# Delay-Locked Loops for Multiphase Clock Generation

*Oliver Yu*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-86
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-86.html

May 16, 2025

**Delay-Locked Loops for Multiphase Clock Generation**

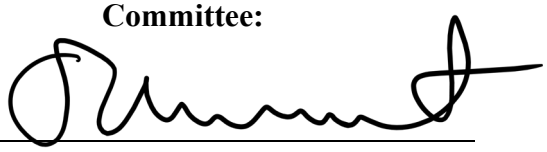
by Oliver Yu

## Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

**Committee:**



Professor Borivoje Nikolic
Research Advisor

5/14/2025

(Date)

* * * * * * *

Professor Jun-Chau Chien
Second Reader

5/15/2025

(Date)

Delay-Locked Loops for Multiphase Clock Generation

by

Oliver Yu

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Borivoje Nikolic, Chair
Professor Jun-Chau Chien

Spring 2025

Delay-Locked Loops for Multiphase Clock Generation

Abstract

Delay-Locked Loops for Multiphase Clock Generation

by

Oliver Yu

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Borivoje Nikolic, Chair

The trend towards chiplet-based architectures not only places strict requirements on high-speed die-to-die interconnects, as defined within the Universal Chiplet Interconnect Express (UCIe) standard, but also creates opportunities to innovate on the designs of these interconnects. Clocking circuits such as delay-locked loops (DLLs) play a crucial role in these high-bandwidth communication links by generating evenly spaced clock phases to support serialization and timing alignment between high-speed signals. As per-lane data rates reach 16 GT/s and higher, clock precision and phase linearity requirements become more stringent. This work is motivated by these demands and explores the design of a digitally controlled DLL architecture targeting the performance needs of UCIe-based die-to-die links.

A digitally-controlled multiphase DLL operating at 8GHz, implemented in the Intel 16 CMOS process, is presented. The DLL achieves 37.96fs RMS jitter and 1.16ps of deterministic jitter while occupying a total area of 32.4$\mu$m by 54.63$\mu$m and consuming 12.5mW of power. The integrated clock lane targets a full 360 degree phase coverage, achieving a 15.6mUI phase resolution with a DNL of 0.2825 LSBs and an INL of 2.245 LSBs. The design was taped out as part of the UCIe module on the Kodiak chip, aiming to contribute to scalable clocking solutions for multi-die systems. This work highlights design challenges and considerations in developing delay locked loops for clocking architectures that meet the precision demands of modern chiplet ecosystems.

To my family and friends.

# Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to express my gratitude to the countless individuals who have supported me throughout my academic journey. Thank you to my advisor Borivoje Nikolic for agreeing to take me on as a Master's student and allowing me the opportunity to work on real, impactful projects. Your insights, guidance, and advice have helped me grow as an individual and student, opening many doors for my future.

Thank you to the UCIe team: Di Wang, Rahul Kumar, and Rohan Kumar. I have learned so much from working with you and simply listening to your discussions. Thank you for all your patience and best of luck in your PhDs and further work with analog/mixed-signal circuits. Additionally, I would like to thank Nikhil Jain and Bob Zhou for their technical contributions, without which this work would also not have been possible.

Shoutout to Evan Frick, Dhruv Vaish, and Leo Huang. Thanks for being amazing roommates – I will miss our conversations, dinners, and Hilgard home deeply.

Thank you to my mom, dad, and brother. I would not be here without your love and support, and I feel truly fortunate to have had such great role models to look up to my entire life.

Lastly, I am beyond grateful to Apple for their generous support through the Apple Masters Scholarship and the Siebel Foundation for their contribution towards furthering my education.

# Chapter 1

# Delay-Locked Loops

## 1.1 Motivation

The slowing of technology scaling has compelled the semiconductor industry to discover alternate solutions to the ever-increasing demand for performance while minimizing costs. In previous decades, technology scaling has enabled improved performance, power and cost per transistor. However, as process nodes reach sub-5nm, the physical limitations of the devices and manufacturing process prevents this continued trend. Instead, new design frameworks must be explored to adapt to the continuously evolving workloads.

In particular, the industry has turned towards chiplets and heterogenous integration as a solution to these challenges. Previous implementations of VLSI design rely on System-on-Chip (SoC) architectures which integrate all required compute, memory, accelerators onto a single monolithic chip. However, larger and more complex dies have greater probability of critical defects that may leave the silicon unusable, lowering overall yield. Additionally, the required design effort for developing designs on newer nodes will only increase with the existing design framework. In contrast, chiplet-based architectures, which fall under the category of System-in-Package (SiP), assemble multiple smaller, function-specific "chiplets" within a single package. Such architectures are becoming increasingly preferred for their improved yield, design flexibility, and scalability at advanced nodes. Standardized high speed die-to-die interconnects provide the necessary technology for developing this chiplet ecosystem, making it a critical area to be researched.

Within die-to-die interconnects, there are multiple layers to the interface that convert between digital data and the transmitted high speed analog signals. Precise alignment between the transferred data and the clock is required to send and receive data at such high speeds while minimizing the error rate of the transfer. Delay locked loops are essential circuits to these high speed digital and mixed signal systems for their ability to address these timing challenges. They create well-defined phase relationships between clock and data signals, ensuring accurate sampling. DLLs have a wide range of applications across these systems which are highlighted in Section 1.4.

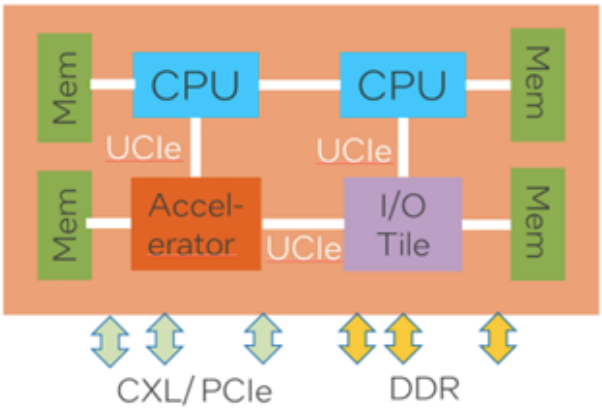



Figure 1.1: An example package connected via UCIe [20].

Delay locked loops are not novel circuits and have been around since 1961 with their use in CMOS design starting in the mid-1980s [17]. Their utility in these modern systems has motivated its continuous innovation, causing design techniques and architectures to evolve over time in order to address the shifting performance requirements. This thesis aims to provide a comprehensive introduction to delay locked loop design techniques along with a detailed analysis of a proposed design, considering the aforementioned techniques. This thesis highlights not only what was designed but also the process and tools used to design the circuit in hopes of benefiting future students working on similar projects. The remaining sections of this chapter further motivate the applications and specific use cases of delay locked loops. We discuss existing delay locked loop design techniques including conventional architectures, general design challenges, and existing state of the art solutions. We then propose a digitally-controlled delay locked loop designed in Intel 16 and being taped out as part of a UCIe-compliant die-to-die interface in the Kodiak chip. Lastly, post-extraction simulation results of the proposed DLL are presented in Section 4, followed by a conclusion and broader discussion about the future directions of this work.

## 1.2 Overview

At a high level, delay locked loops are tunable delay lines in closed loop feedback. They are mixed signal circuits that measure the delay between an input and an output signal (typically clock signals) and use that information to help push the delay towards a locking point where the two signals have a desired phase relationship [17]. Delay locked loops have certain features that make them attractive circuits for a wide range of applications.

### Type I vs. Type II DLLs

There are two flavors of DLLs depending on which signals are compared by the circuit. Type I DLLs will take a reference clock and lock it with some delayed version of the same reference clock. Alternatively, type II DLLs will compare two distinct signals against another (e.g. 2 different clock sources) and attempt to lock the two using similar feedback structures [10]. In this work, type I DLLs are studied and presented.


(a) Type I DLL (b) Type II DLL

Figure 1.2: A comparison between Type I and Type II DLLs [10].

## 1.3 PLLs vs. DLLs

Phase-locked loops (PLLs) are a broader class of feedback circuits that perform phase synchronization between two signals, aiming to minimize any phase error between them [17]. This synchronization can be achieved in two primary ways: by delaying the reference clock or by adjusting the output frequency. Circuits that delay the reference clock are typically referred to as DLLs, whereas those that change the output frequency are conventionally classified as PLLs.


(a) DLL (b) PLL

Figure 1.3: A comparison between PLL and DLL architectures [13].

DLLs exhibit advantages over PLLs in specific applications. As first-order systems, DLLs are unconditionally stable, unlike PLLs which are higher-order systems and require more careful stability analysis. Additionally, DLLs offer superior jitter performance because they do not accumulate jitter in the same manner that PLLs do [13]. However, most DLLs do not inherently support frequency synthesis unless specific architectures, such as multiplying DLLs (described briefly in Section 1.4) are employed. DLLs are generally simpler, consume

lower power, and are more suitable for applications where frequency synthesis is unnecessary. Structurally, PLLs use voltage-controlled or digitally controlled oscillators, whereas DLLs rely on voltage-controlled or digitally controlled delay lines (Figure 1.3). While both share the fundamental goal of phase alignment, the increased complexity of PLLs introduces a broader range of design challenges compared to the more straightforward design of DLLs [4].

## 1.4 Applications of Delay-Locked Loops

Delay locked loops are often used within the context of clocking, due to their ability to create precise phase relationships in the presence of frequency, process, voltage and temperature variations. They have a variety of applications within this area, as outlined in the following section, with the primary emphasis of this work being their application to multiphase clock generation.

### Multiphase Clock Generation

For high speed die-to-die interfaces, it is important to have a precise clock signal that can sample data at a specific location to minimize any probability of errors in the data transfer. In order to do so, having equally spaced phases of a clock signal will allow peripheral circuitry to sample data with the optimal clock phase. Generating these precise phases of clock requires the use of a feedback circuit to ensure that the phases remain tolerant to frequency and PVT variations. [17]

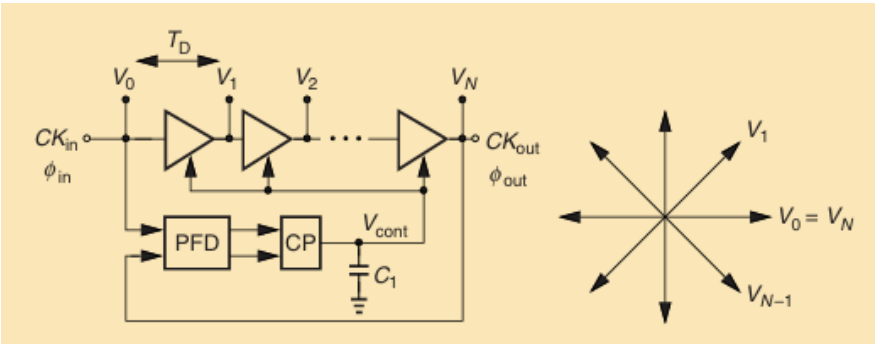
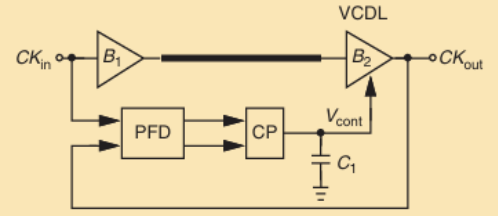



Figure 1.4: A diagram of a DLL for multiphase clock generation [17].

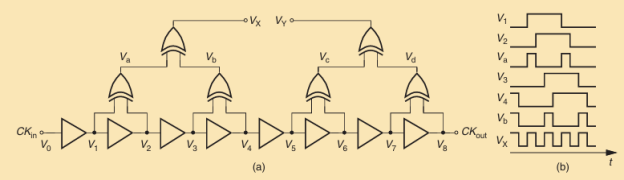
Figure 1.4 portrays a diagram of DLL used for multiphase clock generation. The core principle is that the entire delay line will lock to some fraction or multiple of the reference clock period. Upon lock, the tapped outputs from each delay cell should theoretically yield equally spaced phases of the reference clock. The work presented in this thesis focused on using delay locked loops specifically for multiphase clock generation and the challenges that are associated with it.

## Deskew/Zero-delay Buffering

During clock distribution, clock signals may accumulate some nontrivial delay due to the parasitics of the distribution network. This can result in the edges of the clock signal arriving at different sequential elements with varying delays, leading to clock skew which can degrade the performance of the digital circuit. Such delays cannot be matched with open loop control and therefore requires the use of feedback to ensure that the distributed clocks are in phase with the original reference clock [17] [22]. Figure 1.5a demonstrates an example where the DLL will update the $B_2$ delay such that $\mathrm{CK_{in}}$ is phase aligned to $\mathrm{CK_{out}}$, despite experiencing an unknown delay $B_1$.




(a) Deskew




(b) Frequency Multiplication

Figure 1.5: Examples of DLLs configured for deskew and frequency multiplication [17].

## Frequency Multiplication

Although previously DLLs were presented as circuits for non-frequency synthesis applications, in certain scenarios, DLLs can be configured to generate new frequencies. However, the generated frequencies are typically limited to specific integer multiples of the reference clock frequency. One simple method of frequency multiplication involves using an edge combiner circuit implemented as a hierarchy of XOR gates [17] (Figure 1.5b). DLLs designed for frequency synthesis applications may experience mismatch between the clock phases and spurs within the output waveform both undesirable effects that would need to be addressed within the design. A multiplying DLL (MDLL) is another flavor of delay-locked loop that is embedded within a secondary control loop enabling frequency multiplication by aligning and combining delayed clock edges to generate a higher-frequency output.

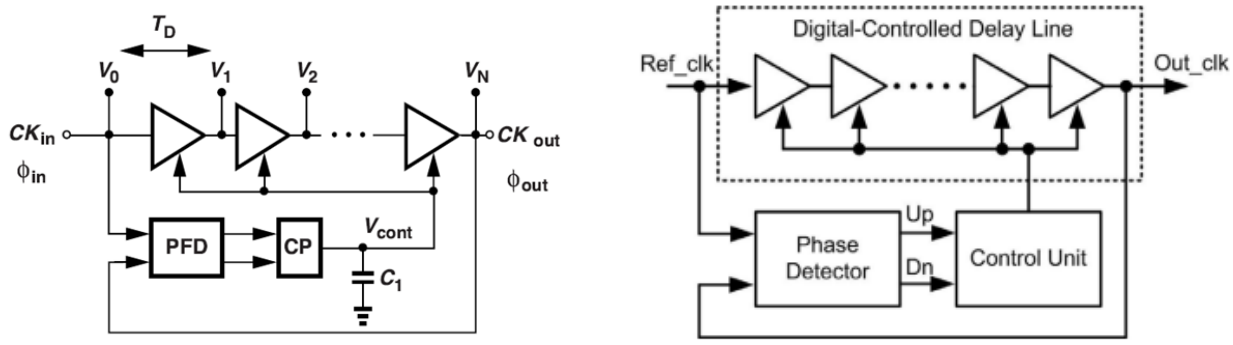
# Chapter 2

# Design Techniques

DLL architectures largely come in two flavors: analog and digital, with a handful of modern architectures utilizing a combination of the two. Each has their distinct advantages and with recent trends in technology scaling, digital DLLs have grown in popularity. This section will first highlight these conventional architectures and their high level tradeoffs. We then follow by describing some basic modeling of DLLs to understand its loop dynamics, followed by a discussion of common implementations of the sub-circuits, and a comparison of state-of-the-art DLL designs.

## 2.1 Conventional DLL Architecture

A conventional analog DLL circuit is shown in Fig. 2.14. It comprises of a phase-frequency detector (PFD), a charge pump (CP), a loop filter (LF), and a voltage-controlled delay line (VCDL). Analog DLLs are capable of generating a high-resolution delay step with low jitter and low power. However, analog DLLs are more sensitive to PVT variations, generally have greater design complexity, and are not very scalable compared to digital DLLs. Analog DLLs are more susceptible to noise and mismatch especially in the charge pump design whereas digital DLLs have additional quantization noise to consider. Area is dependent on the specific architecture employed as analog DLLs require a large on-chip capacitor whereas digital DLLs may contain large digital control blocks.




(a) Analog DLL (b) Digital DLL

Figure 2.1: Comparison between conventional DLL architectures.

Digital DLLs have gained increasing popularity in modern high-speed systems due to their wide phase capture range, tolerance to PVT variations, improved control capabilities, and overall design portability across process nodes. Functionally, digital DLLs share the same core concept as their analog counterparts. However, unlike analog DLLs that rely on analog feedback loop consisting of a charge pump and capacitor to generate a precise DC voltage, digital DLLs use clocked control logic to adjust a multi-bit digital code that configures the delay elements in quantized steps.

With continued technology scaling, digital circuits have become faster and more efficient, driving this transition toward more digitally-based clocking architectures. Some modern digital DLLs can be implemented entirely using synthesizable RTL, including the delay line, control logic, and phase detector. These architectures benefit from compatibility with standard digital design flows, enabling faster design iterations, easier verification, and quick portability across technology nodes. The fully digital implementations may suffer at higher speed operations where precise physical design is required and often require additional calibration techniques.

Leveraging digital VLSI automation tools and layout generators, designers can achieve scalable and replicable implementations, reducing manual effort and design time. These advantages make digital DLLs especially attractive in applications requiring rapid development and tolerance to manufacturing variability. As previously highlighted, analog DLLs can offer superior performance in specific metrics such as power and jitter assuming a highly optimized design. However, their requirement for complex analog circuit techniques to overcome non-idealities particularly in the charge pump design greatly increases complexity and limits scalability. Many modern architectures rely on flexible digital control with custom analog delay elements which can be tailored towards the required specifications of the circuit.

## 2.2 DLL Modeling

To better understand the loop dynamics, it is helpful to develop models of delay locked loop systems.

### S-domain Model

We begin by creating a continuous-time s-domain model to analyze the DLL circuit, enabling us to characterize overall loop dynamics. We assume that the loop bandwidth is much smaller than the reference clock frequency, or in other terms that the DLL settling/locking time is much longer than one reference clock period. This assumption allows us to neglect the finite delay through the delay line in our initial modeling [17] [5]. The s-domain representation of a typical charge-pump based DLL is demonstrated in Figure 2.2 where $K_{\mathrm{PD}}, K_{\mathrm{CP}}$, and $K_{\mathrm{DL}}$ are the gains for the phase detector, charge pump and delay line, respectively.

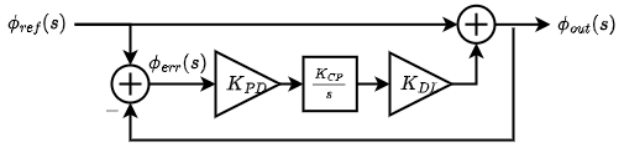



Figure 2.2: S-domain model of a DLL.

By solving for the closed-loop transfer function using the model in Figure 2.2, we derive the all-pass result in Equation 2.2.

$$\phi_{\mathrm{out}}(s) = \phi_{\mathrm{ref}}(s) + (\phi_{\mathrm{ref}}(s) - \phi_{\mathrm{out}}(s))K_{\mathrm{PD}}\frac{K_{\mathrm{CP}}}{s}K_{\mathrm{DL}} \tag{2.1}$$

$$\phi_{\mathrm{out}}(s) = \phi_{\mathrm{ref}}(s) \tag{2.2}$$

To understand the all-pass behavior of the DLL intuitively: 1) At low input frequencies, the loop gain is high, causing the output to closely track the input. 2) At high input frequencies, the loop gain rolls off, and the input phase fluctuation propagates directly to the output with a delay imposed by the delay line [17]. Furthermore, calculating the phase margin of our system, we see that the loop gain under this model is given by:

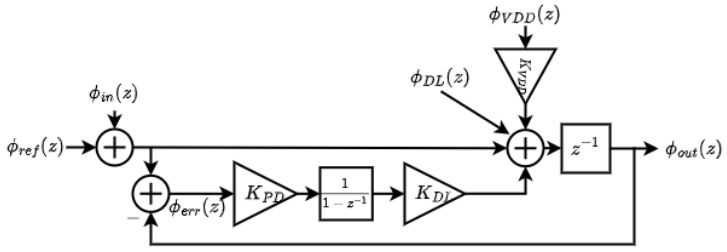
$$\mathrm{LG}(s) = \frac{K_{\mathrm{PD}}K_{\mathrm{CP}}K_{\mathrm{DL}}}{s} \tag{2.3}$$

This is the transfer function of an integrator, which has a phase margin of 90° and is unconditionally stable. However, the assumption that the loop bandwidth is much smaller than the reference clock frequency imposes limitations. In reality, the DLL exhibits peaking in its transfer function that cannot be captured by the continuous-time s-domain model. To account for this, we instead adopt a z-domain model, treating the DLL as a discretely sampled system [10].

## Z-domain Model

The primary limitation of the s-domain model is that it fails to account for the fact that the output clock $\phi_{\text{out}}$ is a delayed version of the input clock $\phi_{\text{in}}$ [10]. Because of this, the model cannot properly distinguish between input jitter and output jitter and may incorrectly adjust the delay line in the wrong direction leading to jitter peaking. To reduce jitter peaking, the loop bandwidth must be decreased [5]. This allows jitter to propagate to the output before the loop attempts to correct it, thereby avoiding the over-correction that results in peaking. However, this introduces a tradeoff between jitter peaking and loop settling time as a narrower bandwidth slows the loop's response towards the lock point [10].

A z-domain model of a conventional DLL is shown in Figure 2.3a, with additional jitter sources included. The resulting input-to-output transfer function (Equation 2.4) is no longer an ideal all-pass response, but instead exhibits jitter peaking near the loop bandwidth as shown in Figure 2.4a.




