A 24Gb/s PAM-4 Clock and Data Recovery Circuit With High Jitter Tolerance

J.I.Bas

TU Delft Delft University of Technology

A 24Gb/s PAM-4 Clock and Data Recovery Circuit With High Jitter Tolerance

by

J.I.Bas

To obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on March 11 2024.

Student number: 4386388
Project Duration: September, 2022 - February 2024
Thesis Committee: Dr. Masoud Babaie, TU Delft, Chair
Dr. Fabio Sebastiano , TU Delft, Academic Supervisor

TU Delft

Abstract

The escalating demand for higher data rates in modern communication networks are pushing more transmitters and receivers to use a modulation technique with more spectral efficiency, like pulse amplitude modulation 4-level (PAM-4).

On the receiver side, phase detection for PAM-4 has proven to be difficult with most receivers using phase detection for non return to zero (NRZ) data. This neglects most transitions and thus some phase information is lost. This results in low bandwidth and jitter tolerance, which is a problem in noisy communication systems where it will lead to a high bit error rate (BER).

This thesis explores an integrated PAM-4 clock and data recovery (CDR) circuit utilizing a novel PAM-4 bang bang phase detector (BBPD) considering all data transitions. A digital oscillator with variable gain is used in order to achieve high jitter tolerance as-well as low jitter generation. at $24Gb/s$ the CDR consumes $8mW$ and generates $487fs$ of jitter. and has a 1 UI at $30MHz$.

Acknowledgements

First, I would like to thank my supervisor Dr. Masoud Babaie for his guidance during my thesis. His support kept me motivated and helped me reach new heights. The plentiful technical discussions we had were both enjoyable and insightful.

I also want to extend my gratitude to Dr. Fabio Sebastiano for reading my thesis and serving as committee.

My sincere appreciation goes to my daily supervisor, Rishabh Gurbaxani, for his help during the project. His work ethic inspired me during my thesis, and will continue to do so after.

I extend my special thanks to the Coolgroup and its members for the technical discussions and assistance, as well as for the lunch meetings and social events.

Finally, I want to thank my parents and friends for their continued support throughout the year, both in my academic life and my personal life.

J.I.Bas
Delft, March 2024

Contents

1 Introduction

This chapter provides an introduction to the background information required to understand the thesis project. It outlines an overview of the basic clock and data recovery (CDR) concepts as well as pulse amplitude modulation (PAM) and recent developments in academia. Following this, it outlines the motivation and specifications of the thesis project. Finally, a thesis outline is presented along with the contributions of each chapter.

1.1. Clock and Data Recovery

In wireline communications, a data stream is transmitted over a channel from a transmitter (TX) to a receiver (RX). A general block diagram is presented in Fig. 1.1. Data is serialized prior to being equalized and transmitted across a channel. At the receiver side, the incoming data is first amplified with a continuous time linear equalizer (CTLE) to compensate for channel loss. In most applications, the transmitter clock is not transmitted; this would require extra power and an extra channel. Therefore, the receiver has no information about the frequency and phase of the incoming data. A clock recovery circuit is used to align the receiver clock with the incoming data. With the recovered clock, a retimer or deserializer is used to create synchronized output data. Together, they make a CDR.

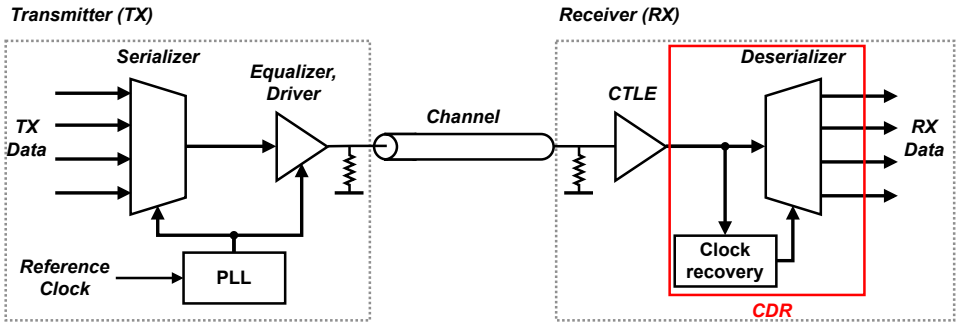


Figure 1.1: General TX-RX architecture.

Most CDRs are based on a phase-locked loop (PLL) design, using the data as a reference instead of a crystal oscillator. A CDR always consists of a few necessary components, as depicted in Fig. 1.2. The incoming data is compared to the CDR clock by the phase detector (PD), a defining element in the CDR that will be discussed later. Subsequently, the phase information is transmitted through a filter to adjust the frequency of a voltage-controlled oscillator (VCO) to align its phase and frequency with the data. The updated clock is then used for a renewed phase comparison and data extraction.

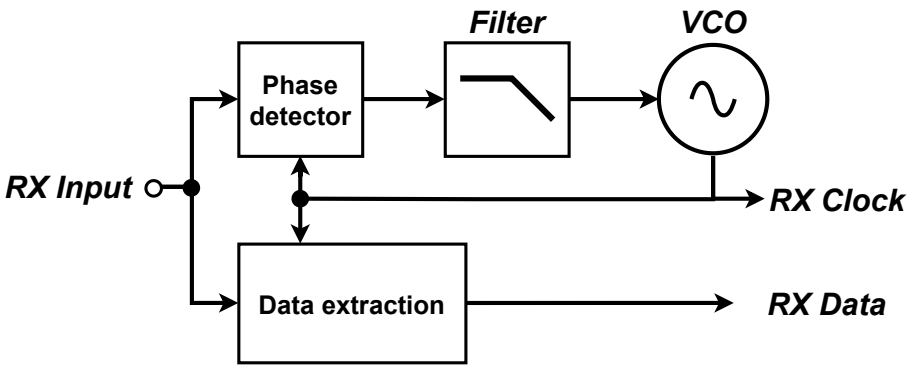


Figure 1.2: Standard CDR block diagram

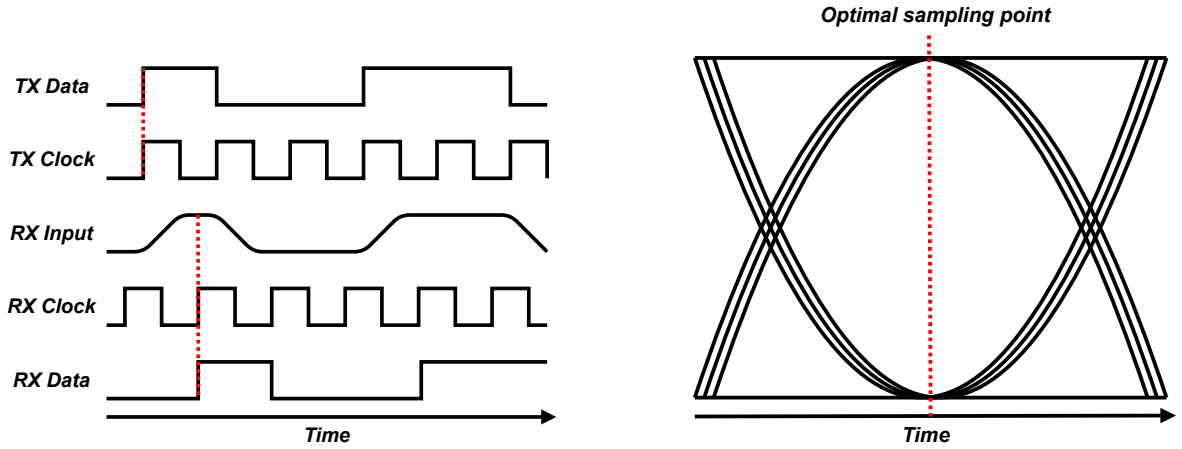
Since CDR topologies vary widely, comparing the state-of-the-art designs is complicated. The absence of a universally accepted figure of merit (F.O.M), due to the various, sometimes conflicting design metrics, adds to the challenge of comparison. The optimal CDR choice depends on the specific implementation requirements.

The main design metrics for evaluating a CDR are as follows:

- **Data rate:** The maximum rate at which data can be processed by the CDR.
- **Jitter tolerance:** The maximum amount of jitter the CDR can tolerate before bits are misinterpreted and bit errors occur.
- **Jitter generation:** The jitter that is generated by the CDR on the output clock.
- **Jitter transfer:** The amount of jitter the CDR transfers from input to output.
- **Power consumption:** The power consumed by the CDR, typically measured in pJ/bit for a fair comparison.

Jitter tolerance and jitter transfer directly oppose each other. For jitter tolerance the input jitter needs to be tracked accurately to the output in order to keep a valid data sample, thus increasing jitter transfer. The bandwidth of the CDR needs to be increased in order to have better tracking. To minimise jitter transfer the input jitter needs to be filtered out and a small bandwidth is required, thus decreasing jitter tolerance.

An important part of data recovery is data re-timing. This is done by sampling the data with the recovered clock. An optimal point exists where the bit error rate (BER) is minimal. This point is in the middle of a bit where amplitude noise, timing noise (jitter) and intersymbol interference (ISI) are least noticeable. This process is illustrated in Fig. 1.3. Half a symbol shift is therefore required between the received data rising edge and internal clock rising edge.



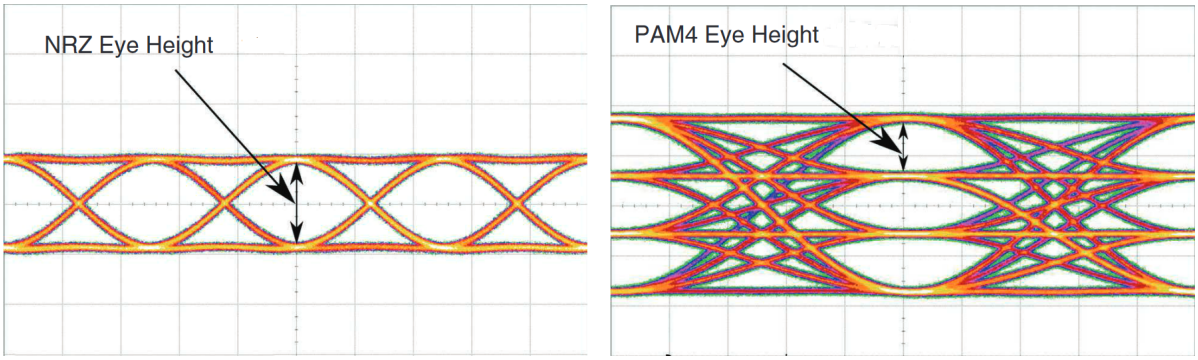
(a) TX and RX waveforms

(b) Eye diagram and optimal sample point

Figure 1.3: CDR data recovery.

1.1.1. Pulse Amplitude Modulation

The previously discussed phase detectors were illustrated using non-return-to-zero (NRZ) data. A form of pulse amplitude modulation (PAM) with two levels, where the data is encoded in the amplitude of the signal. PAM-4 utilizes four different amplitudes, an eye diagram for NRZ data is illustrated in Fig.1.4a and for PAM-4 in 1.4b. Due to the higher possible states, the data rate is twice the baud rate. This is beneficial when high data rates are required. Loss in the medium is frequency-dependent, therefore it is better to transmit data at a lower frequency. However, since the states are only separated by $\frac{1}{3}$ of the signal amplitude, the signal to noise ratio is $-9.5dB$ lower than NRZ data without channel loss. This creates an optimum modulation type depending on the channel loss and data rate [1].



(a) NRZ Eye diagram[1]

(b) PAM-4 Eye diagram[1]

Figure 1.4: Eye diagrams for different PAM modulation techniques

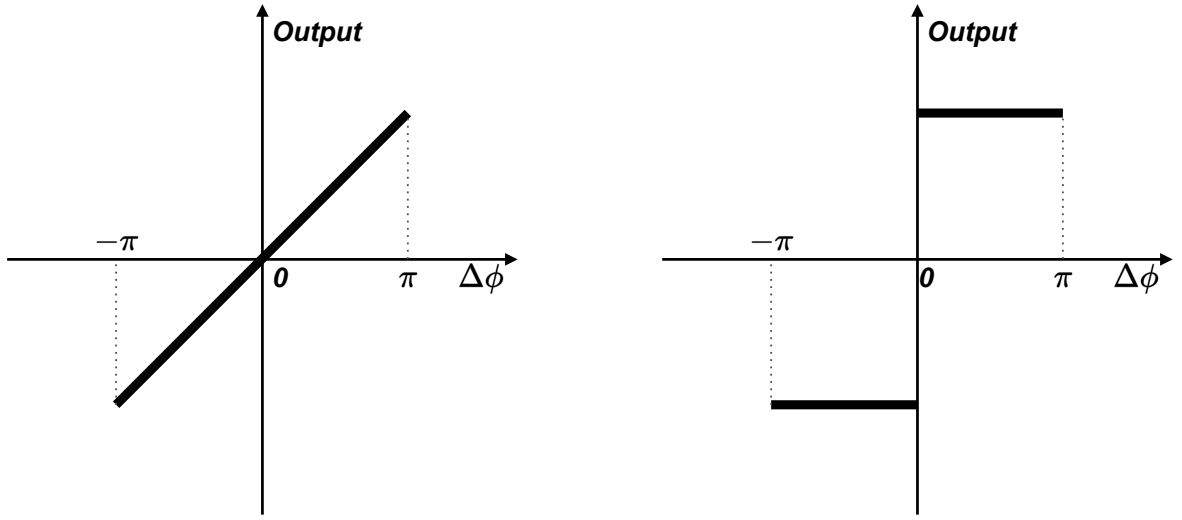
1.2. Phase Detection

One of the most defining blocks of a CDR is the PD. In order to understand the challenges in PAM-4 phase detection, the topic of this thesis, it is beneficial to first analyze the phase detection methods of NRZ data. Not all common methods will be covered. Many topologies rely on the binary nature of NRZ data to extract the phase information and are difficult to adjust to PAM-4 data, with the Hogge PD being a prime example [2].

Phase detectors can be divided into two classes based on the type of information they provide to the rest of the CDR loop. A linear phase detector has a linear relation between input and output, as plotted in Fig. 1.5a. This generally allows for faster phase and frequency locking, but phase detection can be more difficult since the phase detector needs to provide a linear relation between the input phase error and its output. Linear PDs are mostly used in analog CDR loops where complex calculations can be

done easily in the current domain at no extra cost.

The Bang-Bang phase detection (BBPD) provides only binary information (early or late) to the loop filter, as illustrated in Fig. 1.5b. This simplicity allows for easy implementation in digital PLL-based CDRs. Since the phase information is already 1-bit quantized, no additional step is required to move from the analog to the digital domain. However, the simplicity comes at a cost. Since both large and small phase errors result in the same update to the VCO, the CDR must either over- or under-compensate the VCO phase error. Under compensation will result in slow locking while over compensation will result in additional jitter generation due to the increased phase step (dithering jitter) creating a trade-off between the two. A more in-depth analysis of BBPD is done in Chapter 3 3.



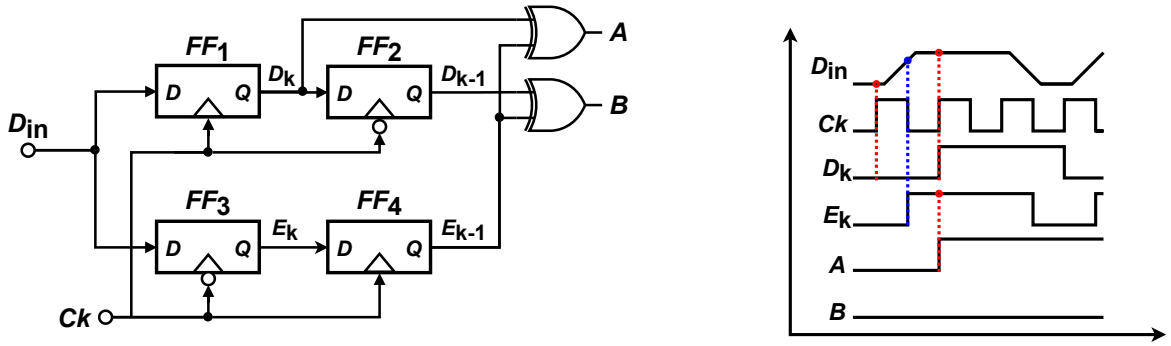
(a) Linear PD transfer characteristic.

(b) Bang-Bang PD transfer characteristic.

Figure 1.5: Phase detector classes

1.2.1. Alexander Phase Detector

The Alexander PD [3] uses positively and negatively clocked flip-flops to detect the phase error. The topology is presented in Fig.1.6a. To illustrate the functionality, the waveforms for a lagging clock are presented in Fig.1.6b. The input data is sampled at the positive clock flank with FF_1 to create D_k and at the negative clock flank with FF_2 to create E_k. D_k represents the data samples and E_k represents the edge or transition samples. On the subsequent clock flanks, the states are propagated to the next flip-flops in order to store the samples. The Alexander PD compares the transition sample E_{k-1} to both its preceding bit D_{k-1} and its proceeding bit D_k to determine the phase information. The samples of E_k are not taken at rail-to-rail voltage since the sample is taken during a transition; the flip-flop acts as a 1-bit quantizer. If E_{k-1} and D_k are opposite, it means that the edge sample was taken closer to D_{k-1} and the clock phase is leading the input data. Signal B becomes high in this case. If E_{k-1} is equal to D_k, as illustrated in the figure, the phase is lagging and signal A becomes high.



(a) Alexander PD topology.

(b) Lagging waveforms of the Alexander PD.

Figure 1.6: Alexander PD.

Comparing whether the transition E_{k-1} was closer to D_k or D_{k-1} can be extended to PAM-4 data. This does pose extra challenges since a 1-bit quantizer is not sufficient for the multi-level data.

1.2.2. Baud-Rate Phase Detector

While the Alexander PD takes samples with twice the bit rate, a Mueller Muller PD [4] detects phase error with only data samples. Therefore, it is also known as a "Baud-Rate PD". Several types of algorithms exist, each with its own applicable data patterns. Let us examine an algorithm for baud rate detection to understand the basic principles.

In the waveforms and samples of Fig.1.7, the clock signal CK is lagging the data stream D_{in} significantly. The phase detector takes samples at each positive clock flank to create S_k. The samples are also resolved to the nearest data point to create D_k. For data patterns with one transition and two consecutive bits, the following formula can determine phase information:

$$\begin{aligned} \textit{Phase error} &= (s_k \cdot d_{k-1}) - (s_{k-1} \cdot d_k) \\ \textit{If } & d_k \neq d_{k-1} \\ \textit{And } & d_{k-1} = d_{k-2} \end{aligned}$$

The first three samples are $s_k = -0.25$, $s_{k-1} = 0.5$, and $s_{k-2} = 0.25$ relative to the mid voltage. The resolved data d_k, d_{k-1} and d_{k-2} is $-1, 1, 1$ respectively, and a quantized data stream will correspond to this. Applying the aforementioned algorithm: $-0.25 \cdot 1 - 0.5 \cdot -1 = 0.25$ yields a positive value corresponding to a lagging clock. For samples $s_{k-4} = 0.25$, $d_{k-5} = -0.5$ and $d_{k-2} = 0.25$, the same can be computed: $0.25 \cdot -1 - -0.5 \cdot 1 = 0.25$ again indicates a lagging clock. Not all data patterns provide phase information. Alternating bits s_{k-2}, s_{k-3}, s_{k-4} will result in 0 for lagging and leading cases.

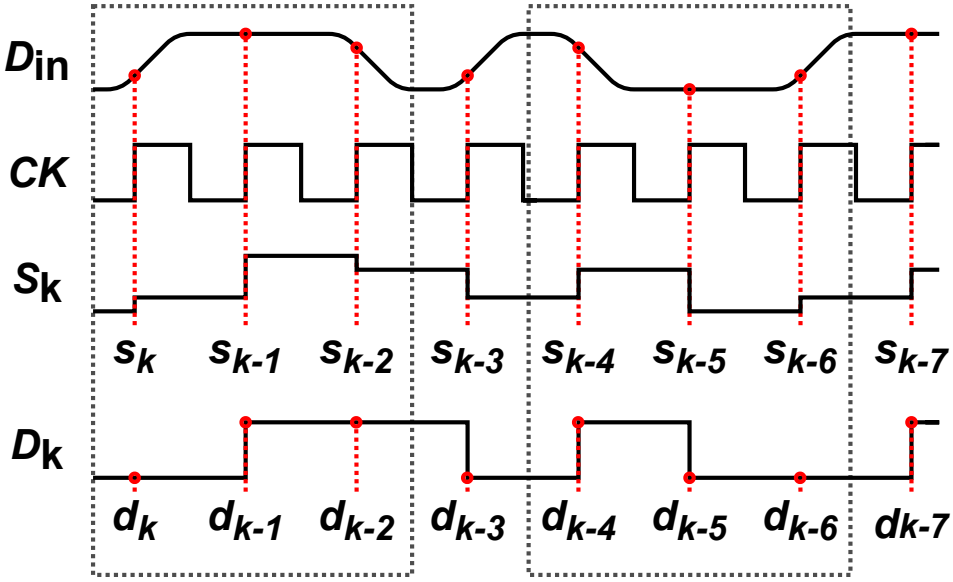


Figure 1.7: Baud-Rate PD waveforms.

Baud-rate PD relies on ISI and the channel attenuation to compute the phase information. The preceding bit attenuates the amplitude of the proceeding bit. The point at which the data sample is taken determines how much ISI is measured. Although the reduced sample rate is beneficial for high-speed data links, there are several drawbacks. As mentioned earlier, not all data patterns provide phase information, requiring some signal processing. Other baud rate detectors may use different algorithms requiring different data streams. Furthermore, the accuracy of the phase information depends on the level of ISI; if ISI is too low to be measured (also dependent on the sample resolution), no phase information can be detected. In the previous example, phase information can only be extracted when the clock shifts enough for ISI to occur. As the data stream has limited ISI, the clock can wander freely in a large range, resulting in high jitter.

Due to these intrinsic qualities of the Mueller-Muller PD, it is widely adopted in long-range high-speed communication systems where the benefits of the lower sample rate become crucial, and significant ISI occurs due to frequency-dependent attenuation of the channel. Since the Mueller Muller PD relies on ISI, less equalization is needed in these systems. However, a high-speed ADC is required, usually leading to high power consumption. The principle of baud-rate PD can be extended to PAM-4 with more complex algorithms.

1.3. State of the Art

As the industry transitions towards PAM-4 wireline communication, more research is focused on enhancing receivers. The phase detector is a relatively new subject and there is no dominant method yet. This section discusses several state-of-the-art PAM-4 phase detectors, their primary advantages, and their limitations.

1.3.1. PAM-4 Linear PD Based CDR

In [5], a quarter-rate linear phase detector (QLPD) is implemented using a sample and hold technique. The detailed schematic of the proposed CDR can be seen in Fig.1.8. Since the CDR runs at a quarter rate of the baud rate, four data recovery paths are required, each using a different clock phase. Only two phase detection paths are used to reduce power consumption. The CDR creates a voltage proportional to the phase error and converts it to current using a V/I amplifier. A simple RC low pass filter is used to update the LC-VCO. The VCO is used in a second PLL to generate the multi-phase clock required for quarter rate operation. The data path employs a sample and hold technique with 3 comparators to slice the data and a decoder to create an MSB and LSB output.

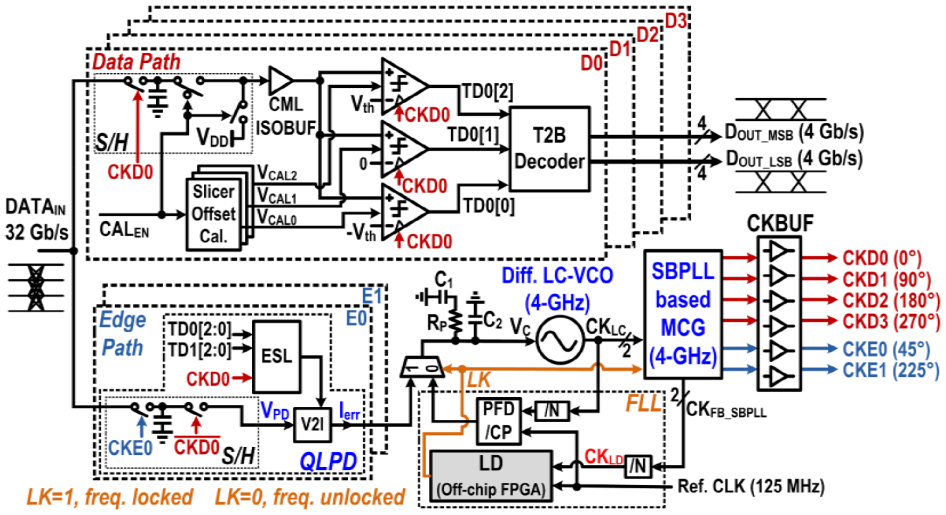


Figure 1.8: Block diagram of [5].

The phase detector schematic is presented in Fig.1.9. At $CKE0$, the differential data is sampled. Since only major transitions are considered, the ideal middle voltage is $0V$, a phase offset will result in a voltage offset from this ideal point. A minor transition from -1 to +1 also results in a middle voltage of $0V$, but due to the different slope, this transition would result in a different gain. Edge selection logic (ELS) uses the data paths to connect or disconnect the V/I stage. When a non-major transition occurs, no current should be provided to the loop filter since it will not provide correct information. Rising and falling transitions result in different polarities for the same phase offset. In order to compensate for this problem, the ESL switches the positive and negative input to the V/I stage between rising and falling transitions.

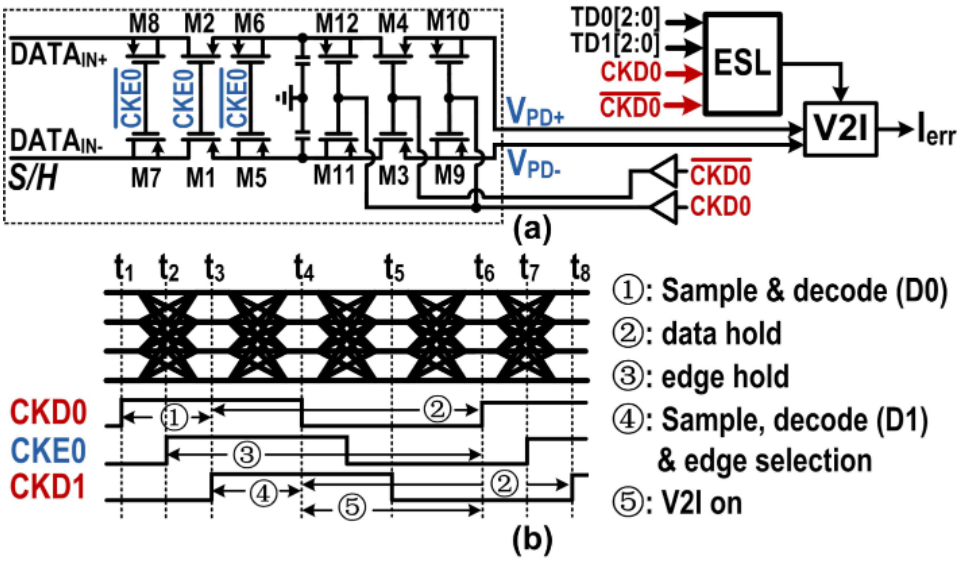


Figure 1.9: QRLPD schematic and waveforms [5].

The phase detection method shows a few issues. Although linear PD is good for avoiding the dithering jitter of bang-bang phase detection, due to the different slopes and middle voltages, it becomes impossible to consider all data transitions. Since only major transitions are considered and there are only two phase detectors, the update density is low at 0.0625. This results in a low jitter tolerance of $1UI$ at $2MHz$. Due to the linear phase detection, accompanied by the LC oscillator and second PLL loop for phase generation, the CDR has very low jitter generation of $352fs$ while only consuming $0.46pJ/bit$. The CDR is fabricated in $40nm$ technology.

1.3.2. PAM-4 Alexander PD Based CDR

In [6], a PAM-4 CDR is proposed, based on the bang-bang Alexander PD. The block diagram can be seen in Fig.1.10. Due to the high data rate, parallel PDs and data extraction units are used. Current mode logic is used to update a VCO operating at the baud rate. A latch-based clock divider is used to create the necessary clock phases for phase detection and data recovery.

