Cho, S. Oh, J. Lee and T. B. Cho, "A 14nm FinFET Sub-Picosecond Jitter Fractional-N Ring PLL for 5G Wireless Communication," *2018 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*, Philadelphia, PA, USA, 2018, pp. 40-43, doi: 10.1109/RFIC.2018.8429027.

[11] M. Madhvaraj, S. Mir and M. J. Barragan, "A self-referenced on-chip jitter BIST with sub-picosecond resolution in 28 nm FD-SOI technology," *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, Patras, Greece, 2022, pp. 1-6, doi: 10.1109/VLSI-SoC54400.2022.9939620.

[12] M. Gal-Katziri and A. Hajimiri, "A Sub-Picosecond Hybrid DLL for Large-Scale Phased Array Synchronization," *2018 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, Tainan, Taiwan, 2018, pp. 231-234, doi: 10.1109/ASSCC.2018.8579340.

[13] B. I. Abdulrazzaq, I. A. Halin, R. M. Sidek, S. Shafie, N. A. M. Yunus and S. Kawahito, "Sub-picosecond jitter resolution wide range digital delay line for SoC integration," *2015 IEEE International Circuits and Systems Symposium (ICSyS)*, Langkawi, Malaysia, 2015, pp. 44-48, doi: 10.1109/CircuitsAndSystems.2015.7394062.

[14] P. A. J. Nuyts, P. Reynaert, and W. Dehaene, "Continuous-Time Digital Design Techniques," in *Continuous-Time Digital Front-Ends for Multistandard Wireless Transmission*, Cham: Springer International Publishing, 2014, pp. 125-185, doi: 10.1007/978-3-319-03925-1_4.

[15] R.J. Baker, *CMOS: Circuit Design, Layout, and Simulation*, 4th ed. Hoboken, NJ: Wiley, IEEE Press, 2019.

[16] N.H.E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. Boston, MA: Pearson, 2004.

[17] A.S. Sedra and K.C. Smith, *Microelectronic Circuits*, 7th ed. New York, NY: Oxford Univ. Press, 2015.

[18] P. Chen, C. Chung and C. Lee, "A portable digitally controlled oscillator using novel varactors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 5, pp. 233-237, May 2005, doi: 10.1109/TCSII.2005.846307.

[19] W. Wang, H. Zhou, F. Ye and J. Ren, "An 8-bit 4fs-step digitally controlled delay element with two cascaded delay units," *2015 IEEE 11th International Conference on ASIC (ASICON)*, Chengdu, China, 2015, pp. 1-4, doi: 10.1109/ASICON.2015.7517034.

[20] J. I. Morales, F. Chierchie, P. S. Mandolesi and E. E. Paolini, "Design and Evaluation of an All-Digital Programmable Delay Line in 130-nm CMOS," *2019 XVIII Workshop on Information Processing and Control (RPIC)*, Salvador, Brazil, 2019, pp. 209-213, doi: 10.1109/RPIC.2019.8882166.

[21] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: design and analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 871-878, Oct. 2003, doi: 10.1109/TVLSI.2003.810787.

[22] M. Maymandi-Nejad and M. Sachdev, "A monotonic digitally controlled delay element," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 11, pp. 2212-2219, Nov. 2005, doi: 10.1109/JSSC.2005.857370.

[23] R. Zhang and M. Kaneko, "Robust and Low-Power Digitally Programmable Delay Element Designs Employing Neuron-MOS Mechanism," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 4, Sept. 2015, doi:10.1145/2740963.

[24] T. Van den Dries, H. Ingelberts, S. Boulanger and M. Kuijk, "A 5 ps resolution, 8.6 ns delay range digital delay line using combinatorial redundancy," *2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, Lausanne, Switzerland, 2019, pp. 21-24, doi: 10.1109/PRIME.2019.8787802.

[25] M. Saint-Laurent and M. Swaminathan, "A digitally adjustable resistor for path delay characterization in high-frequency microprocessors," *2001 Southwest Symposium on Mixed-Signal Design (Cat. No.01EX475)*, Austin, TX, USA, 2001, pp. 61-64, doi: 10.1109/SSMSD.2001.914938.