(a) Z-domain model.

$$\frac{\phi_{out}(z)}{\phi_{in}(z)} = \frac{(1 + K_{PD}K_{DL}) - z^{-1}}{z - (1 - K_{PD}K_{DL})} \tag{2.4}$$

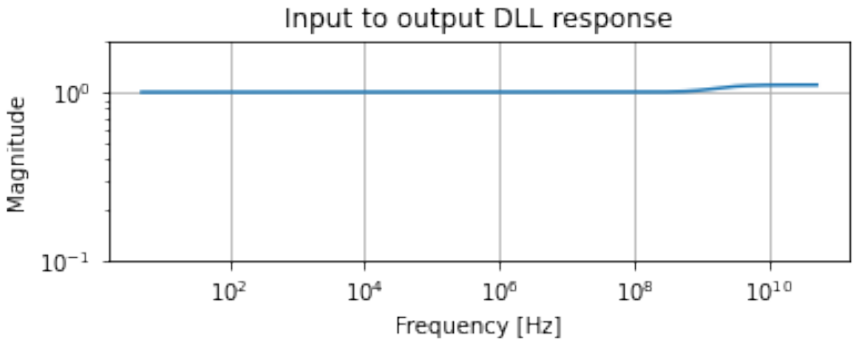
$$\frac{\phi_{out}(z)}{\phi_{V_{\text{DD}}}(z)} = \frac{K_{\text{DD}}}{K_{PD}K_{DL}\frac{1}{1-z^{-1}}z^{-1}} \tag{2.5}$$

$$\frac{\phi_{out}(z)}{\phi_{DL}(z)} = \frac{1}{K_{PD}K_{DL}\frac{1}{1-z^{-1}}z^{-1}} \tag{2.6}$$

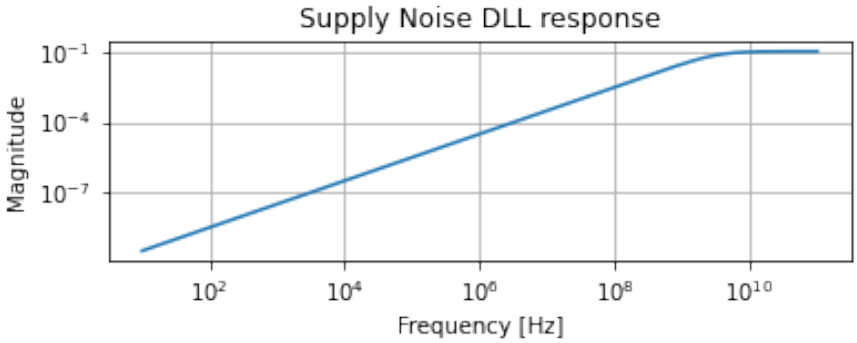
(b) Transfer functions.

Figure 2.3: DLL z-domain model.

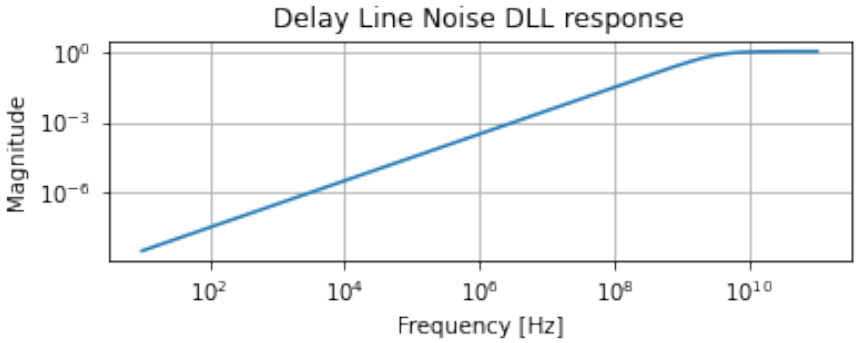
We can also use the z-domain model to offer deeper insight into the jitter transfer characteristics and feedback behavior of DLLs. Both the jitter introduced by supply ($\phi_{V_{\text{DD}}}$) and delay line noise ($\phi_{\text{DL}}$) have a high-pass transfer function, meaning it is suppressed at low frequencies but appears at the output at higher frequencies (Figure 2.4). Further behavioral modeling can be done capture certain performance tradeoffs, thus motivating architectural design decisions.




(a) Input response.




(b) Supply response.




(c) Delay line response.

Figure 2.4: Modeled DLL frequency responses.

## 2.3 DLL Subcircuits

Now, we present examples and descriptions of circuit implementations for the various components of delay locked loops, including delay lines, phase detectors, and the control loop.

## Delay Lines

Delay lines are classified as either voltage-controlled delay lines, in which a DC voltage sets the delay, or digitally-controlled delay lines, where a digital signal programs a discrete delay. For digitally controlled delay lines, three main specifications must be addressed:

- **Minimum delay ($D_{\mathbf{min}}$)**
- **Maximum delay ($D_{\mathbf{max}}$)**
- **Delay resolution ($d_r$)**

The minimum and maximum delays which set the tuning range are determined by the operating frequencies of the circuit and the extent of tolerable PVT variations. The delay resolution is typically set by the required phase resolution from a system perspective. An example transfer function of a generic DCDL is presented in Figure 2.5, which demonstrates quantized delay steps as a function of a digital code. Other considerations include linearity, jitter and variability which will impact the overall function of the delay line.

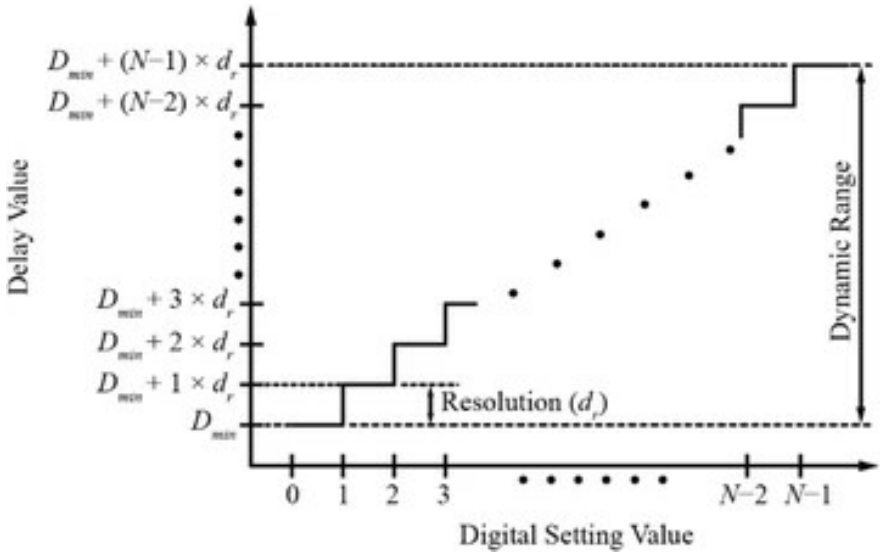
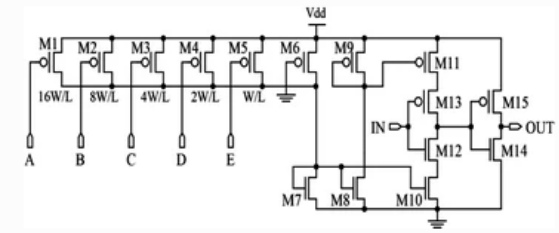


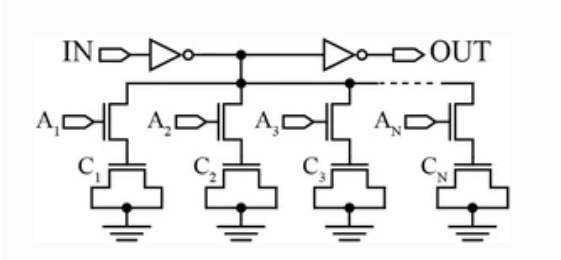

Figure 2.5: A generic DCDL transfer function [22].

There are a couple broad groups of delay lines: gate delay and sub-gate delay [1]. Gate-based delay lines rely on cascading a variable number of fixed delay elements to generate various delay steps. Alternatively, sub-gate delay elements can achieve much finer delay resolutions by tuning the RC delay of the circuit by either modifying the drive strength of a device or altering its load capacitance. Adjusting either of these parameters changes the effective propagation delay through the circuit. Figure 2.6 illustrates common architectures used to implement sub-gate delay elements for fine-tuning. Among these, the current-starved

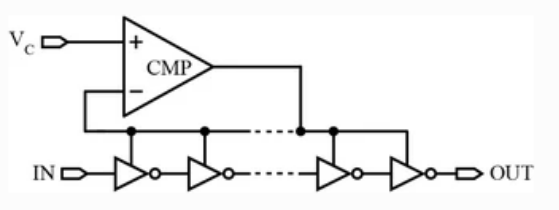
inverter and the shunt-capacitor-loaded inverter are two of the more widely utilized designs in literature.



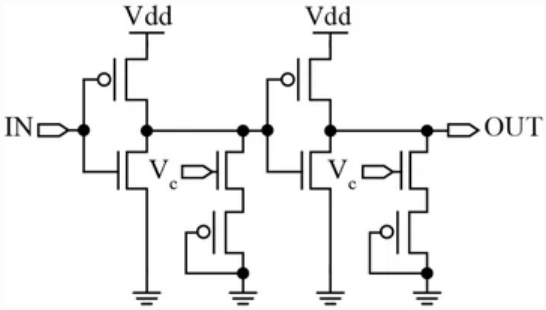

(a) Current-starved delay line.




(b) Shunt-capacitor delay line.




(c) Supply-regulated delay line.




(d) MOS-diode delay line.

Figure 2.6: Diagrams of delay line designs [1].

A high-level comparison between current starved, shunt capacitor and digital gate delay lines was conducted in [25], with the results below (ranked from 1 = Best to 3 = Worst). From this study, it is important to note that while gate based delay cells have the widest tuning range they suffer from limited phase resolution, making them largely unsuitable for high-speed interconnect applications. Between current-starved and shunt-capacitor delay lines, neither topology consistently outperforms the other; the choice depends heavily on the target performance metrics.

Table 2.1: A high-level comparison of 3 delay line topologies [25].

| **Topology** | **Delay Range** | **Resolution** | **Power** | **Area** | **Linearity** | **Process Robustness** | **Temp Robustness** |
|---|---|---|---|---|---|---|---|
| Current Starved | 3 | 2 | 1 | 1 | 3 | 3 | 2 |
| Shunt Capacitor | 2 | 1 | 2 | 2 | 2 | 2 | 1 |
| Digital Gate Based | 1 | 3 | 3 | 3 | 1 | 1 | 3 |

## Phase Detectors

Phase detectors can be broadly categorized into two types: linear and non-linear. In linear phase detectors, the output is proportional to the input phase error which typically gets integrated over to generate an adjustment to a control voltage. These are often utilized in analog-controlled DLLs, where continuous control signals adjust the delay of the delay line.

In contrast, non-linear phase detectors, which largely consist of binary/bang-bang phase detectors, only output the sign of the phase error, without information about its magnitude. Though suitable for digitally controlled DLLs, these detectors inherently introduce quantization error and limit-cycle behavior, creating additional challenges for high precision locking, which will be discussed. Their generic transfer functions are shown in Figure 2.7.

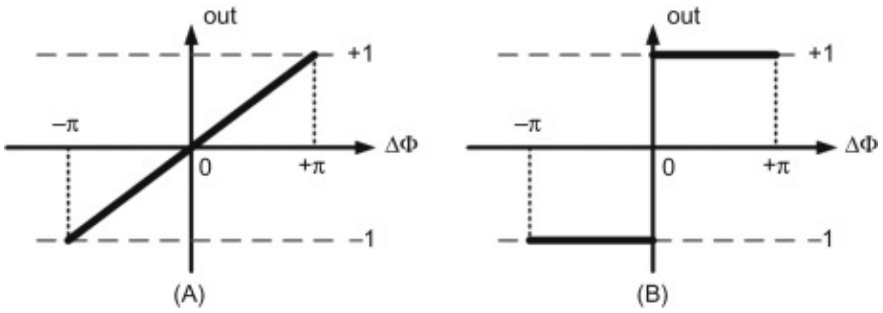



Figure 2.7: Linear and binary phase detector transfer functions [24].

### Linear Phase Detectors

The most common linear phase detector is the phase frequency Detector (PFD), whose schematic and operation are illustrated in Figure 2.8. Its functionality is straightforward: when the reference clock leads the output clock, the UP signal is asserted for a duration proportional to the phase error. Conversely, when the reference clock lags the output clock, the DOWN signal is asserted accordingly [13].

Slowly the loop should push the delay line in a direction as to minimize the phase error. When lock is achieved, there would ideally be 0 phase error and thus neither UP nor DN should be high, or their pulse widths will be exactly equal as to cancel each other out. In both cases, the resulting control voltage should be constant indicating that the DLL has locked.

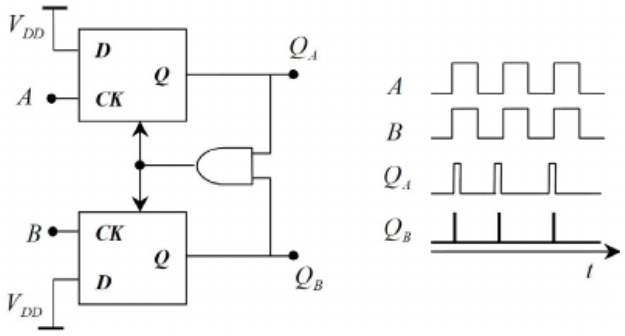



Figure 2.8: Schematic and operation of a conventional phase-frequency detector.

### Binary/Bang-Bang Phase Detectors

A binary or bang-bang phase detector (BBPD), in its simplest form, consists of a flip-flop that uses internal positive feedback to determine whether the output clock leads or lags the reference clock. This mechanism helps minimize the likelihood of entering a metastable state during phase detection. A key design parameter is the sampling window ($d_{\text{sw}}$), defined by the setup and hold times of the flip-flop, which determines the detector's phase resolution. Minimizing this window is essential, as it directly impacts the minimum achievable phase error in the locked DLL. While the clk-to-q delay of the flip-flop is not as critical, it may become relevant if it contributes significantly to the loop latency of the DLL which may affect the locking response of the system [22].

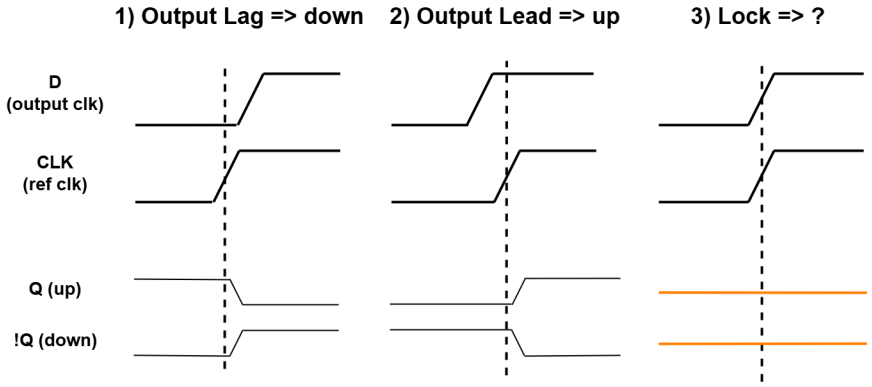



Figure 2.9: Bang-bang phase detector operation.

## Control Loop

For analog DLLs, the control loops typically consists of a charge pump followed by a capacitor which converts the phase error output from the phase detector into a change in voltage.

Digital DLLs span a range of control mechanisms. In its simplest form, the control circuitry employs shift-register-based control to incrementally adjust a thermometer coded the delay line. This digital selection can also be down with up/down counters driving binary-weighted delay elements to achieve finer resolution and faster convergence. To further improve locking speed, hybrid or adaptive control strategies such as successive approximation register (SAR) or time-to-digital converter-based updates can be employed. In the most general sense, the control unit is a state machine that uses the phase error over time and converts it to a digital code to update the delay line.

## 2.4 Challenges with Digital Delay Locked Loops

### Minimize Locking Phase Error

The ultimate goal of delay locked loops is to minimize the phase error of the DLL outputs when in locking mode. There are a couple primary considerations to determine the phase error [4]:

1. **Delay line resolution**: The resolution of the delay line determines how fine the delay line adjustment is and therefore how close the locking point is the desired phase alignment.

2. **Phase detector deadzone**: The precision of the phase detectors determines what phase error is detectable and can be corrected for.

Together, these design parameters influence the DLL's ability to minimize phase error. Both must be designed with the same target resolution in mind. In a design with a very fine delay line resolution but a coarse phase detector, the loop will never be able to achieve less phase error than what is detectable by the phase detector, therefore limiting the capability of the DLL to track the phase. On the other hand, a coarse delay line with a very precise phase detector will similarly be limited by the quantization error of the delay line.

Furthermore, in the context of multiphase clock generation, there also exists phase mismatch or multiphase skew. This is when the entire delay line is correctly locked but the produced intermediate phases are not equally spaced. Such errors typically originate from poor matching in the design, either through differences in the output loads or layout parasitics.

### Tuning Range vs. Phase Resolution

When designing digitally controlled delay lines (DCDLs), the total tunable delay range is inherently determined by the number of tuning steps multiplied by the delay resolution per step, assuming a linear response. As a result, increasing the resolution or decreasing the delay per step will have the impact of narrowing the overall delay range. To extend the

range without sacrificing resolution, designers can increase the number of tuning bits, though this typically comes at the expense of added complexity and area. In many cases, especially when wide tuning ranges are required, delay lines are implemented with both coarse and fine tuning controls. This approach enables broad frequency coverage while still maintaining the fine resolution necessary for precise control.

## Loop Stability

Analyzing the feedback behavior of digitally controlled DLLs is somewhat challenging due to the nonlinear characteristics of the phase detector and control unit. While advanced techniques such as linearizing the nonlinear model in presence of noise exist [8], such methods fall outside the scope of this work.

Instead, a more intuitive approach is presented in [22], which draws parallels to continuous-time systems by modeling loop delay in terms of phase, enabling a conceptual "phase margin" analysis. In systems using binary phase detectors, some degree of limit cycling is unavoidable since the detector can only indicate whether the output phase is leading or lagging. In other words, the true phase relationship will be never perfectly aligned as there is no method to encode zero phase error for a binary phase detector as shown in the discontinuity of the BBPD transfer function in Figure 2.7.

However, depending on the nature of the feedback loop, there is still possibility of "positive feedback" which leads to cycling between more than just two codes around the lock point. As proposed in [22], *a digital DLL remains stable as long as a DCDL correction is not issued before the effect of the previous correction has been fully evaluated by the control mechanism.* In other words, stability is a function of the relationship between loop delay and the delay line update rate.

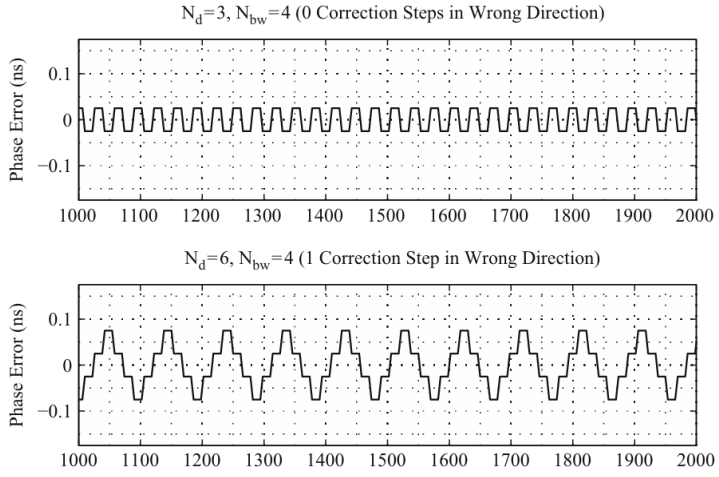



Figure 2.10: Impact of loop latency/stability on DLL code cycling [22].

Loop delay can stem from multiple sources, including flip-flop synchronization after the phase detector and latency through the digital control logic. If the cumulative loop latency exceeds a certain threshold, it reduces the system's effective phase margin. This introduces a form of positive feedback, where a correction is applied after the next phase comparison has already been made, resulting in a misstep before the system can react and correct the error. This behavior is illustrated in Figure 2.10, where erroneous corrections in the wrong direction are clearly visible, resulting in cycling between four different delay codes as opposed to just two. Such positive feedback was observed in the 2024 Q3 DLL design, and solutions to address it are discussed in Section 3.8.

### Phase Detector Precision

Several key issues arise in phase detector design that can have a large impact on the behavior of the DLL. One major concern is input offset caused by mismatched edge rates between the reference and output clocks, which can degrade locking accuracy [13]. Additionally, PDs often exhibit nonlinearity near zero phase error, the intended locking point, leading to undefined behavior if the detector output becomes metastable. This region of uncertainty, commonly referred to as the "deadzone", is determined by the setup and hold times of the flip-flops used in the PD [22]. To ensure robust locking and minimal residual phase error, it is recommended to minimize both the offset and the deadzone in the phase detector design.

## 2.5 Modern Delay Locked Loops

Modern multiphase clock generation requires higher speed, lower jitter, lower power, and minimal area. Here is an overview of some of the state of the art architectures which proposed solutions to the previously mentioned challenges.