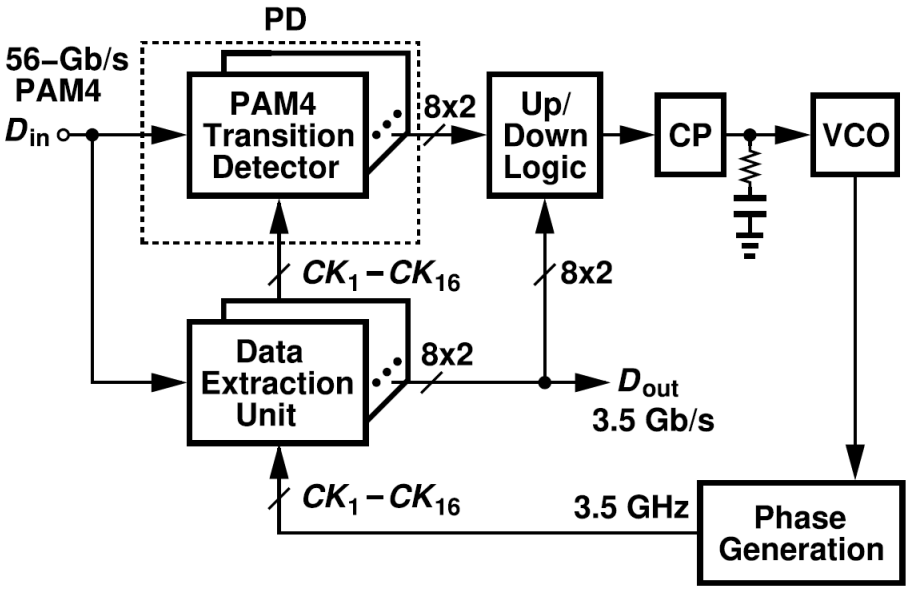


Figure 1.10: Block diagram for [6].

Each phase detector takes three consecutive samples: The leading bit, V_A, the lagging bit V_B, and the transition V_E. The sum $V_{sum} = V_A + V_B - 2V_E$ shows if the transition was closer to V_A or V_B and is calculated in the current domain with four V/I stages, one extra is needed for offset calibration. Again, a problem arises with the ambiguity of the polarity of V_{sum}. If a falling transition occurred, a leading clock results in a high V_{sum} but a rising transition results in a negative V_{sum}. To counter this problem, the direction, $V_B - V_A$, is calculated in a similar fashion. The output of the phase detectors cannot immediately control the charge pumps due to non-transitions where the VCO should not be updated. Three charge pumps are controlled by each phase detector and activated by the data extraction unit when a transition occurred. The phase detector considers all data transitions leading to a high jitter tolerance of $1UI$ at $10MHz$.

The CDR is fabricated in 28nm technology and has an incredibly low power consumption of $0.14pJ/bit$, while maintaining $574fs$ jitter generation. There are still a few downsides to this design, first the use of an LC oscillator lowers the jitter generation but increases the area, this is especially important since this design is meant for die-to-die communication. Since the phase detection method requires a lot of calculation, a high loop delay is introduced, lowering the phase margin. Finally, since each level transition activates a charge pump, a different amount of charge pumps are used for different transitions. Major transitions activate all three while minor transitions only activate one. This leads to a variable phase detector gain and makes the loop difficult to optimize.

1.3.3. Baud-Rate PAM-4 PD Based CDR

In [7], a pattern-based baud rate phase detector is used. A diagram of the complete CDR and PD can be seen in Fig.1.11. Four phase detectors are placed in parallel to allow for a quarter rate operation. The majority voter (MV) assesses the major status among the phase detectors. Using MV instead of individual PD outputs relaxes the speed at which the digital low pass filter (DLF) has to operate. The DCO receives updates with a proportional path directly from the MV. Digitally synthesized logic is employed for the integral path. Data recovery is facilitated using two comparators and reusing the phase detector comparators in combination with a time-based decoder, omitting the use of an extra

comparator.

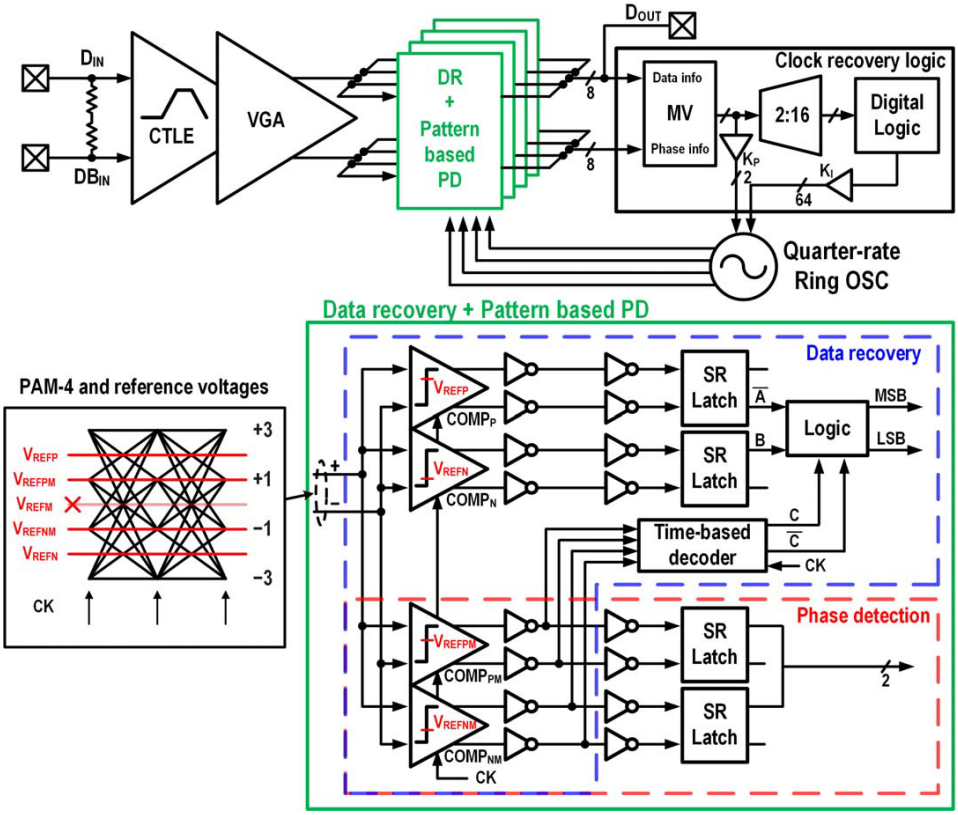
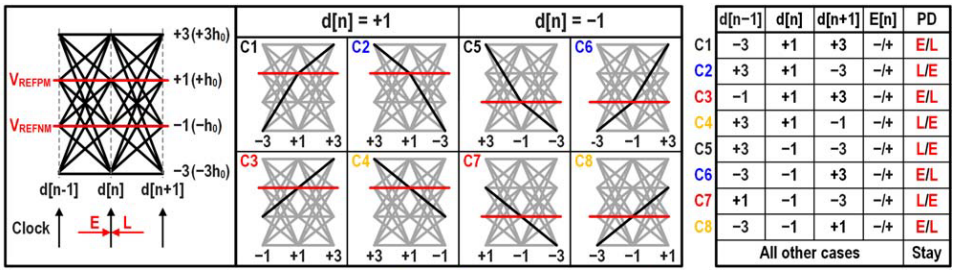


Figure 1.11: Block diagram for [7].

.

The phase detector uses three consecutive data points as a pattern. A transition diagram is shown in 1.12. Each phase detector has two comparators with reference points -1 and 1. When two consecutive data transitions occur with either -1 or 1 as the middle data point, it can be determined if the data falls early or late depending on the output of the comparators. Only an update density of 0.125% is achieved with this method and with the majority voter algorithm this further falls to 0.103%. When no equalization is present the locking points of transitions $C_{3,4,7,8}$ differ from $C_{1,2,5,6}$. Therefore the phase detector has two options, one where all eight transitions are considered and one for under-equalized signals where only $C_{1,2,5,6}$ are considered.



	d[n−1]	d[n]	d[n+1]	E[n]	PD
C1	−3	+1	+3	−/+	E/L
C2	+3	+1	−3	−/+	L/E
C3	−1	+1	+3	−/+	E/L
C4	+3	+1	−1	−/+	L/E
C5	+3	−1	−3	−/+	L/E
C6	−3	−1	+3	−/+	E/L
C7	+1	−1	−3	−/+	L/E
C8	−3	−1	+1	−/+	E/L
All other cases					Stay

Figure 1.12: Transition diagram for [7].

The CDR is fabricated in $28nm$ technology. The drawbacks and advantages of baud rate phase detectors become clear in this design. With a power consumption of $0.83pJ/bit$ including the continuous time linear equalization (CTLE), it is a very efficient design. However, the jitter performance is lacking with an integrated jitter from $1kHz$ to $100MHz$ of $430fs$, a smaller bandwidth than previously discussed papers. The jitter tolerance exceeds the mask for CEI-56G-VSR, which corresponds to a $1UI$ jitter

tolerance at $0.8MHz$, a relatively low standard.

1.4. Thesis Motivation

As discussed before, wireline PAM-4 transceivers are becoming more popular due to the higher spectral efficiency the modulation technique provides. Many ADC-based PAM-4 CDR's exist [8] [9] [10]. High-speed ADCs have a large power consumption, making these designs inefficient. Transceivers with phase detection in the analog domain [11], [12] [13] have provided an adequate solution for this. However, these designs do not consider all data transitions, which can improve both jitter generation and tolerance.

To better understand this a phase domain model of an analog CDR is presented in Fig.1.13. The output clock phase tracks the input data phase:

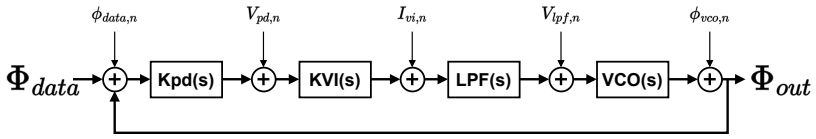


Figure 1.13: General phase domain model for CDR.

$$\Phi_{out} = H_{cl}\Phi_{in} \tag{1.1}$$

$$H_{cl} = \frac{Kpd(s) \cdot KVI(s) \cdot LPF(s) \cdot VCO(s)}{1 + Kpd(s) \cdot KVI(s) \cdot LPF(s) \cdot VCO(s)} \tag{1.2}$$

A better tracking of the input phase means more jitter tolerance is achieved. This can be done by increasing the bandwidth or tuning the gain of each individual stage, as done in [14] [15]. However, increasing the phase detector gain is most beneficial. Each stage provides an additional noise source. Due to the closed-loop operation of the CDR, each noise source will be suppressed by the gain of the preceding block. The output-referred noise of each source can be expressed as:

$$\phi^2_{out,data,n} = \phi^2_{data,n}|H_{cl}(s)|^2 \tag{1.3}$$

$$\phi^2_{out,pd,n} = \phi^2_{pd,n}|H_{cl}(s)|^2\left|\frac{1}{Kpd(s)}\right|^2 \tag{1.4}$$

$$\phi^2_{out,vi,n} = \phi^2_{vi,n}|H_{cl}(s)|^2\left|\frac{1}{Kpd(s) \cdot KVI(s)}\right|^2 \tag{1.5}$$

$$\phi^2_{out,lpf,n} = \phi^2_{lpf,n}|H_{cl}(s)|^2\left|\frac{1}{Kpd(s) \cdot KVI(s) \cdot LPF(s)}\right|^2 \tag{1.6}$$

$$\phi^2_{out,vco,n} = \phi^2_{vco,n}|H_{cl}(s)|^2\left|\frac{1}{Kpd(s) \cdot KVI(s) \cdot LPF(s) \cdot VCO(s)}\right|^2 \tag{1.7}$$

Looking at these equations, each noise source, except for the input noise, can be suppressed with a higher phase detector gain. The phase detector gain is linearly proportional to the transitions considered and transition density.

$$Kpd(s) \propto \alpha_T \tag{1.8}$$

Since the transition density of PAM-4 is 0.75 instead of 0.5 for NRZ, this provides an opportunity for improvement in jitter generation and jitter tolerance of the CDR.

The objective of this thesis is to design a PAM-4 CDR for wireline communication with low jitter generation and high jitter tolerance by considering all data transitions for phase detection. To have a practical

design, a high data rate, low power consumption, and low area should also be achieved. The aimed design specifications are presented in table 1.1.

Design Metric	Specification
Modulation	PAM-4
Data rate	24Gb/s
Jitter generation	<500fs
Jitter tolerance	1 UI at 30MHz
Power consumption	0.5pJ/bit
Oscillator Type	Ring oscillator

Table 1.1: Design requirements.

1.5. Thesis Structure

The thesis is structured into several chapters to describe the design and analysis of the PAM-4 CDR:

- **Chapter 1: Introduction** - This chapter introduces the thesis and discusses background information for PAM-4 CDR.
- **Chapter 2: System-Level Design** - A novel phase detection method is proposed and based on this a system-level design is presented.
- **Chapter 3: System-Level Analysis** - A linear method is proposed to analyze the system and choose optional design parameters.
- **Chapter 4: Circuit Design** - This chapter elaborates on the design of the RF and mixed-signal blocks in the CDR.
- **Chapter 5: Register Calibration Loop** - The calibration loop is discussed and analyzed.
- **Chapter 6: Simulation Results** - The simulation results are discussed and compared to the expected performance.
- **Chapter 7: Conclusion** - This chapter concludes the thesis and provides suggestions for improvement and future work.

2

System-Level Design

This chapter delves into the system-level design and expands on the architecture of the PAM-4 CDR. First, the difficulties of PAM-4 phase detection will be discussed, and a novel phase detection method will be proposed. Subsequently, from the functionality requirements, a system-level design will be constructed for the whole CDR.

2.1. Phase Detection Method

The theoretical benefits of a PAM-4 considering all data transitions have been explained in 1.4. However, this poses a few problems in practice. The adoption of PAM-4 complicates the phase detector due to the quaternary nature of the input. Unlike NRZ data, where only transitions between maximum and minimum voltage are considered, PAM-4 modulation involves the use of voltages between these levels for data symbols. This leads to many more transitions, as illustrated in Fig. 2.1. The transitions differ in three fundamental aspects: different slopes, different directions and different middle voltages, making a universal approach to each transition impossible.

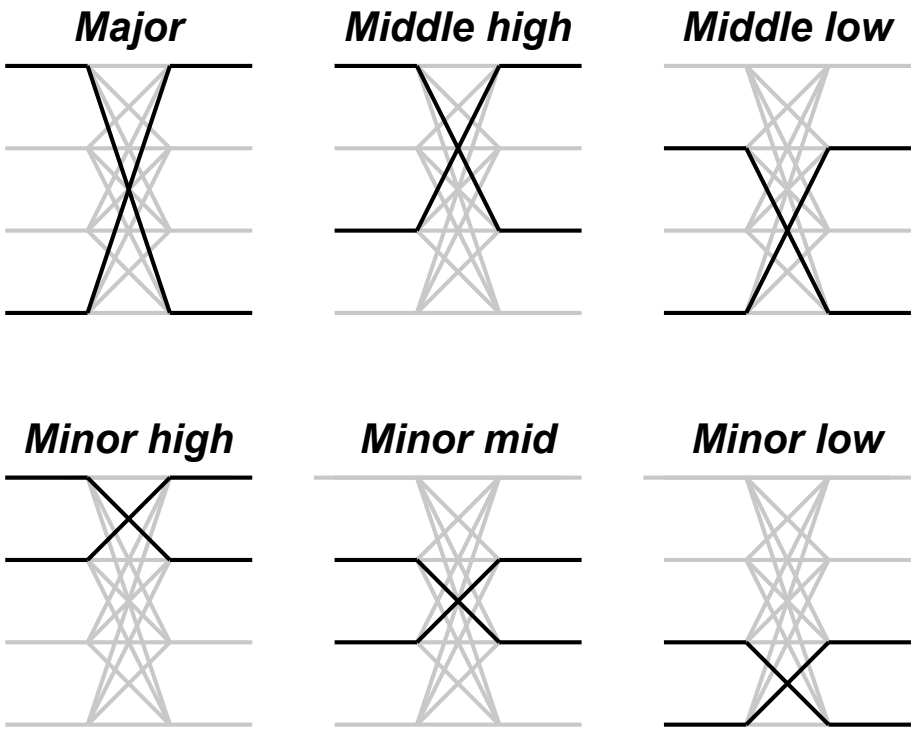


Figure 2.1: Different transition types in PAM-4 modulation.

Fig. 2.2 presents a graphical illustration of the proposed phase detection method. To utilize all data transitions for clock recovery, the PD will take three samples. First, a data sample (shown in red), next, a transition sample (shown in blue) and finally, a second data sample (again in red). With the two data samples, the expected middle voltage (shown in green) can be determined. This reference voltage is compared to the transition sample to determine if the phase is leading or lagging with respect to the data. This method ensures all twelve transitions can be considered for determining the phase error.

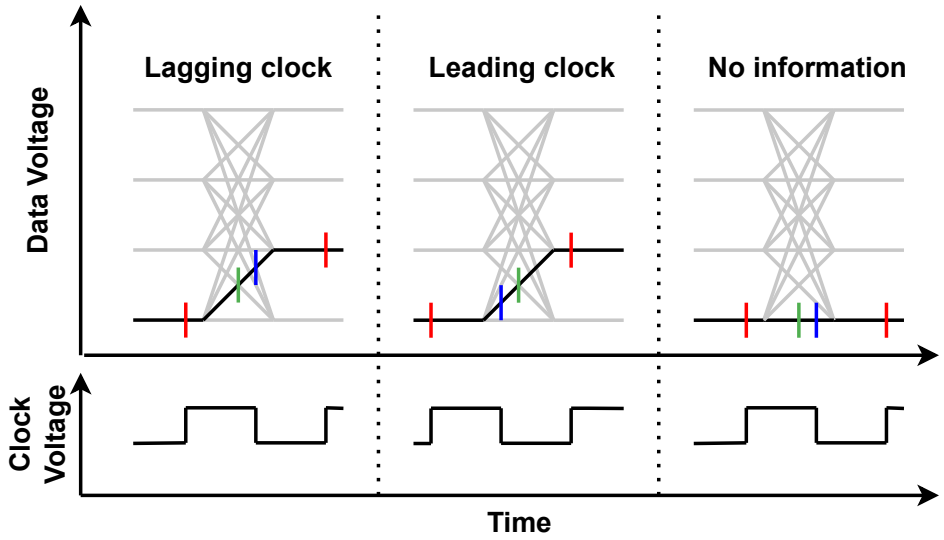


Figure 2.2: Graphical illustration of the phase detection method.

The comparison of the reference voltage and transition sample alone is insufficient for determining phase information. Due to the consideration of both rising and falling transitions, a leading clock can result in either a higher or a lower sample voltage. This can be seen in the leading and lagging clock examples in Fig. 2.2. In order to circumvent this problem, the sign of the comparison needs to be evaluated along with the direction of the transition. An additional consideration arises when two consecutive bits occur. this will result in no information on the phase of the data, and therefore the clock should not be updated.

The use of reference voltages introduces potential challenges. The attenuation of the channel is unknown, and variations in transition times can further complicate the determination of the middle voltage (in relation to time). This renders the exact middle voltage unknown. If predetermined reference voltages are used, the locking points of the transitions can differ. To address this, a calibration loop that can adjust the reference voltages to an optimal point is essential.

The phase detection method and challenges described above can be summarized into several circuit functions for the CDR, a flowchart is presented in Fig. 2.3.

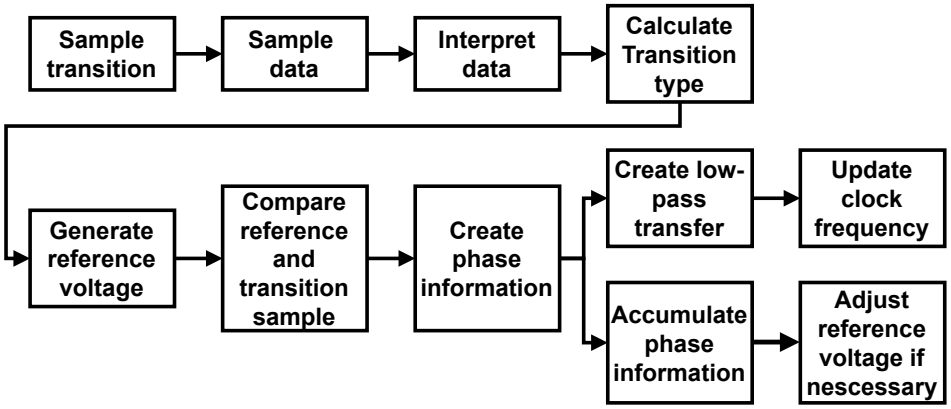


Figure 2.3: Flowchart of circuit functions

2.2. Block Diagram

The proposed top-level block diagram of the system can be seen in Fig. 2.4. In order to facilitate enough computation time, eight phase detectors operating at $\frac{1}{8}$ of the baud rate, operate in parallel. Given that data extraction is required for phase detection, the PD's also generate synchronized output data. The PD's provide tertiary information to the digital loop filter (DLF). To limit complexity, a bang-bang PD is employed by the system.

The DLF consists of a proportional path for phase error correction and an integral path for frequency errors [16]. The digitally controlled oscillator (DCO) generates 16 clock phases for the operation of the PDs. Finally, the calibration loop provides the reference voltages to each phase detector and receives phase information from one PD. Due to the low frequency of the calibration loop, phase information of only one PD is sufficient for calibration. Considering all eight PD outputs for calibration will result in unnecessary power consumption while not significantly improving performance. This block diagram ensures the effective operation of the PAM-4 CDR.

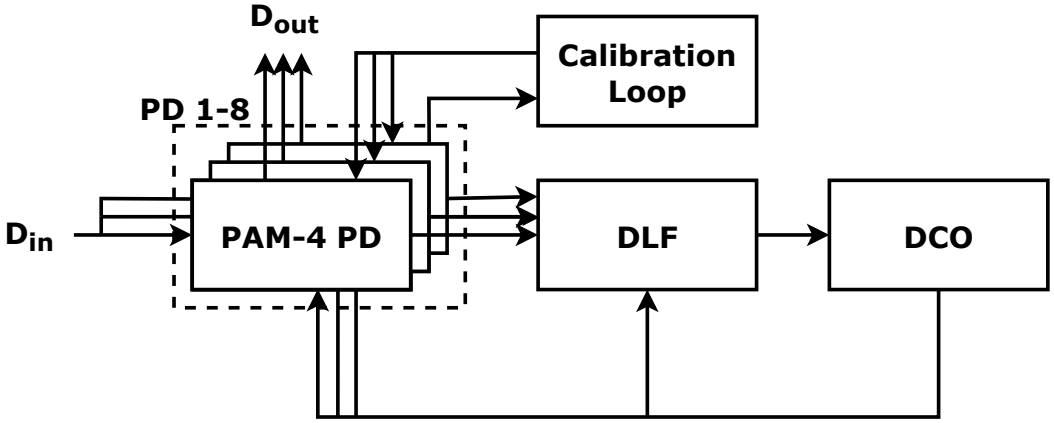


Figure 2.4: Block diagram of the PAM-4 CDR

2.3. PAM-4 Phase Detector

The PD is the most complex block. A block diagram can be seen in Fig. 2.5. The accommodating timeline of the calculation steps and data can be seen in Fig. 2.6

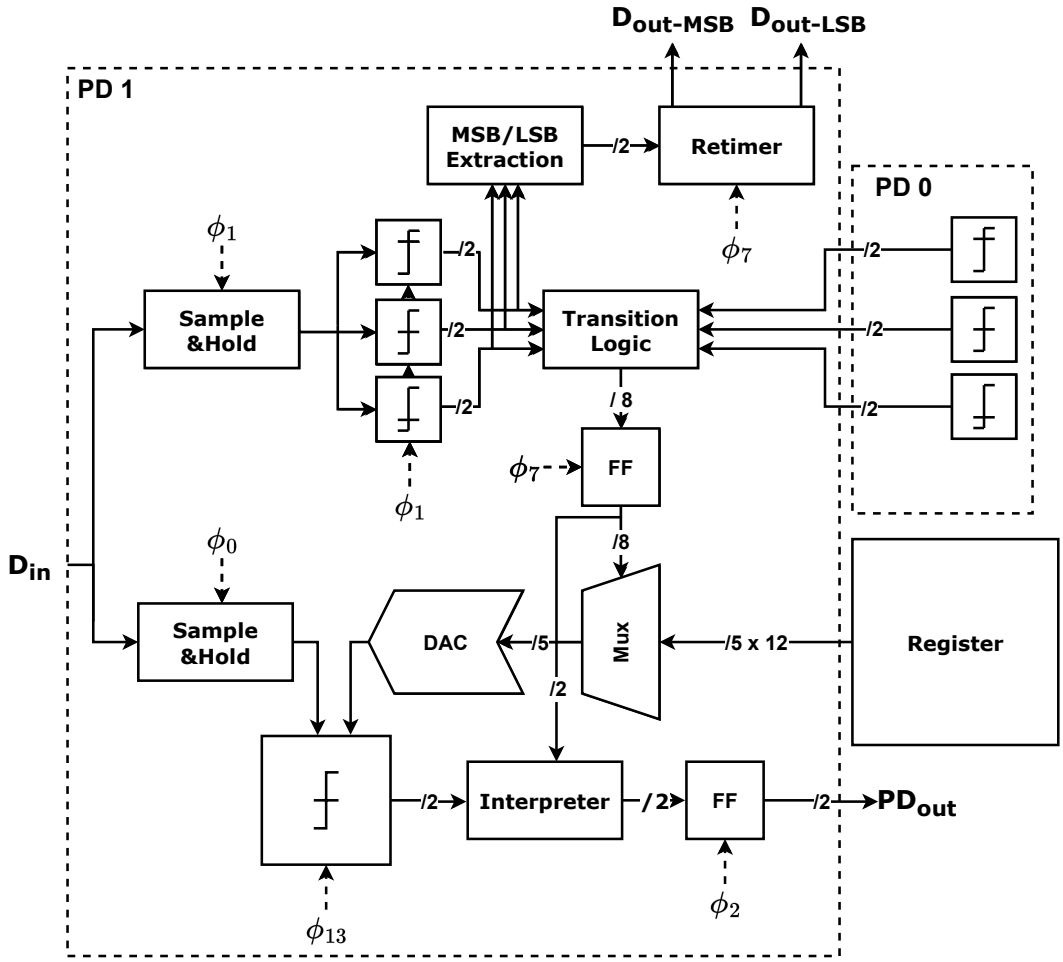


Figure 2.5: Block diagram of the PAM-4 Phase Detector

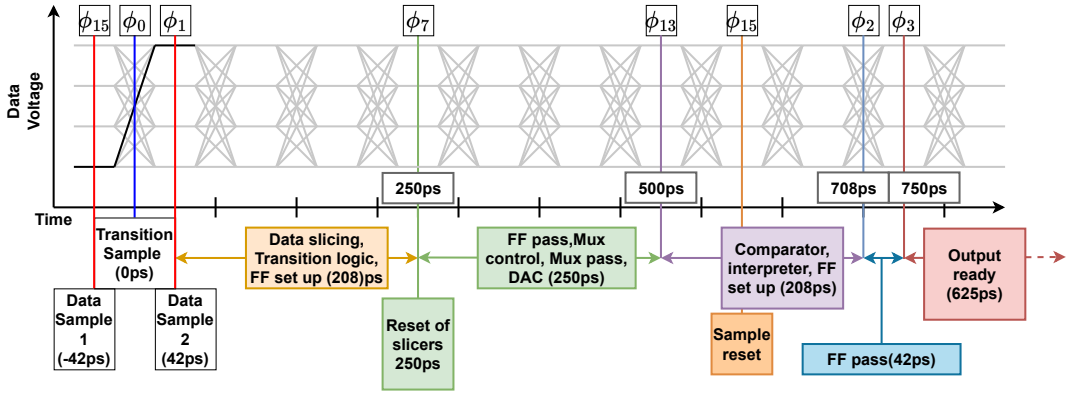


Figure 2.6: Timeline of calculation steps for $12GHz$ input data

The sample-and-hold circuits are designed to capture voltage samples of the incoming data at ϕ_0 and ϕ_1, corresponding to the transition and data, respectively. The timeline positions the transition sample at $0ps$ since this is the moment the phase information is measured.

The data sample is sliced by three comparators at ϕ_1 to determine the corresponding symbol. Each phase detector only takes one data sample, the leading data sample is sliced by the previous PD. The transition logic calculates which transition occurred with the two symbols. To maintain the validity of

the transition logic outputs after the slicers reset, flip-flops are used as buffers. This step takes $208ps$ and is ready at ϕ_7.

The flip-flops generate control signals for the multiplexer (MUX). This can be done directly with the flop-flop output due to the MUX design. A binary digital-to-analog converter (DAC) is used to create a reference voltage. The MUX selects the bits of the register corresponding to the correct transition and provides them to the DAC. The register is incorporated in the calibration loop outside of the phase detector. All PDs require the same information. This step takes $250ps$ and is ready at $500ps$ after the transition sample.

The generated reference voltage is compared to the transition sample using a comparator. However, as discussed previously, this comparison alone does not provide phase information. An interpreter is needed to combine the comparator information with the direction of the transition and to create tertiary phase information in the case of non-transitions. Finally, the phase information needs to be buffered to maintain its validity for a complete clock cycle. The last step takes $208ps$ and is done at ϕ_2, 9 symbols after the initial measurement. The information is passed to the DCO at ϕ_3 due to the delay of the flip-flop.

2.4. DLF and Oscillator

The digital low-pass filter needs to create a low-pass characteristic for the phase information provided by the phase detectors. The DLF consists of a proportional path and an integral path. The block diagram is presented in Fig. 2.7.