[26] X. Zhang, Z. Ma, J. Yu and L. Xie, "Memristor-based programmable delay element," *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Guilin, China, 2014, pp. 1-3, doi: 10.1109/ICSICT.2014.7021414.

[27] C. Chung and C. Lee, "An all-digital phase-locked loop for high-speed clock generation," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 2, pp. 347-351, Feb. 2003, doi: 10.1109/JSSC.2002.807398.

[28] S. Ghosh, M. Miura-Mattausch, T. Iizuka, H. Kikuchihara, H. Rahaman and H. J. Mattausch, "History Effect on Circuit Performance of SOI-MOSFETs," *2020 International Symposium on Devices, Circuits and Systems (ISDCS)*, Howrah, India, 2020, pp. 1-5, doi: 10.1109/ISDCS49393.2020.9262980.

[29] K. A. Jenkins, S. Kim, S. P. Kowalczyk and D. Friedman, "Impact of SOI History Effect on Random Data Signals," *2007 IEEE International Conference on Integrated Circuit Design and Technology*, Austin, TX, USA, 2007, pp. 1-4, doi: 10.1109/ICICDT.2007.4299531.

Appendix A

ICV3 Verilog Code for Generating the Structured Netlist

```
// Programmable Delay Version 1

`timescale 1 ps / 1 ps
module pd_v1( input in, input [10:0] sel, output out);

wire w_gd2;
wire [3:0] w_gd4;
wire [7:0] w_gd8;
wire [15:0] w_gd16;
wire [31:0] w_gd32;
wire [64:0] w_gd64;
wire [127:0] w_gd128;
wire [255:0] w_gd256;
wire [511:0] w_gd512;
wire [1023:0] w_gd1024;
wire [10:0] muxi; // mux in
wire [9:0] muxo; //mux out
genvar i;

MXT2 MUX10(.A(muxo[9]),.B(muxi[10]),.S0(sel[0]),.Y(out));//after gd1
BUFH gd1_0(.A(muxo[9]),.Y(muxi[10]));
MXT2 MUX9 (.A(muxo[8]),.B(muxi[9]),.S0(sel[1]),.Y(muxo[9]));//after gd2
BUFH gd2_0(.A(muxo[8]),.Y(w_gd2));
BUFH gd2_1(.A(w_gd2),.Y(muxi[9]));
MXT2 MUX8 (.A(muxo[7]),.B(muxi[8]),.S0(sel[2]),.Y(muxo[8]));//after gd4
generate  //gd4
  for(i = 0; i < 4; i = i + 1)
    begin
      if(i == 0)
        BUFH gd4_i(.A(muxo[7]),.Y(w_gd4[i]));
```

```
31        else if (i < 3)
32          BUFH gd4_i(.A(w_gd4[i-1]),.Y(w_gd4[i]));
33        else
34          BUFH gd4_i(.A(w_gd4[i-1]),.Y(muxi[8]));
35      end
36  endgenerate
37
38  generate  //gd8
39    for(i = 0; i < 8; i = i + 1)
40      begin
41        if(i == 0)
42          BUFH gd8_i(.A(muxo[6]),.Y(w_gd8[i]));
43        else if (i < 7)
44          BUFH gd8_i(.A(w_gd8[i-1]),.Y(w_gd8[i]));
45        else
46          BUFH gd8_i(.A(w_gd8[i-1]),.Y(muxi[7]));
47      end
48  endgenerate
49
50  generate  //gd16
51    for(i = 0; i < 16; i = i + 1)
52      begin
53        if(i == 0)
54          BUFH gd16_i(.A(muxo[5]),.Y(w_gd16[i]));
55        else if (i < 15)
56          BUFH gd16_i(.A(w_gd16[i-1]),.Y(w_gd16[i]));
57        else
58          BUFH gd16_i(.A(w_gd16[i-1]),.Y(muxi[6]));