The DLL proposed in [16] is a digital DLL that aims to address the problems of wide tuning range, high bang-bang jitter, and multiphase skew. The proposed architecture includes a reconfigurable delay line. Such a delay line allows for several modes of operation based on the desired range of operation. To address jitter and skew concerns, the DLL relies on digital logic to both sequentially update the delay cell delays and perform a calibration procedure to ensure matched delays between all 4 elements of the delay line. Ultimately, this work demonstrates using a flexible and reconfigurable delay line design in parallel with additional correction techniques to achieve wide tuning range, minimal multiphase skew and reduced jitter.

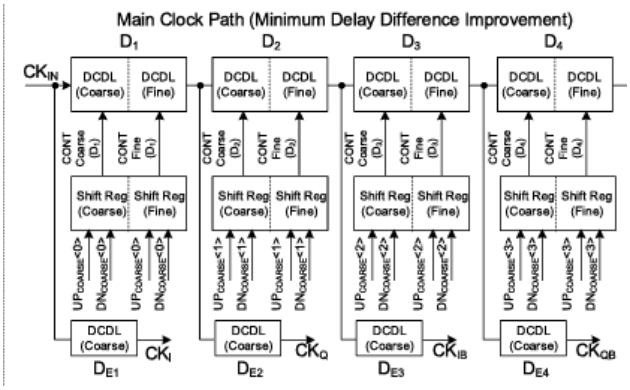



Figure 2.11: Reconfigurable DCDL proposed in [16].

Another solution to achieving wide range operation was presented at ISSCC 2025, where various transceiver architectures were proposed from both industry and academia. In a TSMC/AMD collaboration, a delay locked loop design was presented employing 12 inverting delay stages each with coarse and fine tuning capabilities via current starving and shunt capacitance loading (Figure 2.12) [12]. Their DCDL produces 12 equally spaced stages which serve as inputs into a phase interpolator that further divides the clock into finer granularity. Their clock lane is capable of achieving a 0.9375-degree phase adjustment resolution with 1.06-LSB INL and 0.37-LSB DNL. Furthermore, their use of input and output dummy delay cells highlights the importance of matched drive strengths and loads for each tapped output of the delay line. With such stringent performance requirements, any mismatched capacitance or routing differences can easily introduce phase error, thus reducing eye width for the clocking path and overall BER.

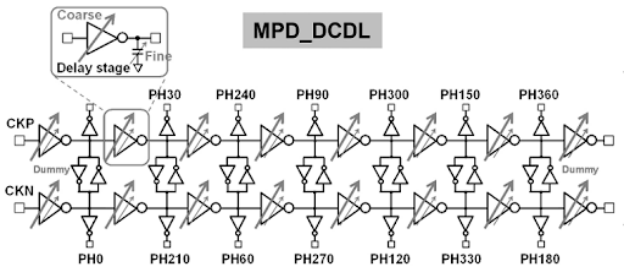
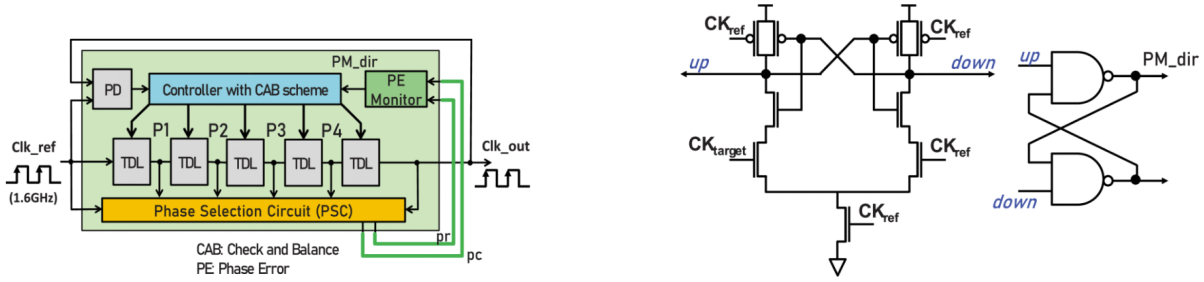



Figure 2.12: DCDL proposed by TSMC/AMD [12].

Modern delay-locked loop architectures have also targeted the increasing demand for higher phase precision. In particular, [4] proposes several techniques to address phase mismatch issues that arise in conventional DLLs. Such approaches can suffer from phase errors due to process and layout-induced variations. The proposed design employs a cell-based delay line combined with a supplementary calibration scheme that actively compensates for

phase mismatches after locking. While their proposed work seems effective, adding these calibration loops introduces additional overhead and design complexity, potentially shifting the precision burden away from the core DLL to the calibration circuit itself. Moreover, such schemes may not scale as efficiently to higher-frequency designs. The work also explores the use of a sense amplifier flip-flop based phase detector, which offers a smaller deadzone compared to conventional standard-cell-based phase detectors, further improving phase detection accuracy.




(a) DLL Diagram.

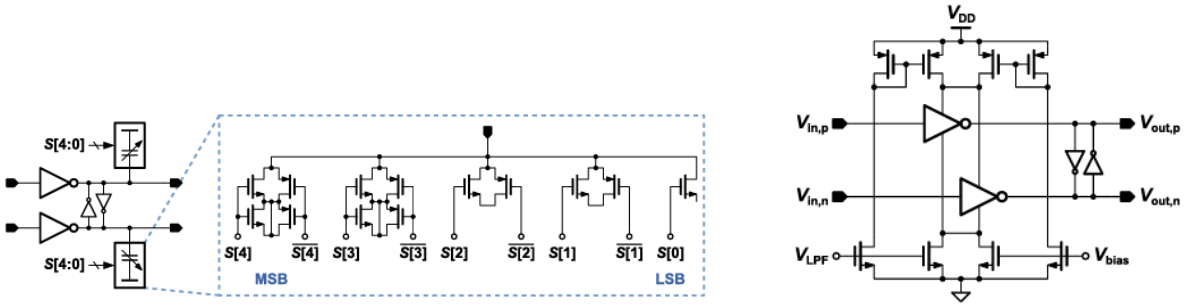
(b) SAFF-based Phase Detector.

Figure 2.13: Multiphase calibration DLL proposed in [4].

While becoming less popular with technology scaling, [23] proposes an 16-phase analog DLL that can achieve very low phase error. The paper describes traditional challenges with conventional CP-based analog DLLs in particular with the charge pump design. This paper emphasizes how modern analog DLL architectures suffer from phase mismatch issues from mismatch between the pull-up and pull-down paths of the charge pump. These variation sources can be minimized through precise design and symmetric layout techniques. Furthermore, another takeaway from this paper is the requirement for proper load matching on the delay line. Any sort of asymmetry can lead to noticeable reductions in performance at such high speeds. Overall, while this thesis is not focused on analog delay locked loop design techniques, this paper demonstrates several core design principles that are universal across DLL designs and serves as a reference for performance comparisons between analog and digital DLLs.

The last paper demonstrates some tradeoffs between power efficiency, jitter, and area, all of which are important specs to optimize for depending on the design requirements [2]. For high speed links, power consumption is arguably one of the largest concerns when it comes to the increasing bandwidth requirements for high speed links. As a solution, two different delay locked loops are proposed one that implements a shunt capacitor based digitally controlled delay line and one that relies on a voltage controlled current starved inverter based VDCL, which results in lower power consumption. Nevertheless, contrasting the two designs side by side there are clear pros and cons to doing either analog or digital design when it comes

to delay locked loops [2]. Furthermore, the methodology for designing the shunt capacitor based delay line was referenced for the presented work.




(a) Proposed shunt capacitor DCDL.

(b) Proposed current starved DCDL.

Figure 2.14: Two proposed DCDLs in [2].

Additionally, Cadence recently presented a UCIe-compliant die-to-die link at ISSCC 2025. Notably, their transmitter lane does not employ a DLL for its multiphase clock generation, instead relying on a PLL[1] to generate the necessary clock phases for both data and clock phase interpolators [14]. Despite this architectural difference, the work is significant and worth highlighting here, as the reported power consumption of 297 mW for the full UCIe module, corresponding to only 0.29 pJ/bit, represents one of the most power-efficient implementations in the industry to date.

## Summary of Published DLLs

A summary of the performance of the presented designs is shown in Table 2.2.

[1]Details on the multiphase PLL architecture were not provided within the paper.

Table 2.2: A summary of recently published DLLs with similar target specifications.

| **Spec** | **Park**[16] | **Chang**[4] | **Yang**[23] | **Angeli**[2] | **Angeli**[2] | **Lin**[12] | **Melek**[14] |
|---|---|---|---|---|---|---|---|
| **Year** | 2021 | 2024 | 2023 | 2020 | 2020 | 2025 | 2025 |
| **Process** | 28nm | 90nm | 40nm | 65nm | 65nm | 3nm | 3nm |
| **Control** | Digital | Digital | Analog | Digital | Mixed | Mixed | – |
| **Delay Element** | NAND DCDL + PI | – | Current Starved | Shunt Capacitor | Current Starved | Coarse Current Starved + Fine Shunt Capacitor | N/A (uses PLL for multiphase generation) |
| **Frequency** | 1.3-4GHz | 1.6GHz | 2-7.4GHz | 2.5GHz | 2.8-3.6G | 4-8GHz | 2-8GHz |
| **Phase Detector** | – | SAFF | PFD | PDC | PDC + $\Delta\Sigma$ + LPF | – | – |
| **Jitter** | 1.82 (RMS) 12.5p (P2P) | 0.57ps | 0.496ps | 1.2p | 0.86p | – | – |
| **Power** | 6.5 mW | – | 18.3 mW | 4.1 mW | – | 0.6 pJ/bit [1] | 0.29 pJ/bit[2] |
| **Power Efficiency** | 1.6 | – | 2.4 | 0.66 mW/Ghz | 0.9 mW/GHz | – | – |
| **Area** | – | 0.076 $mm^2$ | 0.0168 $mm^2$ | 0.0048 $mm^2$ | – | – | – |

[1]Power consumption reported for full UCIe module at 32Gbps.
[2]Power consumption reported for full UCIe module at 16Gbps.

# Chapter 3

# Design of 8GHz Delay-Locked Loop for UCIe in Intel 16 PDK

We propose a design for an 8 GHz delay locked loop for a UCIe-compliant interface developed in Intel 16 CMOS process as part of the Intel University Shuttle Program. This design was first developed for the 2024 Q3 tapeout on the Sirius Chip and was updated in the 2025 Q2 tapeout aiming to address some of the pitfalls of the previous design.

## 3.1 DLL Specifications

The Universal Chiplet Interconnect Express (UCIe) specification outlines transmitter requirements across a wide range of data rates, from 2 Gbps up to 32 Gbps. In this work, we focus on the 16 Gbps mode, which corresponds to an 8 GHz operating frequency, targeting a bit error rate (BER) of 1e-15. The system utilizes 16 parallel lanes to achieve the required aggregate bandwidth. Key timing specifications at this data rate include a phase adjustment resolution of 16 milli-unit intervals (mUI), a total jitter budget of 96 mUI peak-to-peak, and a deterministic jitter budget of 48 mUI peak-to-peak as summarized in Table 3.1 [20].

Since the UCIe specification only defines the performance requirements for the overall transmitter, additional work was necessary to map these system-level requirements down to specifications for the clock lane and DLL. In particular, the phase adjustment resolution played a critical role in determining the number of output phases required from the clock generation circuitry. Based on the specification, a phase adjustment resolution of 16 mUI, relative to a clock unit interval (UI) of 62.5 ps, was required. To meet this requirement, a hierarchical interpolation scheme was used: the DLL generates 8 coarse phases, while the phase interpolator further subdivides adjacent phases into 16 intermediate phases, achieving the necessary resolution. Together, the DLL and PI enable both coarse and fine-grained phase adjustments as required by the transmitter.

The specified data rates impose corresponding clock frequencies, which directly determine the required operating frequency and tuning range of the DLL. Supporting an 8 GHz clock

Table 3.1: Summary of UCIe transmitter specifications.

| **Specification** | **Value** |
|---|---|
| Target Data Rate | 16 Gbps |
| Operating Frequency | 8 GHz |
| Clock UI | 62.5 ps |
| Target BER | 1E-15 |
| PHY Power Consumption | 1pJ/bit |
| Number of Lanes (N) | 16 |
| Phase Adjustment Resolution | 16 mUI (1ps) |
| Total Jitter (Peak-to-Peak) | 96 mUI |
| Deterministic Jitter (Peak-to-Peak) | 48 mUI |

frequency places stringent demands on both the speed and range of the delay elements.

The UCIe specification targets a power efficiency of 1 pJ/bit, which encompasses the complete adapter and physical layer circuitry including the transmitter, receiver, phase-locked loop (PLL), clock distribution network, and all other related components involved in the PHY [20]. No explicit power consumption target was specified for the DLL, other than that it should be comparable to modern designs and avoid excessive power usage. Regarding area, this was largely constrained by the integration team which allocated a total of 50$\mu$m x 70$\mu$m for the DLL. Finally, a preliminary timing budget was created to roughly determine the jitter specifications per block. A summary of the derived target specifications for the DLL and clock lane design is provided in Table 3.2.

Table 3.2: Summary of DLL/clock lane target specifications.

| **Specification** | **Target** |
|---|---|
| Process | Intel 16 |
| Frequency | 8GHz |
| DLL Number of Phases | 8 (45 degrees apart) |
| Deterministic Jitter (Clock Lane) | $<$ 1.3 ps |
| RMS Jitter (Clock Lane) | $<$ 86 fs |
| Power | Minimize |
| Area | $<$ 50$\mu$m x 70$\mu$m |

## 3.2 Mixed-Signal Circuit Design Flow

Before diving into the actual design of the delay locked loop, it is important to understand the design flow and tools required to design a mixed signal circuit from scratch to being "tapeout-ready". For high-precision delay lines, the sensing and delay elements are largely designed with analog techniques, as cell-based designs synthesized from RTL cannot achieve the precision required. However, as described in previous sections, there is growing popularity in relying on digital logic for more complex and intricate control schemes. Modern delay locked loop designs likely require both analog and digital design tool chains, which I outline in the following sections.

### Analog/Mixed Signal Circuit Generation (BAG)

In traditional analog design workflows, Cadence Virtuoso's Layout XL remains the standard tool for developing and laying out circuits. However, layout continues to be a bottleneck in the analog design process, often consuming substantial time for any sort of modification and update to the schematic. This challenge is further amplified in high-speed circuit design, where layout parasitics significantly impact performance and thus require constant iteration.

Berkeley Analog Generator, also known as BAG, was developed to address these pain points through Python-based schematic and layout generators. These generators, when properly constructed, allow designers to easily parameterize their circuits and quickly iterate on new configurations [7], without having to re-layout the circuit each time. BAG's internal APIs help ensure that generated layouts are DRC and LVS clean, reducing the additional overhead of having to manually clean your error-prone design. It also supports automatic inclusion of tap cells and allows for hierarchical layout generation, making it scalable for more complex analog/mixed-signal blocks that contain multiple layers of hierarchy. BAG helps streamline the design process while preserving the precision and flexibility essential to analog design. BAG was utilized for the design of all individual analog components, only relying on Layout XL for the module integration.

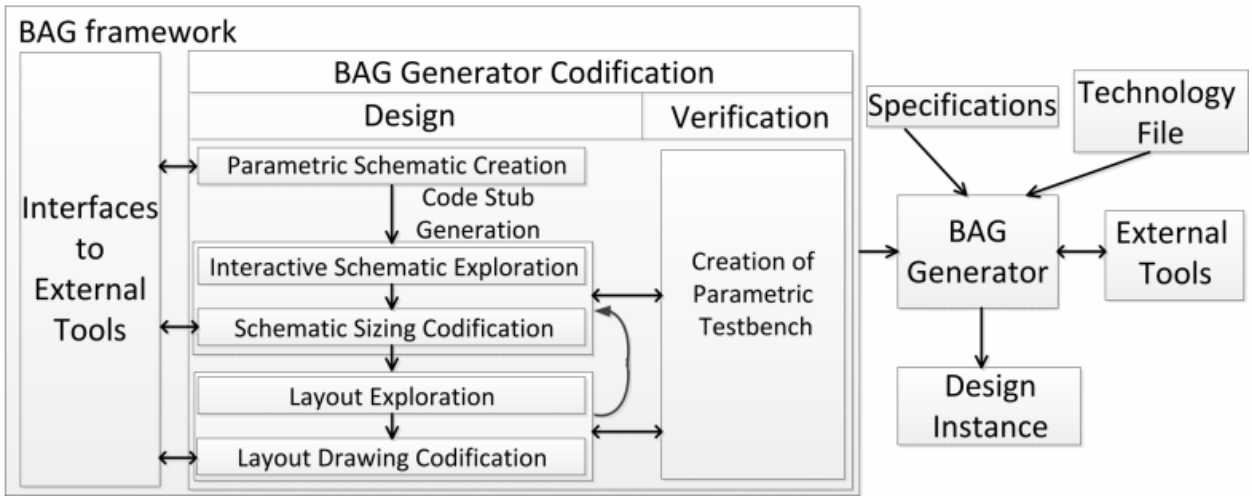



Figure 3.1: Diagram of BAG design flow [7].

## Hammer

Hammer is the primary digital VLSI design tool used at UC Berkeley, which provides a modular and reproducible framework for generating GDS layouts from RTL. It enables users to write and verify RTL, perform synthesis, and execute place-and-route on a digital design abstracting away a lot of the direct interfacing with the verbose VLSI tools. Hammer leverages plugin-based backends to interface with industry-standard EDA tools such as Cadence Genus and Innovus as well as DRC/LVS tools like Calibre and ICV. Hammer automatically generates the necessary TCL scripts and configuration files required for each stage of the design starting from synthesis to place and route and then signoff flows [11].

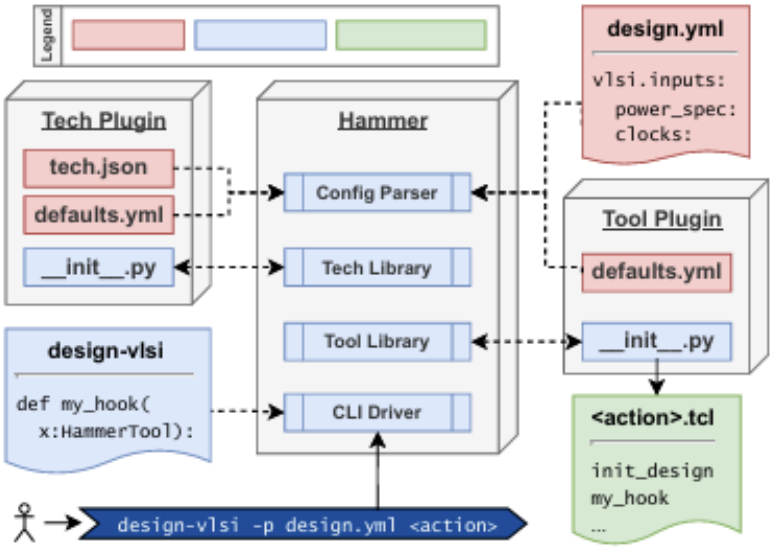



Figure 3.2: A diagram of Hammer design flow from [11].

There are multiple ways to use Hammer. For large SoC designs, Hammer is integrated as part of the broader Chipyard framework capable to producing full SoC designs. In this project, Hammer was used outside of Chipyard in a standalone workspace to design and iterate on small digital components that reside within larger analog blocks. In this setup, designers pass in Verilog source files alongside YAML-based configuration files that specify synthesis and physical design constraints, including timing, IO placement, and area bounds. This abstraction simplifies the integration of constraints to ensure consistency between synthesis targets and floorplan requirements. In this work, Hammer was used to implement the digital integrator logic, ensuring streamlined integration with the rest of the analog components of the delay locked loop [9].

## Cadence Virtuoso

While individual sub-blocks for this project were designed using BAG and Hammer, the overall integration was done within Layout XL in Cadence Virtuoso. Completed analog sub-modules would get imported automatically via BAG into Virtuoso where layout views could be hierarchically instantiated directly into the cellview. The Hammer workflow would generate the necessary collateral for any digital blocks (gds, schematic netlist) from which could be imported into Virtuoso and similarly instantiated within the design. Routing at the top level module was done manually in Virtuoso, with Calibre plugins allowing DRC and LVS to be run directly from Virtuoso as well. Most of the block level verification was also done within Virtuoso.

### Simulator Options

A summary of the different simulator options for mixed signal design is shown in Table 3.3.

Table 3.3: Summary of circuit simulators.

| Simulator | Developer | Use Case | Strengths | Weaknesses |
|---|---|---|---|---|
| Spectre | Cadence | Analog, Mixed-Signal | High precision, Virtuoso Integration | Slow for digital, large mixed-signal designs |
| Xcelium (co-simulation) | Cadence | Digital/AMS Co-simulation | Best for large mixed-signal designs | Complex setup, can be slower for analog heavy simulations |
| VCS | Synopsys | Digital Logic | Very fast, pure digital simulation | No AMS support |

This project primarily relied on Spectre, which is the main simulator for any analog circuit design. Since the designed circuits are not overly large and complex, Spectre is capable of simulating the digital blocks without too much additional overhead. While Xcelium analog/digital co-simulation would also be an option, the digital blocks were small enough such that there was minimal speedup from using co-simulation. Furthermore, setting up Xcelium on the BWRC servers is also a non-trivial task and constant version bumps made it more difficult to manage in comparison with Spectre which was much easier to maintain within Virtuoso. VCS was used in this project to simulate and verify standalone digital modules designed in Hammer workspaces. This RTL simulation could be performed quickly with waveform debugging done through Verdi or DVE.

## 3.3 Delay-Locked Loop Architecture

The overall proposed architecture for the multiphase digitally controlled delay locked loop is demonstrated in Figure 3.3.