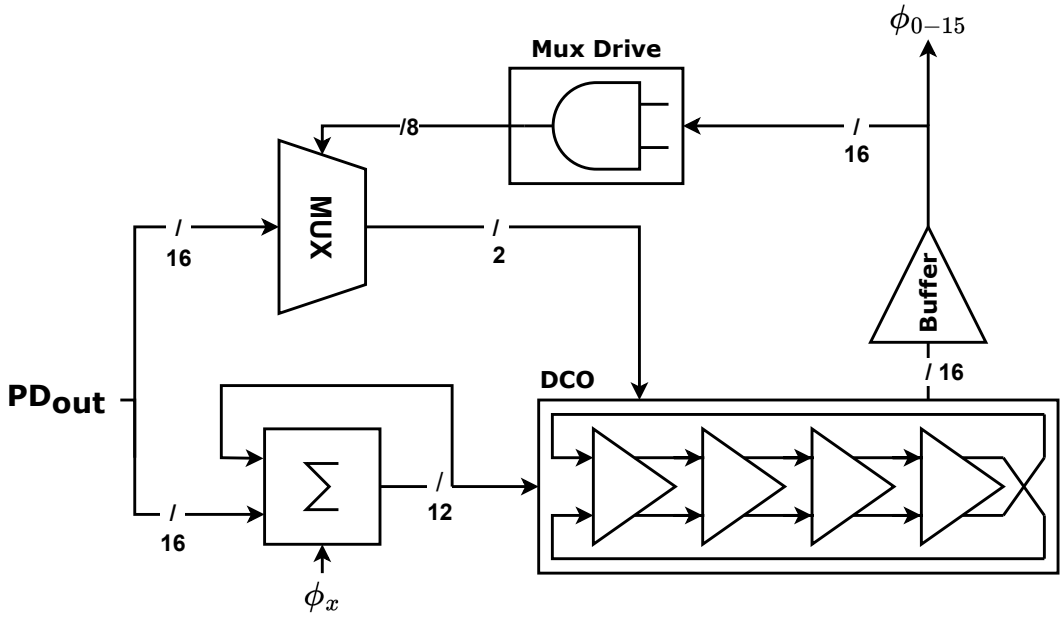


Figure 2.7: Block diagram of the DLF and oscillator

The proportional path of the filter must cycle through the outputs of the eight phase detectors. This needs to happen at the data frequency to maintain the instantaneous characteristic of the proportional path. Given that the oscillator operates at $\frac{1}{8}$ of the data frequency, the clock phases need to be combined to create a smaller duty cycle before they can serve as select signals for the MUX.

The integral path does not require a higher frequency as it has a slower response. During each clock cycle, the outputs of the eight phase detectors are summed and added to a counter that controls the integral path of the DCO. The number of bits in the integral path determines the locking range according to Eq. 2.1, with K_{DCO} representing the DCO gain, ρ the integral path gain and N the number of bits in the integral path. The 12 bits in the final design corresponds to a locking range of $50MHz$ in the

positive and negative direction.

$$\textit{Locking Range} = K_{DCO} \cdot \rho \cdot (2^N - 1) \tag{2.1}$$

The DCO generates 8 complementary clock phases, resulting in a total of 16 clock phases. All the clock cycles are necessary to provide the stages of the phase detector with the correct control signals. Additionally, a clock buffer is essential since not every clock phase will be loaded equally, this imbalance would affect the phase spacing which could lead to timing errors.

2.5. Reference Calibration Loop

The final block is the reference calibration loop, which is necessary for adjusting the reference voltages. This loop is needed to compensate for two phenomena. The first is unequal rise and fall times, as illustrated in Fig. 2.8. If the data experiences unequal rise and fall and the same predetermined references are used for rising and falling transitions, the locking points of rising and falling transitions will not be aligned. The locking point of the CDR will be the average of all locking points, assuming all transitions have an equal occurrence. However, since the locking point of each transition does not align with the locking point of the complete CDR, unnecessary jitter will be generated. Therefore, the reference voltages need calibration to address this issue and align all locking points.

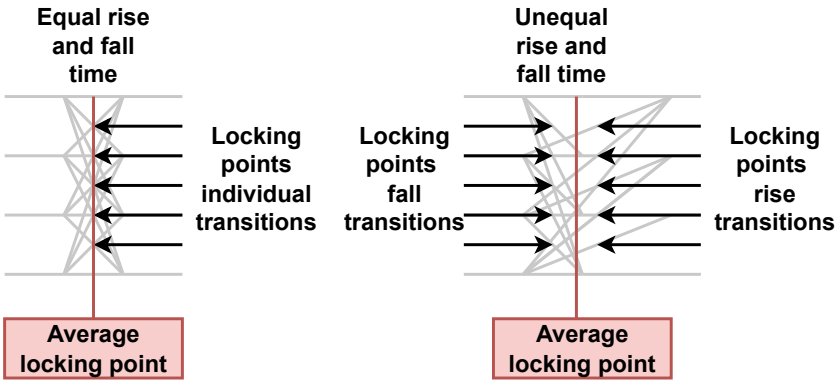


Figure 2.8: Effect of unequal rise and fall transition times on locking point.

The second phenomenon involves deterministic non-idealities in the sampling. Due to the unequal middle voltage of each transition, the overdrive voltage of the sampling switch will differ between transitions. This results in unique time constants, and thus, different delays. This leads to a positive offset for falling transitions and a negative offset for rising transitions. Non-deterministic non-idealities, like the comparator offsets or DAC Differential nonlinearity (DNL), cannot be compensated with this loop since these phenomena are different for each phase detector.

A block diagram of a single calibration loop is illustrated in Fig. 2.9. In total, twelve loops are required, one for each transition. This diagram depicts the functional aspects. The actual implementation of the loop will be designed using register-transfer level (RTL) code. The loop can operate at a very low frequency compared to the data frequency, an external clock CK_{ext} will provide this low-frequency signal. The output of one phase detector needs to be buffered to the lower clock frequency in order to maintain validity during the low-frequency clock.

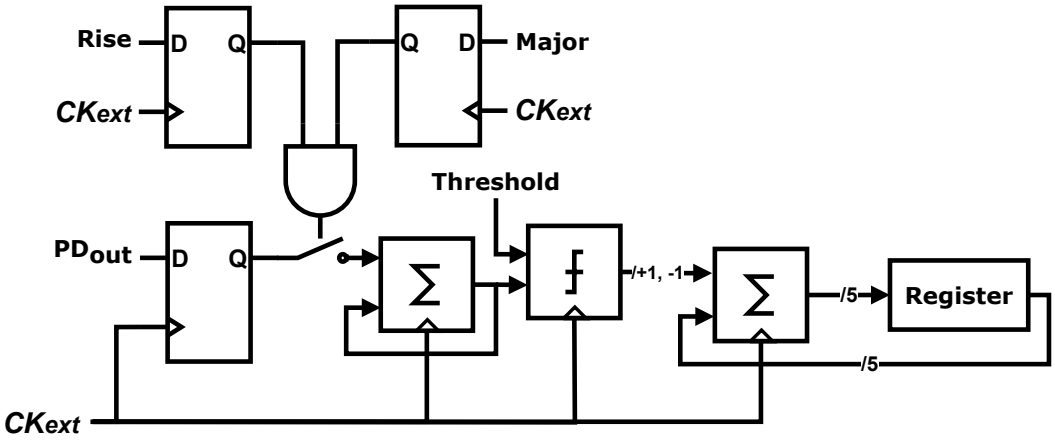


Figure 2.9: Calibration loop block diagram.

The buffered transition signals, PD_{out}, $Rise$ and $Major$ for the major rising transition, will activate the correct calibration loop. The phase information will be summed and accumulated each time the loop is activated. If the accumulated phase information reaches a threshold the register voltage needs to be adjusted. The accumulated phase information needs to be compared to both a positive and a negative threshold since the locking point of a transition can both be early or late with respect to the global locking point. The adjustment of the register can both be $+1$ or -1 depending on the direction of the transition and the phase information. The register value is adjusted with an accumulator. After the register is updated the accumulated phase information needs to be reset. The calibration loop is further explained in Chapter 5.

3

System-Level Analysis

In order to analyze the noise sources in the system and to determine the required oscillator gain, a phase domain model is required. First, a discrete-time domain model is presented. Subsequently, the noise contributions are introduced and the complete model is translated to the phase domain. The CDR is simulated in Matlab to derive the required system parameters, the expected performance and to verify the phase domain model.

3.1. Discrete Time Domain Model

The time discrete or Z-domain model of the proposed PAM-4 CDR is presented in Fig. 3.1.

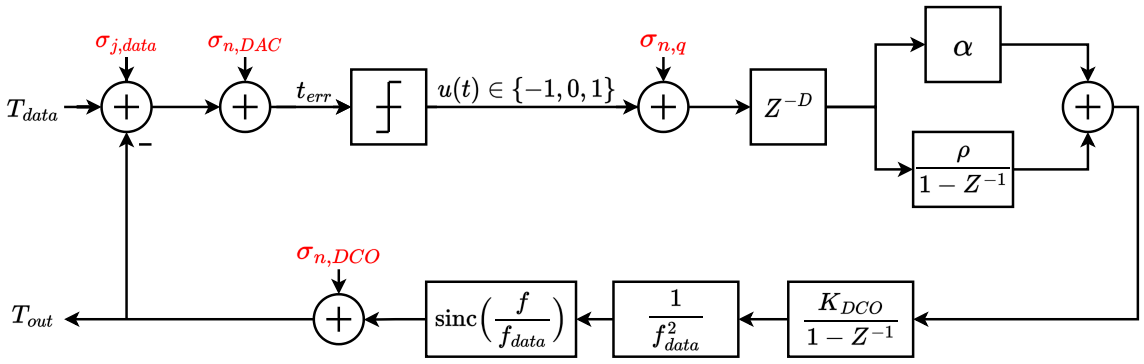


Figure 3.1: Discrete time domain model of the CDR.

The parameter of interest T_{data} denotes the midpoint of the data transition. The CDR attempts to align the sampling time T_{out} with T_{data} by minimizing t_{err}. The phase detector converts the incoming timing error into tertiary information: either ”early”, ”late”, or ”neutral” in the absence of a transition. A delay is also added to the system by the phase detector due to the calculation time required for the phase information. The information is then passed through a digital low-pass filter with proportional gain α and integral gain ρ, and the oscillator frequency is updated with gain $K_{DCO}\frac{Hz}{LSB}$. The DCO gain is multiplied by a factor $\frac{1}{f_{data}^2}$ in order to move to a time step update. A sinc function is needed to model the zero-order hold (ZOH) characteristic of the oscillator [17].

3.1.1. BBPD Linearization

The main obstacle in creating a linear model arises from the non-linear gain of the BBPD. The transfer function of the phase detector can be characterized by the sign function 3.1.

$$K_{pd}(t_{err}) = \begin{cases} -1 & \text{if } t_{err} < 0 \\ 0 & \text{if } t_{err} = 0 \\ 1 & \text{if } t_{err} > 0 \end{cases} \tag{3.1}$$

A linear approximation can be created by introducing a quantization error as discussed in [18]. Here, K_{pd} remains constant while a variable quantization error creates the properties of the sign function. This is illustrated in Fig. 3.2.

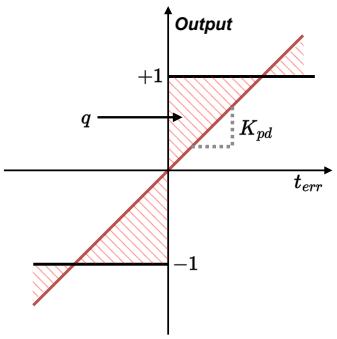


Figure 3.2: Linearization of the BBPD gain.

$$u(t) = K_{pd} t_{err} + q(t) \tag{3.2}$$

In the given expression $u(t)$ represents the output, K_{pd} denotes the linearized gain and $q(t)$ represents the quantization error. The BBPD gain can be linearized due to the effect of jitter. In the presence of a minor timing error between the CDR oscillator and the input data, jitter can result in a sample with incorrect information. A smaller timing error and higher jitter increase the likelihood of missampling and will thus result in a lower average output. This creates a linear gain behavior for small phase errors [19]. K_{pd} can be estimated by minimizing the quantization error. Following the analysis of [18], K_{pd} can be expressed by equation 3.3.

$$K_{pd} = \sqrt{\frac{2}{\pi}} \frac{\alpha_T}{\sigma_{t_{err}}} \tag{3.3}$$

Where α_T denotes the transition density and $\sigma_{t_{err}}$ represents the timing error between DCO rising clock edge and the middle of the input data transition. It is important to note that this analysis only remains valid when the timing error of the system stays within the linear gain region, about $2\sigma_{t_{err}}$. The CDR will still work with larger input errors, but its behaviour can not be linearized anymore. In order to determine $\sigma_{t_{err}}$, all the noise sources in the complete system and their transfer functions must be known.

Moving on to the next block, the DCO, the oscillator's gain is specified in frequency but can be transformed into a time adjustment for T_{out}.

$$f_{DCO} = f_{DCO,0} + K_{DCO} \cdot W \tag{3.4}$$

$$\frac{1}{T_{DCO}} = \frac{1}{T_{DCO,0}} + K_{DCO} \cdot W \tag{3.5}$$

$$\Delta T = T_{DCO} - T_{DCO,0} \tag{3.6}$$

$$\frac{-\Delta T}{T_{DCO} T_{DCO,0}} = K_{DCO} \cdot W \tag{3.7}$$

$$\tag{3.8}$$

Assuming $T_{DCO} \approx T_{DCO,0}$, which is the case for $K_{DCO} \cdot W \ll f_{DCO,0}$ with W representing the dimensionless control word.

$$\Delta T = \frac{K_{DCO}}{f_{dco}^2} \tag{3.9}$$

3.1.2. Noise Sources

In the system, four distinct noise sources exist: input noise, DAC quantization noise, phase detector quantization noise, and oscillator phase noise. Only the computation of quantization noise sources is needed. The input noise arises from the implementation and the oscillator noise results from the circuit design.

BBPD Quantization Noise

Recognizing that the power of the PD output $u(t)$ is equal to the transition density and using equation 3.3, it follows that the PD quantization noise power is only dependent on the transition density [18]. The output referred noise due to the quantization error, is of course dependent on more parameters.

$$\mathbb{E}[u^2] = \mathbb{E}[(K_{pd} t_{err}) + q)^2] \tag{3.10}$$

$$\sigma_u^2 = \frac{1}{N-1} \sum_{k=0}^{N-1} (u[k] - \mu)^2 = \alpha_T \tag{3.11}$$

$$\alpha_T = K_{pd}^2 \sigma_{t_{err}}^2 + \sigma_q^2 \tag{3.12}$$

$$\sigma_q^2 = \alpha_T - \frac{2}{\pi} \alpha_T^2 \tag{3.13}$$

DAC Quantization Noise

The calculation of DAC quantization noise is not straightforward. Fig. 3.3 depicts the consequences of quantized reference voltages. Instead of the ideal voltage points the nearest quantized voltage has to be used. This quantization error alters the locking point of an individual transitions. As the shift is unequal for the transitions, the locking points of individual transitions become misaligned. The time shift of locking point Δt can be described by equation 3.14.

$$\Delta t = t_{trans} \cdot \frac{V_q - V_{ideal}}{V_{trans}} \tag{3.14}$$

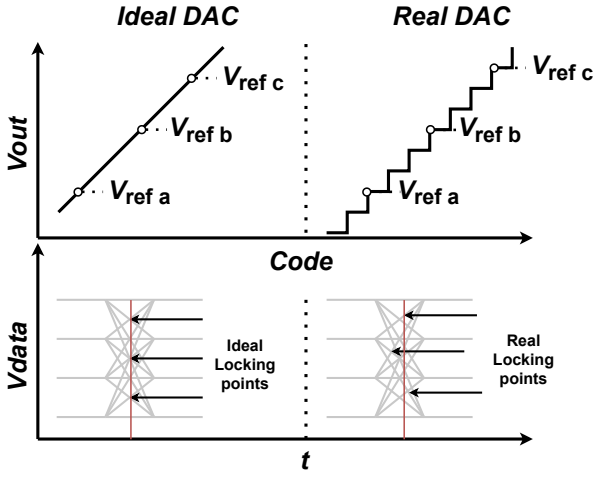


Figure 3.3: Illustration of the shift in locking point due to quantization noise

t_{trans} represents the transition time, V_q denotes the quantized reference voltage, V_{ideal} the ideal reference voltage and V_{trans} the voltage difference between data levels for a certain transition. This formula assumes linear transitions, if this is not the case the slope has to be used around the locking point.

A shift of the locking point in successive transitions is analogous to an instantaneous shift in the input data. The power of the quantization error is equal to the power of a discrete time signal containing the shift of each locking point.

$$\sigma_{q,DAC}^2 = \frac{1}{N-1}\sum_{k=0}^{N-1}(\Delta t[k]-\mu)^2 \tag{3.15}$$

Here, $\sigma_{q,DAC}^2$ denotes the power or variance of the quantization error, $\Delta t[k]$ is a vector containing the shift of each locking point. N is the number of transitions, and μ is the average locking point. The total number of transitions is 16 in PAM-4 since non-transitions also have to be considered. The shift of a non-transition is zero since no locking point exists.

Alternatively, one can assume that the ideal reference codes are randomly positioned. The quantization noise can now be approximated similar to calculating the DAC quantization noise in [20]. This approximation gives more insight into the effects of design parameters. Since only twelve reference codes exist, the quantized references can fall on unfavorable positions and have a higher quantization error. Eq. 3.18 can either over or under estimate the actual quantization noise due to this uncertainty. V_{DAC} represents the full DAC swing, N the number of DAC bits, t_{trans} the transition time and V_{swing} the amplitude of the input signal.

$$\sigma_{q,DAC}^2 = \frac{1}{A_{LSB}}\int_{\epsilon=\frac{A_{lsb}}{2}}^{\epsilon=\frac{-A_{lsb}}{2}} A_{error}^2(\epsilon)\,d\epsilon \tag{3.16}$$

$$\begin{aligned}\sigma_{q,DAC}^2 =& \frac{2}{16}\frac{1}{A_{LSB}}\int_{\epsilon=\frac{A_{lsb}}{2}}^{\epsilon=\frac{-A_{lsb}}{2}} \epsilon^2\cdot(\frac{t_{trans}}{V_{swing}})^2\,d\epsilon + \\ & \frac{4}{16}\frac{1}{A_{LSB}}\int_{\epsilon=\frac{A_{lsb}}{2}}^{\epsilon=\frac{-A_{lsb}}{2}} \epsilon^2\cdot(\frac{t_{trans}}{\frac{2}{3}V_{swing}})^2\,d\epsilon + \\ & \frac{6}{16}\frac{1}{A_{LSB}}\int_{\epsilon=\frac{A_{lsb}}{2}}^{\epsilon=\frac{-A_{lsb}}{2}} \epsilon^2\cdot(\frac{t_{trans}}{\frac{1}{3}V_{swing}})^2\,d\epsilon\end{aligned} \tag{3.17}$$

$$\sigma_{q,DAC}^2 = \frac{65}{16}\left(\frac{V_{DAC}}{(2^N-1)\sqrt{12}}\right)^2\left(\frac{t_{trans}}{V_{swing}}\right)^2 \tag{3.18}$$

3.2. Phase Domain Model

The translation from the Z-domain to the S-domain can be accomplished using the following formula:

$$Z = e^{j\omega T_{data}} \tag{3.19}$$

$$\text{For } f << f_{data} \tag{3.20}$$

$$Z = 1 + \frac{s}{f_{data}} \tag{3.21}$$

$$Z^{-1} = 1 - \frac{s}{f_{data}} \tag{3.22}$$

The open loop transfer function of the system is now given by:

$$H_{ol} = K_{pd}\cdot e^{-sT_{data}\cdot D}\cdot\left(\alpha+\frac{\rho f_{data}}{s}\right)\cdot\left(\frac{K_{DCO}}{sT_{data}}\cdot\frac{1}{f_{data}^2}\right)\cdot\text{sinc}\left(\frac{f}{f_{data}}\right) \tag{3.23}$$

and in the closed-loop configuration:

$$H_{cl} = \frac{H_{ol}}{1+H_{ol}} \tag{3.24}$$

The sinc function is neglected since the approximation does not hold for larger frequencies where the sinc function becomes noticeable.

$\sigma_{t_{err}}$ can now be calculated using the phase noise profiles and the loop dynamics. It is important to note that the incoming data jitter, DAC quantization noise and quantization noise have white spectra.

$$\mathcal{L}_{tot}(s) = \mathcal{L}_{data}(s) + \mathcal{L}_{q,DAC}(s) + \mathcal{L}_{q}(s) + \mathcal{L}_{DCO}(s) \tag{3.25}$$

$$\mathcal{L}_{tot}(s) = H_{cl}(s)\cdot(\sigma^2_{data}+\sigma^2_{q,DAC})\cdot\frac{(2\pi\cdot f_{data})^2}{\frac{1}{2}f_{data}} \;+\; \frac{H_{cl}(s)}{K_{pd}}\cdot\sigma^2_q\cdot\frac{(2\pi\cdot f_{data})^2}{\frac{1}{2}f_{data}} \;+\; \frac{1}{1+H_{ol}(s)}\cdot S_{\phi,DCO}(s) \tag{3.26}$$

$$\begin{aligned}\sigma^2_{t_{err}} = &\int_{0.5}^{-0.5} H_{cl}(\omega)\cdot(\sigma^2_{data}+\sigma^2_{q,DAC})\cdot\frac{(2\pi\cdot f_{data})^2}{\frac{1}{2}f_{data}}\,d\omega \;+\\ &\int_{0.5}^{-0.5} \frac{H_{cl}(\omega)}{K_{pd}}\cdot\left(\alpha_T-\frac{2}{\pi}\alpha_T^2\right)\cdot\frac{(2\pi\cdot f_{data})^2}{\frac{1}{2}f_{data}}\,d\omega \;+\\ &\int_{0.5}^{-0.5} \frac{1}{1+H_{ol}(\omega)}\cdot S_{\phi,DCO}(\omega)\,d\omega\end{aligned} \tag{3.27}$$

With the equations 3.3 and 3.27 an analytical solution can be derived given the phase noise profile of the DCO and the input noise are known. Alternative analytical approaches for the BBPD gain are presented in [21], [22] and [23]

3.2.1. Limit Cycling

The linear system not only fails in the presence of large disturbances but also in scenarios with minimal noise. The absence of noise prevents the loop from converging to a stable point. Due to the delay and fixed frequency gain, the system toggles between phase states, resulting in spurs at specific frequencies in the phase spectrum. This phenomenon is known as limit cycling [24], [25]. While ring oscillators are less prone to this issue due to the random input data, it remains important to be mindful of this aspect. The minimum noise requirement can be expressed as:

$$\sigma_{t_{err}} \geq \frac{\alpha K_{DCO}(D+1)}{\alpha_T f_{dco}^2\sqrt{3}} \tag{3.28}$$

3.3. MATLAB Time Domain Model

A time domain simulation of the system was created in Matlab, similar to the approach in [26], and compared to the linearized phase domain model. Matlab code can be found in appendix A. The results are presented in Fig. 3.4. It can be seen that the quantization noise is slightly underestimated. This can be attributed to the timing error occasionally falling outside the linear region.

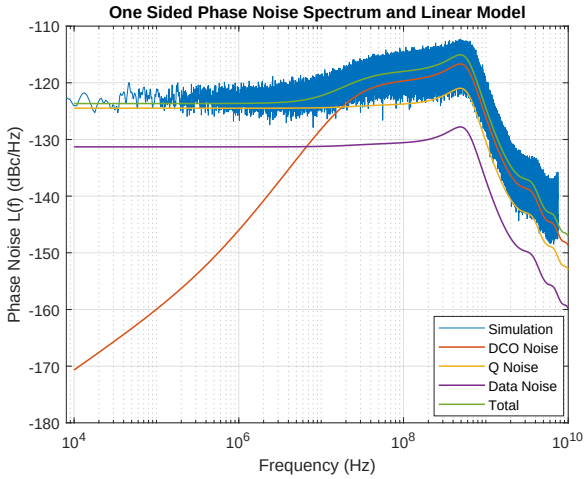


Figure 3.4: Comparison of linear model and time domain simulation

The circuit implementation of the phase detector introduces a few critical non-idealities to the system. These can have a significant impact on the system and should therefore be added to the Matlab model. The modelled non-idealities are:

- **Feed-through:** When the sample switch is closed, some feed-through still occurs, attenuating the sample.
- **Comparator settling**: If the difference between the sample and the reference voltage is small, the comparator cannot settle in the given time, leading to no phase information.
- **Low bandwidth integral path**: The integral path is not updated at the data rate but at the DCO frequency.
- **Deviations from the ideal oscillator gain**: The final DCO will have non-ideal gain due to mismatch and compression.

These effects cause the timing error to deviate from the Gaussian distribution assumed in [18]. The PD gain can be calculated with the simulated decisions and timing error.

$$K_{pd} = \frac{\mathbb{E}[u(t) \cdot t_{err}]}{\mathbb{E}[t_{err}^2]} \tag{3.29}$$

And for the quantization error:

$$q[k] = u[k] - K_{pd} \cdot t_{err}[k] \tag{3.30}$$

$$\sigma_q^2 = \frac{1}{N-1} \sum_{k=0}^{N-1} (q[k] - \mu)^2 \tag{3.31}$$

This way, the noise contributions of the quantization and oscillator can be compared, and an optimal K_{dco}, ρ, and N can be chosen.

3.4. Parameter Design

The system has three tunable parameters: the number of DAC bits N, the oscillator gain K_{dco}, and the integral path gain ρ with respect to α. Here, α is set to one.

3.4.1. Number of Bits

The DAC should be designed with a minimal number of bits to reduce power consumption. However, choosing too few bits will result in additional phase noise. The power of the quantization noise can be calculated using equations 3.14 and 3.15, as discussed previously. The noise is shaped by the closed-loop transfer function. Furthermore, the DAC noise reduces the PD gain, increasing the contribution

of other noise sources. The additional phase noise due to the DAC thus consists of both its direct and indirect contributions, making it difficult to calculate its overall impact analytically. In Fig. 3.5, the total phase noise and the direct DAC quantization phase noise are plotted for $N = 4, 5, 6$ for a worst-case scenario of a maximum transition time of the incoming data. $N = 5$ is chosen since it adds only $40fs$ of jitter while significantly reducing power consumption. In practice, the added jitter will most likely be less due to the sinusoidal transition waveform having a steeper slope around the locking point.

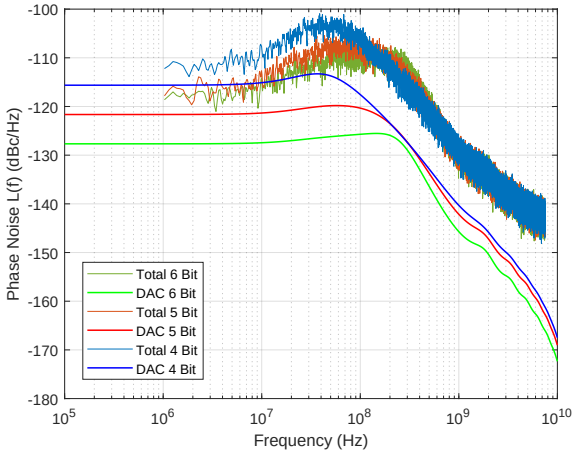
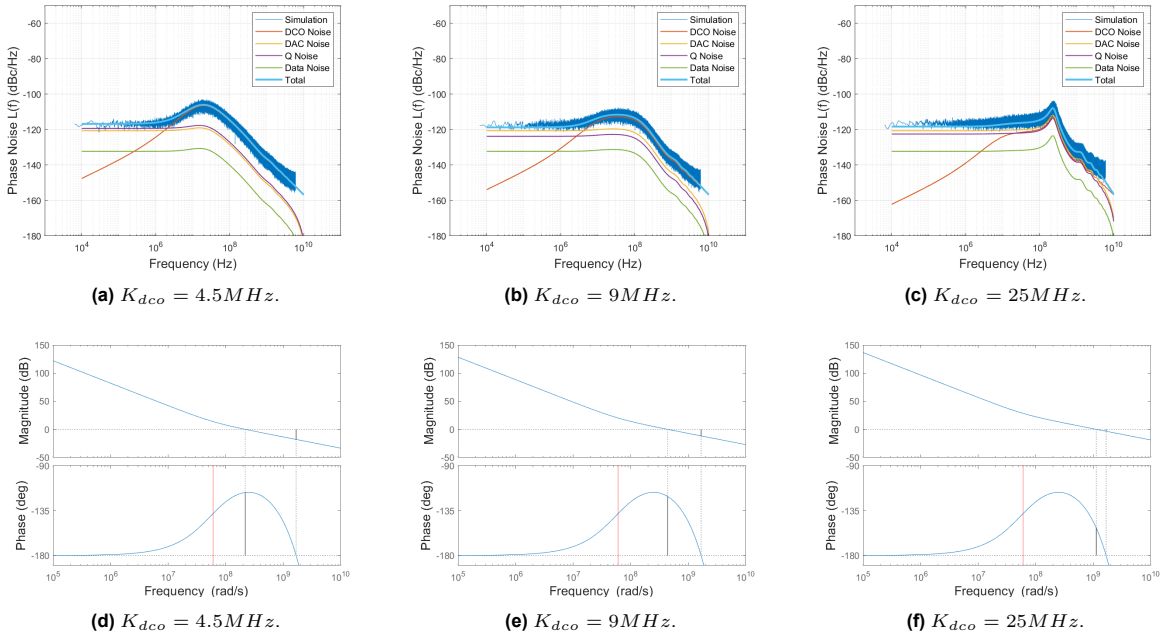


Figure 3.5: DAC quantization noise contribution

3.4.2. Optimal Parameter Choice for Jitter

The only parameters that can be tuned without changing the circuit architecture are K_{dco} and ρ. In Fig. 3.6, the phase noise and open loop transfer functions are plotted for three different oscillator gains: $4.5MHz$, $9MHz$, and $25MHz$. If the gain is too low, i.e., $4.5MHz$, the oscillator thermal noise is not sufficiently suppressed, and therefore, jitter will be too high. If the gain is too large, the phase margin degrades, and limit cycling occurs. This also leads to an increase in jitter generation. The optimal K_{dco} is $9MHz$.