59      end
60  endgenerate
61
62  generate  //gd32
63    for(i = 0; i < 32; i = i + 1)
64      begin
65        if(i == 0)
66          BUFH gd32_i(.A(muxo[4]),.Y(w_gd32[i]));
67        else if (i < 31)
68          BUFH gd32_i(.A(w_gd32[i-1]),.Y(w_gd32[i]));
69        else
70          BUFH gd32_i(.A(w_gd32[i-1]),.Y(muxi[5]));
71      end
72  endgenerate
73
74  generate  //gd64
75    for(i = 0; i < 64; i = i + 1)
76      begin
77        if(i == 0)
78          BUFH gd64_i(.A(muxo[3]),.Y(w_gd64[i]));
79        else if (i < 63)
80          BUFH gd64_i(.A(w_gd64[i-1]),.Y(w_gd64[i]));
81        else
82          BUFH gd64_i(.A(w_gd64[i-1]),.Y(muxi[4]));
83      end
84  endgenerate
```

```
85
86 generate  //gd128
87   for(i = 0; i < 128; i = i + 1)
88     begin
89       if(i == 0)
90         BUFH gd128_i(.A(muxo[2]),.Y(w_gd128[i]));
91       else if (i < 127)
92         BUFH gd128_i(.A(w_gd128[i-1]),.Y(w_gd128[i]));
93       else
94         BUFH gd128_i(.A(w_gd128[i-1]),.Y(muxi[3]));
95     end
96 endgenerate
97
98 generate  //gd256
99   for(i = 0; i < 256; i = i + 1)
100    begin
101      if(i == 0)
102        BUFH gd256_i(.A(muxo[1]),.Y(w_gd256[i]));
103      else if (i < 255)
104        BUFH gd256_i(.A(w_gd256[i-1]),.Y(w_gd256[i]));
105      else
106        BUFH gd256_i(.A(w_gd256[i-1]),.Y(muxi[2]));
107    end
108 endgenerate
109
110 generate  //gd512
111   for(i = 0; i <= 512; i = i + 1)
112    begin
113      if(i == 0)
114        BUFH gd512_i(.A(muxo[0]),.Y(w_gd512[i]));
115      else if (i < 512)
116        BUFH gd512_i(.A(w_gd512[i-1]),.Y(w_gd512[i]));
117      else
118        BUFH gd512_i(.A(w_gd512[i-1]),.Y(muxi[1]));
119    end
120 endgenerate
121
122 generate  //gd1024
123   for(i = 0; i <= 1024; i = i + 1)
124    begin
125      if(i == 0)
126        BUFH gd1024_i(.A(in),.Y(w_gd1024[i]));
127      else if (i < 1024)
128        BUFH gd1024_i(.A(w_gd1024[i-1]),.Y(w_gd1024[i]));
129      else
130        BUFH gd1024_i(.A(w_gd1024[i-1]),.Y(muxi[0]));
131    end
132 endgenerate
133
134 MXT2 MUX0 (.A(in),.B(muxi[0]),.S0(sel[10]),.Y(muxo[0]));//after gd1024
135 MXT2 MUX1 (.A(muxo[0]),.B(muxi[1]),.S0(sel[9]),.Y(muxo[1]));//after gd512
136 MXT2 MUX2 (.A(muxo[1]),.B(muxi[2]),.S0(sel[8]),.Y(muxo[2]));//after gd256
137 MXT2 MUX3 (.A(muxo[2]),.B(muxi[3]),.S0(sel[7]),.Y(muxo[3]));//after gd128
138 MXT2 MUX4 (.A(muxo[3]),.B(muxi[4]),.S0(sel[6]),.Y(muxo[4]));//after gd64
```

```
139 MXT2 MUX5 (.A(muxo[4]),.B(muxi[5]),.S0(sel[5]),.Y(muxo[5]));//after gd32
140 MXT2 MUX6 (.A(muxo[5]),.B(muxi[6]),.S0(sel[4]),.Y(muxo[6]));//after gd16
141 MXT2 MUX7 (.A(muxo[6]),.B(muxi[7]),.S0(sel[3]),.Y(muxo[7]));//after gd8
142
143 endmodule
```