The DLL receives an 8GHz differential signal from the phase locked loop through an input buffer. The reference clock then passes through a delay line consisting of 6 pseudo-differential delay components that are tunable via switched shunt capacitor arrays. The phase detector compares the phase error between the positive reference clock and a delayed version of the clock (`out<7>`) which sends an `up` signal to the digital integrator. This digital integrator, clocked at 2GHz, will sample the output of the phase detector and update the DLL digital code accordingly to either increase or decrease the overall the delay of the delay line. This update will continue until a stable locking point is reached in which the 360 degree phase shifted clock edge is aligned to the reference clock.

At this point, we expect that the sum of four core delay cells, not including the dummy input and output, to be locked and therefore, tapping the differential outputs of the delay

line would yield 8 equally spaced clock phases each separated by 45 degrees. In closed loop operation, these phases should track over PVT variations.

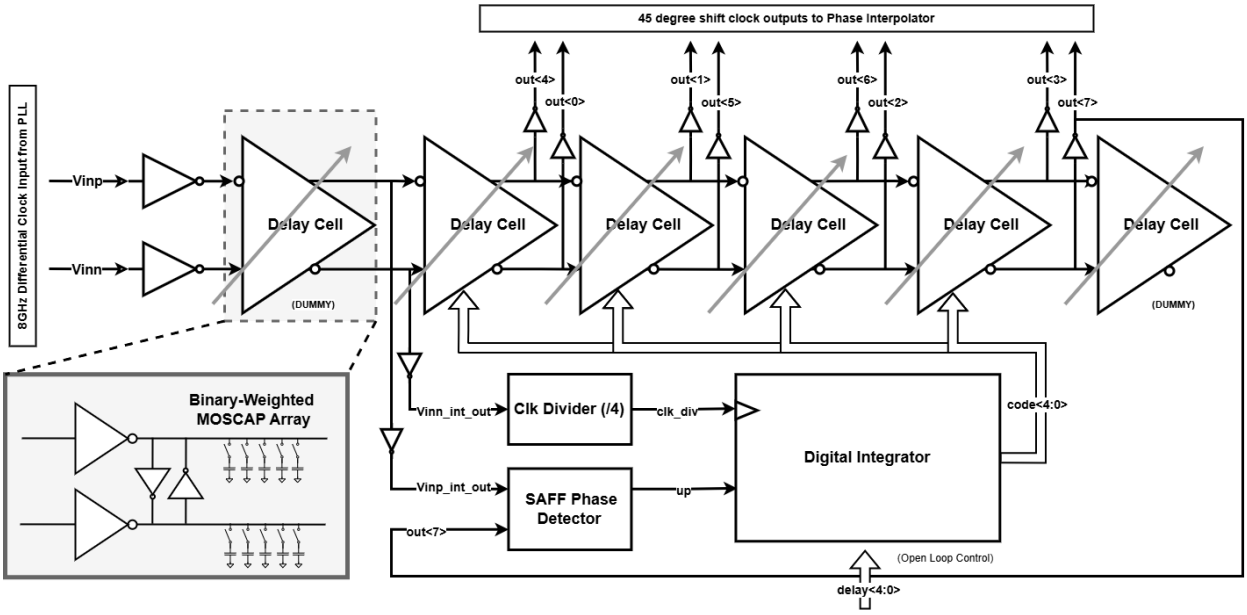



Figure 3.3: A block diagram of the proposed DLL architecture.

In the following sections, we present detailed descriptions of the different sub-blocks of the DLL in addition to the considerations and challenges during the design process.

## 3.4 Digitally Controlled Delay Line

### Description

The digitally controlled delay line (DCDL) consists of 6 identical pseudo-differential delay cell whose delay is tuned via a 5-bit binary-weighted shunt capacitor array as shown in Figure 3.4 with schematic and layout views of the DCDL in Section 3.4.

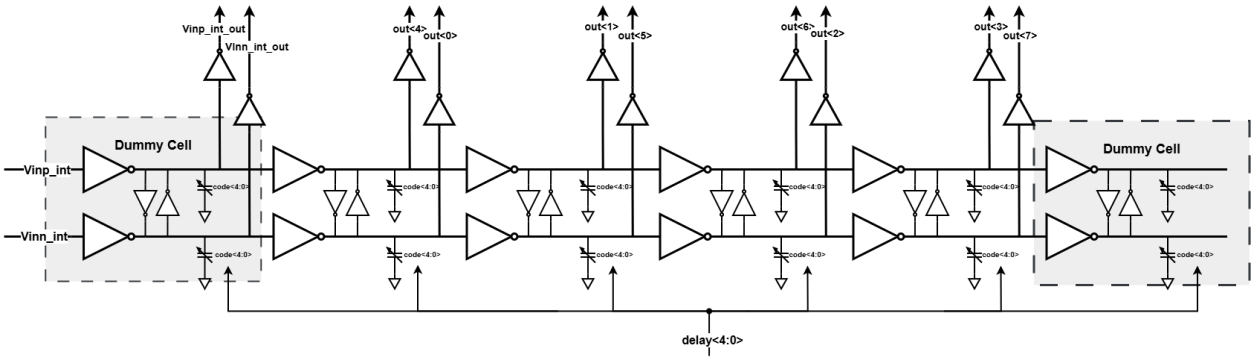



Figure 3.4: A diagram of the proposed DCDL.

## Design Process

The first step in the DCDL design process was selecting the topology and tuning mechanism. The tight phase precision requirement necessitated a high-resolution delay line, effectively ruling out gate-based designs and narrowing the options to current-starved and shunt-capacitor-based implementations. The shunt-capacitor approach was chosen for its more linear delay response and simplicity.

Initial design exploration was performed using ideal `analogLib` components to estimate the required capacitance and inverter drive strength for achieving a linear RC delay as a function of the control code. The objective was to ensure monotonic, predictable delay behavior covering the required tuning range. Since only 8GHz operation was targeted, the tuning range was only required to cover PVT variations around the center frequency.

Ideal components were then replaced with custom MOS capacitors and ideal switches with CMOS switches implemented with PDK devices. The capacitor bank design followed the methodology outlined in [2], characterizing effective capacitance of the MOSCaps against their width, fingers, and multiplier. Cross-coupled inverters were used to maintain duty cycle in differential signaling, and output buffers isolated the core delay line from the load presented by the phase interpolator. After completing the initial schematic, transistor sizing was iteratively optimized until a monotonic delay response was achieved.

## DCDL Design Challenges

**Meeting Tuning Range for 6GHz and 8GHz Operations** Initially the design was intended to target both 12GT/s and 16GT/s operation (6 GHz, 8 GHz respectively). The target was to achieve a phase resolution finer than 1 ps, which requires that the cumulative delay of four consecutive delay elements sum to approximately 62.5 ps for 8 GHz operation and 83.33 ps for 6 GHz operation.

Achieving these delay targets ensures that the delay cells align properly with the clock period for both frequencies. While initial design efforts aimed to support both frequencies,

time constraints led to a decision to prioritize performance for the 8 GHz mode. To extend the effective tuning range and maintain fine resolution across different process, voltage, and temperature (PVT) corners, mixed-mode control strategies which may require incorporating multiple tuning mechanisms such as current starving should be considered.

**8GHz Tuning Range Across Corners** Maintaining reliable operation across corners also proved challenging, as significant variations in passive device values were observed in the provided model files. In particular, the process variations on passive components such as capacitors and resistors are modeled as uniform multipliers applied across all devices, resulting in large shifts in the effective tuning range of the digitally controlled delay line (DCDL), saturating the delay range at extreme corners. Careful design of the tuning range of the shunt capacitor array and transmission gate sizing was essential to mitigate these variations and maintain sufficient tuning range and performance robustness across all conditions.

**Maintaining DCDL Linearity and Monotonicity** Another major design consideration involved maintaining linearity and monotonicity of the DCDL while achieving the required tuning resolution. Increasing the number of tuning bits would naturally extend the achievable range; however, each added bit doubles the area requirement of the shunt capacitor array and increases parasitic loading from the associated switches. Larger transmission gate switches reduce the on-resistance, which is beneficial for minimizing delay distortion, but they also introduce significant parasitic capacitance, which degrades delay cell performance. With capacitance resolution already in the the femtofarads, such parasitics had non-trivial impacts.

Special attention was given to the impact of CMOS switch on-resistance ($R_{\text{on}}$) and parasitic capacitance, where $C_{\text{par,switch}} \propto W_{\text{switch}}$, illustrating the tradeoff between switch sizing and parasitic loading. As capacitance increases, the influence of $R_{\text{on}}$ becomes more significant, modifying the effective capacitance seen at the output and therefore resulting in non-monotonic delay steps.

An example of a non-monotonic DCDL delay function from an earlier iteration of the design is demonstrated in Figure 3.5. Visually, you can observe glitches around code 8, 16, 24, with the MSB code switch at code 16 resulting in a non-monotonic corresponding to DNL greater than 1 LSB. Such non-monotonicity has the potential to prevent the DLL from locking.

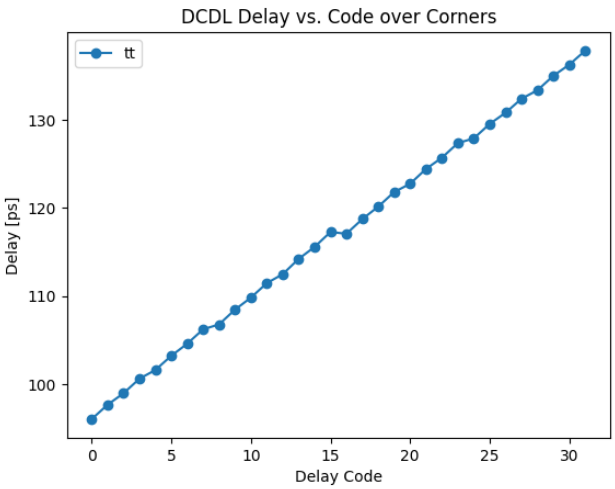


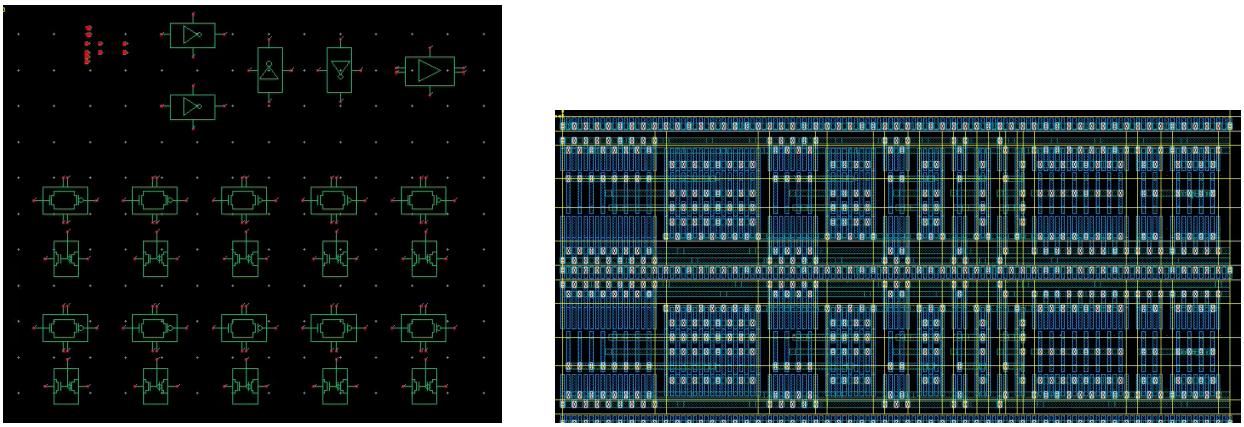

Figure 3.5: Example of non-monotonic DCDL delay.

Originally, the design targeted a 6-bit resolution for shunt capacitor tuning, but due to challenges in designing a large enough MSB capacitor while meeting the minimum delay, the final implementation reduced the number of tuning bits to 5. This decision represented a trade-off between achieving sufficient tuning range, minimizing loading effects, and preserving the linear, monotonic behavior necessary for predictable DCDL operation and DLL locking.

## DCDL Generators[1]

The schematic and layout of the digitally controlled delay line were designed using BAG and composed of hierarchical instantiations of inverters, transmission gates, and MOScaps. The use of a generator-based approach significantly accelerated design iterations by enabling easy regeneration of the layout without manual rework. Parameters such as the device sizing of the main driving inverters, cross-coupled inverters, transmission gate switches, and MOS capacitors could be adjusted programmatically, saving significant design effort. An example of a single delay cell layout is shown in 3.6. To generate the full delay line, multiple instances of the delay cell were stacked vertically with necessary row taps placed between instances.

[1]The schematic and layout generators were based on prior work by Di Wang and modified for the 2025 Q2 Tapeout.



(a) Schematic of delay cell.

(b) Layout of delay cell.

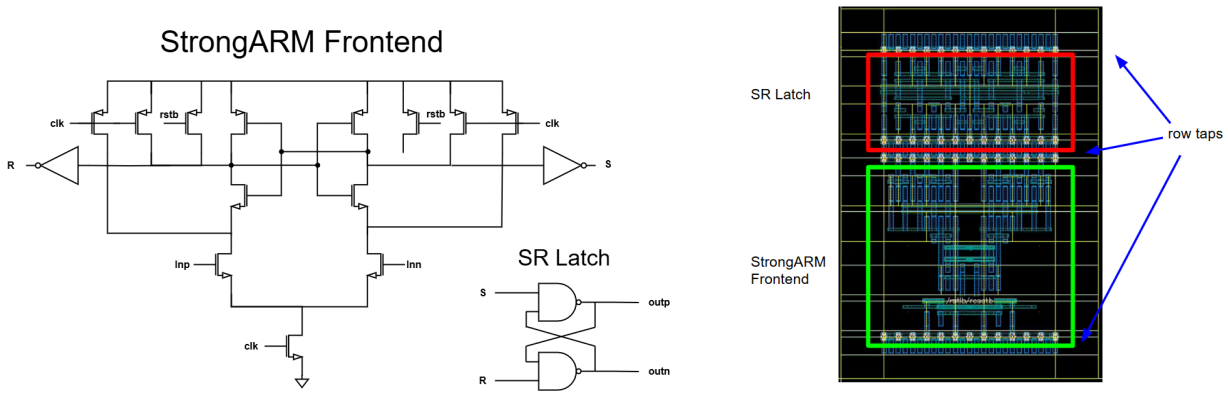
Figure 3.6: Schematic and layout of one pseudo-differential delay cell created from BAG generator.

## 3.5 Phase Detector

A sense amplifer flip-flop was designed as the bang-bang phase detector. The previous iteration of the design used a direct instantiation of a D flip-flop from the provided Intel PDK. As explained in Section 2.4, the phase detector accuracy is largely dependent on its deadzone which is determined by the setup and hold time requirements of the flip flop. As the standard cell is not optimized to be used as a comparator, the DLL locked to a point slightly offset from the ideal phase. This introduced phase mismatch which degraded the overall linearity of the phase shift for the clock lane.

To achieve a lower deadzone and avoid this issue, a sense-amplifier based flip-flop (SAFF) was used. This sense-amplifier based flip flop is a modified version of the BAG3 StrongArm Flop from the `bag3_digital` library. Using the analog generators allowed for rapid instantiation a DRC and LVS clean implementation of a sense-amplifier flip-flop, significantly accelerating the development process. Iterating on the transistor sizings and additional I/Os took minutes as opposed to having to manually update the layout each time which could take hours.

The design of the sense-amplifier flip-flop is relatively straightforward and consists of 2 cascaded components: (1) StrongARM Frontend (2) SR Latch. Additional reset circuitry is required to ensure that the SR latch starts up in a valid and known state as to avoid an invalid states that might lead to scenarios where the latch is stuck at a given value.




(a) Schematic (b) Layout

Figure 3.7: Design of sense amplifier based flip-flop used as phase detector.

## 3.6 Digital Integrator

The digital integrator is responsible for driving the digital code to the delay line and was written in Verilog, synthesized/place-and-routed using Hammer, and verified using VCS. The digital integrator in the DLL acts primarily as a counter, incrementing or decrementing the delay code based on the direction of the phase error as indicated by the phase detector. It also supports open-loop operation, allowing the feedback loop to be disabled and the DLL delay code to be set directly through MMIO registers. This enables deterministic control for testing purposes. To ensure proper reset behavior and initialize the integrator to a known state at startup, a reset synchronization scheme is employed. Specifically, the reset signal passes through a pair of cascaded flip-flops, ensuring that its de-assertion of the reset signal aligns with the rising edge of the clock, thereby minimizing the probability of entering a metastable state.

Due to the inability to close timing at 8GHz (reference clock frequency), the digital integrator operates four times slower than the reference clock at 2GHz, motivating the need for an additional clock divider. The pin placements for the integrator are specified in Listing 3.1. The locations are specified such all control signals can be easily routed to the boundary of the DLL and that the `acc[4:0]` and `accb[4:0]` outputs are aligned with the DCDL input pins. Furthermore, Listing 3.2 highlights the SDC constraints used to define pin capacitances and additional timing constraints, ensuring that timing is met and edge transitions are sharp enough. The full integrator layout is demonstrated in Figure 3.8.

```
# Pin placement constraints
vlsi.inputs.pin_mode: generated
vlsi.inputs.pin.generate_mode: semi_auto
vlsi.inputs.pin.assignments: [
  {pins: "acc[4]", layers: [""], side: "top", location:[_, _]},
  {pins: "acc[3]", layers: [""], side: "top", location:[_, _]},
  {pins: "acc[2]", layers: [""], side: "top", location:[_, _]},
  {pins: "acc[1]", layers: [""], side: "top", location:[_, _]},
  {pins: "acc[0]", layers: [""], side: "top", location:[_, _]},
  {pins: "accb[4]", layers: [""], side: "top", location:[_, _]},
  {pins: "accb[3]", layers: [""], side: "top", location:[_, _]},
  {pins: "accb[2]", layers: [""], side: "top", location:[_, _]},
  {pins: "accb[1]", layers: [""], side: "top", location:[_, _]},
  {pins: "accb[0]", layers: [""], side: "top", location:[_, _]},
  {pins: "clk", layers: [""], side: "right"},
  {pins: "up", layers: [""], side: "top", location:[_, _]},
  {pins: "EN", layers: [""], side: "bottom"},
  {pins: "OCL", layers: [""], side: "bottom"},
  {pins: "reset", layers: [""], side: "bottom"},
  {pins: "delay*", layers: [""], side: "bottom"},
  {pins: "DLL_code*", layers: [""], side: "bottom"},
]
```

Listing 3.1: YAML configuration for integrator pin placements (layer names and exact coordinates not shown).

```
vlsi.inputs.custom_sdc_constraints: [
  "set_units -time 1.0ps",
  "set_units -capacitance 1.0fF",
  "set_driving_cell -lib_cell <standard cell name> [all_inputs]",
  "set_max_delay 100 -from {clk} -to [get_ports acc*]",
  "set_false_path -from [get_ports reset]",
  "set_load 6 [get_ports acc*]",
  "set_load 6 [get_ports acc*]",
  "set_load 5 [get_port clk]"
]
```

Listing 3.2: YAML configuration for integrator SDC constraints (standard cell name not shown).

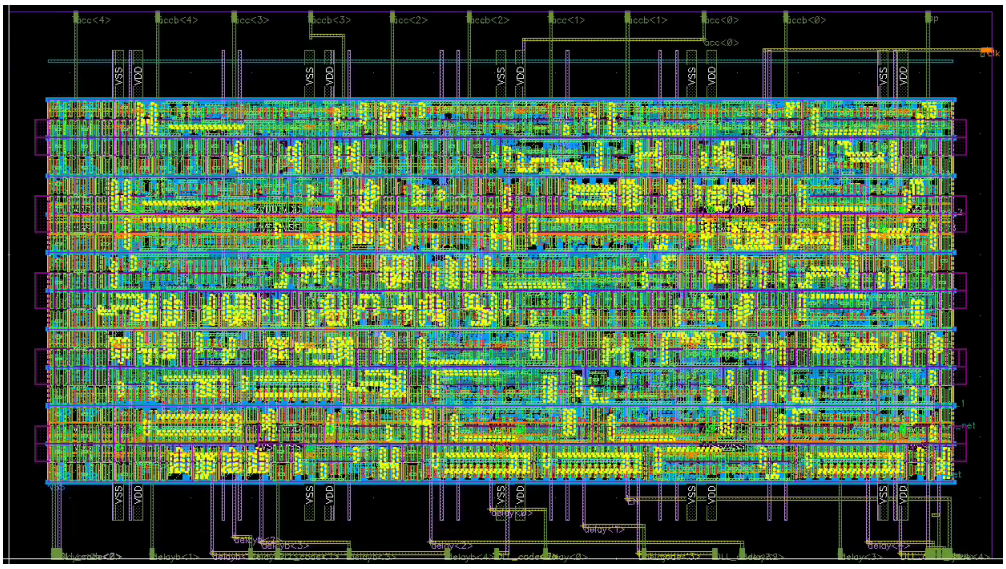


Figure 3.8: Image of integrator layout (9$\mu$m by 16.2$\mu$m).

## DLL Debug Signal

To enable observability and debuggability of the delay locked loop, additional digital signals were exposed from the digital integrator and routed to memory-mapped I/O registers. These signals correspond to the current digital code asserted by the DLL during operation, providing insight into the DLL's internal state. However, because the digital integrator operates in a different clock domain (2GHz) than the core digital system ($\sim$ 500MHz), a standard clock domain crossing, such as handshake protocols or asynchronous FIFOs, would typically be required to avoid metastability issues on the outputs from the DLL. Attempting to read the DLL output code directly through an MMIO register without synchronization risks metastable or glitchy outputs.