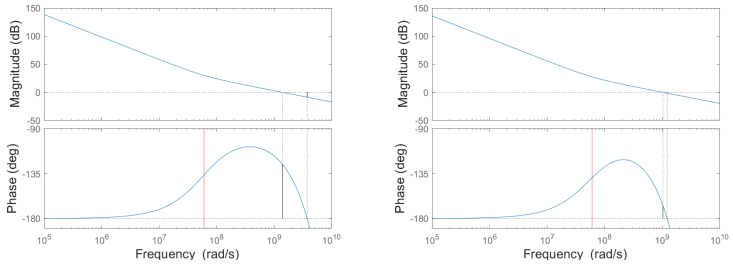
(a) $K_{dco} = 4.5MHz$. **(b)** $K_{dco} = 9MHz$. **(c)** $K_{dco} = 25MHz$.

(d) $K_{dco} = 4.5MHz$. **(e)** $K_{dco} = 9MHz$. **(f)** $K_{dco} = 25MHz$.

Figure 3.6: One sided PN spectrum and Bode Plot for different K_{dco}.

The phase margin experiences degradation due to delay. This effect is illustrated in the two cases

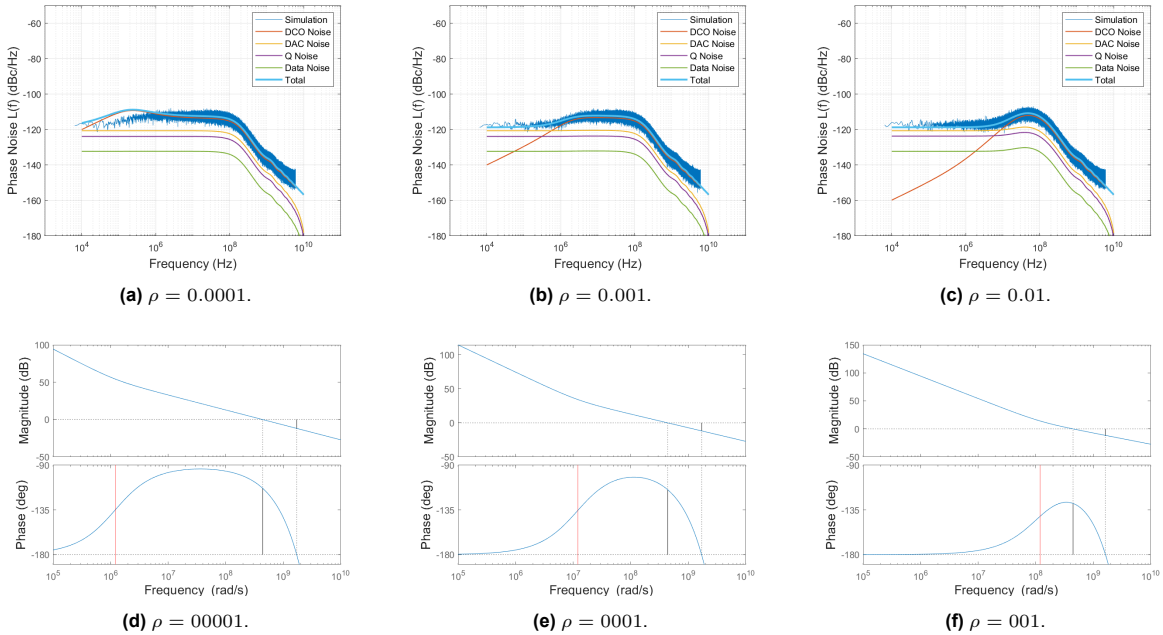
examined in Fig. 3.7. In Fig. 3.7a, a delay of 5 symbols is used, enabling a large open-loop gain and thus a large K_{dco}. In Fig. 3.7b, the delay is increased to 15 symbols while maintaining K_{dco}. It is clear that the phase margin has degraded significantly, resulting in a peak in the phase noise spectrum. Since a high K_{dco} is needed to sufficiently suppress the DCO noise, it is important that the delay is minimized in the system. If it cannot be minimized, it can be beneficial to increase the power consumption in the DCO, lowering the phase noise and thus the required K_{dco} for optimal suppression. In the final system, a delay of 10.5 bits is achieved.



(a) $Delay = 5symb.$ **(b)** $Delay = 15symb.$

Figure 3.7: Phase margins of the CDR for different delays.

The second adjustable parameter is the integral gain, ρ. Fig. 3.8 presents the Bode plot and phase noise for three distinct values of ρ. As ρ increases, the zero in the DLF shifts to a higher frequency as illustrated in Fig. 3.8d - 3.8f by the red line, giving more prominence to the proportional path. If ρ is chosen too low, as shown in Fig. 3.8a, the DCO noise is not suppressed enough, and flicker noise will start to have an influence. Note that flicker noise is absent in the time-domain simulation but only present in the linear model. A lower ρ also reduces the locking range for a given number of bits in the integral path according to Eq. 2.1. If ρ is chosen too high, as shown in Fig. 3.8c, it will reduce the phase margin, which is clearly visible in the Bode plot in Fig. 3.8f. A final value of $\rho = 0.001$ is used. This leads to a minimal reduction in phase margin while suppressing the flicker noise and maintaining enough locking range.



(a) $\rho = 0.0001.$ **(b)** $\rho = 0.001.$ **(c)** $\rho = 0.01.$

(d) $\rho = 00001.$ **(e)** $\rho = 0001.$ **(f)** $\rho = 001.$

Figure 3.8: One sided PN spectrum and bode Plot for different ρ.

3.5. Jitter Tolerance

Jitter tolerance is the maximum allowed jitter amplitude at a specific frequency in order to maintain an acceptable bit error rate (BER). The linearized model is unusable for calculating jitter tolerance since cycle slipping needs to occur. The phase error is too large for the linear model to stay valid.

To ensure a reasonable BER, we aim to keep the phase error below h, measured in degree or unit intervals UI, h depends on the CDR's performance and incoming signal quality. A graphical illustration of h can be seen in Fig. 3.9, with t_{data} representing the bit period, t_{trans} the transition time and σ_{rj} the random jitter between the clock and the data. The clock can drift a certain amount h before a bit error occurs.

In an ideal case, h equals to π radians or $0.5UI$ (half the bit) since ideally, the clock is situated in the middle of a bit. For PAM-4 modulation, it's crucial to avoid sampling in transition regions to prevent errors, especially for major transitions quickly crossing to the next symbol level. Random jitter also needs to be considered. To achieve a BER of $10^{-}12$, $7\sigma_{rj}$ needs to be taken into account. h can be calculated in UI with equation 3.32.

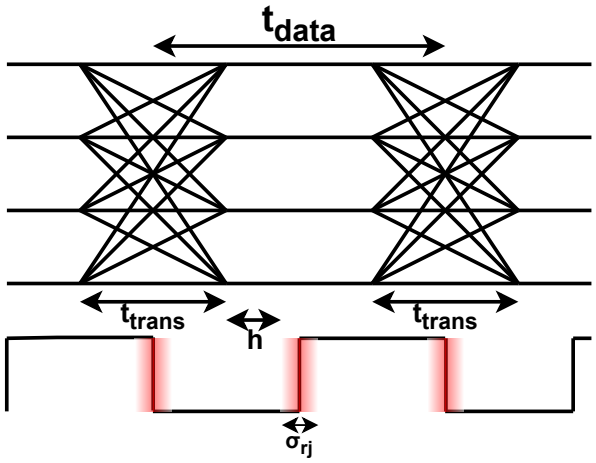


Figure 3.9: Graphical description of h.

$$h = \frac{t_{data} - \frac{1}{2}t_{rise} - \frac{1}{2}t_{fall}}{2t_{data}} - \frac{7\sigma_{rj}}{t_{data}} \tag{3.32}$$

3.5.1. Cycle Slipping

Since the linear model is invalid, another method is needed to find the point where $t_{error} > h$. There are three regions in the jitter transfer function. At high frequencies, the transfer is flat since the CDR cannot keep up with the input jitter. At medium frequencies, the proportional path is dominant and the transfer has a slope of $-20dB/dec$. At low frequencies, the proportional path is dominant and the transfer has a slope of $-40dB/dec$.

In the flat region, the jitter tolerance is equal to h since no jitter is transferred from input to output due to the high frequency.

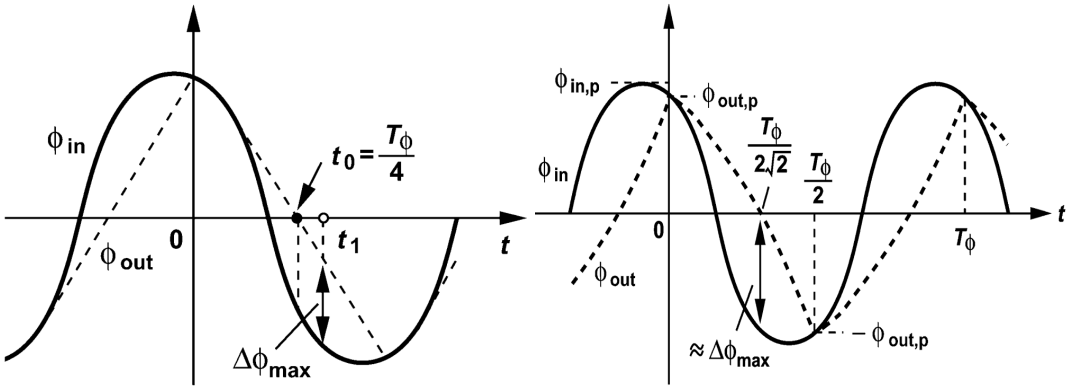
In the $-20dB$ region, the proportional path of the CDR can compensate for phase errors. To approximate the maximum tolerable jitter, let us examine the case presented in Fig. 3.10a. In this figure, the excess input and output phase are plotted when a sinusoidal jitter is applied to the input data. Since in this region the frequency is too high for the proportional path to have an effect, only the linear phase compensation of the proportional path is considered. The phase change of the input is too large for

the CDR to keep up. With each time step, a larger phase error between input and output is created. The maximum phase error $\Delta\phi_{max}$ occurs when the slopes of the input and output are equal. After this point, the output starts to catch up. Since this point is difficult to approximate, the zero crossing of the excess output phase is considered, here the phase error $\approx \phi_{max}$. The following derivation finds an approximate formula for the jitter transfer.

$$\begin{aligned}
\phi_{in}(t) &= \phi_{in,a}\cos(\omega t + \delta) \\
\phi_{out}(t) &= \phi_{out,a} - K_{DCO}\alpha_T \\
\Delta\phi_{max} &= \phi_{in,a}\cos(\omega\frac{T}{4} + \delta) \\
\delta &= \arccos\Big(\frac{\pi^2 K_{DCO}\alpha_T}{\phi_{in,a}\omega_{in}}\Big)
\end{aligned}$$

Equating $\Delta\phi_{max}$ to h and solving for $\phi_{in,a}$ gives:

$$|G_{JT}| = \frac{\sqrt{\pi^4(\alpha_T K_{DCO})^2 + h^2\omega_{in}^2}}{\omega_{in}} \tag{3.33}$$



(a) Dominant proportional path. **(b)** Dominant integral path.

Figure 3.10: In and output phase error [19].

The low-frequency region is the The $-40dB/dec$ region, where the integral path of the system is dominant. This region can be approximated with a similar method for a quadratic output phase instead of a linear one. A graphical illustration is given in Fig. 3.10b. Now the approximate maximal phase error occurs at $t = \frac{T}{2\sqrt{2}}$.

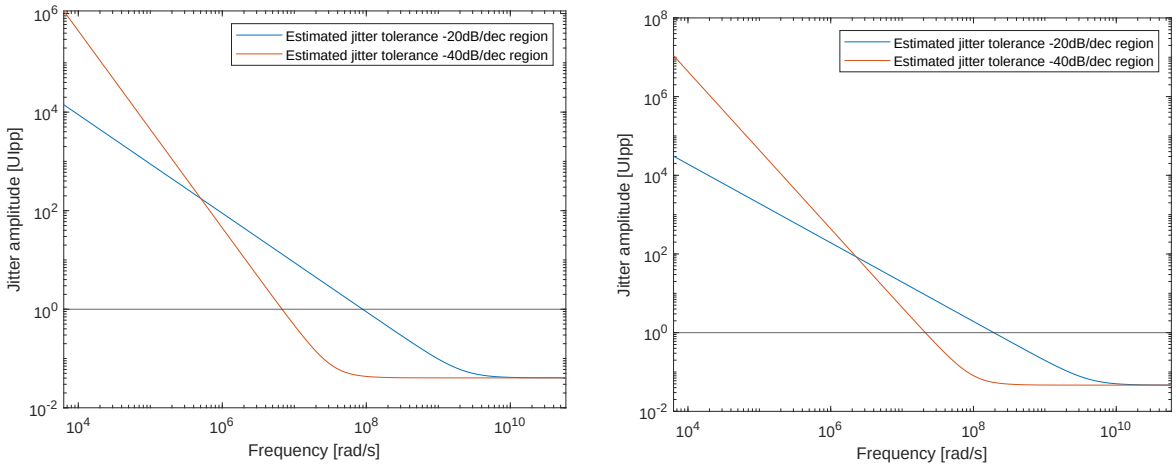
$$\begin{aligned}
\phi_{out}(t) &= \phi_{out,a} - \int \alpha_T K_{DCO}\rho f_{data} t dt \\
\phi_{out}\Big(\frac{T}{2}\Big) &= -\phi_{out,a} = \phi_{out,a} - \frac{1}{2}\alpha_T K_{DCO}\rho f_{data}\frac{T^2}{4} \\
\Delta\phi_{max} &= \phi_{in,a}\cos(\omega_{in}\frac{T}{2\sqrt{2}} + \delta) \\
\delta &= \arccos\Big(\frac{\pi^2\alpha_T K_{DCO}\rho f_{data}}{4\phi_{in,a}\omega_{in}^2}\Big)
\end{aligned}$$

Equating $\Delta\phi_{max}$ to h and solving for $\phi_{in,a}$ gives:

$$|G_{JT}| \approx \frac{\sqrt{0.06c^2 - 0.31ch\omega_{in}^2 + h^2\omega_{in}^4}}{\omega_{in}^2} \tag{3.34}$$

$$c = \alpha_T K_{DCO}\rho f_{data}\pi^2 \tag{3.35}$$

The result can be seen in Fig. 3.11a for $K_{dco} = 9MHz$. With a jitter tolerance of $1UI$ at $14MHz$, the jitter tolerance is lower than the desired specification of $30MHz$. However, the jitter tolerance can be easily increased with a higher K_{dco} due to the high transition density. In high-jitter environments, where the jitter tolerance is more important, this does not lead to an increase in jitter generation due to the reduced K_{pd}. If $K_{dco} = 19.5MHz$, the jitter tolerance increases to $1UI$ at $31MHz$ as can be seen in Fig. 3.11b. It is therefore proposed that the oscillator can switch between two frequency gains.



(a) JGEN optimised path with $K_{DCO} = 9MHz$.

(b) JTOL optimised path with $K_{DCO} = 19.5MHz$.

Figure 3.11: Estimated jitter tolerance.

4

Circuit Design

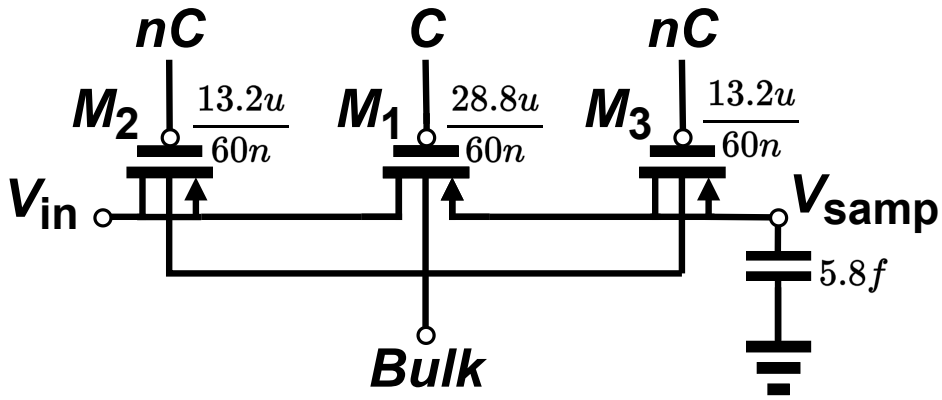
This chapter covers the circuit design at the transistor level of the phase detector, digital low-pass filter, and oscillator. It also discusses the layout of the phase detector and verifies the functionality of each block.

4.1. Phase Detector

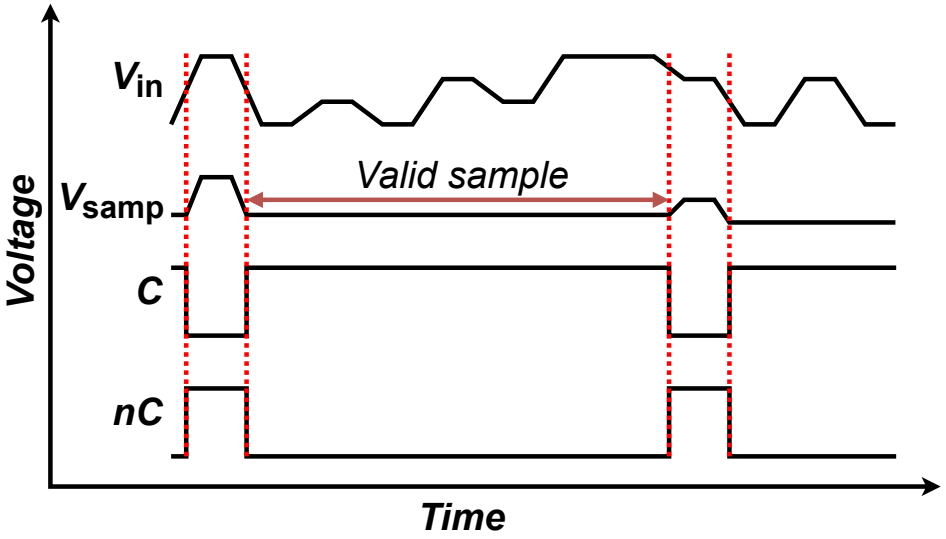
The phase detector consists of several analog and digital blocks, each requiring different design considerations. The primary design bottleneck of the phase detector is the delay. As seen in section 3.4.2, a lower delay in the phase detector allows for a larger oscillator gain, increasing the suppression of the dominant noise source, the DCO. Power consumption is also an important factor since, in the complete CDR, 8 parallel phase detectors exist. An accurate delay can only be found with a layout; therefore, the layout of the phase detector is needed to accurately illustrate the functionality of the CDR.

4.1.1. Sample and Hold

The sample and hold circuit is responsible for taking a sample of the incoming data and holding it before it is used by the comparator. The comparators require a input signal larger than the PMOS threshold voltage. Raising the signal after the sample and hold circuit would introduce an additional step and thus, additional delay. In order to circumvent this the incoming data ranges from $0.6V$ to $1.1V$. In future work, an level shifter is needed to raise the signal to this level. Although sampling a higher voltage signal is more difficult due to the increased resistance of PMOS transistors, the timing budget in the phase detector is more important. In order to be able to compensate for the increased resistance of the PMOS transistors, a bulk connection is used to increase the overdrive voltage. The sample and hold circuit consists of a large switch with dummy switches and a sampling capacitor. The schematic and its waveforms can be seen in Fig. 4.1. The sampling circuit is the same for both the data sample and the transition sample to ensure equal time constants. The sampling capacitor cannot be made too large due to the increased charging time, the bottom limit is set by the discharging of the capacitance due to leakage current and, most importantly, the kickback of the comparators.



(a) Schematic.



(b) Waveforms.

Figure 4.1: Sample and Hold Circuit.

The waveforms are presented in Fig.4.1b. The control signals C and nC are combined clock signals with a higher duty cycle for C (87.5%) and a lower duty cycle for nC (12.5%). This is done for two reasons: if direct clock signals with 50% duty cycle would be used, four sampling capacitors would be connected at all times, increasing the load on the level shifter. Now only one sampling capacitor is connected at all times, reducing the load. Additionally this also ensures the transition sample stays valid for a longer time increasing the calculation time budget. The transition sample needs to be compared to the generated reference voltage before the switch opens and a new sample is taken. With the lower duty cycle the calculation budget increases from 4 symbols, to 7 symbols.

$M1$ is the main sampling switch, its width is large to reduce on-resistance. Increasing it indefinitely will result in the parasitic capacitances becoming dominant over the sampling capacitance and thus will have no benefit.

Since $M1$ is relatively large, problems arise in the circuit. Charge injection occurs when switch $M1$ turns on. This can easily be mitigated by the dummy switches $M2$ and $M3$ having half the size of $M1$. The dummy switches turn on when switch $M1$ turns off, absorbing the injected charge.

Another problem is the drain-source capacitance C_{ds}. This will result in a capacitive division with the sample capacitor when the switch is off, and feed-through of the data will occur, as can be seen in Fig.4.2. Without shielding, the feed-through on the sample would reach 15.7% of the input, leading to large errors in the transition sample. Since most of C_{ds} comes from the metal connecting the doped

regions, metal shields can be placed over the gate with a strong connection to ground to insulate the drain and source metal connections [27]. The schematic layout can be seen in Fig.4.3. This lowered the feed-through to about 0.5% in simulations.

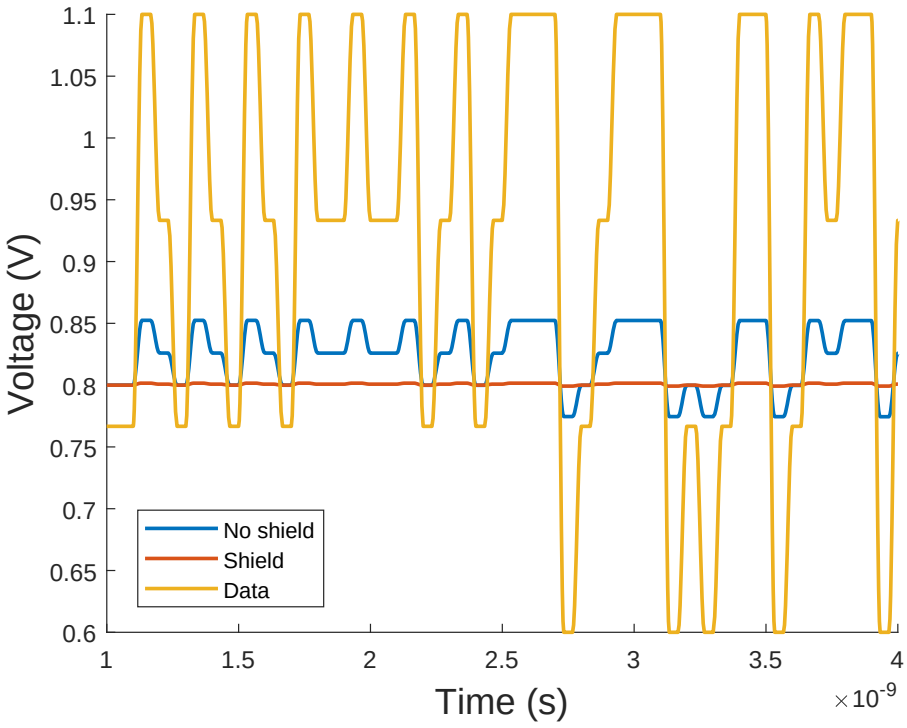


Figure 4.2: Sample voltage with and without shielding.

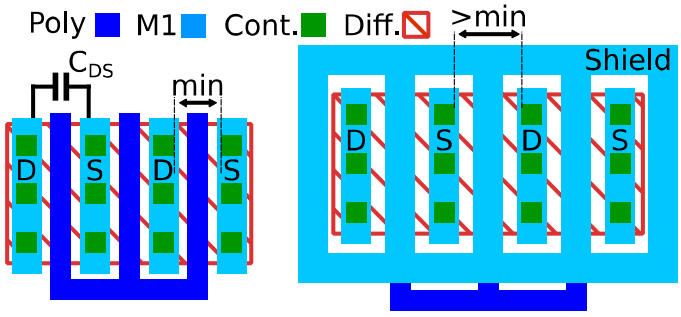


Figure 4.3: Schematic shield layout [27].

A transient simulation is depicted in Fig.4.4 for the two different major transitions. The control signals C and nC were created by NOR and NAND gates from the clock signals. It can be observed that V_{samp} tracks V_{in}, although slightly lagging, when C is low. Since V_{samp} lags V_{in} the falling transition results in a higher sample and a lower sample in the rising transitions. This shows that it is difficult to use predetermined voltages, however since the sample and hold circuits are the same for each phase detector, and thus the time constant, the calibration loop can adjust the reference voltages to its closest value. This is further explained in Chapter 5. It is also visible that charge gets injected and absorbed during the switching of C and nC. Finally, it can be seen that the feed-through is minimal when C is high. The component parameters can be seen in Fig. 4.1a

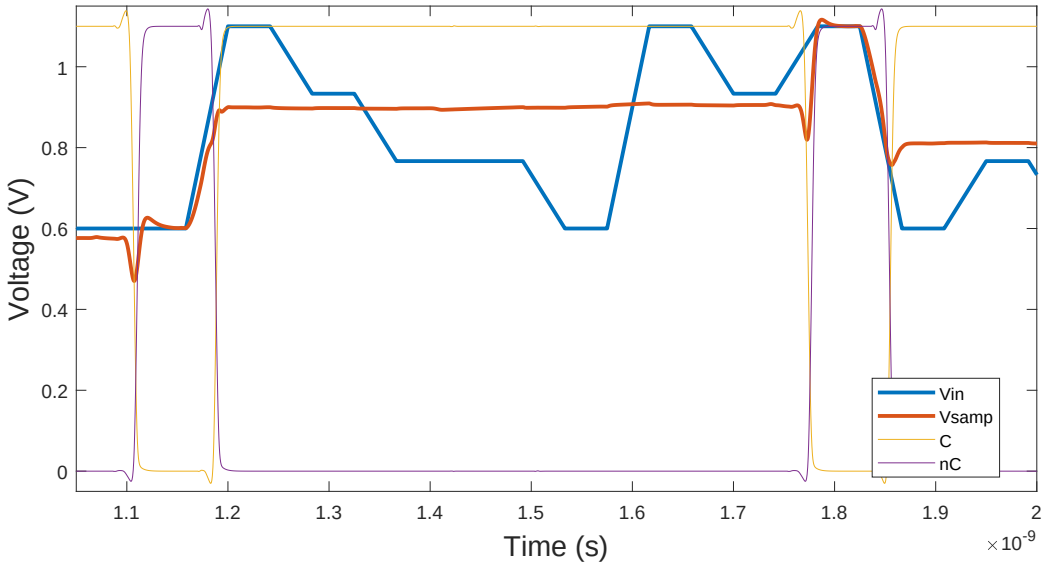


Figure 4.4: Transient S/H simulation

4.1.2. Comparator

In the phase detector, four comparators are needed: Three to slice the PAM-4 data and one to compare the transition sample. A StrongARM latch is used for all comparators, and the schematic is depicted in Fig.4.5. The working of a StrongARM latch is well-established and extensively discussed in [28].

The StrongARM latch comprises two complementary inverters, M_3 to M_6. The input transistors, M_1 and M_2, pull down nodes X and Y. Whichever node is pulled down faster creates the stronger inverter. The strongest inverter turns the weaker inverter off, and the circuit is latched in a stable state.

A few modifications have been introduced to enhance performance. Buffers are added to the output to address unequal loading and ensure sufficient drive to the next stage. Additionally, instead of resetting nodes X and Y to V_{DD}, they are reset to $V_{DD} - V_{th}$ and equalized with M_{10}. This adjustment reduces kickback to the input, which needs to be minimized to allow for a smaller sampling capacitor. This adjustment increases the reset time, which is not an issue in this design.

The size of transistors creates a trade-off between mismatch and speed. The comparator has two equal branches where the parasitic capacitances of all transistors need to be discharged. Increasing the size of M_{3-8} will result in more capacitance in each branch resulting in a slower settling of the comparator. However if the size is decreased the mismatch between the capacitances on each branch will be more resulting in more offset, if one branch has more parasitic capacitance the input voltage needs to be higher to achieve the same discharge time. The exception is the input pair $M_{1,2}$, where larger transistors result in both less mismatch and faster comparison since they supply the current required to drain the capacitance in each branch. However these transistors have an upper limit since they increase the kickback experiences on the input. Since the optimization focus of this design is primarily on speed, all transistors, except for the input pair, are chosen close ti minimal size.