Appendix B

Genus Command Log Output File for ICV4

```
# Cadence Genus(TM) Synthesis Solution, Version 20.10-p001_1, built Dec 11
    2020 16:29:55
# Date: Wed Nov 02 11:26:30 2022

read_hdl dpd/verilog/pd_v3.v
elaborate
set_dont_touch [get_db insts]
syn_generic
init_design
write_design -base_name dpd/outputs/Nov2/pd3 -innovus
```

Appendix C

Genus Synthesized Verilog Output for ICV4

In the code below ... is to signify that some lines were removed to reduce the overall length.

```
// Generated by Cadence Genus(TM) Synthesis Solution 20.10-p001_1
// Generated on: Nov  2 2022 11:31:52 CDT (Nov  2 2022 16:31:52 UTC)

// Verification Directory fv/pd_v3

module pd_v3(in, sel, out);
  input in;
  input [9:0] sel;
  output out;
  wire in;
  wire [9:0] sel;
  wire out;
  wire [9:0] muxi;
  wire [8:0] muxo;
  wire [2:0] w_gd4;
  wire [6:0] w_gd8;
  wire [14:0] w_gd16;
  wire [30:0] w_gd32;
  wire [62:0] w_gd64;
  wire [126:0] w_gd128;
  wire [254:0] w_gd256;
  wire [510:0] w_gd512;
  wire w_gd2;
  MXT MUX0(.A(in), .B(muxi[0]), .S0(sel[9]), .Y(muxo[0]));
  MXT MUX1(.A(muxo[0]), .B(muxi[1]), .S0(sel[8]), .Y(muxo[1]));
  MXT MUX2(.A(muxo[1]), .B(muxi[2]), .S0(sel[7]), .Y(muxo[2]));
```

```
27    MXT MUX3(.A(muxo[2]), .B(muxi[3]), .S0(sel[6]), .Y(muxo[3]));
28    MXT MUX4(.A(muxo[3]), .B(muxi[4]), .S0(sel[5]), .Y(muxo[4]));
29    MXT MUX5(.A(muxo[4]), .B(muxi[5]), .S0(sel[4]), .Y(muxo[5]));
30    MXT MUX6(.A(muxo[5]), .B(muxi[6]), .S0(sel[3]), .Y(muxo[6]));
31    MXT MUX7(.A(muxo[6]), .B(muxi[7]), .S0(sel[2]), .Y(muxo[7]));
32    MXT MUX8(.A(muxo[7]), .B(muxi[8]), .S0(sel[1]), .Y(muxo[8]));
33    MXT MUX9(.A(muxo[8]), .B(muxi[9]), .S0(sel[0]), .Y(out));
34
35    BUFH gd1_0(.A(muxo[8]), .Y(muxi[9]));
36    BUFH gd2_0(.A(muxo[7]), .Y(w_gd2));
37    BUFH gd2_1(.A(w_gd2), .Y(muxi[8]));
38    BUFH gd4_0(.A(muxo[6]), .Y(w_gd4[0]));
39    BUFH gd4_1(.A(w_gd4[0]), .Y(w_gd4[1]));
40    BUFH gd4_2(.A(w_gd4[1]), .Y(w_gd4[2]));
41    BUFH gd4_3(.A(w_gd4[2]), .Y(muxi[7]));
42    BUFH gd8_0(.A(muxo[5]), .Y(w_gd8[0]));
43    BUFH gd8_1(.A(w_gd8[0]), .Y(w_gd8[1]));
44    BUFH gd8_2(.A(w_gd8[1]), .Y(w_gd8[2]));
45    BUFH gd8_3(.A(w_gd8[2]), .Y(w_gd8[3]));
46    BUFH gd8_4(.A(w_gd8[3]), .Y(w_gd8[4]));
47    BUFH gd8_5(.A(w_gd8[4]), .Y(w_gd8[5]));
48    BUFH gd8_6(.A(w_gd8[5]), .Y(w_gd8[6]));
49    BUFH gd8_7(.A(w_gd8[6]), .Y(muxi[6]));
50    BUFH gd16_0(.A(muxo[4]), .Y(w_gd16[0]));
51    ...
52    BUFH gd16_15(.A(w_gd16[14]), .Y(muxi[5]));
53    BUFH gd32_0(.A(muxo[3]), .Y(w_gd32[0]));
54    ...
55    BUFH gd32_31(.A(w_gd32[30]), .Y(muxi[4]));
56    BUFH gd64_0(.A(muxo[2]), .Y(w_gd64[0]));
57    ...
58    BUFH gd64_63(.A(w_gd64[62]), .Y(muxi[3]));
59    BUFH gd128_0(.A(muxo[1]), .Y(w_gd128[0]));
60    ...
61    BUFH gd128_127(.A(w_gd128[126]), .Y(muxi[2]));
62    BUFH gd256_0(.A(muxo[0]), .Y(w_gd256[0]));
63    ...
64    BUFH gd256_255(.A(w_gd256[254]), .Y(muxi[1]));
65    BUFH gd512_0(.A(in), .Y(w_gd512[0]));
66    ...
67    BUFH gd512_511(.A(w_gd512[510]), .Y(muxi[0]));
68 endmodule
```