A simplified observability method was utilized in place of the standard alternate clock domain crossing techniques because full precision and low-latency updates were not strictly necessary for the intended debug functionality. The goal was to monitor the general behavior of the DLL, particularly which codes the DLL centered around when locked. As such, even with the possibility of occasional metastable reads, the system could still effectively capture meaningful insights into the general DLL behavior. During closed-loop operation, the DLL output code is expected to fluctuate among a small range of values around the locking point. In contrast, during open-loop operation, the output code should remain constant and match the asserted `delay[4:0]` input value provided to the DLL.

Additionally, a prime number of clock cycles was selected for the readout update period. This ensured that the sampled DLL code would eventually capture any transitions even if the DLL toggles between two or more adjacent codes. Without this consideration,

synchronizing the update to MMIO register with the internal cycling of the delay code could otherwise incorrectly suggest that the DLL is locked to a single static code. Although there remains some risk of metastability due to the lack of synchronization between the core clock and the divided analog clock, the slowed update rate of the debug output greatly reduces the probability of consecutively reading metastable results. This, simplified MMIO-based observability mechanism was deemed sufficient for both closed-loop monitoring and open-loop verification during chip bringup and measurement.

An RTL simulation is shown in Figure 3.9 which demonstrates the behavior of `DLL_code[4:0]` when the DLL is switching between 2, 3, and 4 codes. The waveform highlights that the `DLL_code[4:0]` will not mistakenly produce a static code if the underlying behavior of the DLL code is still cycling between multiple codes.

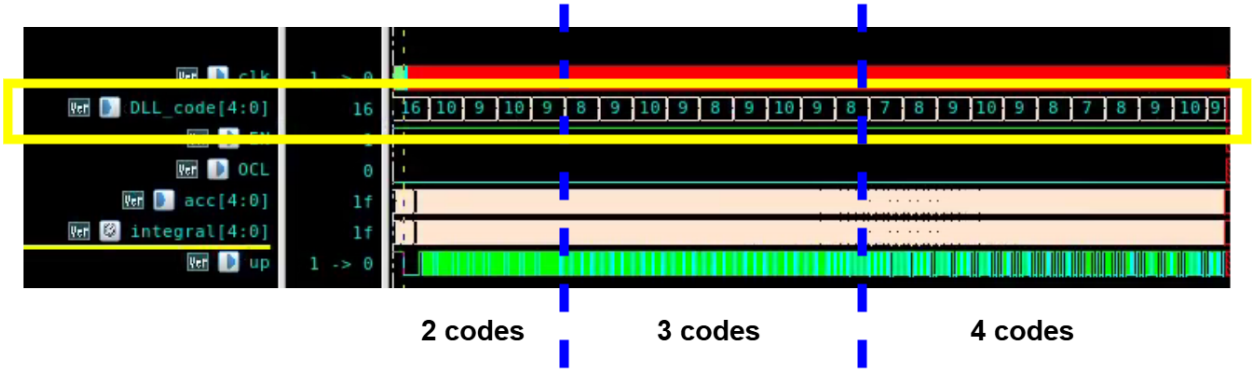
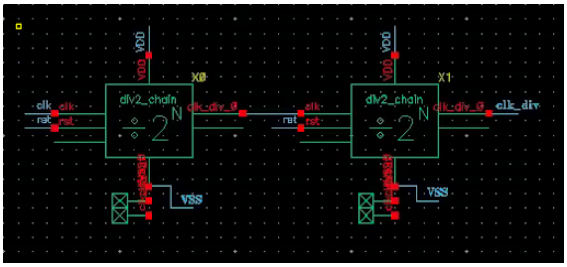



Figure 3.9: Waveform view of `DLL_code[4:0]` debug signal.

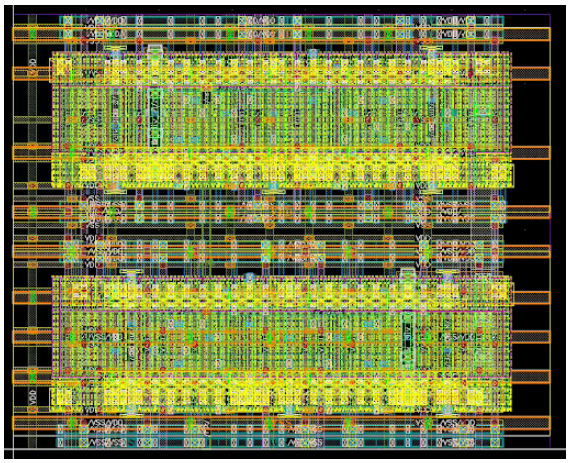
## 3.7 Additional Sub-circuits

### Clock Divider

As described in Section 3.6, a clock divider is required in order to clock the digital control loop at a lower frequency so that digital timing margins are met. This clock divider also leverages the utility of BAG to quickly generate functional and clean schematics and layouts. In particular the `bag3_digital` clock divider is used to generate two instances of divide by 2 clock dividers. These dividers are cascaded together to form a divide by 4 clock divider whose output clocks the digital integrator. The layout view of the clock divider is shown in Figure 3.10.




(a) Clock divider schematic.

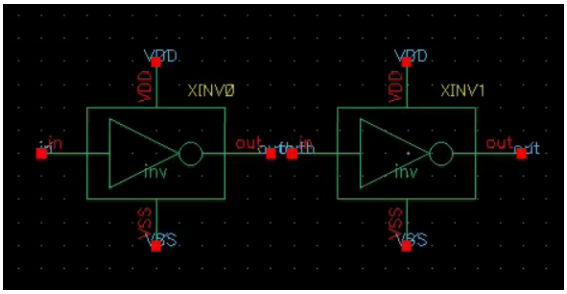


(b) Clock divider layout.

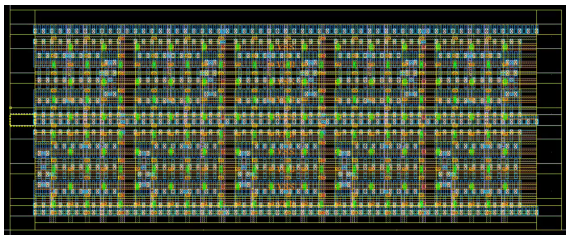
Figure 3.10: Clock divider design.

## Buffers

Additional buffer chains were required to drive the DCDL control inputs, particularly in configurations with larger capacitive loads resulting from increased switch sizes. Without these additional buffers, the integrator would not have sufficient drive strength to properly switch the delay line, compromising the DLL's locking behavior. These buffers were implemented using BAG and instantiated directly within the top-level layout. They consist of 5 parallel instantiations (one for each bit of the delay code) of 2-stage inverter chains with fanout of 4 to minimize propagation delay while efficiently increasing the signal drive strength.




(a) Buffer schematic.



(b) Buffer bank layout

Figure 3.11: Buffer bank design.

### Dummy Loads

Additional dummy instances are typically required to balance the load on matched signals, which are the DLL output phases (`out<7:0>`). Without these matched dummy loads, the RC delay between the DLL outputs and the PI inputs may vary, introducing phase mismatches across the output phases. The inclusion of these dummy phase detectors can be observed in the layout view of the complete DLL shown in Figure 3.15.

## 3.8 Design Optimizations

Some of the main design optimizations and improvements made in this iteration of the DLL were:

- Improved locking condition
- Load matching
- Delay code cycling due to loop latency/instability

### Improved Locking Condition

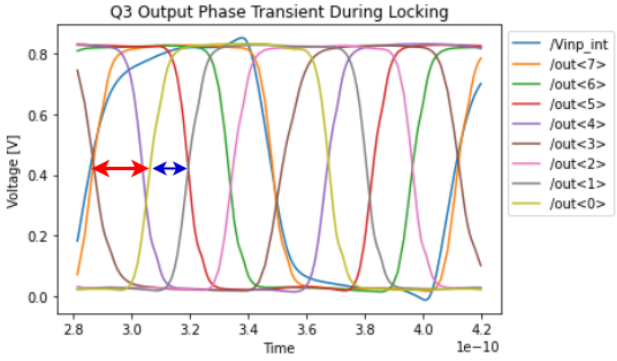
As described in Section 2.4 the performance of the DLL in closed loop is only as good as its ability to properly detect phase error. This includes the precision of the phase detector, in particular its deadzone, as well as the accuracy of the compared signals themselves. If the two inputs to the phase detector are already noisy or skewed versions of the original signals they are trying to compare, then the correction will be wrong regardless of how accurate the phase detector is.

1. Minimize phase detector deadzone
2. Match the delays and loading of reference and output clocks

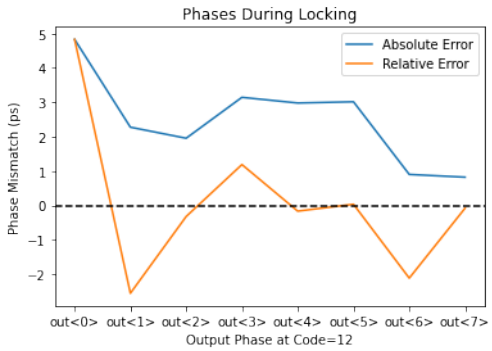
In the previous design, the use of a simple D flip-flop (DFF) as the phase detector introduced a relatively wide dead zone, which resulted in poor locking precision and consequently larger phase errors. Additionally, discrepancies in the signals being compared exacerbated the issue: one signal (`Vinp_int`) was taken before the output buffer, while the other was taken after the output buffer (`out<3>`). This mismatch not only caused differences in edge sharpness which degrades the phase comparison accuracy, but also introduced an extra fixed delay through the output buffer. As a result, the DLL would inadvertently lock to a delay longer than the intended value, further increasing the overall phase error, which can be directly seen in the transient and linearity plots in Figure 3.12.

This static phase offset has downstream effects on the phase shift linearity of the tapped DLL outputs. When the locking condition is not accurately established, the delay line phases

are no longer equally spaced, degrading the linearity that is critical for the clock lane phase adjustment.



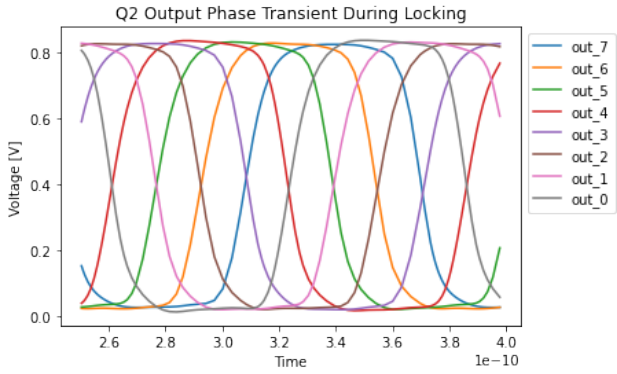

(a) Transient During Lock



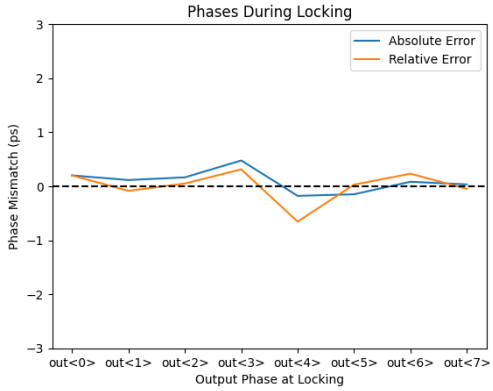

(b) Phase Linearity

Figure 3.12: Impact of imprecise locking condition on output phase linearity.

Therefore, ensuring a more accurate locking condition was identified as a crucial first step toward reducing phase error at the DLL outputs. By minimizing mismatch between compared signals and improving the sensitivity of the phase detector, the design achieved more precise control over the delay line, ultimately enhancing the uniformity and predictability of the generated clock phases (Figure 3.13). After these optimizations, the phase mismatch improved considerably from 4.85ps (0.31 LSBs) to just 0.65ps (0.042 LSBs).




(a) Transient During Lock




(b) Phase Linearity

Figure 3.13: Post-optimization phase mismatch transient and linearity demonstrating equal phase spacing.

## Matched Load and Routing

To address phase mismatch in the DLL outputs, the most critical consideration is ensuring symmetry across all delay cells and their outputs. This symmetry is achieved by introducing input and output dummy structures, which help balance the drive strength and load capacitance seen by each cell.

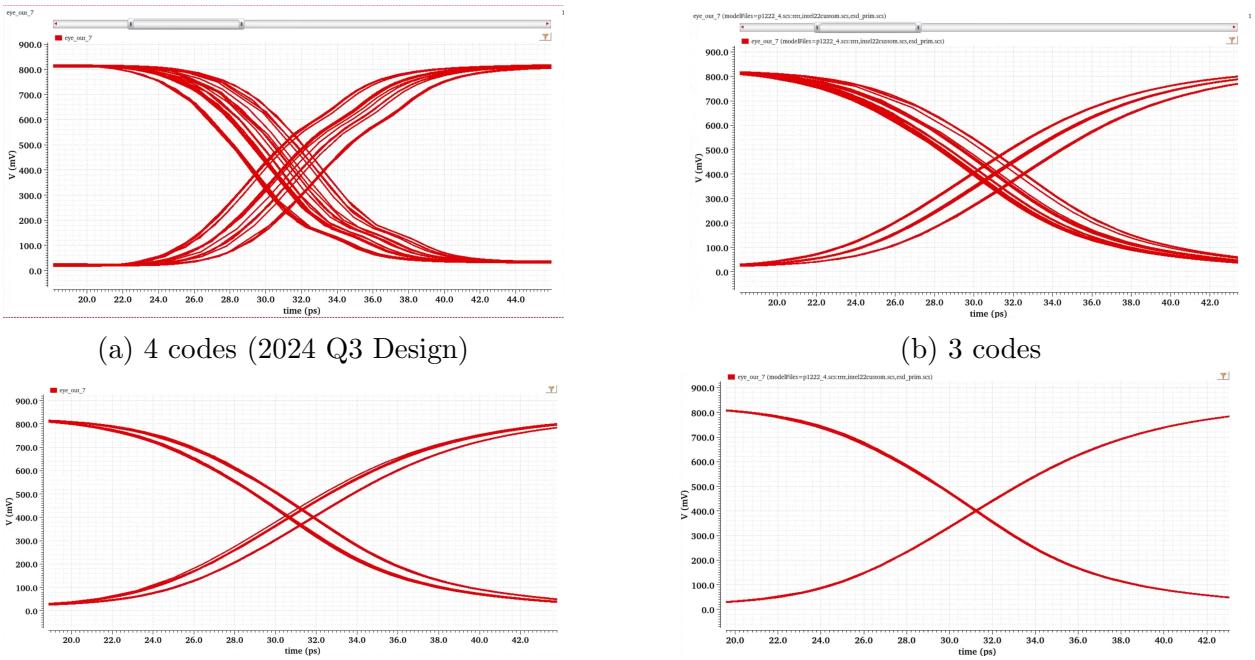
It is essential that not only the delayed outputs but also the reference clock signal used for comparison are load matched. The previous design matched the delayed outputs to each other but overlooked mismatches on the reference clock path, resulting in phase errors during the phase comparison. As described in [17], maintaining identical fanout and loading across all tapped outputs is crucial to ensuring equally spaced phases. Any asymmetries in the delay line manifest directly as deviations from the ideal phase spacing, degrading the performance of the overall system.

The final consideration is routing parasitics, which can significantly affect performance in high-speed designs. Even additional routing delays on the order of 1ps can shift the DLL's locking code by a detectable amount. To avoid this, it is generally good practice to route high-speed signals on higher metal layers, where trace resistance is lower and signal integrity is better preserved. Furthermore, as the DLL outputs should be matched as closely as possible, their routing distances should be relatively similar in length as to avoid additional parasitics on any given output, which introduces additional phase mismatch.

## Delay Code Cycling

Delay code cycling refers to when the digital delay code of the DLL oscillates between multiple values near the locking point. While toggling between two codes is typically unavoidable without additional lock detection circuitry, excessive cycling between multiple codes usually indicates problems in the feedback loop dynamics in terms of the loop stability as described in Section 2.4. In the previous design, the DLL exhibited cycling across four digital codes as shown in Figure 3.14a

As explained in Section 2.4, excessive code cycling in digital control loops can be attributed to insufficient "phase margin" due to excessive loop latency. Ideally, the delay line should be updated before the next phase comparison occurs to ensure that the control loop converges in the correct direction. However, if the delay update takes longer than the update period, subsequent phase comparisons may incorrectly interpret the phase relationship, causing the system to oscillate between multiple codes. In this design, the total loop latency—comprising the phase detector latency, integrator propagation delay, and the delay required for the updated signal to propagate through the delay line—must be less than the update period of 500 ps (corresponding to the 2GHz digital clock). Simulations revealed that, for certain delay code values, the delay update did not complete in time, leading to erroneous corrections and, consequently cycling between four codes.




(a) 4 codes (2024 Q3 Design)

(b) 3 codes

(c) 2 codes

(d) Ideal

Figure 3.14: Eye diagram threshold crossings for four cases of delay code cycling.

Table 3.4: Summary of measured deterministic jitter as a result of delay code cycling.

| No. of Codes | Measured DJ at Threshold |
|---|---|
| 4 | 3.8 ps |
| 3 | 2.6 ps |
| 2 | 1.16 ps |
| Ideal | $\sim$ 0 ps |

To mitigate this issue, an analysis of the loop latency was done to determine why the update would occur after the next decision was already made. After determining the sources of latency, the most promising solution was to constrain the integrator's propagation delay using the static timing constraints (SDC) in the digital flow. This helped reduce the overall loop latency and ensured that the delay line could be updated reliably within the allowable window, improving locking behavior and reducing delay code dithering to acceptable levels. In particular `"set_max_delay 100 -from {clk} -to [get_ports acc*]"` was added to the integrator design YAML file to specify that that the max delay between the rising edge of the `clk` to the `acc[4:0]` outputs would be 100ps. This would help ensure that the total latency between two consecutive phase decisions would sum to less than 500ps and thus always correct the delay in the right direction.

While the code cycling was reduced from four codes down to two codes, there still remains the deterministic jitter associated with the toggling between the two codes as seen in Figure 3.14c. If this amount of deterministic jitter is not tolerable during operation, then the DLL can be operated in open loop control, where a single DLL code is driven via MMIO registers.

## 3.9 DLL Integration

The full layout DLL integration, which combines all previously described blocks, was done within Virtuoso Layout XL and shown in Figure 3.15. This layout view includes all routing between sub-blocks in addition to connected power straps up to top metal layer. The integrated DLL is 32.4$\mu$m wide and 54.63 $\mu$m tall.

## 3.10 Clock Lane Integration

The DLL constitutes a critical component within the transmitter (TX) data tile, contributing to the timing control of the high-speed serial interface. It is integrated as part of the overall clock lane along with a phase interpolator and duty cycle corrector, which altogether is responsible for phase-aligning the clock signal with a resolution of up to 16mUI. This resolution corresponds to a phase shift of 2.88 degrees, enabling precise timing adjustments required for high-speed data sampling and serialization.

Figure 3.16 depicts the TX Data Tile and where the DLL and larger clock lane fit into the broader subsystem within UCIe. Figure 3.17 portrays a block level diagram of the clock lane module which integrates a single DLL, a PI, and two DCCs.

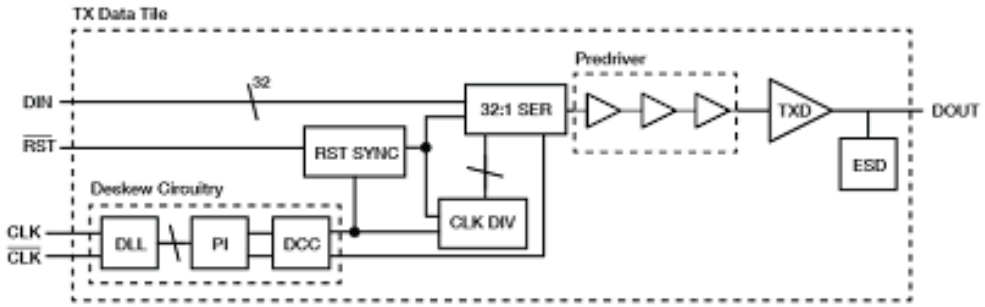



Figure 3.16: Diagram of TX data tile [20].