An overview of transistor dimensions is provided in Fig. 4.5. Switches M_7 and M_8 are set to the minimum size, as there is enough time to ensure a full reset to V_{DD} for nodes W and Z. Transistors M_3 to M_6 are chosen close to the minimum size, as they increase capacitance at nodes $W - Z$ and consequently slow down settling. In contrast, transistors M_1 and M_2 are relatively large compared to M_3 and M_4, supplying the necessary current for discharging nodes X and Y. Increasing the size further would lead to higher kickback. Finally, transistor M_9 is chosen with twice the width of M_1 and M_2 to ensure enough current is available.

The comparator schematic is simulated with cadence virtuoso and the results are presented in Fig. 4.6 Settling of the comparator nodes in a transient simulation is presented in Fig.4.6a. The settling takes around $60ps$ for an input difference of $5mV$, less than $0.5LSB$. A mismatch Monte Carlo simulation can be seen in Fig.4.6b. The standard deviation of the offset is $19mV$. As 3σ should be taken into account, this would lead to misinterpreted data and phase information. The calibration loop of the reference voltages provides no solution since the mismatch in each comparator is different while the reference voltages are the same for each PD. A second foreground calibration loop is required in future work. This could be done by equalizing the capacitances on node W and Z, as proposed in [29]. Both inputs are connected to a equal common mode voltage and varactors are connected to node W and Z. Comparisons are of the equal input are accumulated and the capacitance on the faster branch increased. This is done until the comparator has an equal decision change with equal inputs.

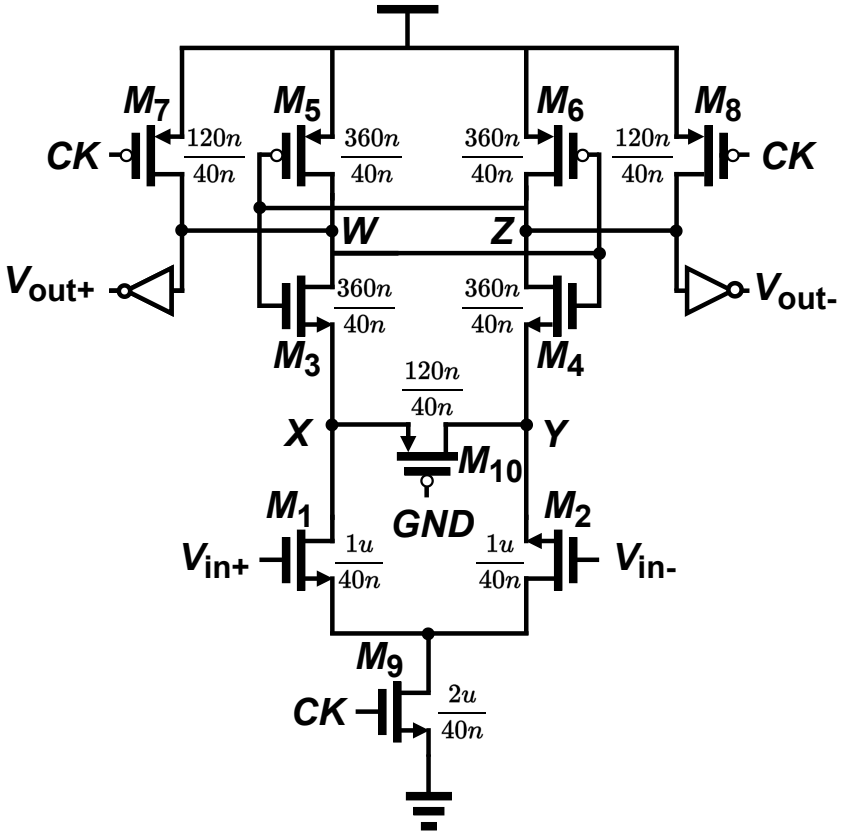
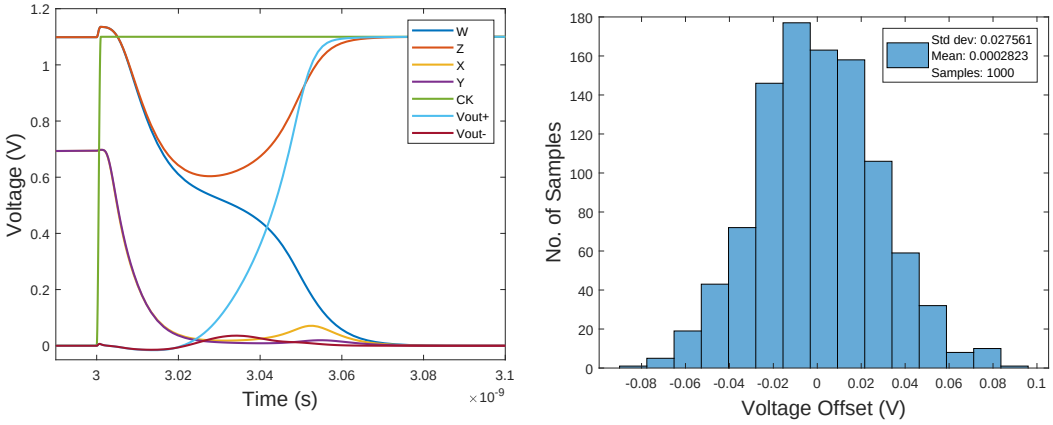


Figure 4.5: StrongARM comparator.



(a) Comparator Settling.

(b) Monte Carlo Mismatch Simulation.

Figure 4.6: Comparator circuit simulations.

4.1.3. Transition Logic

The transition logic calculates which PAM-4 transition occurred for the sampled transition. This happens with standard TSMC LVT logic gates. Fig.4.7 provides an overview of the circuit at the logic gate level. In order to compute the correct transition, the circuit compares the outputs of each comparator to see at which level a crossing occurred. $D_{lead2-0}$ is XOR'd with D_{lag2-0} to create cross signals $X_{a,b,c}$ and $nD_{lead2-0}$ is XNOR'd with nD_{lag2-0} to create non-cross signals $nX_{a,b,c}$. All transitions can be found using this information. It is beneficial to use both comparator outputs to minimize loading of the comparators.

The process for three distinct transitions is illustrated in Fig.4.8. For a $Minor_L$ transition, a transition occurs only at the lowest voltage level. If X_a is high and X_b, X_c are low, this transition must have occurred. For $Middle_H$, X_a must be low and X_b, X_c are high. For $Major$, all XOR signals must be high. It is important to note that not all levels need to be considered for each transition. Looking back at the $Minor_L$ transition, if X_a is high and X_b is low, X_c can only be low and therefore should not be considered.

It is still ambiguous if the transition is rising or falling. This can be computed by comparing the XOR signals with the leading data. If a transition occurred at the lowest level, X_a is high, comparing this signal to the leading data at the lowest level, D_{lead0}, the direction can be determined. If D_{lead0} is positive, the transition must be falling and if it is negative, rising. Since a transition can occur at all three levels, all have to be considered and combined for a valid rise or fall signal. A NAND gate implementation is used to minimize the delay and area. The lagging data can also be used, but since the leading data is available sooner, it is preferred to use these signals.

The power consumption is very low at $100\mu W$ for a data frequency of $12GHz$. The delay of the transition logic is predicted to be around $166ps$ in TT corner post-layout.

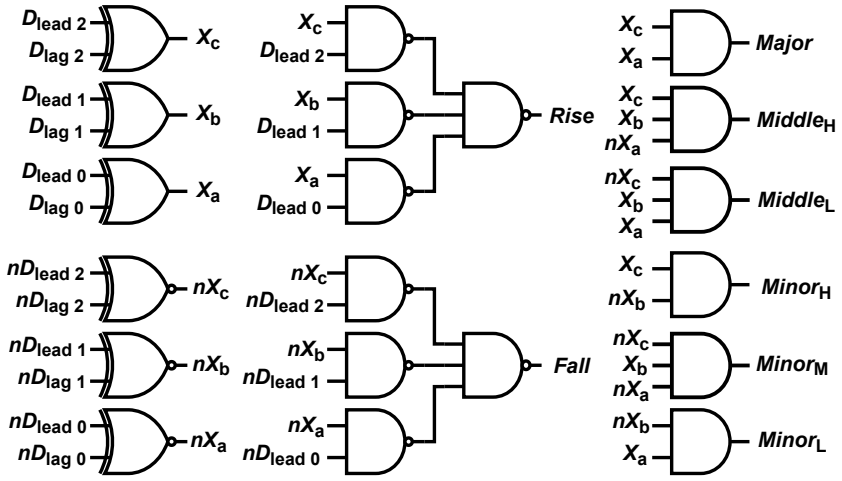


Figure 4.7: Transition logic circuit at gate level.

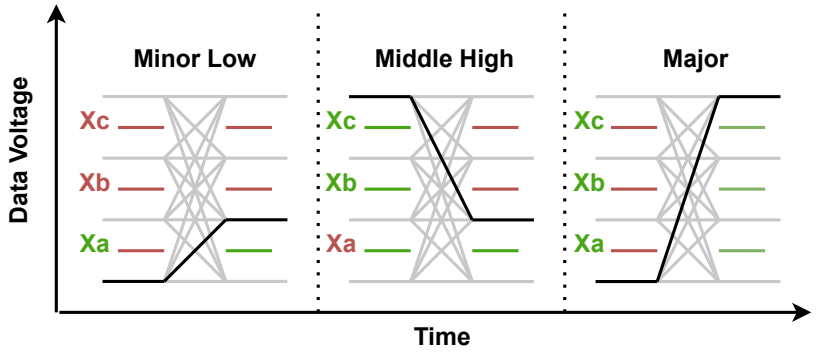


Figure 4.8: Transition determination.

4.1.4. Dynamic Flip-Flop

The conventional TSMC Dynamic Flip-Flop (DFF) has long setup and hold times, introducing undesired delay to the phase detector and thus limiting the maximum data rate. A faster alternative is based on the true single-phase clocking (TSPC) topology with split outputs [30]. This topology is not yet sufficient for high-frequency operation due to the degraded internal low voltage. Replacing the clocked transistors with complementary MOS switches, as used in [31], full swing is achieved at the internal nodes. The schematic is presented in Fig.4.9. The complementary switches can charge or discharge nodes $W - Z$ to full swing. A comparison of the proposed circuit and the standard TSMC-DFF in a frequency division test can be seen in Fig.4.10. The TSMC-DFF fails at an input of $8GHz$ while the TSPC-DFF maintains its functionality until $18GHz$.

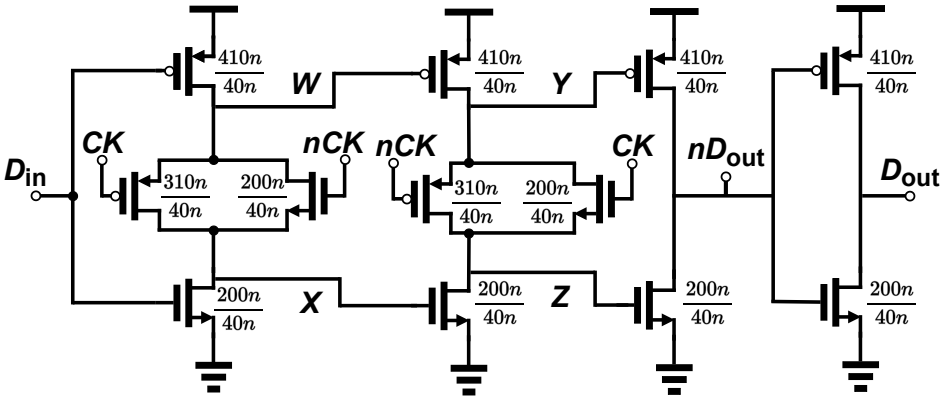


Figure 4.9: Dynamic Flip-Flop Topology.

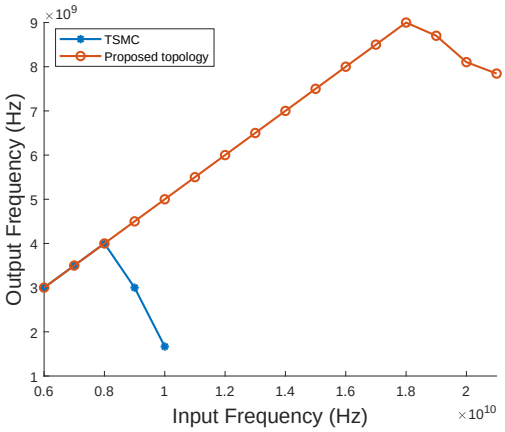


Figure 4.10: Performance of DFF in divide-by-two circuit.

To understand the working of the DFF, a transient simulation is presented in Fig.4.11, where D_{in} switches to positive and $50ps$ later CK goes positive. During low CK, the DFF can be set up. If CK is low, the first inverter stage of the DFF is active, charging W and X to nD_{in}. Node Y will be charged if D_{in} is positive, leaving Z floating and vice versa for a negative D_{in}. When Y or Z is charged, the DFF is set up, this takes about $40ps$. At the positive clock edge, the second inverter stage turns on, connecting Y and Z. This allows the charged state to propagate to the output. At this point, W and X are isolated, ensuring changes on D_{in} cannot propagate to Y and Z. After the positive clock edge, it takes another $50ps$ for D_{out} to become valid.

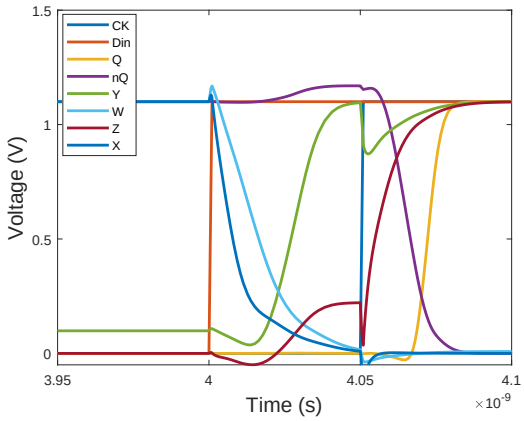


Figure 4.11: Transient simulation.

4.1.5. Multiplexer

A 12-to-1 multiplexer (MUX) is needed to provide the correct register bits to the DAC. Since the registers of each transition contain 5 bits, 5 MUXes are needed in parallel. The conventional implementation employs standard digital logic as can be seen in Fig.4.12. The desired output bit passes through the AND gate only if all the select signals are set to the correct value. In the Fig.D_1 is selected with $S_{0,1} = 00$. Incorrect select signals result in a low output from the AND gate and therefore an OR gate is used to combine the outputs.

This implementation of a MUX is straightforward but has some serious drawbacks. The logic gates will need a large number of inputs and will therefore be slow. The inputs are also selected with binary encoding, this means that the control signals of the transition logic need conversion adding additional delay. Furthermore, this topology scales to 2^n inputs, leading to four redundant lines in this implementation.

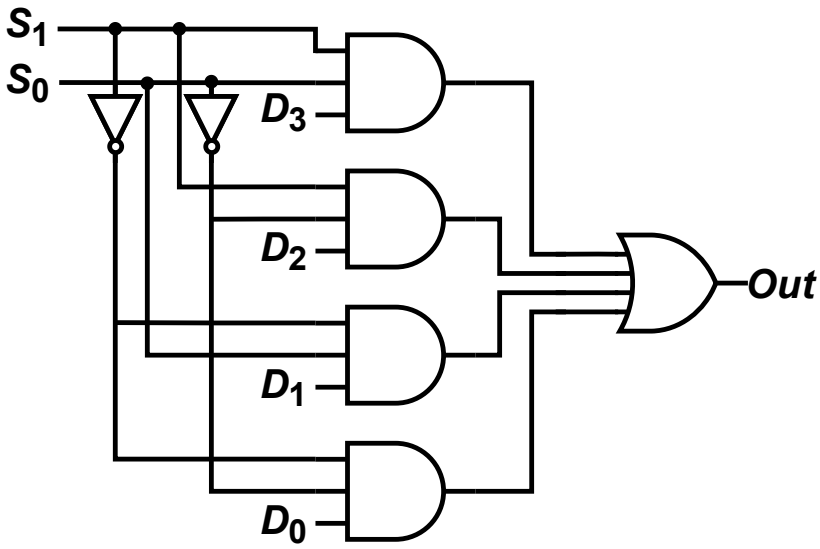


Figure 4.12: Standard MUX topology.

A more effective implementation is a pass gate topology. As illustrated in Fig.4.13. When $S_0 = 1$ both the NMOS and PMOS are turned on, enabling the transmission of bit D_0 to the output. Although complementary transistors add extra output capacitance, an NMOS is needed for passing 0 and a PMOS is needed to pass 1. Without the complementary gates, the output can only be charged to $V_{gs} - V_{th}$.

A drawback of this topology is the large output capacitance due to the 12 input ports. This results in low charging of the output. Using larger transistors will not help since R_{on} scales with $\frac{1}{W}$ and C_{dd} with W. Due to this problem, close to minimal size can be used for the NMOS, while the PMOS should be proportionately sized to maintain equal R_{on} and thus equal delays for negative and positive bits. Another limitation is the need for 12 select signals, requiring the use of additional digital logic, albeit simple, to be added between the transition logic and the MUX.

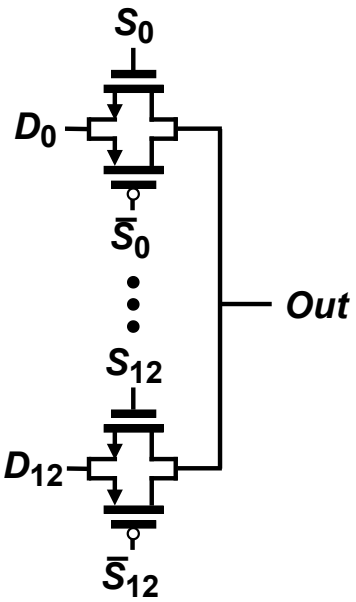


Figure 4.13: Pass gate MUX topology.

Both problems are addressed with the final topology depicted in Fig.4.14. To reduce the output capacitance, the MUX is split into two stages: one for the transition type and one for the direction. No additional logic is required for generating select signals since the DFFs have complementary output. The time constant reduces from $12RC$ to $10RC$, which is still significant since it takes 5τ to fully charge the output. An inverter is introduced to mitigate this delay. Only 0.69τ is needed to reach $0.5VDD$ and activate the inverter. Adding an inverter between the stages is not beneficial since the delay time is more than $2RC$. To accommodate the unequal loading by the DAC, the inverters of each MUX are scaled proportionally.

A transient simulation in the complete system is presented in Fig.4.15. At $9.9ns$, the DFFs pass their outputs. It can be seen that the negative control signals arrive slightly earlier due to the DFF design. The direction control signals rise faster due to the decreased loading with respect to the transition control signals. The internal nodes X and Z experience significant charging time, while the output node has a propagation delay of about $150ps$ after the setup of the flip-flops.

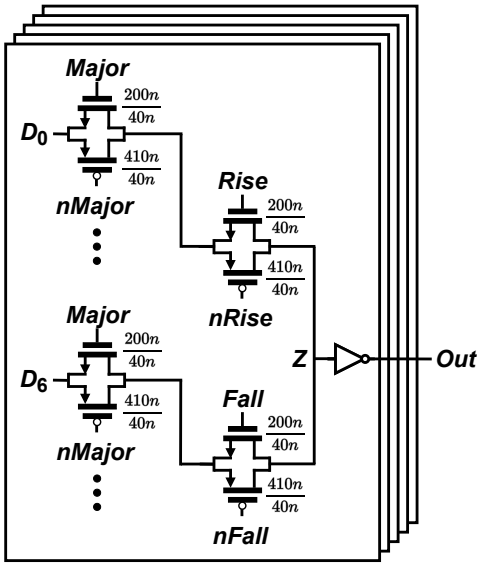


Figure 4.14: Final MUX topology.

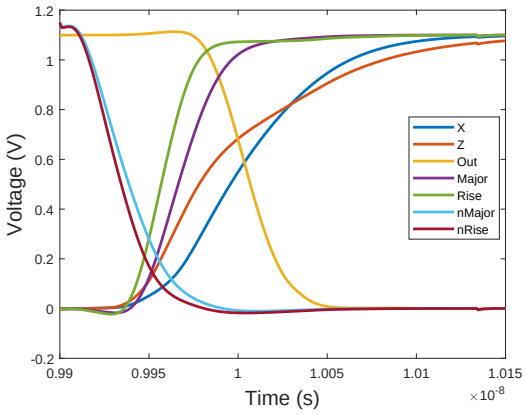


Figure 4.15: Transient simulation of the MUX.

4.1.6. Binary Current Steering DAC

The DAC is tasked with converting a binary code from the MUX into a reference voltage swiftly and with minimal power consumption. Various DAC topologies exist, but current steering is chosen for its superior speed [32], since this is the main limitation of the PD.

In the DAC, the delay time is influenced by power consumption; a time budget is thus required. To ensure the reference voltage is compared before the transition sample resets, the time budget is calculated as the reset time minus the time for the reference code to arrive and one comparator delay, approximately $70ps$.

In section 3.4.1 of the system-level analysis, it was determined that the DAC requires 5 bits. The DAC will drive a sampling capacitor of $5.8fF$ connected to the comparator. The sampling capacitor is needed to equalize kickback. This leads to the following simplified schematic illustrated in Fig.4.16. The DAC needs to charge the capacitor to less than 0.5LSB error. The resistance and current can be calculated by evaluating the case for discharging the capacitor to minimum voltage, leading to Eq. 4.1 and Eq. 4.2. The required resistance is 2800Ω. 2200Ω is chosen to compensate for the additional drain

capacitance, leading to a current of $7.1\mu A$ per current cell.

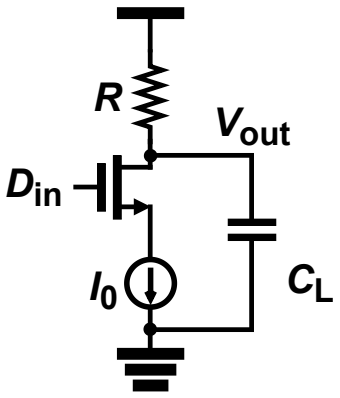


Figure 4.16: Simplified DAC.

$$V_{out}(t) = VDD - I_0R + V_{swing}e^{\frac{-t}{RC_L}}$$

$$R = \frac{-t}{C_L \cdot ln(\frac{1}{2^{n+1}})} \tag{4.1}$$

$$I = \frac{V_{swing}}{2^n \cdot R} \tag{4.2}$$

The current cell is presented in Fig.4.17. M_1 and M_2 steer the current to the correct branch and should be the minimum size. M_3 is used to increase INL and to reduce power consumption. The DAC stays dormant during most of the clock cycle. It only needs to be active when the MUX output is ready and until the comparator is settled. The DAC can therefore be turned on and off to save power consumption; this is done with control signal C. It is important to note that the DAC needs time to charge internal capacitances and therefore should be turned on before the MUX is ready. The duty cycle of C is 0.375 and is constructed from the available clock phases. M_3 can be the minimum size since it adds loading on C and the mismatch requirement can be reached with M_4.

Finally, transistor M_4 deals with the non-linearity of the current sources and sets the current. INL due to output resistance is less important in this design. It is the same for all DACs and therefore the calibration loop can compensate for this. DNL, however, is an issue. It can lead to missing codes and thus a higher quantization error. To avoid missing codes, the DNL should be less than $0.5LSB$. The maximum DNL due to mismatch for a binary DAC occurs at the MSB switch. It can be calculated with Eq.4.3 [33]. This allows for a maximum σ_i of $637\mu A$. Since the mismatch scales with area according to Eq.4.4, the width and length can be increased until the mismatch is acceptable.

The size is chosen to be $W/L = 550n/350n$. The final DAC is simulated in cadence virtuoso, the results are presented in Fig. 4.18. Fig.4.18b presents the mismatch of the current cell in a Monte Carlo simulation, with $\sigma_i = 467n$ the $DNL_{max} = 0.37LSB$. The INL is 0.53. Fig.4.18a shows the transient simulation for the DAC. A modified version of the MUX simulation is presented where all bits switch leading to a worst-case scenario. Again at $9.9ns$, the DFF clock switches to positive. One clock phase later the DAC control signal becomes high. At $10.1ns$, $200ps$ after the DFF passes its output, V_{out} reaches $0.5LSB$ error.

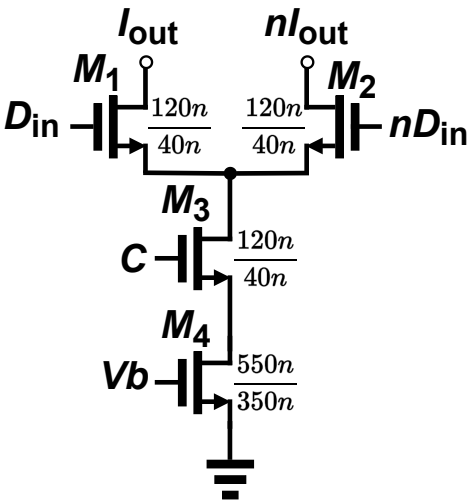
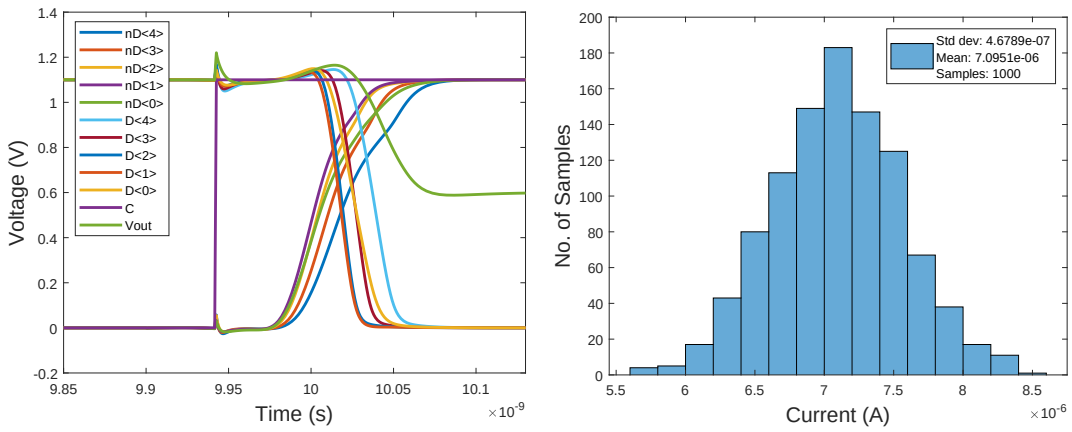


Figure 4.17: DAC current source.

$$DNL_{max} = \sqrt{2^n - 1}\frac{\sigma_i}{\Delta I} \tag{4.3}$$

$$\sigma_i = \frac{A}{\sqrt{WL}} \tag{4.4}$$



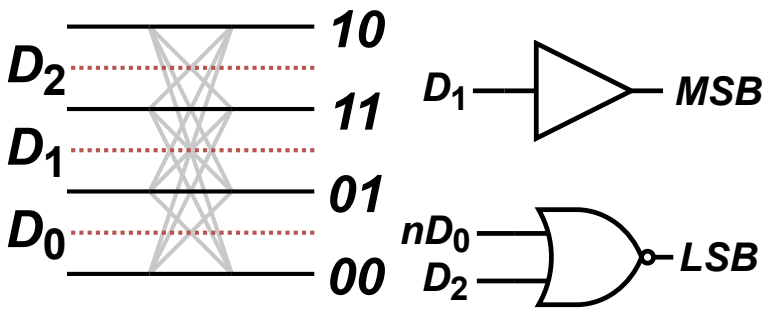
(a) DAC settling.

(b) MC simulation current source.

Figure 4.18: DAC circuit simulations.

4.1.7. Data extraction

The comparator outputs need to be converted into two data streams, one for the MSB and one for the LSB. There are three data points from the comparator which can be used. Fig. 4.19 shows the data extraction block. Illustration 4.19a shows the PAM-4 data points and the comparator data points. Grey encoding is used; this ensures a missample of a symbol only results in one bit error instead of two. For the MSB, we can directly look at D_1; if this is high, the MSB is high. The LSB is high when the signal is above D_0 but below D_2. The simple implementation can be seen in Fig.4.19b.



(a) Grey encoding illustration. **(b)** Data extraction circuit.

Figure 4.19: Data extraction.

4.1.8. Interpreter

The final phase detector block is the interpreter. This block converts the information from the sample comparison into phase information. It utilizes the complementary output of the comparator and the buffered rise and fall signals from the transition logic. The interpreter has to consider five possible cases, each illustrated in Fig.4.20. The comparator decision, CDR task, and PD output are also presented for each case. The DCO is controlled by PMOS transistors; therefore, $Out < 1 : 0 >= 11$ corresponds to less current flowing to the DCO and thus a lower frequency. After the interpreter, the output gets inverted twice: once by the DFF (the negative output is faster than the positive) and once by the DLF MUX, resulting in the same bits being presented to the DCO.