Appendix D

Innovus Command Log for ICV4

In the code below ... is to signify that some lines were removed to reduce the overall length.

```
#  Innovus Command Logging File
#  Created on Thu Nov 10 11:05:15 2022
########################################################
set_db init_power_nets {VDD}
set_db init_ground_nets {VSS }
set_db init_oa_ref_libs { sc9_45rfsoi_base_svt MSOA_PDK}
#@ source dpd/outputs/Nov2/pd3.invs_setup.tcl
read_sdp_file dpd/outputs/Nov2/pd3.sdp
set_floorplan_row_spacing_and_type 0.14 1
create_floorplan -site sc9_45rfsoi -core_size 120 70 10.08 10.08 10.08
    10.08
set_db place_global_uniform_density true
place_design
create_floorplan -site sc9_45rfsoi -core_size 119.98 70.0 10.08 10.08
    10.08 10.08
set_db place_global_sdp_alignment true
set_db place_global_sdp_place true
set_db place_sdp_legalization sw
set_db place_sdp_max_move_action leave_overlap
set_db place_sdp_max_move_distance 150.0
set_db place_sdp_pre_fixed_cells_blockage_direction y
place_opt_design
set_cell_padding -cells * -right_side 1 -left_side 1 -top_side 1 -
    bottom_side 1
place_opt_design
create_floorplan -site sc9_45rfsoi -core_size 149.94 150 10.08 10.08 10.08
     10.08
place_opt_design -incremental
place_design
read_sdp_file dpd/outputs/Nov2/pd3.sdp
```

```
place_opt_design -incremental
place_opt_design
set_db place_detail_sdp_alignment_in_refine true
place_opt_design
place_design
create_floorplan -site sc9_45rfsoi -core_size 149.94 80 10.08 10.08 10.08
    10.08
add_well_taps -cell BITIE8_A9TS -cell_interval 160  -skip_row 5 -prefix
    WELLTAP
add_rings -nets {VDD VSS} ...
add_rings -nets {VDD VSS} ...
connect_global_net VSS -type pg_pin -pin_base_name VSS -inst_base_name * -
    verbose
connect_global_net VDD -type pg_pin -pin_base_name VDD -inst_base_name * -
    verbose
set_db route_special_via_connect_to_shape { noshape }
route_special -connect {...}
place_opt_design
refine_place
route_design
create_text -layer SXCUT  -point 2.0 2.0 -height 0.05bulk!-label
create_text -layer SXCUT -label 'sub!' -point 2.0 2.0 -height 0.05
check_drc
check_connectivity -type all -error 1000 -warning 50
write_db -oa_lib_cell_view {slkrass pd_v3 layout}
write_netlist -phys -exclude_leaf_cells -flatten_bus dpd/outputs/Nov2/
    pd3_innovus.v
write_sdf dpd/outputs/Nov2/pd3_inn.sdf
write_sdp_file dpd/outputs/Nov2/tmp0.sdp
route_special -connect {...}
create_pg_pin -name VSS -net VSS -geometry M2 5.148 5.245 5.148 5.245
create_pg_pin -name VSS -net VSS -geometry M2 5.148 5.245 5.148 5.245
create_pg_pin -name VSS -net VSS -geometry M2 4.574 4.505 5.763 5.771
create_pg_pin -name VDD -net VDD -geometry M3 7.425 91.258 8.595 92.369
create_pg_pin -name VDD -net VDD -geometry M3 7.425 91.258 8.595 92.369
write_sdf dpd/outputs/Nov2/pd3_inn.sdf
write_netlist -phys -exclude_leaf_cells -flatten_bus dpd/outputs/Nov2/
    pd3_innovus.v
write_db -oa_lib_cell_view {slkrass pd_v3 layout}
```