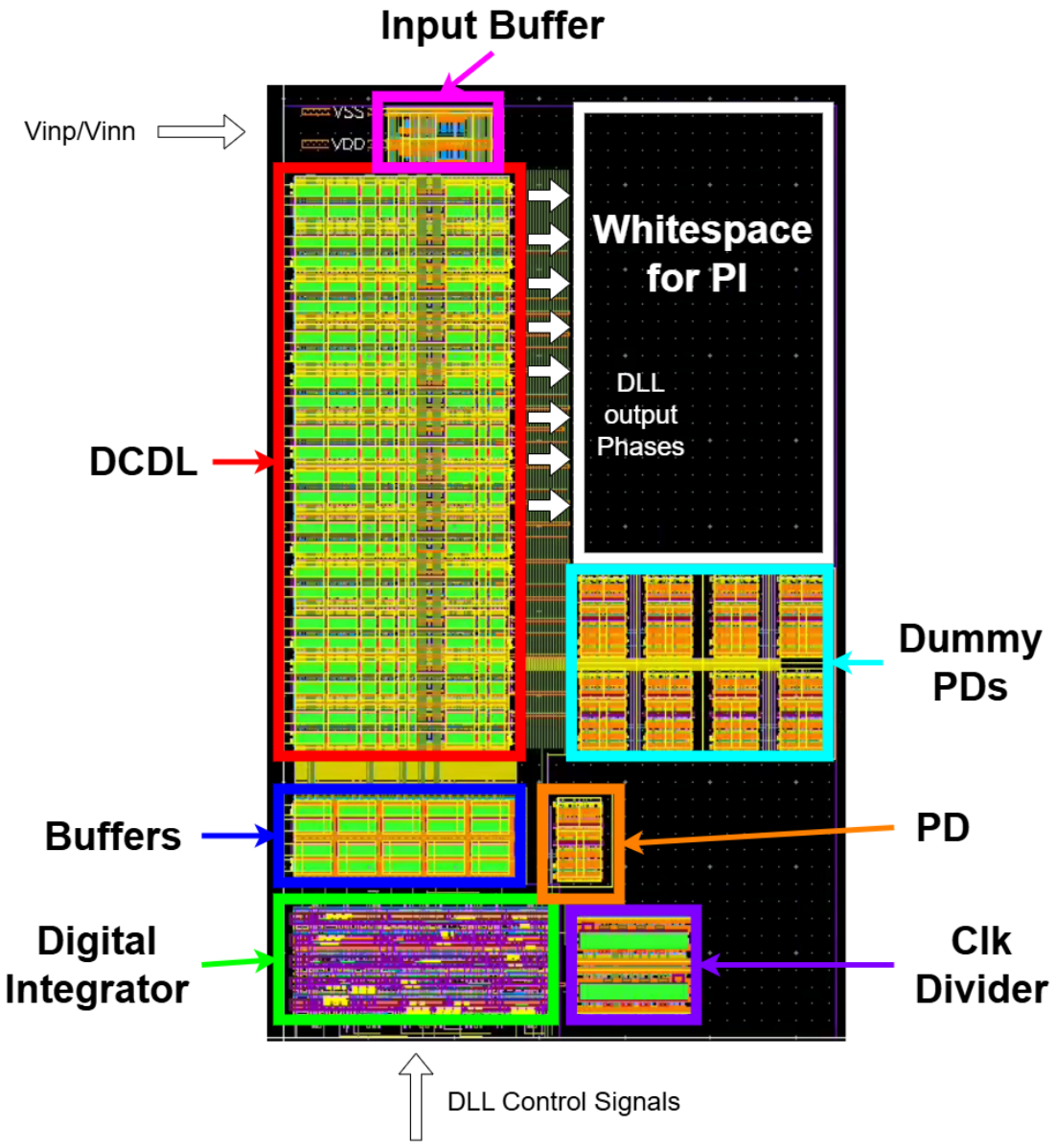



Figure 3.15: Labeled image of full DLL layout including all described sub-blocks.

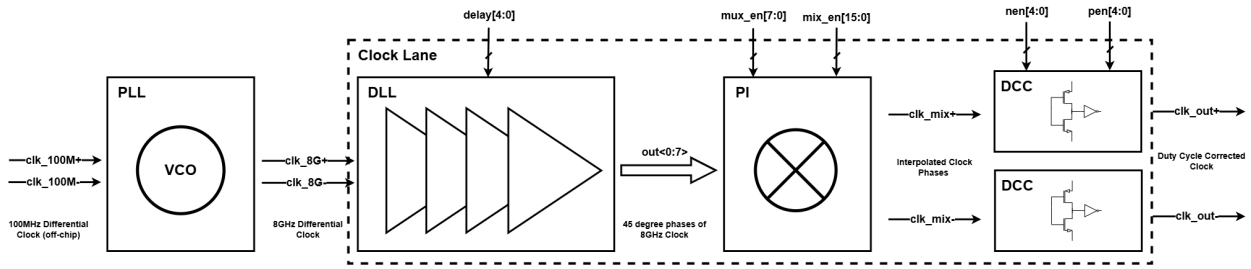



Figure 3.17: Diagram of clock lane on TX data tile (labeled as "Deskew Circuitry" in Figure 3.16).

## Phase Interpolator[2]

Phase interpolators generate intermediate clock phases by linearly combining two adjacent input signals, enabling fine-grained phase resolution beyond the spacing provided by the delay line. In the proposed work, a 5-bit PI is capable of generating 16 interpolated phases from any two adjacent 45-degree phases produced by the DLL. This cascaded DLL-PI system achieves a phase resolution of 2.8125 degrees or 15.625mUI, allowing highly accurate control of the sampling instant of the data. The phase interpolator plays a vital role in maximizing the data eye opening, which is crucial for minimizing bit error rates in high-speed links.

[2]The phase interpolator was designed by Bob Zhou and used from the 2024 Q3 Sirius chip tapeout.

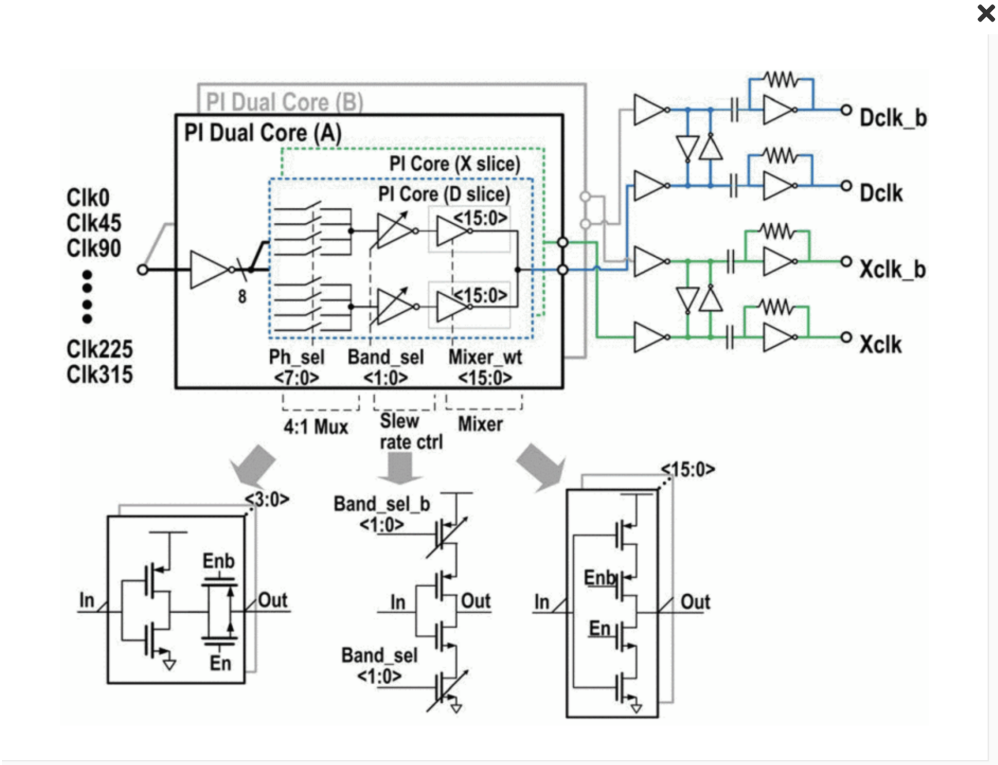



Figure 3.18: Phase interpolator schematic.

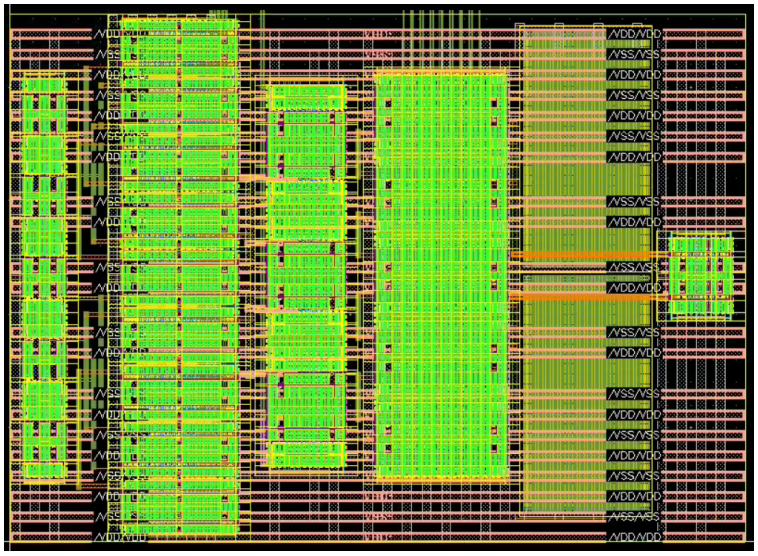
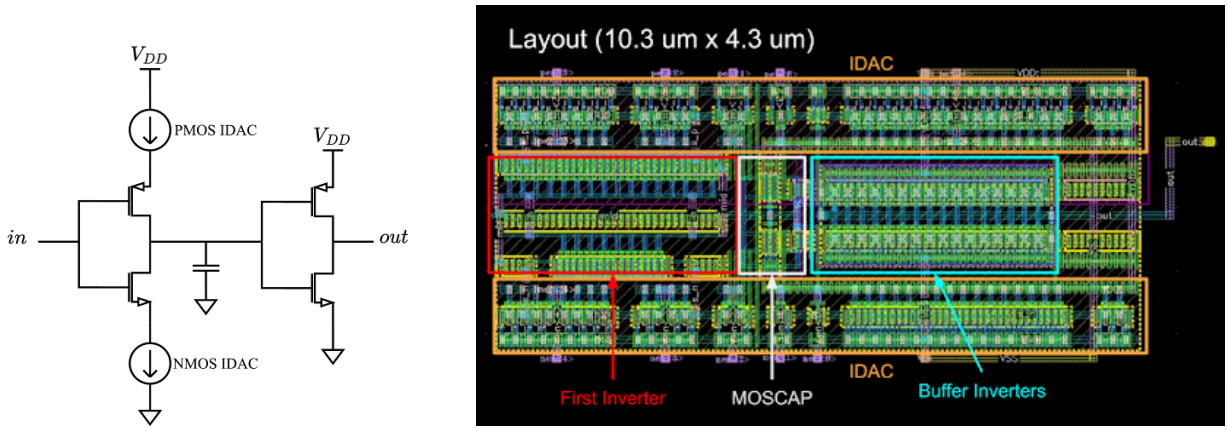


Figure 3.19: Phase interpolator layout.

## Duty Cycle Corrector[3]

To ensure optimal operation of the double date rate signaling, a Duty Cycle Corrector (DCC) is used to restore the duty cycle of the clock signal to 50/50, even in the presence of significant duty cycle distortion up to 40/60. Two instances of the DCC are instantiated within the module, one for each differential output path from the phase interpolator. Each DCC consists of a core inverter stage with a current-starving architecture, controlled via a 5-bit digitally controlled current DAC. The pull-up and pull-down networks are controlled by separate 5-bit digital signals `pen[4:0]` and `nen[4:0]`, respectively. This allows precise adjustment of the inverter's delay characteristics to compensate for the distortion. An inverter buffer follows the core stage to prevent signal loading and to drive the corrected output signal effectively. Additionally, a capacitor is added help fine-tune the delay characteristics.




(a) Schematic.

(b) Layout.

Figure 3.20: Design of duty cycle corrector.

## Full Clock Lane

The full layout of the integrated clock lane is demonstrated in Figure 3.21. The physical design of the clock lane is carefully engineered to support high-speed operation and preserve signal integrity of the signals. Some of the key layout features include:

- All clock lane pins are routed to the module boundary, permitting easy access and integration within the hierarchical block.
- Control signal pins are placed along the bottom edge of the layout
- The 8 GHz clock input from the PLL enters the layout from the top-left corner, while the DCC outputs are routed to the right edge.

[3]The duty cycle corrector was designed by Nikhil Jain and used from the 2024 Q3 Sirius chip tapeout.

- Power straps are manually generated up to a high metal layer, with full internal connectivity to ensure robust power delivery to the analog block.

This module's GDS and netlist files were submitted to the rest of the UCIe team to integrate within the larger TX data tile and UCIe PHY. The designs were verified to be DRC / LVS clean, aside from a handful of density-related errors which were confirmed to be resolvable to by fill and the top-level signoff flow.

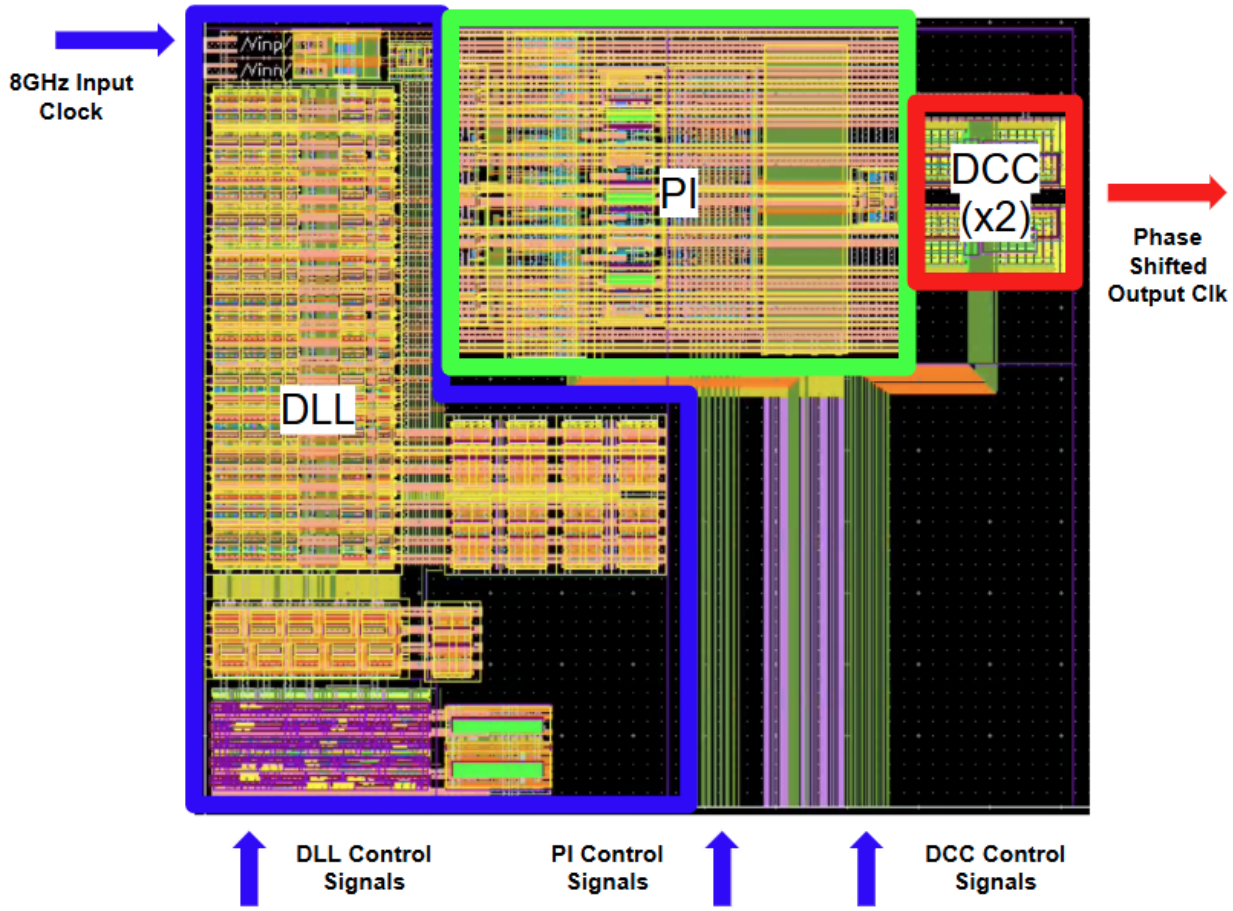



Figure 3.21: Labeled image of full clock lane layout.

# Chapter 4

# Simulation Results

This section presents the post-extraction simulation results of the proposed DLL design in Chapter 3. Results for the standalone delay line, the full delay locked loop, and the integrated clock lane are presented. Key performance metrics such as phase resolution, linearity, jitter, and power consumption are analyzed.

## 4.1 Delay Line Simulations

### Tuning Range (Delay vs. Code)

The measured tuning range of the DCDL is demonstrated across corners. The figure highlights that at the worst case deterministic corners, the ideal reference locking delay of 125ps is still within the tuning range of the DCDL.

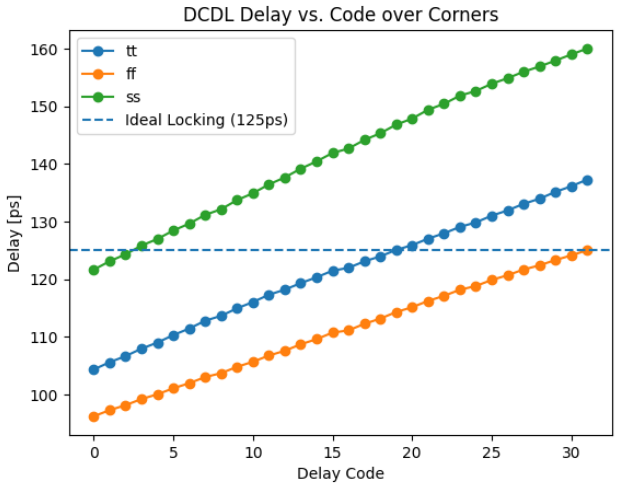
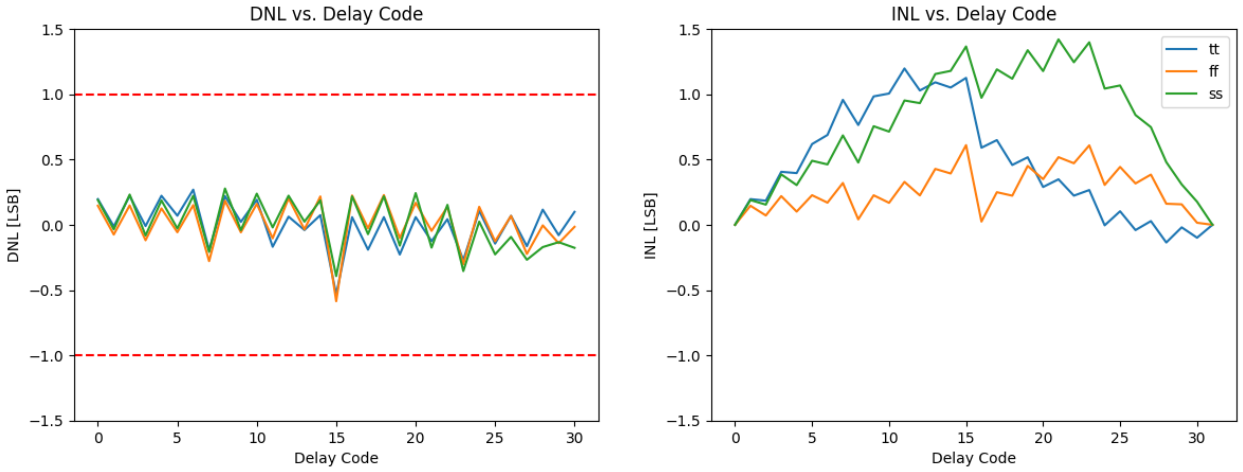



Figure 4.1: DCDL delay vs. code across corners (TT, FF, SS).

## Linearity

The differential (DNL) and integral (INL) non-linearities of the delay line are presented in Figure 4.2. The primary requirement is that the worst-case DNL is less than 1 LSB, which guarantees monotonicity and enables reliable locking of the circuit. Minimizing DNL is also important, as excessive DNL can lead to uneven delay spacing between digital codes, directly contributing to increased jitter and phase mismatches at the DLL outputs.

In contrast, the INL is less critical for DLL performance. Since the DLL operates within a feedback loop, small absolute phase errors introduced by INL can be still corrected as long as locking period is still within the tuning range of the delay line. Additional absolute phase error may be introduced, but poor INL should not impact the DLL's ability to lock in the same way DNL can.



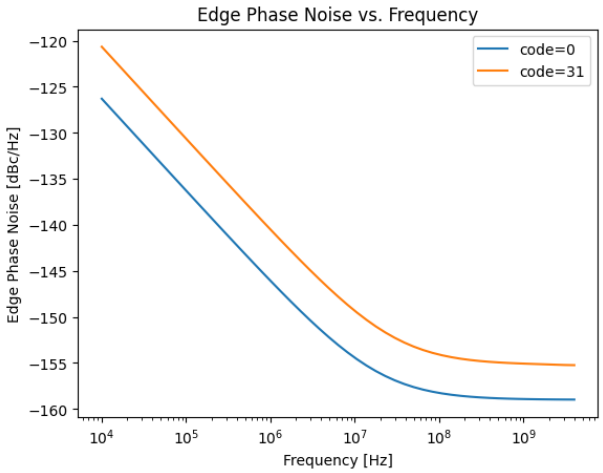

(a) DNL (b) INL

Figure 4.2: Measured differential and integral non-linearity of DCDL.

## Jitter

The following jitter measurements are obtained from the DCDL in isolation. These results are based on Periodic Steady-State (PSS) and PNoise analyses, which use jitter transfer functions and phase noise conversions to estimate the RMS jitter. This approach assumes linear time-invariant behavior around the steady-state operating point and captures small-signal noise contributions. The edge phase noise plots and measured RMS jitter at the minimum and maximum delay codes are shown in Figure 4.3. The calculation for jitter is an integration over the phase noise at the threshold crossing as shown in Equation 4.1. The integration limits $(f_1, f_2)$ are 10kHz to 4GHz (Nyquist frequency) and the carrier frequency $f_0 = 8$GHz.

$$\sigma_{\text{rms}} = \sqrt{\frac{1}{2\pi^2 f_0^2} \int_{f_1}^{f_2} 10^{\frac{\phi(f)}{10}} \, df} \tag{4.1}$$




(a) Edge phase noise of DCDL at `out<7>` for code 0 and code 31.

| Delay Code | RMS Jitter [fs] |
|---|---|
| 0 | 20.55 |
| 31 | 32.23 |

(b) Measured integrated RMS jitter (10kHz-4GHz integration bandwidth).