The logic implementation can be seen in Fig.4.21. Limited logic can be used since complementary comparator signals are available, as well as complementary rise and fall signals. $Out < 1 >$ is always set to 0 and is pulled up when the CDR needs to slow down. $Out < 0 >$ is always set to 1 and is pulled down when the CDR needs to speed up. The truth table can be seen in Tab. 4.1. The output signals of the interpreter are buffered with the previously discussed flip-flop and the inverse output is used, since this output is available sooner.

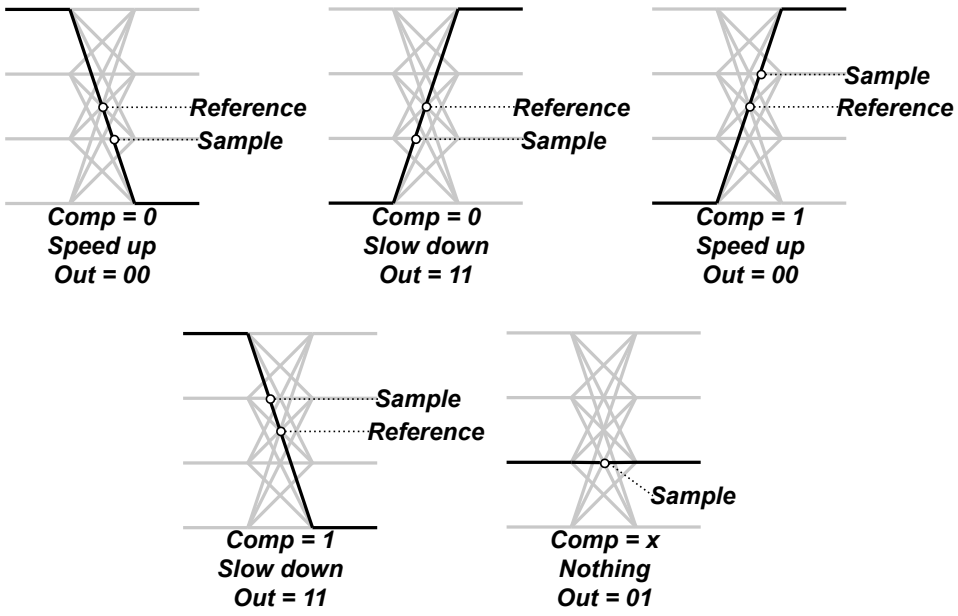


Figure 4.20: Illustration of interpreter decisions.

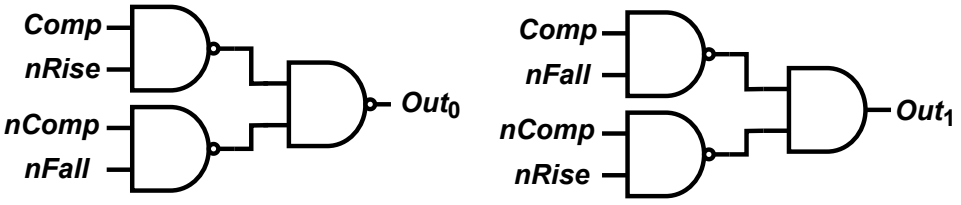


Figure 4.21: Interpreter circuit.

Comp	Rise	Fall	Out
x	0	0	01
1	0	1	11
0	0	1	00
1	1	0	00
0	1	0	11

Table 4.1: Interpreter decisions.

4.1.9. Layout

The layout of the complete PD is presented in Fig.4.22. The individual components are annotated. Note that in the final design, eight PDs are needed. The PD components have been closely integrated to minimize the signal delay between each block.

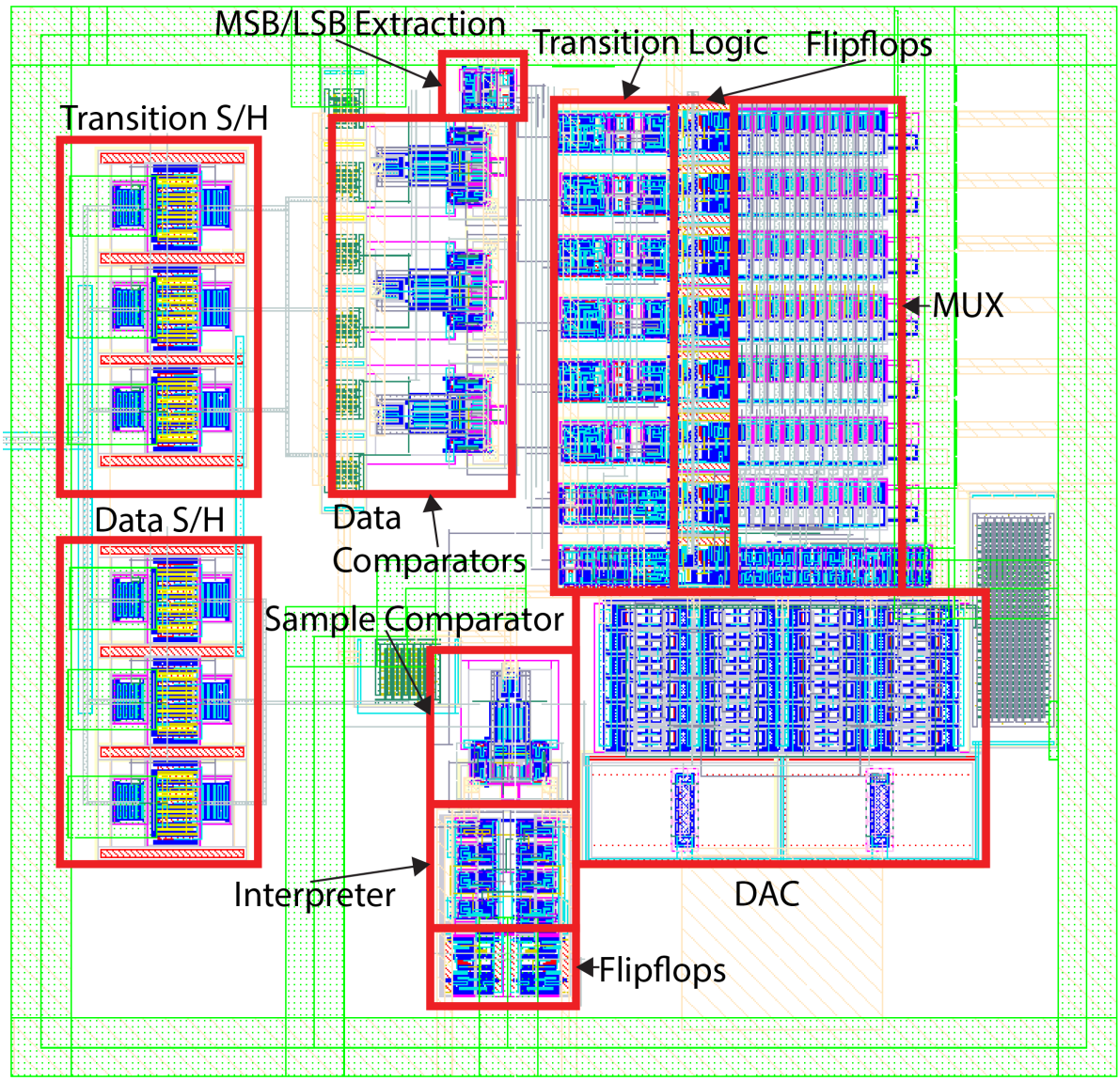
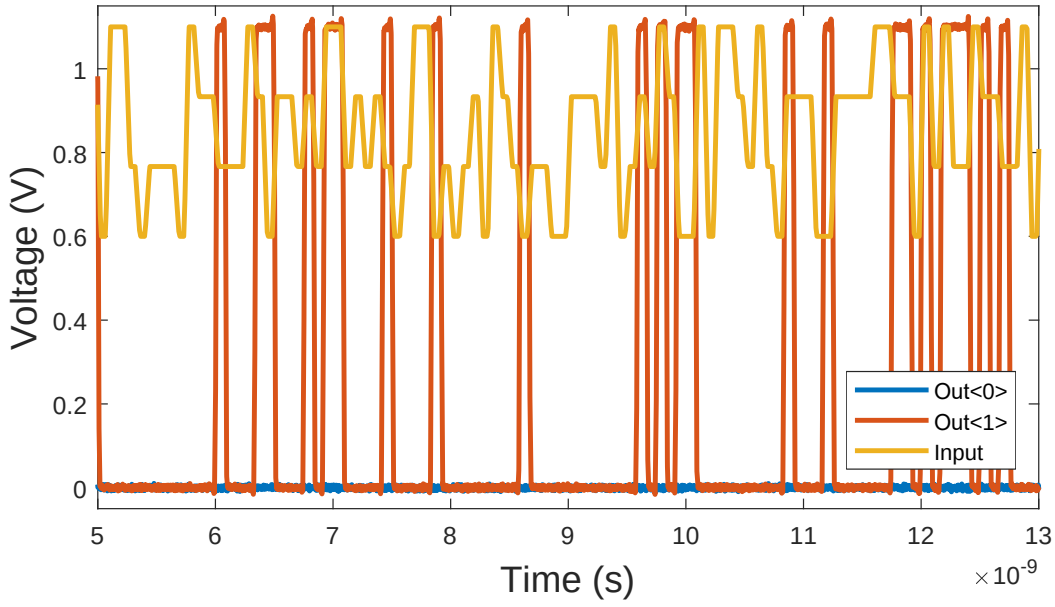


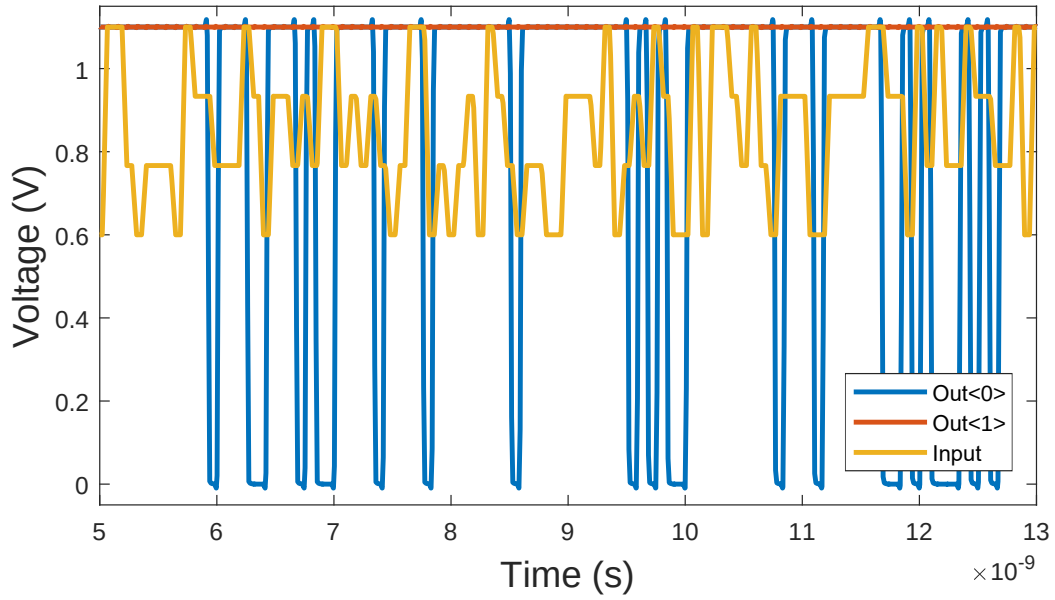
Figure 4.22: Phase detector layout.

The critical delay path is from the DFF that buffers the transition logic output to the start of the reference comparator. This path needs to complete before the sample gets reset at the positive edge Φ_{14}. The DFFs pass their signal at Φ_2, meaning there is a total delay of 9 symbols in the phase detector. The half symbol delay in the proportional path and 1 symbol delay in the DCO since the phase of the DCO is not updated instantaneously. A total delay of 10.5 is achieved.

The functionality of the phase detector is illustrated in Fig.4.23. A transient simulation is shown in Fig.4.23a for an open-loop simulation of the phase detectors and DLF for a lagging oscillator clock. A pseudo-random bit stream (PRBS) is used as input. As can be seen, the output is consistently low, corresponding to the increase in DCO frequency, unless no transition occurs. A case with a leading DCO clock is presented in Fig.4.23b for the same input. As can be seen in the figure, the output is consistently high unless there is no transition; the same behavior occurs for non-transitions in both cases.



(a) Phase detector outputs with a lagging DCO clock.



(b) Phase detector outputs with a leading DCO clock.

Figure 4.23: Post-layout phase detector behaviour.

The power consumption of one PD is $370\mu W$ leading to a total PD power consumption of $3mW$. The area of one PD is $37\mu m^2$. At a data rate of $12GHz$, the maximum speed of the PD, the delay of the PD is 10.5 symbols.

4.2. DLF

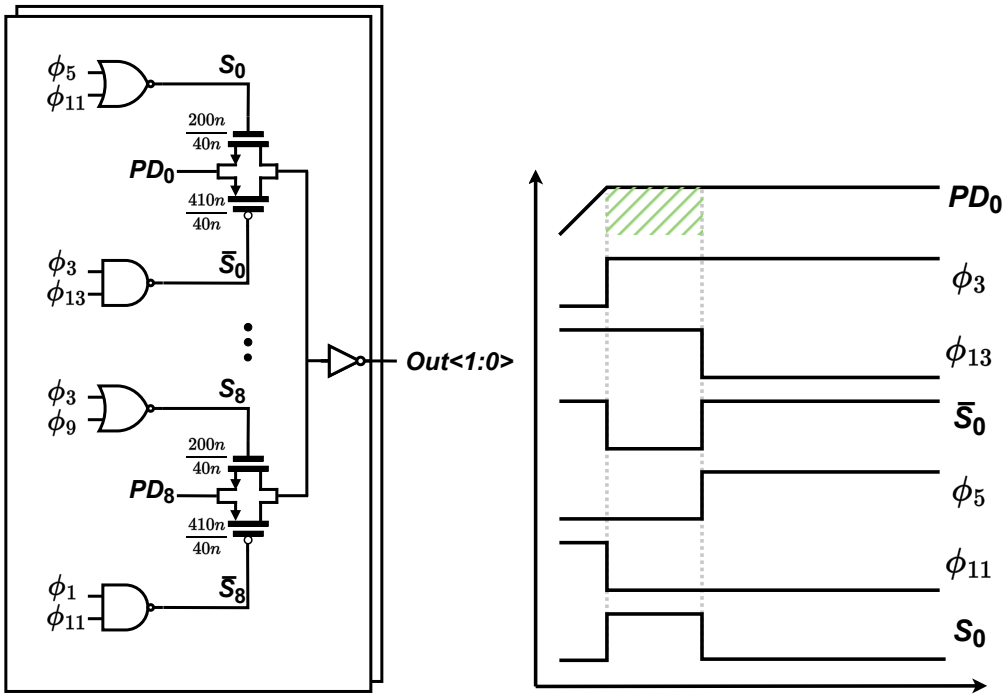
The second main component of the CDR loop is the digital low-pass filter (DLF). Both paths are manually designed. For the proportional path, speed is of the essence. For the integral path, RTL code would lead to high power consumption due to the high-frequency operation.

4.2.1. Proportional Path

The proportional path consists of two 8-to-1 pass gate muxes. The design process is similar to the register MUX implemented in the phase detector. There is no benefit in splitting the MUX into two stages

since the time constant τ remains $8RC$.

The proportional path needs to cycle between each phase detector output with the data frequency. As earlier mentioned, this poses a problem since the oscillator runs at a lower frequency. Low-duty cycle control signals are again constructed from the low-frequency clock phases. The circuit and its waveforms are presented in Fig. 4.24. A circuit diagram is illustrated in Fig.4.24a, and the waveforms in Fig.4.24b show the phase detector taking a transition sample at ϕ_0. After nine symbols, at ϕ_2, the first phase detector output PD_0 is transmitted. Due to the delay of the DFF in the PD, the output becomes valid between ϕ_2 and ϕ_3. The output should then be passed between ϕ_3 and ϕ_5, which equals one symbol. The delay is equal to $9.5 - 10.5$ symbols since the DCO phase update is not instantaneous. Control signal S_0 is constructed with a NOR operation between ϕ_5 and ϕ_{11}. The negative control signal $\overline{S_0}$ is constructed with a NAND operation between ϕ_3 and ϕ_{13}.



(a) Schematic of DLF MUX. **(b)** Waveforms of DLF MUX.

Figure 4.24: Proportional path of the DLF.

4.2.2. Integral Path

The integral path needs to collect the phase information for each individual phase detector, combine them, and add them to an internal counter. The block diagram with sub-circuits is presented in Fig.4.25. The circuit is manually designed instead of with Verilog to ensure minimal gates. The delay does not matter, but since the circuit runs at a high frequency, $\frac{1}{8}f_{data}$, it is important to keep the gates to a minimum.

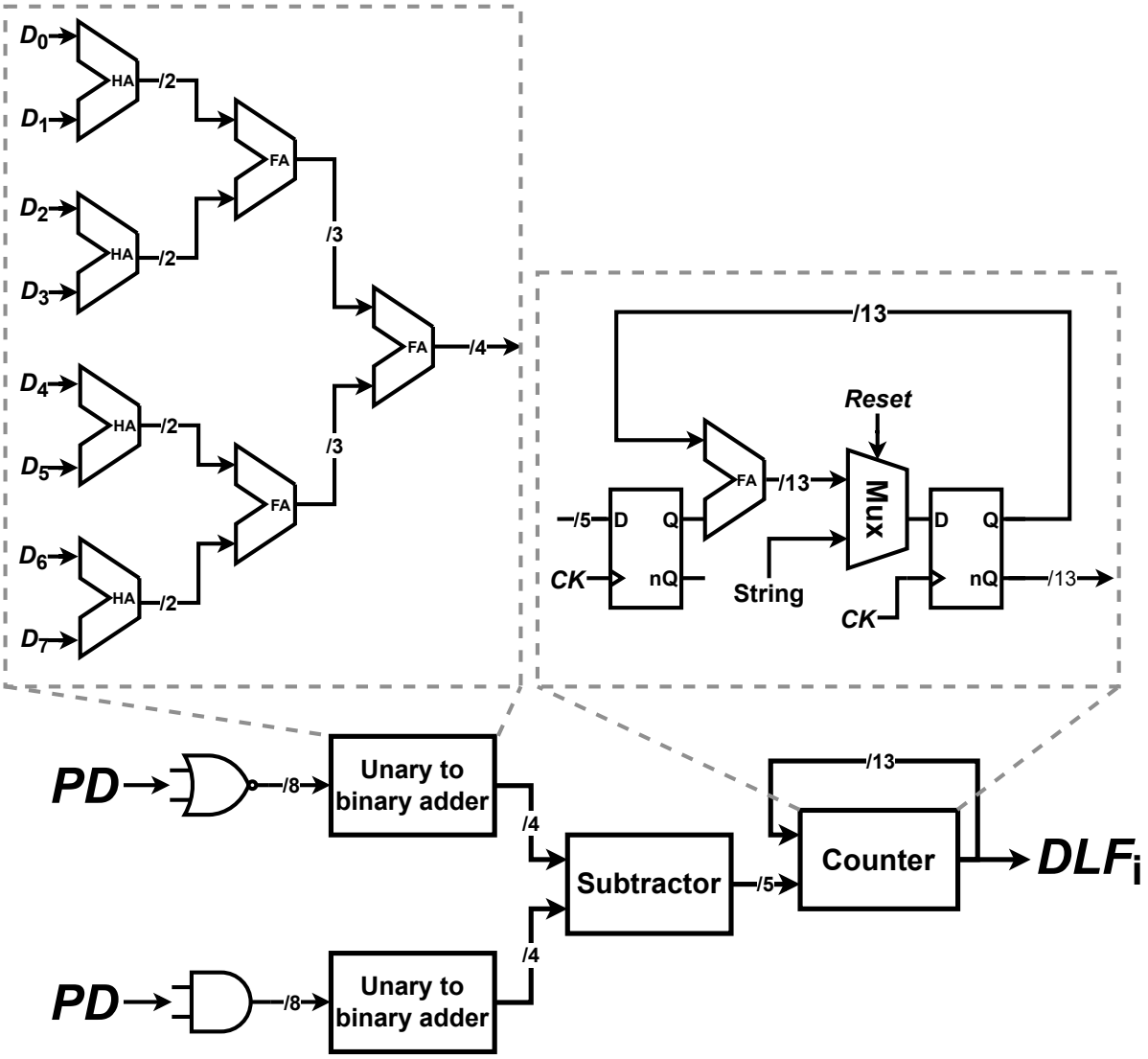


Figure 4.25: DLF schematic for the integral path.

The phase detector outputs become valid one at a time, which can induce racing conditions in the adders, making it uncertain when the output is valid. It is, therefore, necessary to buffer the phase detector outputs to create the DLF input. The phase detector outputs will be applied to two paths, one for the positive increment and one for the negative increment. Since 00 corresponds to a desired increase of the clock, NOR gates are used to create the unary increments, and an AND gate is used to convert 11 into negative increments.

The binary adder is created using a series of adders. For the first level, half adders can be used since the maximum output is 10. The next levels combine the 2-bit data into 3-bit data and the 3-bit data into 4-bit data. For the multiple-bit adders, ripple carry adders are used, with the first full adder replaced by a half-adder since no input carry is needed. The output of these adders is unsigned.

The next stage is a 4-bit ripple borrow subtractor. The output is converted in this stage into 2's complement form. The output of the subtractor is a 5-bit signal since it ranges from -8 to 8.

The counter is the final step in the DLF integral path. The update increment needs to be buffered since the delay of the increment calculation is roughly $400ps$ for a worst-case scenario. This does not leave enough time to complete the calculation in one cycle. The increment is added to the internal 13-bit state of the counter. In 2's complement form, the 5-bit increment can easily be extended to the

longer format by copying the MSB. The output is then loaded into register flip-flops. The complementary output of these registers is used to control the integral path gain.

Lastly, a reset needs to be built-in. This is done by implementing a MUX between the 13-bit counter and the register. If the reset is high, a standard string is loaded into the registers. The string consists of zeros except for the MSB and the first bit not supplied to the DCO.

In Fig.4.26, the waveforms of the 13-bit counter can be seen. Since in a ripple carry adder the output is calculated consecutively, the full delay takes a significant time of $500ps$ if all bits change. The power consumption is low, estimated at $271\mu W$.

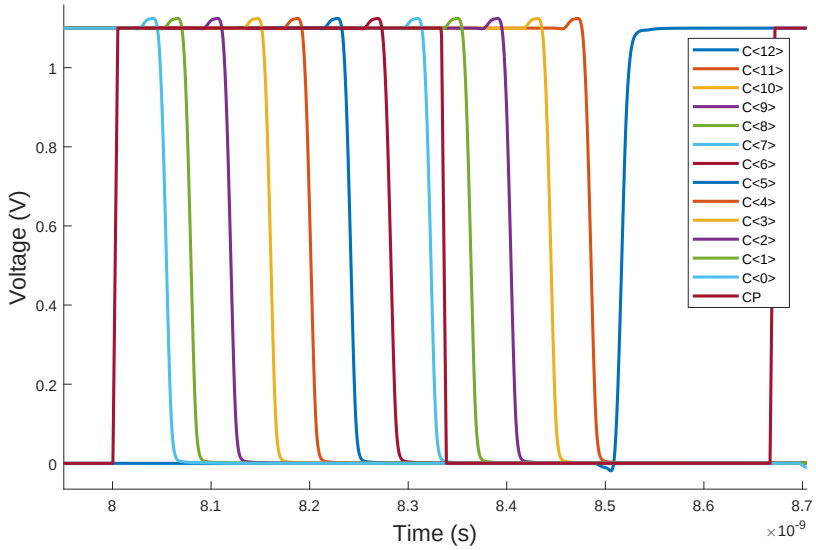


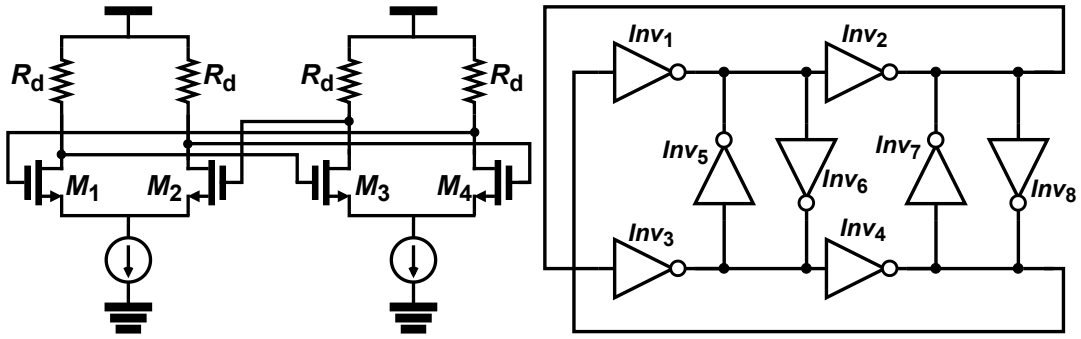
Figure 4.26: Worst case counter ripple carry delay.

4.3. DCO

The final analog component is the DCO. The DCO needs to provide 16 equal clock phases and needs three different gains: a jitter generation optimized proportional gain of $9MHz$, a jitter tolerance optimized proportional gain of $19.5MHz$, and a 13-bit binary integral gain of $9kHz$ for the LSB.

4.3.1. Topology

A ring oscillator is chosen instead of an LC oscillator. Although the LC oscillator has a serious benefit in phase noise performance, the increased area of the inductor makes it an impractical option in most short-reach applications. Since an even number of equally spaced phases is required, only differential oscillators can be used. If a ring of CS-amplifier stages or inverters are used, the circuit will latch up. The oscillators considered, Fig. 4.27 are the differential ring oscillator (see Fig.4.27a) and the pseudo-differential oscillator (see Fig.4.27b). If designed properly, the differential oscillator cannot latch up since current will be steered to one of the two branches in each stage. In the pseudo-differential oscillator, the extra inverters added to the ring oscillator ensure the circuit cannot latch up.



(a) Schematic differential oscillator.

(b) Schematic of a pseudo-differential oscillator.

Figure 4.27: Ring oscillator topologies.

A comparative study between the two oscillator designs has been conducted in [34]. The differential ring oscillator exhibits a better figure of merit (FOM) in the lower frequency range. The differential oscillator exhibits less up-conversion of flicker noise. On the other hand, the pseudo-differential oscillator performs better in the thermal noise region due to its full output swing. Since the phase noise transfer of the DCO has two zeros, one for the DCO itself and one for the DLF, low-frequency noise will be greatly suppressed. Therefore, the inverter-based ring oscillator is a more practical solution for this design. Although the addition of cross-coupled inverters to make the single-ended ring oscillator pseudo-differential does result in a small performance degradation, it still outperforms the differential ring oscillator in the thermal noise region.

To minimize the phase noise in the oscillator, the inverters must be properly sized to minimize the impulse sensitivity function (ISF). When a transition takes place, the oscillator is most sensitive to noise [35]. It is therefore important to minimize the transition times. In the pseudo-differential oscillator, the inverters of the main chain are fighting with the cross-coupled inverters during a transition. The cross-coupled inverters should therefore be made as small as possible. However, making them too weak will stop them from preventing latch-up. The inverter ring should also have equal rise and fall times to minimize the DC components of the ISF and therefore minimize flicker noise up-conversion.

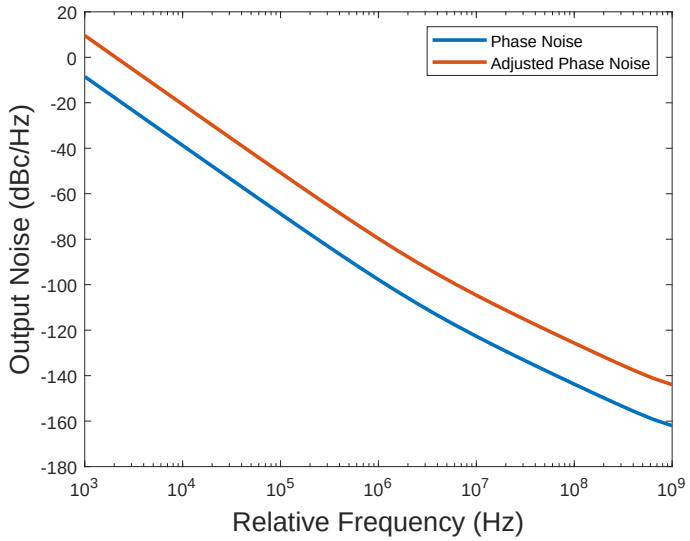


Figure 4.28: Phase Noise of the DCO.

Increasing the transistor sizes results in less flicker and thermal noise in the transistors. However, this comes at the cost of increased capacitance in the oscillator and thus increased power consumption

for the same frequency. The sizing of the transistors can be seen in Tab. 4.2, and the phase noise is illustrated in Fig.4.28. It is important to realize that the oscillator runs at $\frac{1}{8}$ of the data frequency. Adjusted for this, the phase noise is about 18 dBc higher at the data frequency. The flicker noise corner is high at about 5 MHz, and at 1 MHz, the phase noise is slightly more than -80 dBc. The oscillator consumes 2.8 mW. A transient simulation of the oscillator output is shown in Fig.4.29.