Appendix E

Python to Generate SDP for a Specific Buffer

```
import csv
import sys

file = 'gd128.txt'
delay = '128'
name = file.split('.txt')
rows = []
with open(file, 'r') as f:
    while(True):
        line = f.readline()
        if not line:
            break
        elements = line.strip().split("\t")
        row = []
        for item in elements:
            row.append(item)
        rows.append(row)
print(rows)
text = 'datapath pd{}\n{{\n\torigin 0.0 0.0\n'.format(delay)
t2 = '\tcolumn gd{}\n\t{{\n\t\tjustifyBy SW'.format(delay)
count = 1
temp = ''

for j in range(0,len(rows)):
    temp += '\n\t\trow gd{}_{}\n\t\t{{\n\t\t\tjustifyBy SW'.format(delay,j
    +1)
    for i in range(0,len(rows[j])):
        temp += '\n\t\t\tinst gd{}_{}'.format(delay,rows[j][i])
    temp += '\n\t\t}'

```

```
temp += '\n\t}\n}'
t3 = text + t2 + temp

newFile = 'sdp_{}.txt'.format(delay)

with open(newFile,'w') as f:
    f.write(t3)
```

Appendix F

Full Output Text File Generated by Python for the 128 delay step buffer

```
datapath pd128
{
  origin 0.0 0.0
  column gd128
  {
    justifyBy SW
    row gd128_1
    {
      justifyBy SW
      inst gd128_3
      inst gd128_2
      inst gd128_1
      inst gd128_0
      inst gd128_127
      inst gd128_126
      inst gd128_125
      inst gd128_124
    }
    row gd128_2
    {
      justifyBy SW
      inst gd128_4
      inst gd128_33
      inst gd128_34
      inst gd128_63
      inst gd128_64
      inst gd128_93
      inst gd128_94
      inst gd128_123
    }
```

```
    row gd128_3
    {
      justifyBy SW
      inst gd128_5
      inst gd128_32
      inst gd128_35
      inst gd128_62
      inst gd128_65
      inst gd128_92
      inst gd128_95
      inst gd128_122
    }
    row gd128_4
    {
      justifyBy SW
      inst gd128_6
      inst gd128_31
      inst gd128_36
      inst gd128_61
      inst gd128_66
      inst gd128_91
      inst gd128_96
      inst gd128_121
    }
    row gd128_5
    {
      justifyBy SW
      inst gd128_7
      inst gd128_30
      inst gd128_37
      inst gd128_60
      inst gd128_67
      inst gd128_90
      inst gd128_97
      inst gd128_120
    }
    row gd128_6
    {
      justifyBy SW
      inst gd128_8
      inst gd128_29
      inst gd128_38
      inst gd128_59
      inst gd128_68
      inst gd128_89
      inst gd128_98
      inst gd128_119
    }
    row gd128_7
    {
      justifyBy SW
      inst gd128_9
      inst gd128_28
      inst gd128_39
```

```
      inst gd128_58
      inst gd128_69
      inst gd128_88
      inst gd128_99
      inst gd128_118
    }
    row gd128_8
    {
      justifyBy SW
      inst gd128_10
      inst gd128_27
      inst gd128_40
      inst gd128_57
      inst gd128_70
      inst gd128_87
      inst gd128_100
      inst gd128_117
    }
    row gd128_9
    {
      justifyBy SW
      inst gd128_11
      inst gd128_26
      inst gd128_41
      inst gd128_56
      inst gd128_71
      inst gd128_86
      inst gd128_101
      inst gd128_116
    }
    row gd128_10
    {
      justifyBy SW