Figure 4.3: Measured DCDL jitter.

We observe that the delay line contributes greater jitter at higher delay codes. This intuitively makes sense as higher delay codes will introduce more capacitance, thus slowing the edge of the transitions, leading greater susceptibility to noise sources near the threshold crossing and increasing the jitter.

## 4.2 DLL Simulations

### Transient Locking

The most critical requirement of the delay locked loop is its ability to lock. If the circuit cannot properly reach a locking point, then the DLL outputs will not have any guaranteed phase relationship. Figure 4.4, demonstrates the DLL locking to codes 21 and 22 at the TT corner. The measured lock time from the initial reset is 4.3ns at the TT corner. At the SS corner, the DLL toggles around code 4, whereas at the FF corner, the delay line will saturate at code 31.

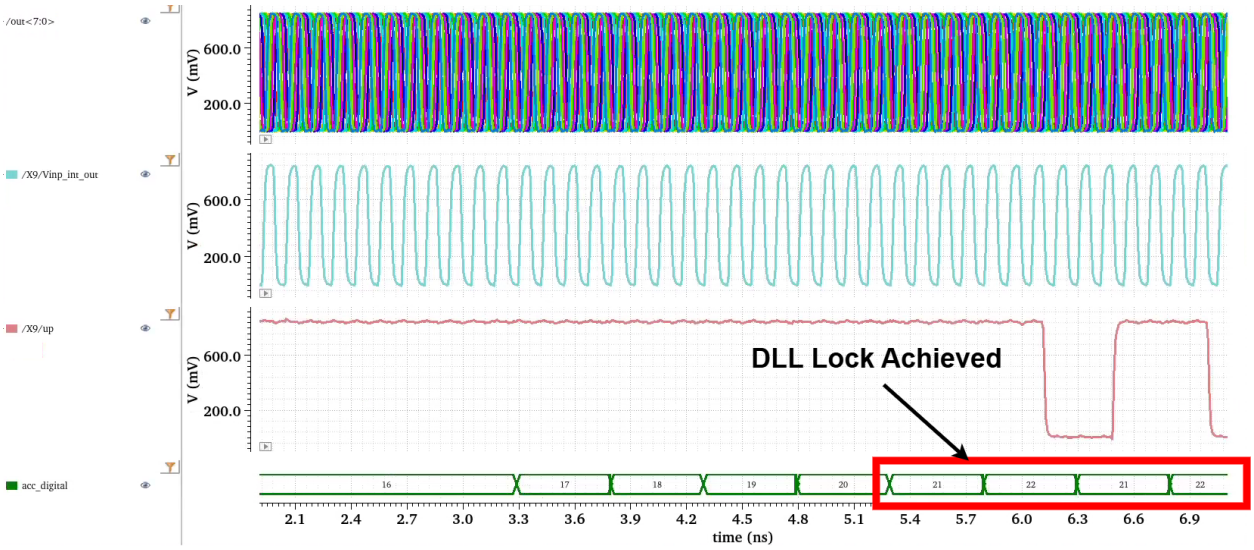
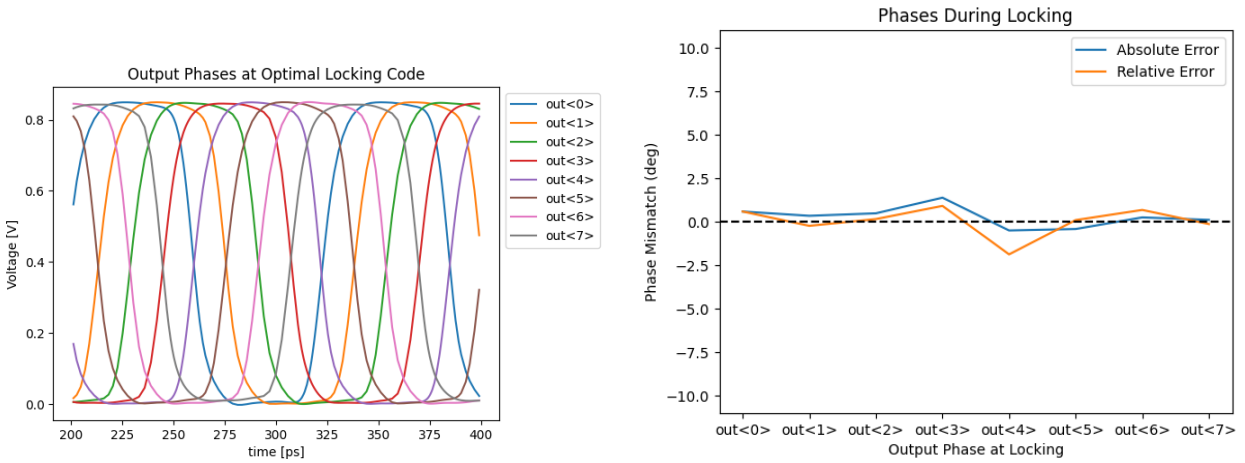



Figure 4.4: DLL locking demonstrated at typical corner.

## Output Phase Linearity

The output phase linearity provides a measure of how equally spaced the clock phases are and how much phase mismatch exists during lock. Ideally, when the DLL is locked we should expect the output phases to be equally spaced 45 degrees apart. For a 8 GHz clock, this results in a 15.625ps spacing between adjacent output phases produced by the DLL. Figure 4.5 illustrates the transient output of the DLL when locked to the optimal delay code and the corresponding linearity of the output phases. The output phase linearity plot demonstrates that during lock, the max phase mismatch from the ideal 15.625ps spacing is 1.89° which corresponds to 0.65ps of error.




(a) Output Phases Transient

(b) Output Phase Linearity

Figure 4.5: DLL output phase linearity (during locking at TT).

## Jitter

### Deterministic Jitter

Deterministic jitter refers to periodic timing variations in the output that are caused by known sources within the system. The DJ is largely dominated by the quantization error of the DCDL. Since the delay code alternates around the locking code, the deterministic jitter can be measured using eye diagram histograms and measurements as described in [19]. Using the histograms generated by the eye measurement tools shown in Figure 4.7, the resulting DJ at locking is calculated to be 1.166ps, roughly equivalent to the delay resolution of the DCDL.

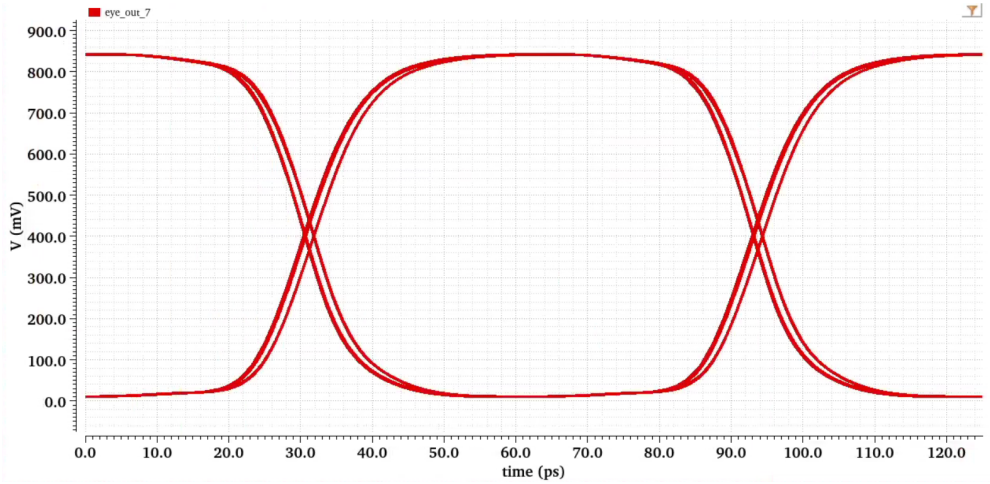



Figure 4.6: DLL eye diagram used for deterministic jitter measurement.

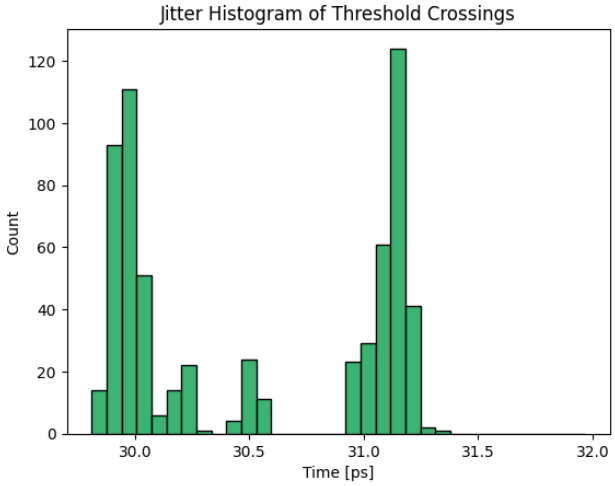



Figure 4.7: DLL threshold crossing histogram from eye diagram in Figure 4.6. 1.166ps of DJ is measured as the difference in the means of the two gaussian distributions.

### Random Jitter

Ideally the random jitter induced by the entire DLL can be measured using a periodic steady state (PSS) and Pnoise simulation as described in Section 4.1. However, due to the extreme non-linearity introduced by the sense amplifier flip-flop, the PSS analysis on the integrated DLL struggles to converge to a steady state, making it unfeasible to use this method for the jitter measurement. As an alternative, jitter can be measured through direct transient simulations with noise sources. While this approach is significantly slower, it does capture

a broader range of jitter sources and doesn't rely on convergence to a steady-state solution. By using eye diagrams and the corresponding eye measurement tools available in Virtuoso, the simulated RMS jitter can be extracted. This method typically yields higher jitter values than the PSS/Pnoise approach, as it accounts for additional nonlinear effects that are not captured in small-signal analyses.

Transient noise analysis was run for 25ns to generate 200 clock cycles worth of noisy threshold crossings from which the jitter at the sampling point was calculated. The noise $f_{\max}$ of the transient simulation was set to 40GHz to capture as much of the high frequency noise as possible while maintain a reasonable simulation time.

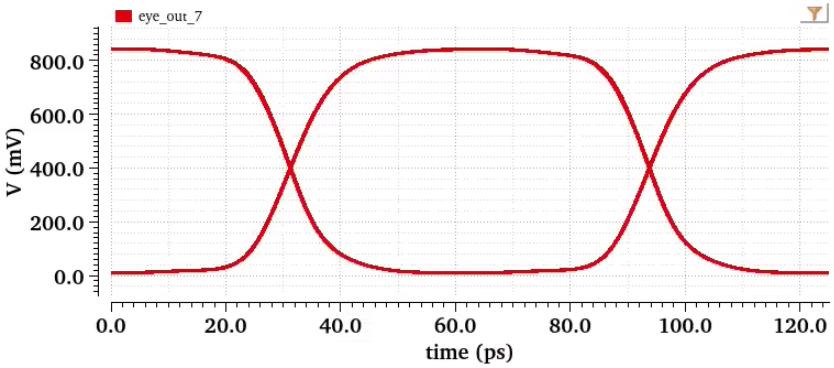



Figure 4.8: Eye diagram of DLL in open loop with transient noise for RMS jitter measurement.

The measured RMS jitter is 37.96 fs when the DLL is locked at delay code 21.

## Power Consumption

Power consumption is a critical design consideration, particularly in high-speed interfaces like UCIe. Although power was not explicitly optimized in this design as correct functionality was prioritized, it remains important to evaluate its overall efficiency for comparison against similar designs. To assess power consumption, a transient simulation was performed over the reset, locking and tracking operation of the DLL. The average power consumption was calculated by integrating the instantaneous power drawn by each circuit block over time. This provides insight into which components contribute most significantly to the overall power budget and informs potential areas for future optimization. This power is also converted to be in terms of pJ/bit while operated at 16Gb/s[1].

[1]Only measures power consumption from DLL and does not include rest of TX or UCIe.

| Sub-block | Power | [pJ/bit]@16Gb/s | % |
|---|---|---|---|
| DCDL | 10.6 mW | 0.66 pJ/bit | 84.8% |
| Phase Detectors | 526 $\mu$W | 0.033 pJ/bit | 4.2% |
| Integrator | 235.7 $\mu$W | 0.015 pJ/bit | 3.9% |
| Clk Divider | 214 $\mu$W | 0.013 pJ/bit | 1.7 % |
| Buffers | 926.7 $\mu$W | 0.058 pJ/bit | 7.4% |
| **Total** | 12.5 mW | 0.78 pJ/bit | – |

Figure 4.9: DLL power summary.

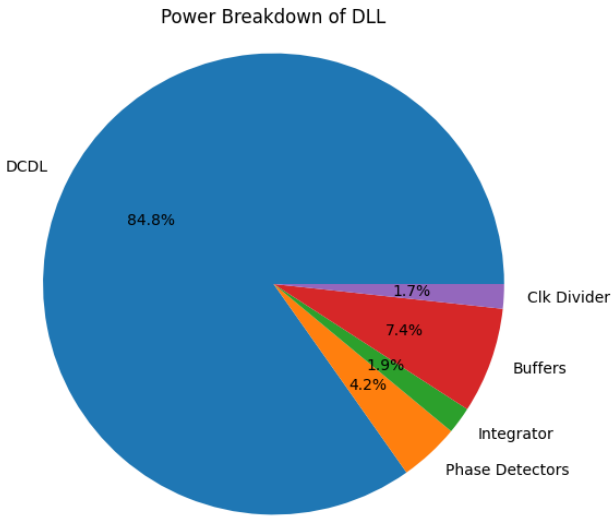



Figure 4.10: DLL power breakdown.

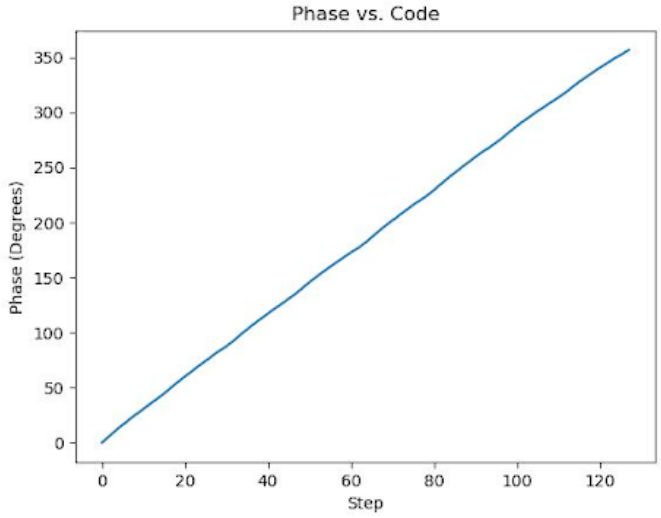
It is obvious that the power consumption is largely dominated by the DCDL, with the other components consuming $< 20\%$ of the overall power. This is one of the primary drawbacks of a shunt capacitor based delay line as increasing the load capacitance to modulate the delay will inevitably increase the amount of energy that is charged and discharged per transition. Alternatively, to improve the power consumption, a current starved based delay cell could be employed, which maintains a constant capacitance but instead changes the rate at which the capacitance is charged.

## 4.3 Clock Lane Simulations

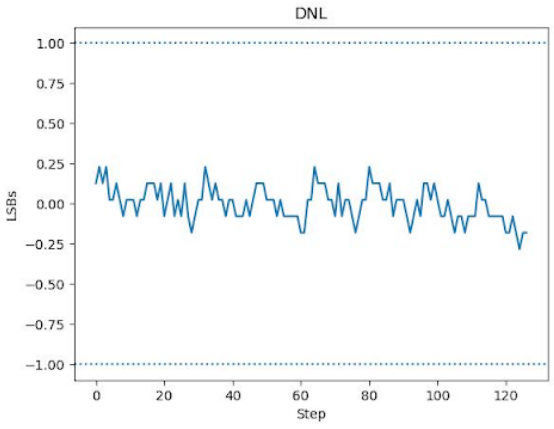
These simulations are performed post-extraction on the full integrated clock lane which includes the DLL, PI, and DCC.

## Phase Linearity

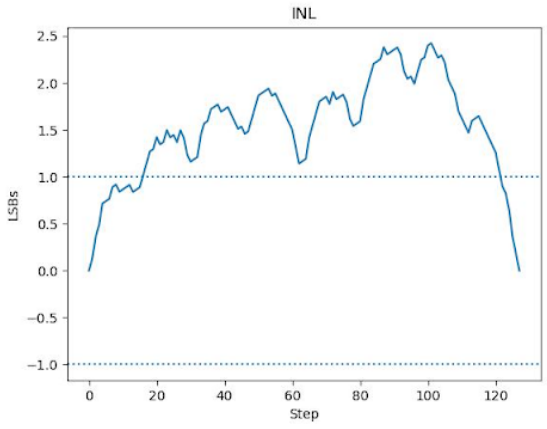
As described in the previous section, the combined DLL, PI, and DCC should be designed to provide phase tuning over a full 360° range of an 8GHz clock, with a minimum resolution of 16mUI. The following plots illustrate the DNL and INL of the generated phase steps over the full 360°. Compared to the 2024 Q3 version, which exhibited DNL values around 1 LSB, the phase step linearity has been significantly improved in the current design.




(a) Phase vs. Code

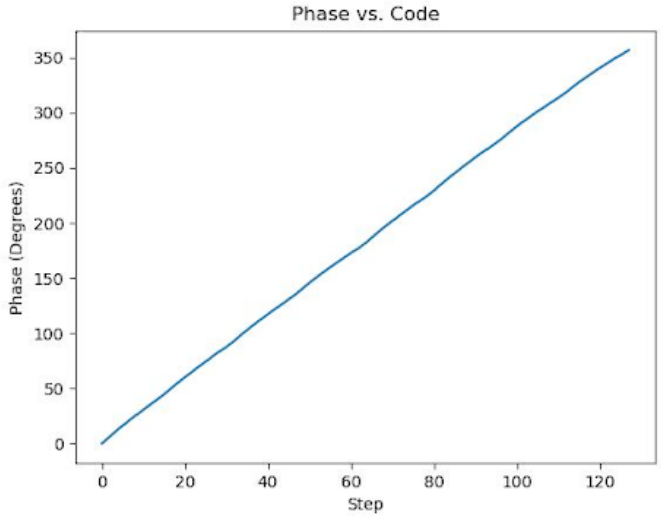



(b) DNL

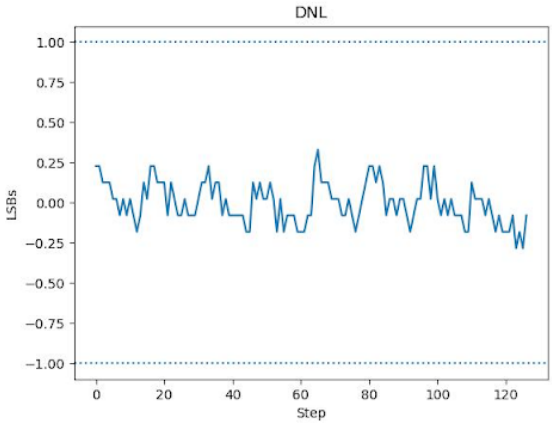



(c) INL

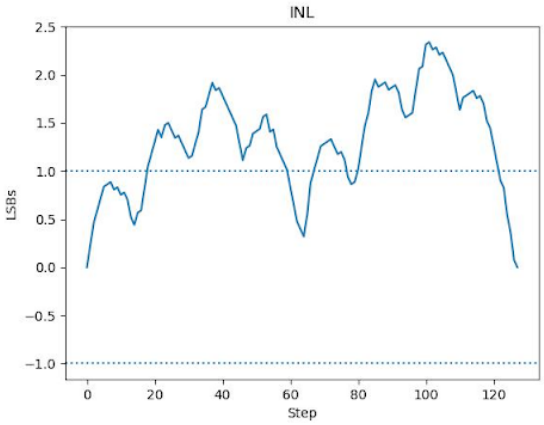
Figure 4.11: Clock lane linearity at TT corner




(a) Phase vs. Code

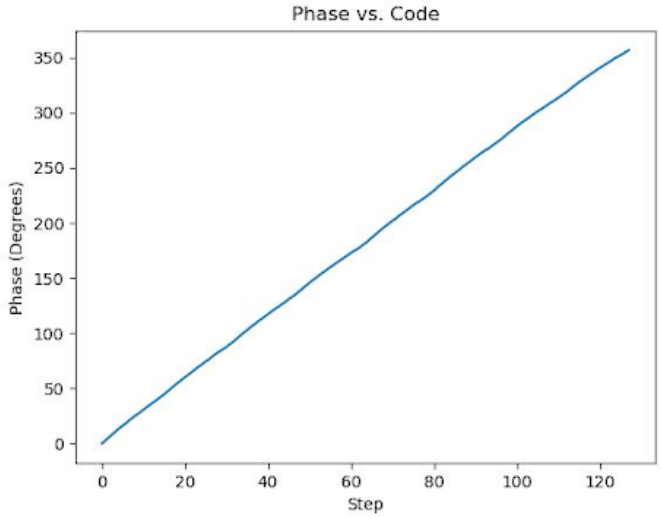



(b) DNL

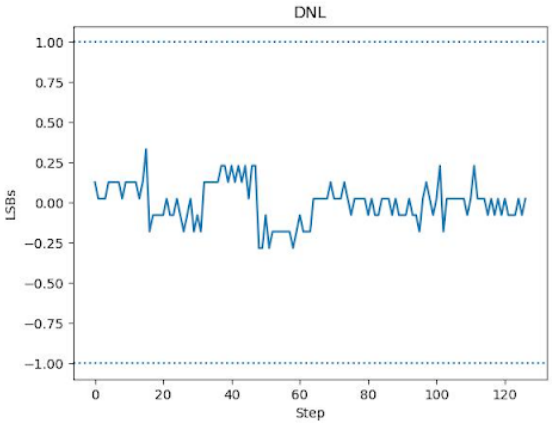



(c) INL

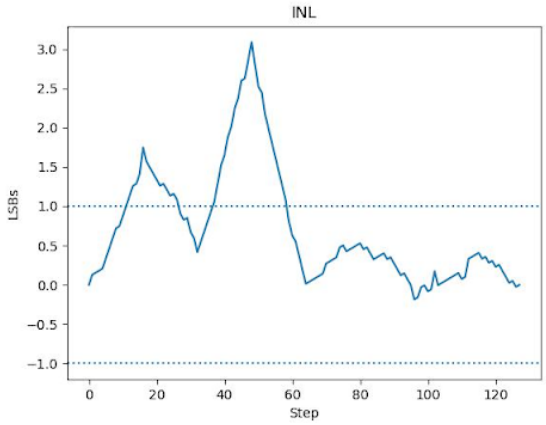
Figure 4.12: Clock lane linearity at FF corner.