Device	M	W	L
NMOS Main	4	1.4u	70n
PMOS Main	4	2.8u	70n
NMOS Latch	4	600n	70n
PMOS Latch	4	1400n	70n

Table 4.2: Oscillator transistor sizing

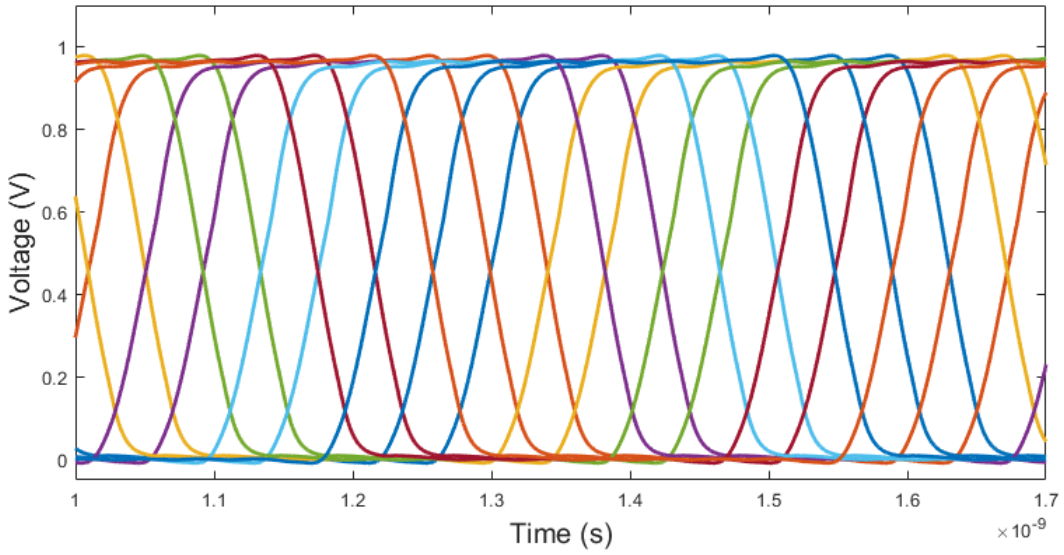


Figure 4.29: Transient simulation of the oscillator.

4.3.2. DCO Tuning

In order to track the phase changes as well as pinpoint the exact data frequency, the DCO frequency should be tunable. As previously discussed, three gain paths should be designed: a proportional gain for jitter generation, one for jitter tolerance, and an integral gain. The base frequency of the design should also be tunable in order to compensate for PVT variations and operate at lower data frequencies.

There are two common methods of tuning the frequency. The first is adding or removing capacitance from the DCO. This can be done by switching capacitors in or out. In the DCO of this CDR, this method is undesirable. Each clock phase needs an equal amount of capacitance, which means that for each bit controlling the DCO frequency, 16 capacitors are needed. This greatly increases the complexity in the design due to the several gain paths required. Mismatch in these capacitors will also cause the phases to become unequally spaced. Mismatch can be especially problematic since the capacitors need to be incredibly small; each capacitor needs to be $\frac{1}{16}$ of the total capacitance added.

A better alternative is manipulating the current available to the DCO. If digitally controlled current sources are placed between the supply and the VDD of the oscillator, the frequency can be altered. A complete schematic is presented in Fig.4.30. By controlling the current available to the DCO, the frequency can be increased or decreased.

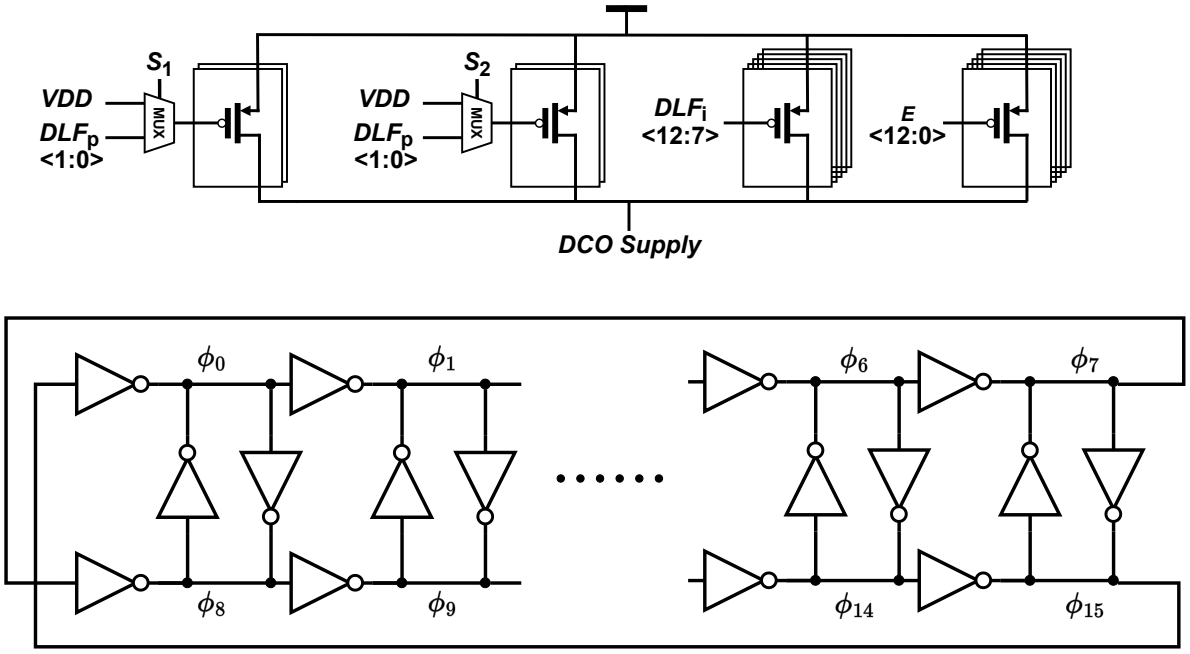


Figure 4.30: Schematic of the DCO

An immediate concern is the way the current sources are implemented. If the gate is set to VSS, the current sources are in the triode region, making the current a function of the drain-source voltage. This would result in compression of the gain when more stages are active. The higher the frequency, the higher the DCO supply voltage, and thus the less current the source provides.

In order to examine this issue, the supply sensitivity, KV_{DD}, needs to be known, and by extension, the variation the DCO supply will experience over the frequency range. The frequency should vary from $f = 1.493GHz$ to $f = 1.507GHz$. Fig.4.31 shows the frequency of the DCO vs. V_{DD}. A very small range is required for V_{DD} from about $993mV$ to $998mV$ due to the large KV_{DD} of $3.91GHz$. However, if V_{DS} is low for the current sources, this can still lead to issues.

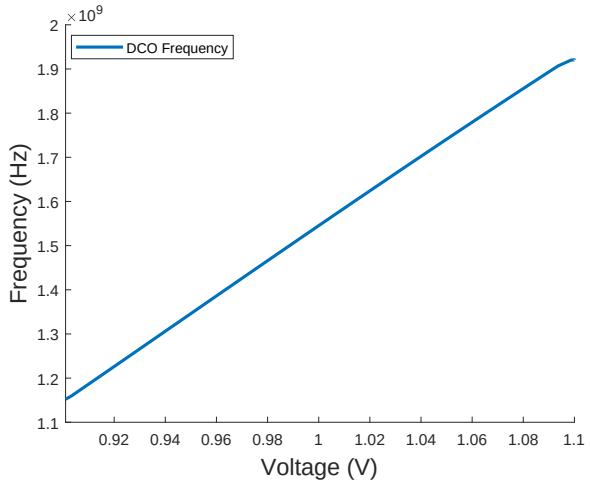


Figure 4.31: DCO frequency vs supply voltage.

Due to the low V_{DS} of $100mV$ available to the current sources, the small change in V_{DS} of $5mV$ has a significant impact on the gain. The second proportional current source shows a compression of 10%. To test this problem, the time-domain model is adjusted for variable DCO gains with a look-up table. Compression will lead to an increase of $30fs$. The best way to compensate for this issue is by creating

a larger gain in the second proportional current source.

With the problem of the triode region examined and mitigated, the current sources can be designed. The dimensions of the current sources can be found in Tab. 4.3. The lower bits of the integral path will result in long transistors with large input capacitance. These bits also change quickly without having much effect on the output. It is therefore beneficial to only count the lower bits internally in the DLF and not update the DCO. A time-domain MATLAB simulation shows that the first 6 bits can be neglected while only adding $2fs$ of jitter. The dimensions of the current sources can be seen in Tab. 4.32. An MC simulation is done for the smallest current source. The standard deviation of 1% is simulated in MATLAB and adds a negligible amount of jitter. In total, the DCO nonidealities, compression, mismatch, and neglected integral bits add $6fs$ of jitter for a total of $431fs$ in the time-domain model. The final design has an F.o.M of $157dBc/Hz$ according to formula 4.5.

$$FoM = -\mathcal{L}(\Delta\omega) + 20log_{10}\Big(\frac{\omega_0}{\Delta\omega}\Big) - 10log_{10}\Big(\frac{P_{DC}}{1mW}\Big) \tag{4.5}$$

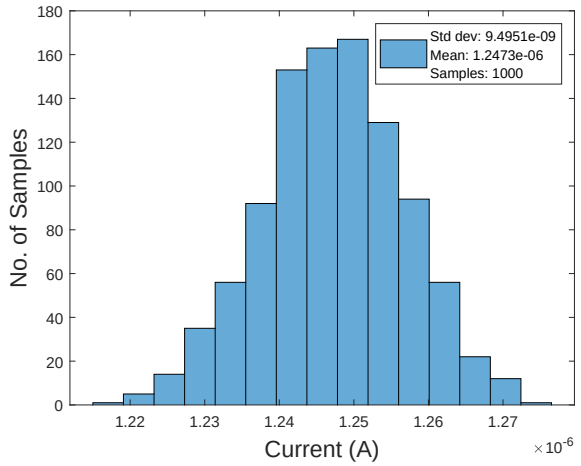


Figure 4.32: Monte Carlo simulation of smallest current source.

Device	M	W	L	Ideal	Actual	Error
Proportional JGEN<0>	1	260n	640n	1.125M	1.142M	1.51%
Proportional JGEN<1>	1	270n	640n	1.125M	1.147M	1.95%
Proportional JTOL<0>	1	290n	320n	2.438M	2.467M	1.24%
Proportional JTOL<1>	1	310n	320n	2.438M	2.429M	0.76%
Integral<12>	1	655n	320n	4.608M	4.625M	0.36%
Integral<11>	1	610n	640n	2.304M	2.324M	0.87%
Integral<10>	1	265n	640n	1.152M	1.156M	0.34%
Integral<9>	1	120n	745n	576K	573.5K	0.44%
Integral<8>	1	120n	1.51u	288K	293.3K	1.84%
Integral<7>	1	120n	3.08u	144K	141K	2.01%

Table 4.3: Dimensions of the current sources for DCO tuning.

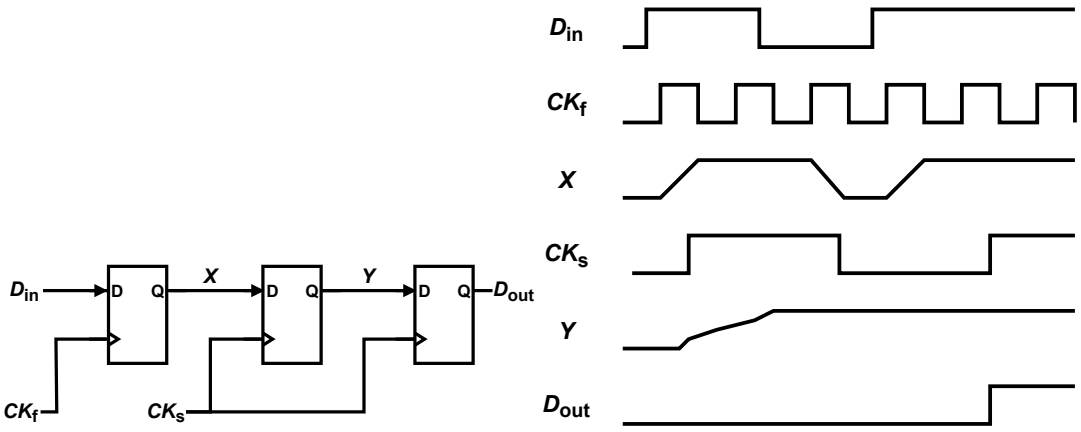
5

Register Calibration Loop

This chapter will present the design of the calibration loop and discuss its functionality and behavior.

5.1. Clock Domain Crossing

The calibration loop receives data from one of the phase detectors. Since the calibration loop runs at a slower clock, the phase detector outputs cannot be used directly. Clock domain crossing (CDC) must be implemented with the use of a synchronizer circuit 5.1. The circuit is presented in Fig. 5.1a and Fig. 5.1b illustrates the waveforms during metastability on node Y.



(a) Multi-flipflop syncronizer. **(b)** Waveforms during metastability.

Figure 5.1: Clock domain crossing.

The fast-changing signals D_{in} are buffered with the fast clock CK_f. A clock phase can be chosen in order to not violate set-up and hold times, since the phase of D_{in} with respect to CK_f is known. However, the phase of the slow clock CK_s is unknown with respect to the fast clock. Metastability can occur for signal Y if the setup time is violated. To mitigate this risk, a second slow flip-flop needs to be added to allow sufficient time for node Y to settle to a valid state before it is used by the calibration loop. Theoretically, metastability can still occur if Y does not settle within one clock period, although this is improbable due to the long clock period. Additionally, the propagation delay of the flip-flop should be longer than the hold time. Another important consideration is that some input data will be neglected when crossing clock domains, as this is inherent to clock domain crossing (CDC).

5.2. Calibration Loop Algorithm

If the reference code of a transition does not align with the sample in the locked state, the calibration loop must gradually adjust the register code for that transition. When the code is not aligned, the transition will have a higher chance to provide one type of phase information. It is crucial that at least one

transition is locked. Since the DCO has discrete frequency steps, the settled frequency will have some offset. This results in a higher probability, albeit very slight, of one type of phase information types being dominant. Consequently, wrongful adjustments of registers may occur if the calibration loop runs for a long time. If no registers are locked, all the values will wander until no phase information can be determined.

An algorithm of the calibration loop can be seen in Fig. 5.2. At the start of the loop, the initial register values are loaded, as well as the threshold. Certain registers are also locked.

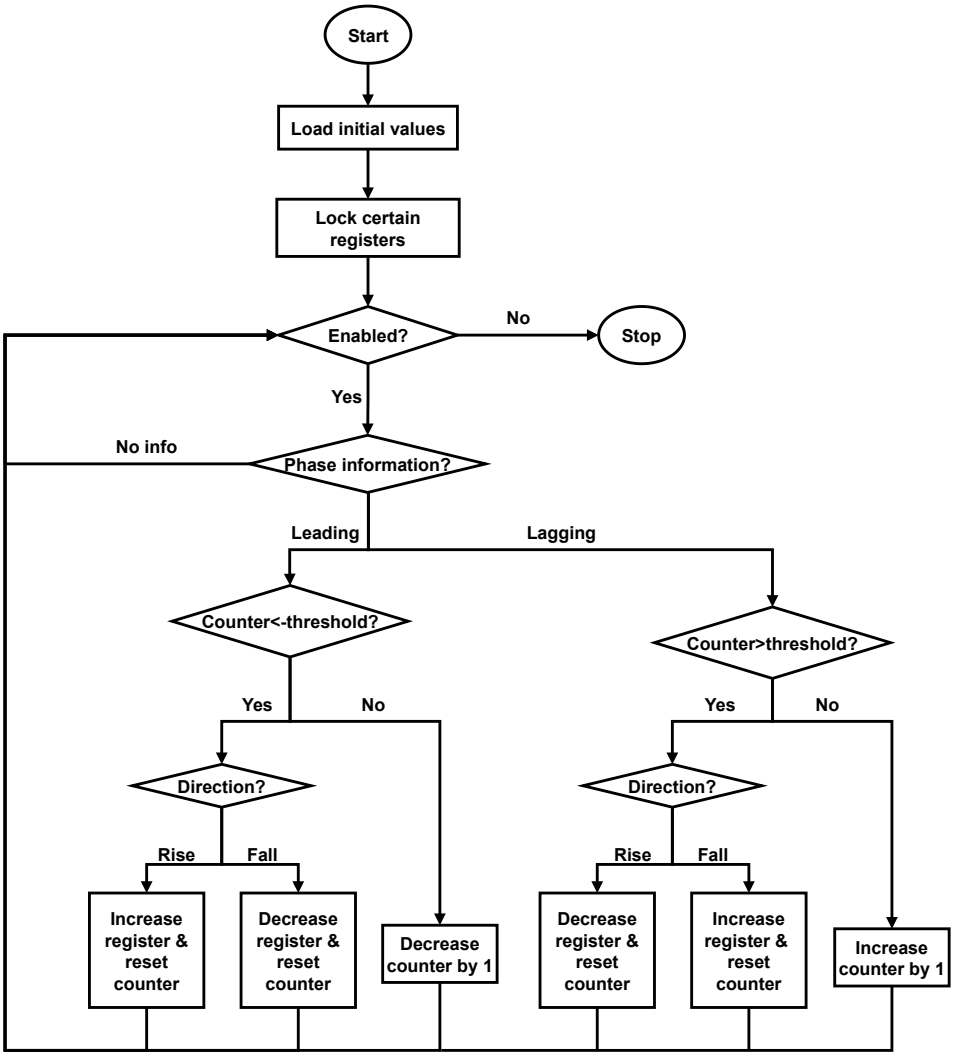


Figure 5.2: Register calibration algorithm.

If the register is enabled by the transition and enable signal, the correct register is selected. The phase information determines the direction of the counter. If there is leading phase information, the counter will be decreased. If there is lagging phase information, the register will be increased. If the counter is full, the register will be adjusted depending on the direction of the transition and the phase information. Leading information for rise signals will result in a positive increment, while fall signals will decrease. For lagging phase information, the direction of the increments is switched.

The calibration loop is written in Verilog code and can be viewed in Appendix B. The functionality was verified in QuestaSim, and an equivalent Verilog-A block was created and simulated in Virtuoso. The transient simulation was run with a locked major fall transition and fall transitions twice as short as rising transitions. Fig. 5.3 plots the codes for all twelve transitions.

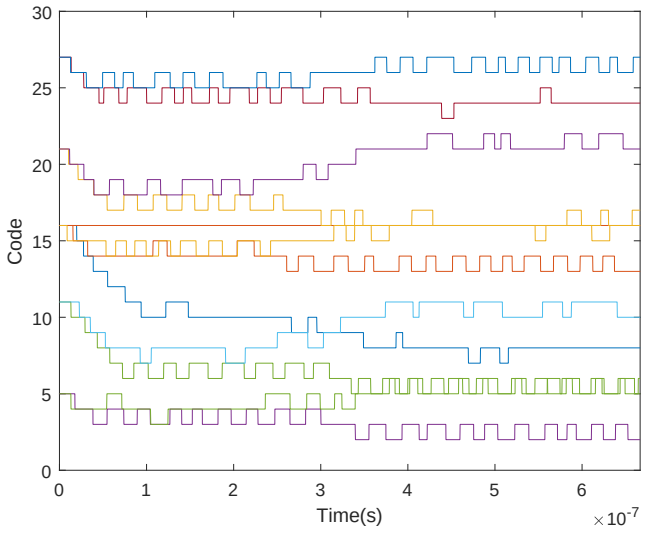


Figure 5.3: Register codes during non equal rise and fall times.

In the initial phase, all reference voltages are set to the middle of the transition. Initially, all codes converge to the average locking point. This is why the falling transitions are adjusted. After this point has been reached, the locked transition slowly pulls the codes of the other transitions to the set locking point. Since the codes will not perfectly match the voltage at the set locking point, the final state will switch between two codes.

5.3. Noise

A drawback of this calibration loop is that low-frequency jitter is passed to the output. This occurs because the loop adjusts the reference voltages alongside the jitter. The magnitude of this effect is determined by the bandwidth of the calibration loop $f_{bw} = \frac{f_{clock}}{Threshold \cdot N}$, where f_{bw} is the bandwidth of the loop, f_{clock} is the frequency of operation, and N is the number of different transitions. To illustrate this, the calibration loop is simulated in MATLAB with a high bandwidth. The result can be seen in Fig. 5.4. In order to minimize the jitter added by this effect, the loop should run at a low frequency. Another reason to run at a low frequency is the power consumption of the loop.

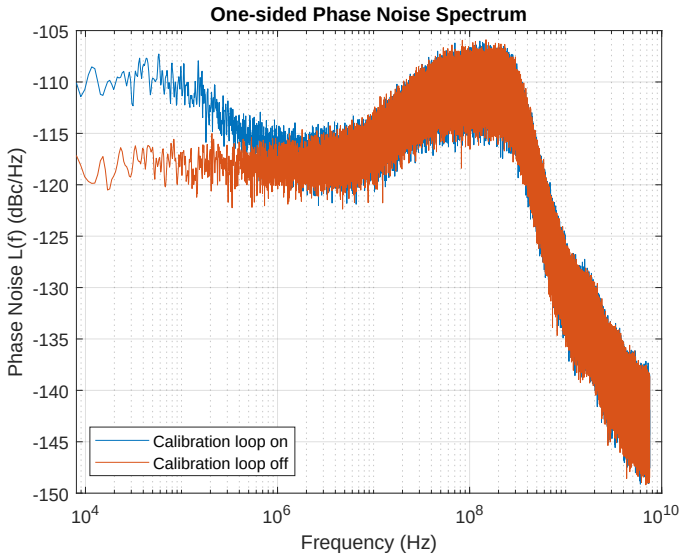


Figure 5.4: Phase noise of CDR with and without calibration

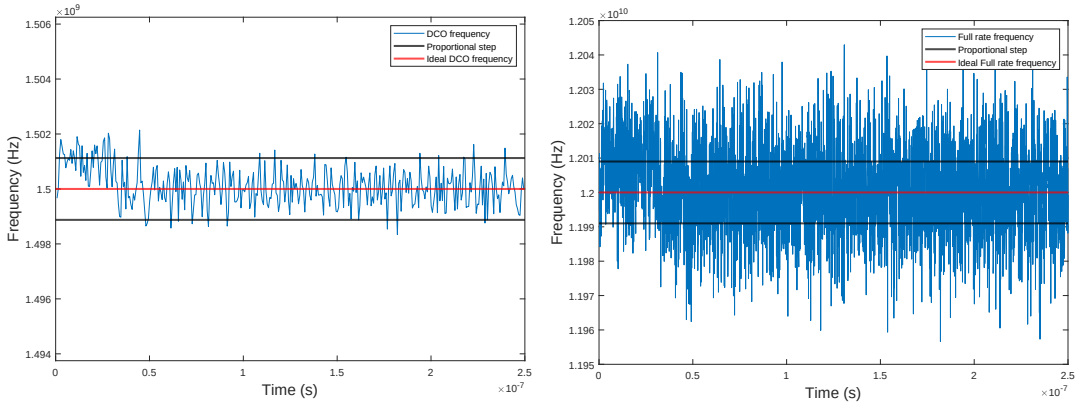
6 Simulation Results

In this chapter, the simulation results of the complete CDR will be discussed and compared to the system requirements.

6.1. Locking Behavior

6.1.1. Phase Locking

The locking behavior is simulated using a pseudo random binary sequence (PRBS) with rise and fall times of $41.66ps$ and amplitude of $0.5V$. In the initial state the DCO is leading the data with $20ps$ or about $\frac{1}{4}UI$ UI. The results are illustrated in Fig. 6.1. Fig. 6.1a shows the DCO frequency. The red line indicates the ideal frequency and the black lines indicate the ideal proportional path frequency steps. As can be viewed in the plot, the CDR immediately resorts to a higher frequency until the phase is corrected. Extracting and comparing the zero crossings of all even (or odd) DCO phases, the full rate operation of the CDR can be illustrated. The frequency plot can be seen in Fig. 6.1b, again the red line indicates the ideal frequency and the proportional steps are shown in black.

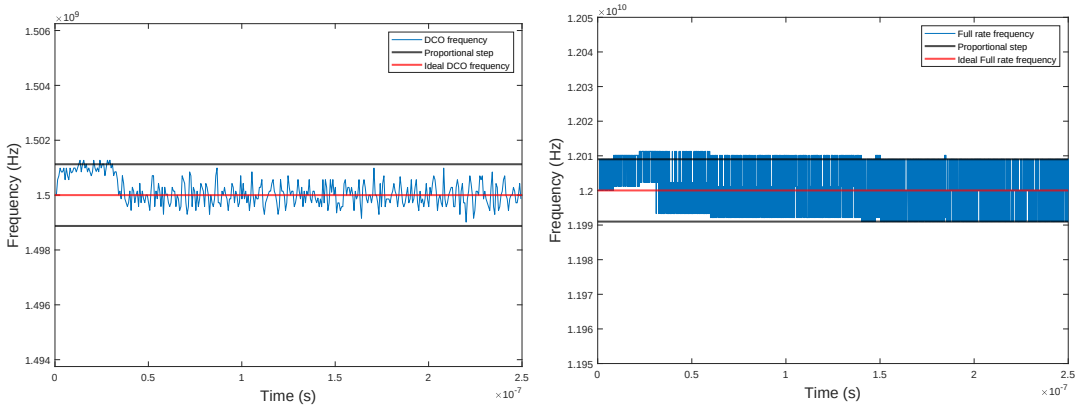


(a) DCO frequency.

(b) Full rate frequency.

Figure 6.1: CDR operation for an initial leading phase.

Due to the noise in the system, it can be difficult to see what happens. To illustrate the decisions the CDR makes, the previous simulation is repeated without input noise and an ideal Verilog-A oscillator. The result are presented in Fig. 6.2. Fig. 6.2a shows the DCO output and Fig. 6.2b, the full-rate frequency. As can be seen in the plot, due to the initial phase error, the integral path gets wrongfully updated. This error is slowly corrected once the CDR has caught up to the initial phase error. Since the frequency will be updated eight times during each clock cycle, the ideal DCO frequency experiences more than three discrete output steps, the full-rate frequency does show only three steps.

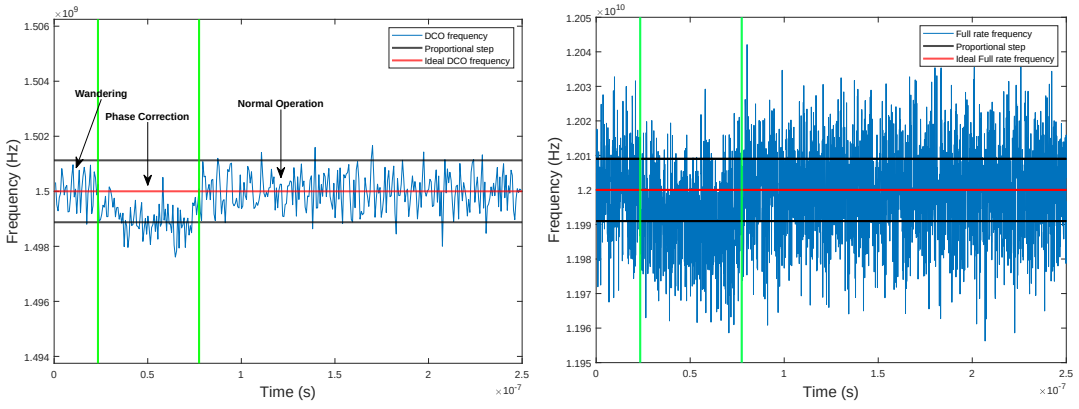


(a) DCO frequency. **(b)** Full rate frequency.

Figure 6.2: CDR operation for an initial leading phase without noise

Since the phase recovery is dependent on data recovery it is important to test the system with a maximal phase error where the data samples fall in the middle of a transition and no accurate data can be recovered. Again a PRBS with rise and fall times of $41.66ps$ and amplitude of $0.5V$ is applied to the input. In the initial state the DCO is leading or lagging by $41.66ps$. The results are presented in Fig. 6.3. Fig,6.3a shows the frequency output for the DCO, while Fig. 6.3b shows the constructed full-rate clock from the individual phases. The CDR response shows three regions, separated with green lines. Initially, due to the inaccurate data recovery, the CDR makes arbitrary decisions and wanders randomly until a point is reached where data can be recovered. Consequently the CDR makes leading phase decisions until the phase error corrected. Finally the CDR starts normal operation.

The simulation is repeated with an ideal verilogA oscillator, to get more insight into the decisions of the CDR, note that input jitter is still present to disturb the system, otherwise the system would stay in its wandering state for longer. The result can be seen in Fig. 6.4. Since the data samples fall on transitions and the transition sample on a data point, some patterns will result in no phase information, some in lagging and some in leading information. The smaller the phase error, the more data patterns can be correctly interpreted. For this reason wrong updates are still given to the CDR even though the phase error is already getting corrected.



(a) DCO frequency. **(b)** Full rate frequency.

Figure 6.3: CDR operation for an initial maximal phase error.