      inst gd128_12
      inst gd128_25
      inst gd128_42
      inst gd128_55
      inst gd128_72
      inst gd128_85
      inst gd128_102
      inst gd128_115
    }
    row gd128_11
    {
      justifyBy SW
      inst gd128_13
      inst gd128_24
      inst gd128_43
      inst gd128_54
      inst gd128_73
      inst gd128_84
      inst gd128_103
      inst gd128_114
    }
```

```
    row gd128_12
    {
      justifyBy SW
      inst gd128_14
      inst gd128_23
      inst gd128_44
      inst gd128_53
      inst gd128_74
      inst gd128_83
      inst gd128_104
      inst gd128_113
    }
    row gd128_13
    {
      justifyBy SW
      inst gd128_15
      inst gd128_22
      inst gd128_45
      inst gd128_52
      inst gd128_75
      inst gd128_82
      inst gd128_105
      inst gd128_112
    }
    row gd128_14
    {
      justifyBy SW
      inst gd128_16
      inst gd128_21
      inst gd128_46
      inst gd128_51
      inst gd128_76
      inst gd128_81
      inst gd128_106
      inst gd128_111
    }
    row gd128_15
    {
      justifyBy SW
      inst gd128_17
      inst gd128_20
      inst gd128_47
      inst gd128_50
      inst gd128_77
      inst gd128_80
      inst gd128_107
      inst gd128_110
    }
    row gd128_16
    {
      justifyBy SW
      inst gd128_18
      inst gd128_19
      inst gd128_48
```

```
193         inst gd128_49
194         inst gd128_78
195         inst gd128_79
196         inst gd128_108
197         inst gd128_109
198       }
199     }
200 }
```

Appendix G

SDP File used for ICV4 in Innovus

In the code below ... is to signify that some lines were removed to reduce the overall length.

```
datapath pd_two
{
    origin 0.0 0.0
    padding  0   0
    row two
    {
        justifyBy SW
        inst gd2_0
        inst gd2_1
    }
}
datapath pd_four
{
    origin 2.0 3.5
    column col_four
    {
        justifyBy SW
        row gd4_one
        {
            justifyBy SW
            inst gd4_0
            inst gd4_3
        }
        row gd4_two
        {
            justifyBy SW
            inst gd4_1
            inst gd4_2
        }
    }
```

```
}
datapath pd8
{
    origin 2.0 3.5
    column gd8
    {
        justifyBy SW
        row gd8_1
        {
            justifyBy SW
            inst gd8_0
            inst gd8_7
        }
        row gd8_2
        {
            justifyBy SW
            inst gd8_1
            inst gd8_6
        }
        row gd8_3
        {
            justifyBy SW
            inst gd8_2
            inst gd8_5
        }
        row gd8_4
        {
            justifyBy SW
            inst gd8_3
            inst gd8_4
        }
    }
}
datapath pd16
{
    origin 2.0 3.5
    column gd16{ ... }
}
datapath pd32
{
    origin 2.0 3.5
    column gd32{ ... }
}
datapath pd64
{
  origin 0.0 0.0
  column gd64{ ... }
}
datapath pd128
{
  origin 0.0 0.0
  column gd128{ ... }
}
datapath pd256
```

```
85 {
86   origin 0.0 0.0
87   column gd256{ ... }
88 }
89 datapath pd512
90 {
91   origin 0.0 0.0
92   column gd512{ ... }
93 }
```