(a) Phase vs. Code




(b) DNL




(c) INL

Figure 4.13: Clock lane linearity at SS corner.

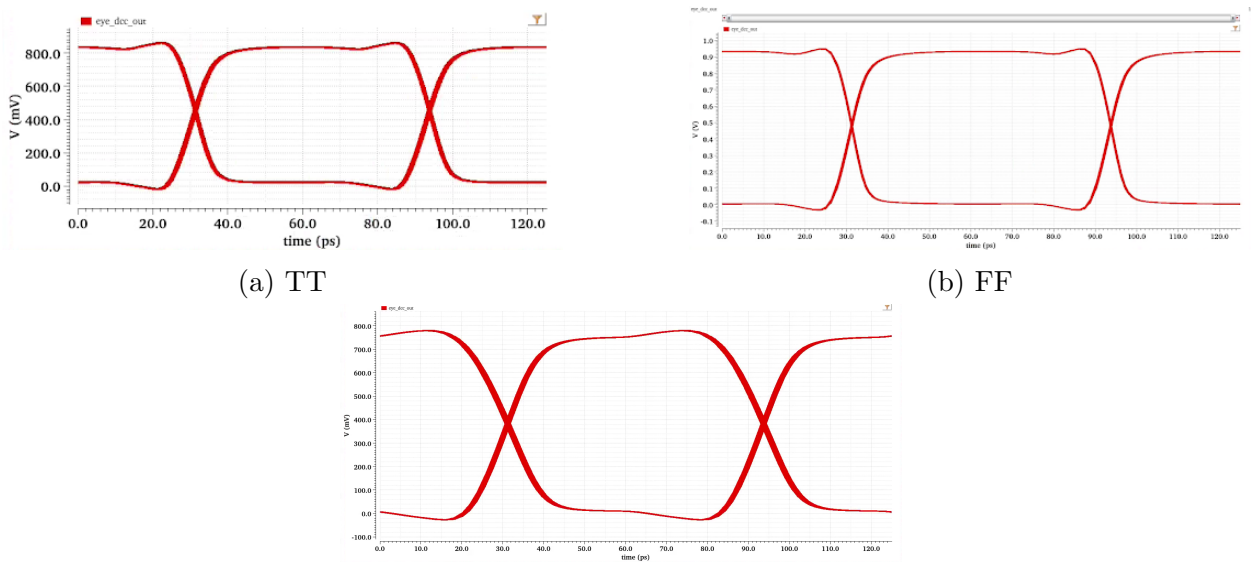
All corners hit the desired specification of max DNL being less than 1 LSB. It is clear from the INL lobes and patterns within the DNL that there is still some amount of systematic non-linearity from both the DLL and PI that can be further investigated and optimized. A summary of these results is shown in Table 4.1.

Table 4.1: DNL/INL summary of full 360 degree phase shift across corners.

| Corner | Max DNL [LSB] | Max INL [LSB] |
|---|---|---|
| TT | 0.2825 | 2.4245 |
| FF | 0.3314 | 2.3411 |
| SS | 0.3336 | 3.087 |

## Random Jitter

Similar to the RJ measurements done with the integrated DLL, the random jitter of the integrated clock lane can be measured using a transient noise simulation. The noisy eye diagrams are highlighted in Figure 4.14 with a summary of the measured random jitter values in Table 4.2.




(a) TT

(b) FF

(c) SS

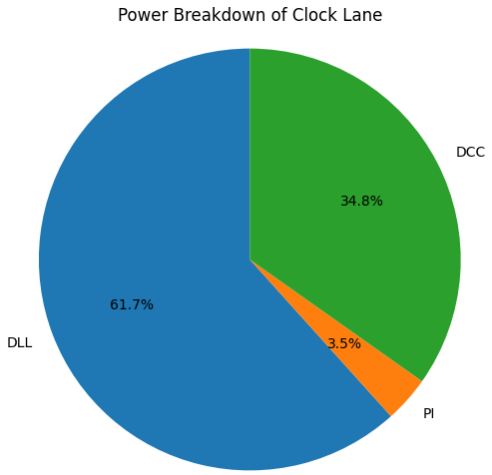
Figure 4.14: Clock lane eye diagrams with transient noise used for RMS jitter measurement.

Table 4.2: Summary of clock lane RMS jitter across corners.

| Corner | RMS jitter |
|---|---|
| TT | 114 fs |
| FF | 54 fs |
| SS | 242 fs |

## Power Consumption

Power consumption of the clock lane is characterized below, with the DLL consuming slightly more than 60% of the total power.




(a) Power Breakdown

| 2 **Block** | **Power** | **Percentage** |
|---|---|---|
| DLL | 12.51 mW | 61.7% |
| PI | 718 uW | 3.5 % |
| DCC | 7.058 mW | 34.8 % |
| **Total** | 20.3 mW | – |

(b) Power Summary

Figure 4.15: Clock lane power consumption.

## Clock Lane Performance Summary

A summary of the integrated clock lane performance is provided in Table 4.3.

Table 4.3: Summary of clock lane performance across PVT.

| **Spec** | **Desired** | **tt, 0.85V** | **ff, T=-40°C, 0.935V** | **ss, T=125°C, 0.765V** |
|---|---|---|---|---|
| Phase Step | $<$ 16 mUI | 15.61 mUI | 15.62 mUI | 15.61 mUI |
| DNL | $<$ 1 LSB | 0.2825 LSBs | 0.3314 LSBs | 0.3336 LSBs |
| INL | – | 2.4245 LSBs | 2.3411 LSBs | 3.087 LSBs |
| RJ | $<$ 86 fs | 114 fs | 54 fs | 242 fs |
| Power | – | 20.3 mW | 31.96 mW | 14.5 mW |

# Chapter 5

# Conclusion

This work explores the design considerations and challenges involved in the design of high speed delay locked loops, presenting a verified post-extraction design taped out in Intel 16nm PDK. Table 5.1 summarizes the achieved specifications of the isolated delay locked loop (not including the PI and DCC).

Table 5.1: Measured specifications of proposed DLL.

| **Specification** | **Measured (TT)** |
|---|---|
| DCDL Phase Resolution | 1.03 ps |
| Lock/Acquisition Time | 4.3 ns |
| Max Output Phase Error (during lock) | 0.65 ps |
| DJ (during lock) | 1.16 ps |
| RJ (during lock) | 37.96 fs |
| Power Consumption | 12.5 mW |
| Area | 32.4 x 54.63 $\mu$m |

The key contributions of this work focus on enhancing the robustness and precision of the DLL beyond its core functionality. Specifically, this work addresses two major limitations observed in previous designs: significant phase mismatch at lock and excessive delay code cycling during operation. To overcome these challenges, several main design techniques were exercised. First, careful load matching was implemented across the delay line to ensure uniform delay cell behavior and minimize multiphase skew, directly reducing mismatch between clock phases. Second, a reduced-deadzone phase detector was developed to improve phase detection sensitivity and accuracy near lock, enabling finer resolution phase correction. Third, the loop latency was intentionally limited to avoid potential instability and minimize delay code cycling. These optimizations were propagated from schematic design through to layout and were integrated within the larger clock lane. Implementing these changes required targeted layout modifications with careful attention to routing and parasitic effects. Collec-

tively, these contributions result in a more precise DLL design capable of being integrated into the broader UCIe system.

## 5.1 Future Work

Despite the submission of this work as part of the 2025 Q2 University Shuttle for Intel 16nm, there still remains significant opportunity for further exploration and optimization of the design. As with most analog and mixed-signal circuits, the design space of DLLs is vast.

To achieve full UCIe compliance, the DLL should support operation at other frequencies, including the 6GHz target. Meeting this requirement would require an additional coarse tuning mechanism in the delay line as designing the shunt-capacitance controlled delay line to achieve sufficient range would be challenging without drastically inflating the area. A promising approach would be to add current starving as an additional tuning knob, as presented in the delay line from [12].

To address the inevitable dithering between two delay codes near lock, several techniques from literature could be employed. These include lock detection schemes that automatically switch to open-loop control when repetitive code patterns or small phase errors are detected [15, 18].

As the tuning range of the presented design was relatively narrow, good output phase linearity was able to be achieved. However, in wider tuning scenarios, phase calibration and correction techniques may be required to equalize the phase outputs after the delay line lock is achieved. Some of these works include implementing additional delay cell tuning [6] and output buffer tuning [3, 21]. Implementations of these calibration mechanisms add non-trivial amounts of overhead however as the loops themselves must also be carefully design as to not compromise the precision of the original circuit.

Lastly, improving jitter and power consumption specifications requires careful bottom up design approach with intentional architectural designs. Using modeling techniques expressed in Section 2.2, further analysis of the sources of jitter can be studied and improved. As seen in the layout and power breakdown of the DLL, modifying and optimizing the DCDL control mechanism and sizings would have the greatest impact on area and power performance.

In summary, this work lays the foundation for a fully integrated digitally-controlled DLL, but the design space remains full of opportunities for further research and improvements.

# Bibliography


[1] Bilal I. Abdulrazzaq et al. "A review on high-resolution CMOS delay lines: towards sub-picosecond jitter performance". In: *SpringerPlus* 5.1 (2016), p. 434. DOI: 10.1186/s40064-016-2090-z. URL: https://springerplus.springeropen.com/articles/10.1186/s40064-016-2090-z.

[2] Nico Angeli and Klaus Hofmann. "Low-Power All-Digital Multiphase DLL Design Using a Scalable Phase-to-Digital Converter". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.4 (2020), pp. 1158–1168. DOI: 10.1109/TCSI.2019.2945086.

[3] Hsiang-Hui Chang et al. "A 0.7-2-GHz self-calibrated multiphase delay-locked loop". In: *IEEE Journal of Solid-State Circuits* 41.5 (2006), pp. 1051–1061. DOI: 10.1109/JSSC.2006.874036.

[4] Shu-Yu Chang and Shi-Yu Huang. "A Check-and-Balance Scheme in Multiphase Delay-Locked Loop". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32.7 (2024), pp. 1253–1262. DOI: 10.1109/TVLSI.2024.3393615.

[5] Chithra. *EE698G: Circuit design for frequency and phase synthesis (2024)*. 2024. URL: https://www.youtube.com/playlist?list=PLP-rjhz_nIi7o76Vc_H0L7wyWCKRJM3t2.

[6] Woo-Seok Choi et al. "3.8 A 0.45-to-0.7V 1-to-6Gb/S 0.29-to-0.58pJ/b source-synchronous transceiver using automatic phase calibration in 65nm CMOS". In: *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*. 2015, pp. 1–3. DOI: 10.1109/ISSCC.2015.7062928.

[7] J. Crossley et al. "BAG: A designer-oriented integrated framework for the development of AMS circuit generators". In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013, pp. 74–81. DOI: 10.1109/ICCAD.2013.6691100.

[8] Nicola Da Dalt. "Linearized Analysis of a Digital Bang-Bang PLL and Its Validity Limits Applied to Jitter Transfer and Jitter Generation". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 55.11 (2008), pp. 3663–3675. DOI: 10.1109/TCSI.2008.925948.

[9] Hammer Developers. *Hammer VLSI Tool Documentation*. Accessed: 2025-04-23. 2025. URL: https://hammer-vlsi.readthedocs.io/en/stable/.

[10] M.-J.E. Lee et al. "Jitter transfer characteristics of delay-locked loops - theories and design techniques". In: *IEEE Journal of Solid-State Circuits* 38.4 (2003), pp. 614–621. DOI: 10.1109/JSSC.2003.809519.

[11] Harrison Liew et al. "Hammer: a modular and reusable physical design flow tool: invited". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. San Francisco, California: Association for Computing Machinery, 2022, pp. 1335–1338. ISBN: 9781450391429. DOI: 10.1145/3489517.3530672. URL: https://doi.org/10.1145/3489517.3530672.

[12] Mu-Shan Lin et al. "36.1 A 32Gb/s 10.5Tb/s/mm 0.6pJ/b UCIe-Compliant Low-Latency Interface in 3nm Featuring Matched-Delay for Dynamic Clock Gating". In: *2025 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 68. 2025, pp. 586–588. DOI: 10.1109/ISSCC49661.2025.10904767.

[13] J.G. Maneatis. "Timing and clocking: Design of high-speed CMOS PLLs and DLLs". In: Jan. 2001, pp. 10–1.

[14] Didem Turker Melek et al. "A 0.29pJ/b 5.27Tb/s/mm UCIe Advanced Package Link in 3nm FinFET with 2.5D CoWoS Packaging". In: *2025 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 68. 2025, pp. 590–592. DOI: 10.1109/ISSCC49661.2025.10904754.

[15] Behzad Mesgarzadeh and Atila Alvandpour. "A Low-Power Digital DLL-Based Clock Generator in Open-Loop Mode". In: *IEEE Journal of Solid-State Circuits* 44.7 (2009), pp. 1907–1913. DOI: 10.1109/JSSC.2009.2020229.

[16] Hyunsu Park et al. "A 1.3–4-GHz Quadrature-Phase Digital DLL Using Sequential Delay Control and Reconfigurable Delay Line". In: *IEEE Journal of Solid-State Circuits* 56.6 (2021), pp. 1886–1896. DOI: 10.1109/JSSC.2020.3045168.

[17] Behzad Razavi. "The Delay-Locked Loop [A Circuit for All Seasons]". In: *IEEE Solid-State Circuits Magazine* 10.3 (2018), pp. 9–15. DOI: 10.1109/MSSC.2018.2844615.

[18] Muhammad Riaz Ur Rehman et al. "A Design of 6.8 mW All Digital Delay Locked Loop With Digitally Controlled Dither Cancellation for TDC in Ranging Sensor". In: *IEEE Access* 8 (2020), pp. 57722–57732. DOI: 10.1109/ACCESS.2020.2982180.

[19] D. Chaberski S. Grzelak M. Zieliński. "Measuring deterministic jitter using time interval measurement system". In: *Proceedings of the International Workshop on ADC Modelling and Testing (IWADC)*. IMEKO. 2008. URL: https://www.imeko.org/publications/iwadc-2008/IMEKO-IWADC-2008-125.pdf.

[20] UCIe Consortium. *Universal Chiplet Interconnect Express (UCIe) Specification Revision 1.1*. Beaverton, OR, USA: UCIe Consortium, Aug. 2023. URL: https://www.uciexpress.org/specifications.

[21] Lin Wu and W.C. Black. "A low-jitter skew-calibrated multi-phase clock generator for time-interleaved applications". In: *2001 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. ISSCC (Cat. No.01CH37177)*. 2001, pp. 396–397. DOI: 10.1109/ISSCC.2001.912690.

[22] Thucydides Xanthopoulos. "Digital Delay Lock Techniques". In: *Clocking in Modern VLSI Systems*. Ed. by Thucydides Xanthopoulos. Boston, MA: Springer US, 2009, pp. 183–244. ISBN: 978-1-4419-0261-0. DOI: 10.1007/978-1-4419-0261-0_6. URL: https://doi.org/10.1007/978-1-4419-0261-0_6.

[23] Jian Yang et al. "A 2.0-to-7.4-GHz 16-Phase Delay-Locked Loop With a Sub-0.6-ps Phase-Delay Error in 40-nm CMOS". In: *IEEE Transactions on Microwave Theory and Techniques* 71.8 (2023), pp. 3596–3604. DOI: 10.1109/TMTT.2023.3242333.

[24] Hanqiao Zhang, Steven Krooswyk, and Jeff Ou. "Chapter 4 - Link circuits and architecture". In: *High Speed Digital Design*. Ed. by Hanqiao Zhang, Steven Krooswyk, and Jeff Ou. Boston: Morgan Kaufmann, 2015, pp. 163–198. ISBN: 978-0-12-418663-7. DOI: https://doi.org/10.1016/B978-0-12-418663-7.00004-6. URL: https://www.sciencedirect.com/science/article/pii/B9780124186637000046.

[25] Renyuan Zhang and Mineo Kaneko. "Robust and Low-Power Digitally Programmable Delay Element Designs Employing Neuron-MOS Mechanism". In: *ACM Trans. Des. Autom. Electron. Syst.* 20.4 (Sept. 2015). ISSN: 1084-4309. DOI: 10.1145/2740963. URL: https://doi.org/10.1145/2740963.

# Appendix A

# Pin Summaries

The following tables outline and describe the pins for each sub-block of the design.

| Signal | Direction | bitwidth | A/D | Notes |
|---|---|---|---|---|
| `delay`[4:0] | in | 5 | D | Open loop control bits. When OCL is set to 1, this replaces the closed loop control bits. Ignored when OCL=0. |
| `delayb`[4:0] | in | 5 | D | Open loop control bits. complementary input to `delay` to drive the CMOS transmission gate . |
| `Vinp` | in | 1 | A | Positive input clock (8GHz from PLL) |
| `Vinn` | in | 1 | A | Negative input clock (8GHz from PLL) |
| `VDD` | inout | 1 | A | N/A |
| `VSS` | inout | 1 | A | N/A |
| `out[7:0]` | out | 8 | A | 8 output phases equally spaced 45 degrees apart. |

Table A.1: Summary of DCDL pins.

| Signal | Direction | bitwidth | A/D | Notes |
|---|---|---|---|---|
| clk | in | 1 | A | Tail of the StrongARM. During LOW will reset internal nodes HIGH and when HIGH will begin sensing and amplifying difference between inp and inn. |
| resetb | in | 1 | A | Resets SR Latch |
| inp | in | 1 | A | Positive input |
| inn | in | 1 | A | Negative input |
| out | out | 1 | A | Latched data value |
| outb | out | 1 | A | Complement of out |
| VDD | inout | 1 | A | N/A |
| VSS | inout | 1 | A | N/A |

Table A.2: Summary of phase detector pins.

| Signal | Direction | Bits | A/D | Notes |
|---|---|---|---|---|
| clk | in | 1 | A | 2GHz clock from clock divider |
| reset | in | 1 | A | Resets integrator to middle code 16 |
| EN | in | 1 | A | Used to disable integrator |
| OCL | in | 1 | A | Used to drive DLL code externally. |
| up | in | 1 | A | Up / down update to the integrator |
| delay[4:0] | in | 5 | D | DLL delay code for open loop control. |
| delayb[4:0] | in | 5 | D | Complement of DLL delay code for open loop control. |
| acc[4:0] | out | 5 | D | Delay code driving the delay line (NMOS switches). If OCL is asserted, acc[4:0] = delay[4:0] |
| accb[4:0] | out | 5 | D | Complement of delay code driving the delay line (PMOS switches). If OCL is asserted, accb[4:0] = delayb[4:0] |
| DLL_code[4:0] | out | 5 | D | Low frequency debug signal driving MMIO register (see Section DLL Debug Signal) |
| VDD | inout | 1 | A | N/A |
| VSS | inout | 1 | A | N/A |

Table A.3: Summary of digital integrator pins.

| Signal | Direction | Bits | A/D | Notes |
|---|---|---|---|---|
| reset | in | 1 | D | Reset signal for the digital integrator |
| resetb | in | 1 | D | Reset signal for the SAFF phase detector |
| EN | in | 1 | D | Enable signal for the DLL. When LOW, DLL signal is held constant. |
| OCL | in | 1 | D | Open loop control. When set to 1, delay/delayb perform open loop control instead of closed-loop feedback control. |
| delay[4:0] | in | 5 | D | Open loop control bits. When OCL is set to 1, this replaces the closed loop control bits. Ignored when OCL=0. |
| delayb[4:0] | in | 5 | D | Complement of delay[4:0] |
| Vinp | in | 1 | A | Positive input phase. |
| Vinn | in | 1 | A | Negative input phase. |
| VDD | inout | 1 | A | N/A |
| VSS | inout | 1 | A | N/A |
| out[7:0] | out | 8 | A | 8 output phases to the PI (out[0] = 45 degree shift, out[7] = 360 degree shift) |
| DLL_code[4:0] | out | 5 | D | Debug signal for observability (Section 3.6) |

Table A.4: Summary of DLL pins.