The simulation is repeated with an ideal verilog-A oscillator, to get more insight into the decisions of the CDR. It is important to note that input jitter is still present to disturb the system, otherwise the system would stay in its wandering state for longer. The frequency plot for the full rate frequency can be seen in Fig. 6.4. Since the data samples initially fall in the middle of a transition and the transition samples on a data points, some patterns will result in no phase information, some in lagging and some

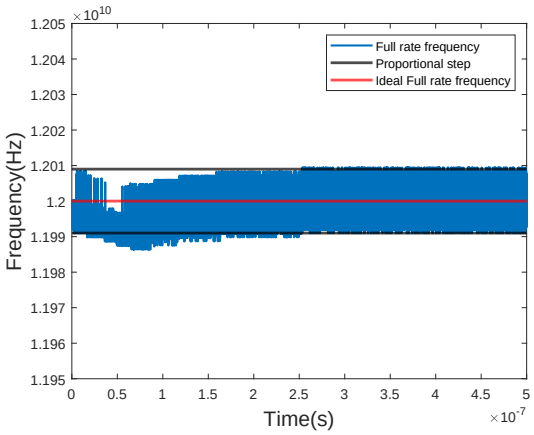
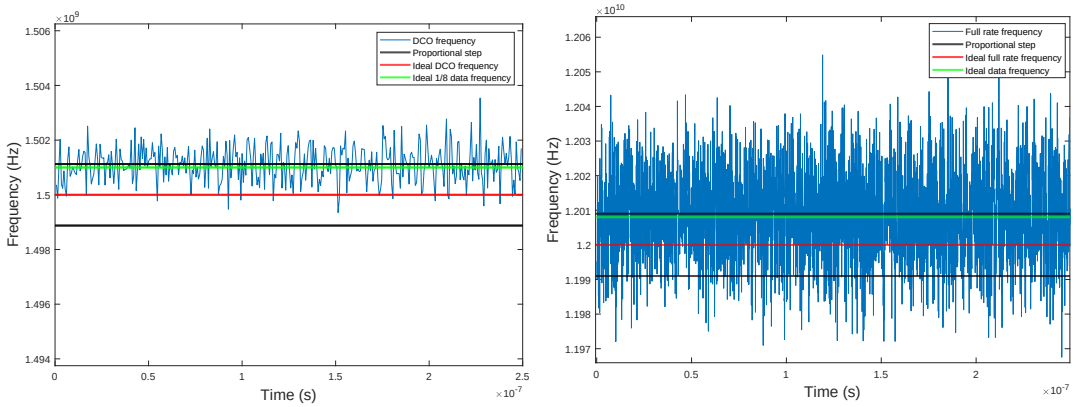


Figure 6.4: Noiseless CDR operation for an initial maximal phase error.

in leading information. The smaller the phase error, the more data patterns can be correctly interpreted. For this reason wrong updates are still given to the CDR even though the phase error is already getting corrected.

6.1.2. Frequency Locking

To test the frequency locking of the CDR a PRBS with rise and fall times of $1ps$, an amplitude of $0.5V$ and a Baud-rate of $12.008GHz$ was applied to the input. This equates to an $1MHz$ offset frequency at the DCO rate or an $8Hz$ offset at the Baud rate. The DCO starts at an initial frequency of $1.5GHz$ The results are presented in Fig. 6.5. The ideal data frequency is shown in green. Fig. 6.5a shows the DCO frequency and Fig. 6.5b shows the full-rate frequency. Since the frequency error is smaller than the proportional loop gain, the CDR can immediately compensate for the phase error. The integral path slowly adjusts to the proper frequency. Cycle slipping does not occur.



(a) DCO frequency.

(b) Full rate frequency.

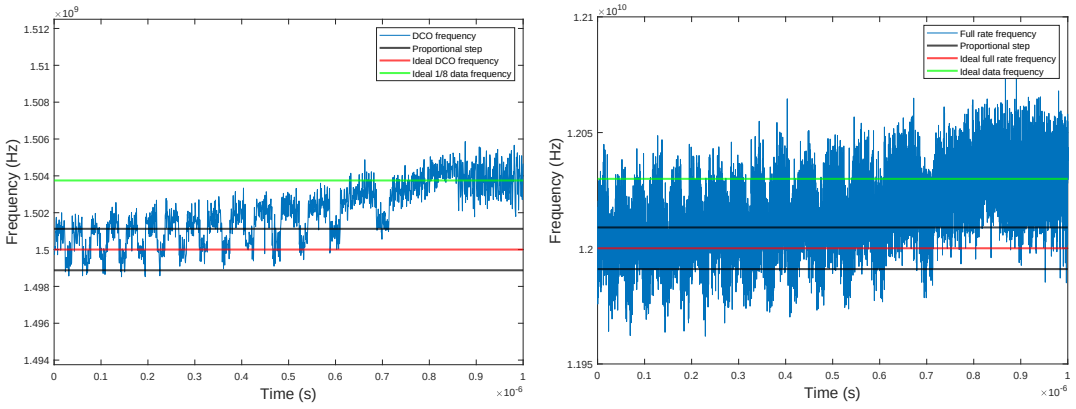
Figure 6.5: CDR operation for an in range frequency error.

The same simulation is run for a data frequency of $12.03GHz$. The results are illustrated in Fig. 6.6. In Fig. 6.6a DCO frequency is presented and Fig. 6.6b shows the full-rate frequency. When the frequency deviation exceeds the proportional path gain, the behaviour of the CDR is radically different showing cycles of leading and lagging phase information until the desired frequency is reached.

Initially the CDR has no phase error. It cannot run at the high data frequency, and thus begins to fall behind the input data resulting in an increasing lagging phase. Once the phase error is $-0.5UI$ a bit is skipped and the CDR will try to align to the next bit. At this point the CDR will interpret the phase information as leading, $-0.5UI$ equals $+0.5UI$ when a bit is skipped. Instead of increasing the

frequency now the CDR will decrease its frequency even though the data frequency is still higher. Subsequently the phase error will continue to increase, but this time at a faster rate since the frequency deviation is higher. An additional $-0.5UI$ is reached and the phase error is zero again. However, the CDR still runs at a lower frequency and thus the cycle repeats.

Even though both leading and lagging information is interpreted in each cycle, the CDR still slowly catches up to the data frequency. When the phase information is interpreted correctly, the frequency deviation from the data frequency is less than when wrong information is misinterpreted. Therefore, each cycle more correct updates are given to the integral path than wrong updates, slowly the integral path will increase the frequency of the DCO until the proportional path can compensate for the remaining frequency error. Since the cycle of correct information is always longer than the cycle of wrongful information, the frequency range is only limited by the DCO range. However, if the offset is extremely large, the frequency will take longer to settle due to the limited updates in each cycle.



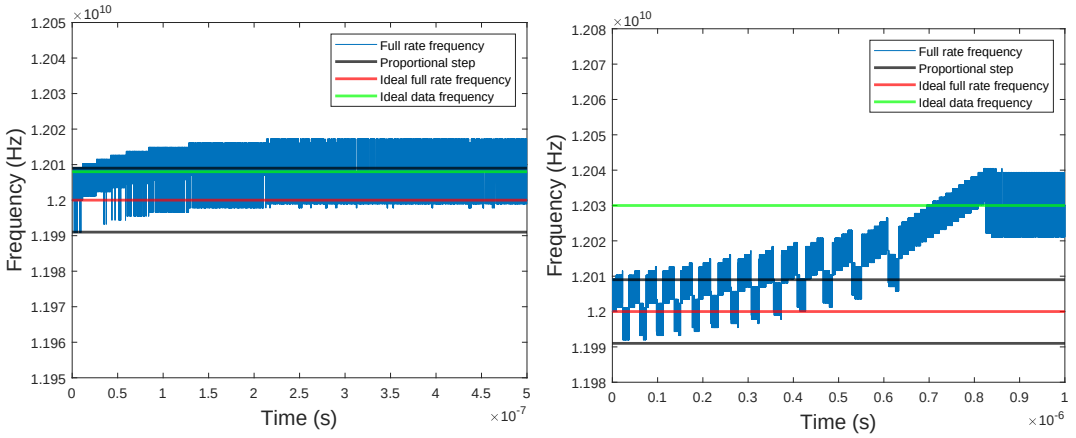
(a) DCO frequency.

(b) Full rate frequency.

Figure 6.6: CDR operation for an out of range frequency error.

Both in range and out of range frequency locking simulations are repeated without DCO noise and input jitter. The results can be seen in Fig. 6.7. In Fig. 6.7a the full rate frequency is plotted for an input data frequency of $12.08GHz$ since the proportional path can compensate for the frequency deviation, no cycle slipping occurs. The DCO will also sometimes lead the data even though the proportional path has not yet fully compensated for the frequency error.

in Fig. 6.7b the full rate frequency is plotted for an input data frequency of $12.03GHz$. The alternating leading and lagging phase updates due to cycle slipping now become more clearly visible. Since the frequency deviation becomes less each cycle, the time of each cycle also increases. Finally it can be seen that as soon as the full rate frequency crosses the input data frequency, the DCO can start to catch up to the data and no more cycle slipping occurs.



(a) In range frequency error

(b) Out of range frequency error

Figure 6.7: CDR operation for frequency errors without noise.

6.2. Jitter Generation and Tolerance

In order to accurately model the jitter generation, data-dependent effects have to be taken into account; therefore, a periodic steady-state simulation will not be possible. The CDR is simulated with a PRBS with $0.5V$ amplitude and $41.66ps$ rise and fall time. The simulation is run for $4\mu s$ in order to obtain a good estimate of the jitter.

A phase noise spectrum can be recovered from the zero crossings of the DCO phases and their deviation from the ideal crossings. The result for can be seen in Fig. 6.8. It is also compared to the phase domain model proposed in Chapter 3. The generated jitter is $486fs$ instead of the predicted $424fs$ by the model. This can be attributed to the nonideal gain in the system and data-dependent effects like ISI. The high flicker noise corner also contributes to the increased jitter.

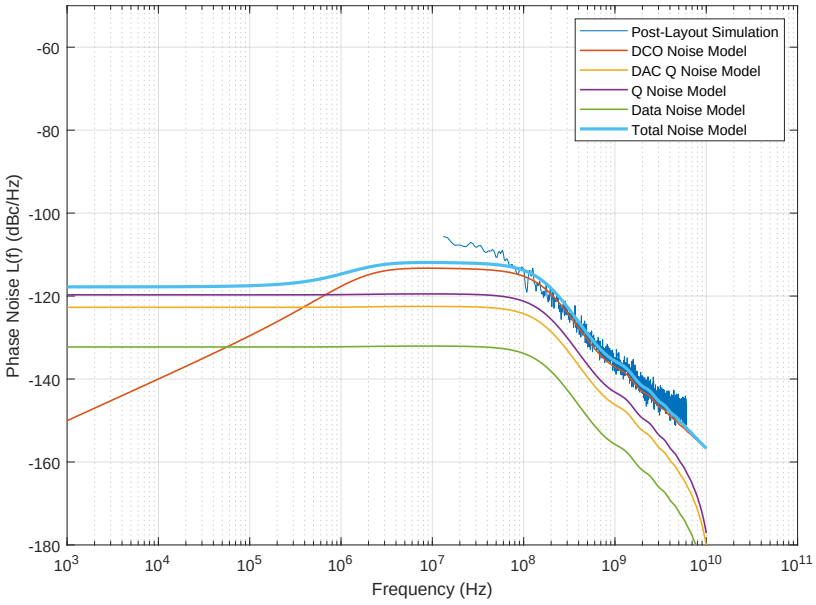


Figure 6.8: Phase noise post layout CDR simulation and Matlab model.

The jitter tolerance cannot be simulated due to the low bit error requirement of 10^{-12}. However with the jitter from a transient simulation and equation 3.33, the jitter tolerance can be estimated. The CDR is again simulated for $4\mu s$ with a PRBS with $0.5V$ amplitude and $41.66ps$ rise and fall time. The proportional gain is set to the jitter tolerance path with K_{DCO} of $19.5MHz$. The jitter generation increases to $543fs$.

This leads to an expected jitter tolerance of $1UI$ at $30.66MHz$. The estimated $-20dB/dec$ region can be seen in Fig. 6.9. A true value for the jitter tolerance can only be measured after the CDR has be fabricated.

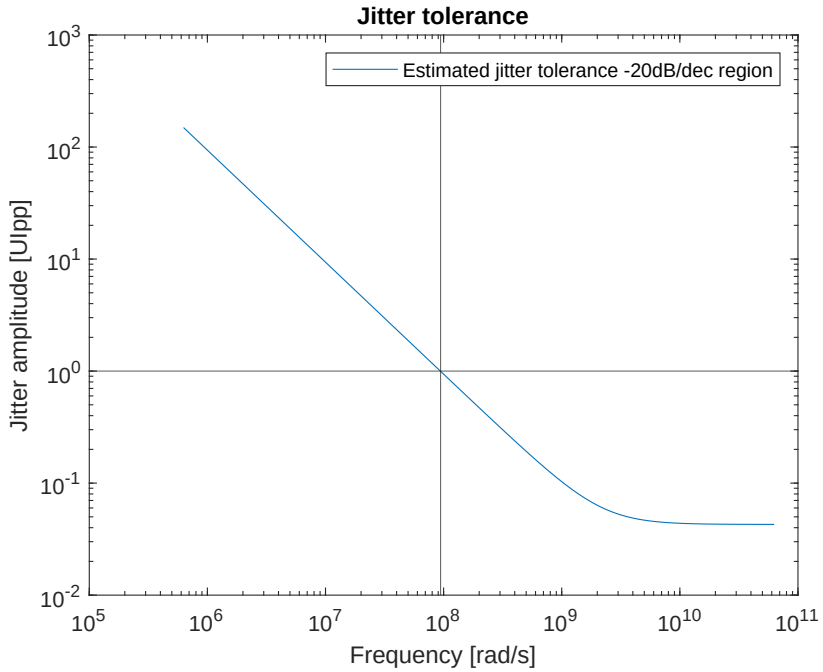


Figure 6.9: Estimated jitter tolerance.

6.3. Power Consumption

The power consumption is presented in Fig. 6.10. The power of the blocks without layout has been estimated using transistors with layout. However, this does not take parasitics due to routing into account. The buffer power is estimated using inverters with maximum width. The total power is $8.03mW$. The buffer needs to consume a significant amount of power since many clock phases need to be provided to eight phase detectors.

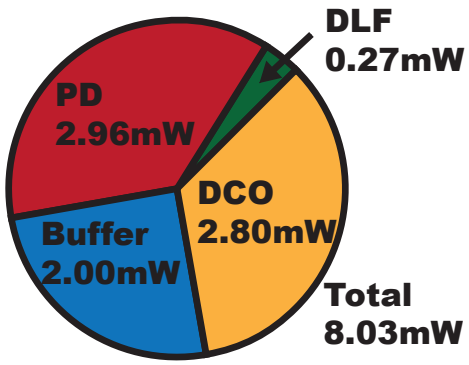


Figure 6.10: Power consumption CDR.

6.4. Comparison Table

The CDR has been compared to other state-of-the-art designs. The results can be seen in Tab. 6.1. The complete CDR shows competitive efficiency at $0.33pJ/bit$. Only [6] shows better efficiency. The jitter generation is also one of the best, with only [5] having better jitter generation due to linear phase

detection. The jitter tolerance outperforms all the other CDRs by a large margin. However, this is an estimate and the true value has to be measured.

	This work	[5]	[6]	[7]	[36]
PD type	BBPD	Linear	BBPD	BBPD	BBPD
Oscillator	Ring	LC	LC	Ring	LC
Data rate (Gb/s)	24	32	56	52	29.1
Power (mW)	8.03	14.7	8	43.1	19.16
Power efficiency (pJ/bit)	0.33	0.46	0.14	0.83	0.66
Jitter (fs)	482	352	574	430*	486
1-UI Jtol (MHz)	30	2	10	0.8	1.8
Technology	40	40	28	28	28

Table 6.1: CDR Comparison table.

7 Conclusion

7.1. Thesis Conclusion

This thesis proposed a PAM-4 CDR with specifications discussed in Chapter 1. A novel phase detection method was introduced in Chapter 2, along with a functional design. Chapter 3 presented a linearized time and phase domain model to determine system parameters, simulated in MATLAB. Chapter 4 covered the circuit design of individual components, including layout details of the phase detector. Chapter 5 delved into the calibration loop and its behavior. Chapter 6 presented simulations of the complete system, including phase and frequency locking, jitter generation and tolerance, power consumption and comparisons to state-of-the-art PAM-4 CDRs.

The proposed BBPD PAM-4 CDR uses a novel phase detection method sampling the middle of a transition, extracting phase information based on the expected middle voltage. This method ensures optimal use of all data transitions, optimizing for jitter tolerance and generation. A digital ring oscillator controlled by a DLF generates the 16 clock phases needed for phase detector operation. Two proportional control paths enable optimization for jitter tolerance and generation. The CDR consumes $8mW$ at a data rate of $24Gb/s$, for a competitive efficiency at $0.33pJ/bit$. Jitter generation is low at $483fs$, while jitter tolerance is very high at $1UI$ at $30MHZ$, due to the high transition density and DCO gain.

7.2. Future Work

7.2.1. Current Design

Since the design has not yet been taped out, several blocks need to be designed to achieve a functional CDR.

Comparator Calibration Loop

Since the comparators have an offset of more than 1 DAC LSB, a calibration loop has to be implemented to create a functional layout. This can be done with the use of varactors connected to both sides, equalizing the capacitance and ensuring a minimal offset voltage.

CTLE

To compensate for channel loss and improve practicality, a CTLE boosting high-frequency components should be designed and implemented for the current CDR. This amplifier also needs to convert the incoming differential data to single-ended data and bias the signal.

Layout

In its current form, only the layout of the PD has been done; the DCO, DLF, and buffer still need layout. An IO ring aswell as a full chip layout also need to be done.

7.3. Design Improvements

7.3.1. Data Rate

The most limiting aspect of the current CDR is the data rate, as industry CDRs can already handle data rates in the order of 100GB/s. In the current design, the computation time of the phase detector causes the most delay and this limits the data rate. If the number of phase detectors is doubled, the data rate could also be doubled. However, the delay would remain the same length and would therefore be doubled if measured in symbols, limiting the maximum DCO gain that can be used further. An LC-oscillator could be used to limit the phase noise contribution of the DCO and thus allow for a lower phase detector gain.

The topology of the phase detector could also be improved. The current phase detector consumes a lot of power and calculation time to convert the register code into an analog voltage. A solution could be to move the DAC outside of the phase detector as illustrated in Fig. 7.1. Here, one DAC is used for each transition. The reference voltage is pre-loaded into a capacitor bank, allowing for a low-power, low-speed DAC. The DAC only needs to provide enough current to compensate for kickback and noise and can therefore be low power. The delay is also reduced since the phase detector only has to wait until the input of the comparator is at a high enough voltage.

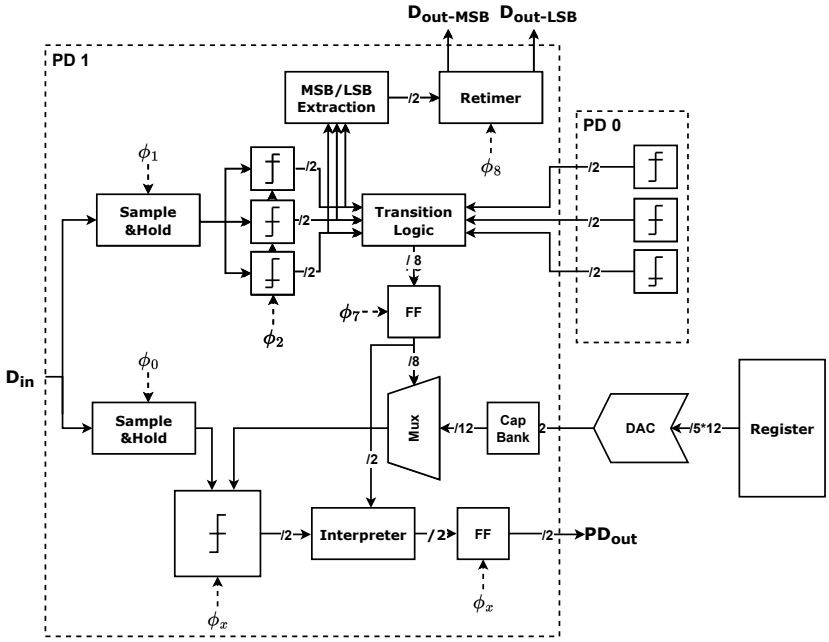


Figure 7.1: Proposed Phase detector block diagram.

7.3.2. Variable DCO Gain

The jitter tolerance of the CDR can be greatly improved by modifying the DCO gain. In the current design, the DCO should be manually tuned in order to reach a higher jitter tolerance, which is not practical. A second loop could be employed to tune the DCO gain based on the PD outputs, increasing the gain when all phase detectors provide the same phase information. A block diagram is presented in Fig. 7.2. This would allow accurate tracking of large input jitter while maintaining the low jitter generation in normal conditions. Of course, careful design is necessary if a second loop is used, as instability can easily occur if both loops have the same bandwidth. This could be mitigated by resetting the adjusted gain to its nominal value once the PDs start to report conflicting information. Another method is to have the bandwidth of the gain adaptation loop lower than the main CDR loop.

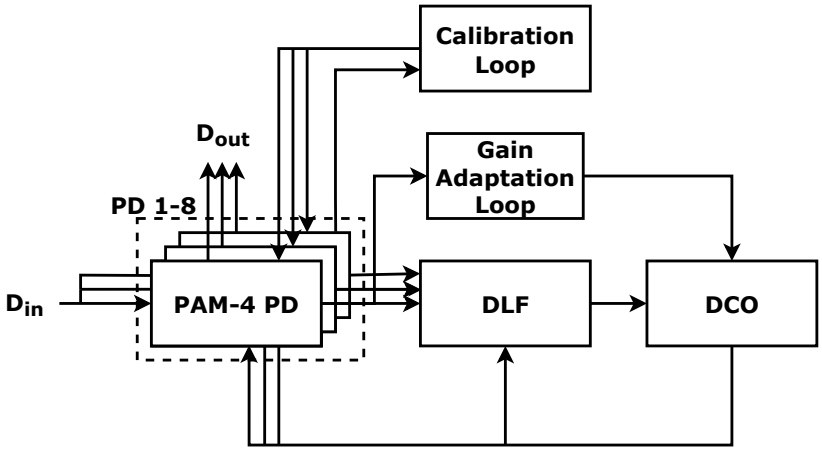


Figure 7.2: CDR with gain adaptation loop.

References

[1] J. Van Kerrebrouck, T. De Keulenaer, R. Pierco, *et al.*, "Nrz, duobinary, or pam4?: Choosing among high-speed electrical interconnects," *IEEE Microwave Magazine*, vol. 20, no. 7, pp. 24–35, 2019. DOI: 10.1109/MMM.2019.2909517.

[2] C. Hogge, "A self correcting clock recovery curcuit," *Journal of Lightwave Technology*, vol. 3, no. 6, pp. 1312–1314, 1985. DOI: 10.1109/JLT.1985.1074356.

[3] J. Alexander, "Clock recovery from random binary signals," *Sep*, vol. 26, pp. 541–542, 1975.

[4] K. Mueller and M. Muller, "Timing recovery in digital synchronous data receivers," *IEEE Transactions on Communications*, vol. 24, no. 5, pp. 516–531, 1976. DOI: 10.1109/TCOM.1976.1093326.

[5] Z. Zhang, G. Zhu, C. Wang, L. Wang, and C. P. Yue, "A 32-gb/s 0.46-pj/bit pam4 cdr using a quarter-rate linear phase detector and a self-biased pll-based multiphase clock generator," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 10, pp. 2734–2746, 2020. DOI: 10.1109/JSSC.2020.3005780.

[6] G. Hou and B. Razavi, "A 56-gb/s 8-mw pam4 cdr/dmux with high jitter tolerance," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2. DOI: 10.23919/VLSICircuits52068.2021.9492414.

[7] S. Park, Y. Choi, J. Sim, *et al.*, "A 0.83pj/b 52gb/s pam-4 baud-rate cdr with pattern-based phase detector for short-reach applications," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 118–120. DOI: 10.1109/ISSCC42615.2023.10067541.

[8] M. Pisati, F. De Bernardinis, P. Pascale, *et al.*, "A 243-mw 1.25–56-gb/s continuous range pam-4 42.5-db il adc/dac-based transceiver in 7-nm finfet," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 6–18, 2020. DOI: 10.1109/JSSC.2019.2936307.

[9] B.-J. Yoo, D.-H. Lim, H. Pang, *et al.*, "6.4 a 56gb/s 7.7mw/gb/s pam-4 wireline transceiver in 10nm finfet using mm-cdr-based adc timing skew control and low-power dsp with approximate multiplier," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 122–124. DOI: 10.1109/ISSCC19947.2020.9062964.

[10] S. Kiran, S. Cai, Y. Luo, S. Hoyos, and S. Palermo, "A 52-gb/s adc-based pam-4 receiver with comparator-assisted 2-bit/stage sar adc and partially unrolled dfe in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 3, pp. 659–671, 2019. DOI: 10.1109/JSSC.2018.2878850.

[11] E. Depaoli, H. Zhang, M. Mazzini, *et al.*, "A 64 gb/s low-power transceiver for short-reach pam-4 electrical links in 28-nm fdsoi cmos," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 6–17, 2019. DOI: 10.1109/JSSC.2018.2873602.

[12] B. Dehlaghi, S. Shahramian, J. Liang, *et al.*, "A 1.41-pj/b 56-gb/s pam-4 receiver using enhanced transition utilization cdr and genetic adaptation algorithms in 7-nm cmos," *IEEE Solid-State Circuits Letters*, vol. 2, no. 11, pp. 248–251, 2019. DOI: 10.1109/LSSC.2019.2938677.

[13] C. Wang, L. Wang, Z. Zhang, M. K. Mahmoudabadi, W. Shi, and C. P. Yue, "A 52-gb/s sub-1-pj/bit pam4 receiver in 40-nm cmos for low-power interconnects," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 46–55, 2021. DOI: 10.1109/OJCAS.2020.3034819.

[14] S.-C. Chang and S.-I. Liu, "A 5-gb/s adaptive digital cdr circuit with ssc capability and enhanced high-frequency jitter tolerance," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 161–165, 2021. DOI: 10.1109/TCSII.2020.3008777.

[15] J. Liang, A. Sheikholeslami, H. Tamura, Y. Ogata, and H. Yamaguchi, "6.7 a 28gb/s digital cdr with adaptive loop gain for optimum jitter tolerance," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 122–123. DOI: 10.1109/ISSCC.2017.7870291.

[16] R. C. Walker, "Designing bangbang plls for clock and data recovery in serial data transmission systems," in *Phase-Locking in High-Performance Systems: From Devices to Architectures*. 2003, pp. 34–45. DOI: 10.1109/9780470545492.ch4.

[17] M. Pimenta, Ç. Gürleyük, P. Walsh, D. O'Keeffe, M. Babaie, and K. A. A. Makinwa, "A 200-µw interface for high-resolution eddy-current displacement sensors," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1036–1045, 2021. DOI: 10.1109/JSSC.2020.3044027.

[18] M.-J. Park and J. Kim, "Pseudo-linear analysis of bang-bang controlled timing circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 6, pp. 1381–1394, 2013. DOI: 10.1109/TCSI.2012.2220502.

[19] J. Lee, K. Kundert, and B. Razavi, "Analysis and modeling of bang-bang clock and data recovery circuits," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, pp. 1571–1580, 2004. DOI: 10.1109/JSSC.2004.831600.

[20] M. Pelgrom, *Analog-to-Digital Conversion*. Jan. 2017, ISBN: 978-3-319-44970-8. DOI: 10.1007/978-3-319-44971-5.

[21] N. D. Dalt, "Markov chains-based derivation of the phase detector gain in bang-bang plls," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 11, pp. 1195–1199, 2006. DOI: 10.1109/TCSII.2006.883197.

[22] Y. Choi, D.-K. Jeong, and W. Kim, "Jitter transfer analysis of tracked oversampling techniques for multigigabit clock and data recovery," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 11, pp. 775–783, 2003. DOI: 10.1109/TCSII.2003.819070.

[23] G. Marucci, S. Levantino, P. Maffezzoni, and C. Samori, "Analysis and design of low-jitter digital bang-bang phase-locked loops," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 1, pp. 26–36, 2014. DOI: 10.1109/TCSI.2013.2268514.

[24] N. Da Dalt, "A design-oriented study of the nonlinear dynamics of digital bang-bang plls," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 1, pp. 21–31, 2005. DOI: 10.1109/TCSI.2004.840089.

[25] N. Da Dalt, "Linearized analysis of a digital bang-bang pll and its validity limits applied to jitter transfer and jitter generation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3663–3675, 2008. DOI: 10.1109/TCSI.2008.925948.

[26] R. Staszewski, C. Fernando, and P. Balsara, "Event-driven simulation and modeling of phase noise of an rf oscillator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 4, pp. 723–733, 2005. DOI: 10.1109/TCSI.2005.844236.

[27] G. Kiene, R. W. J. Overwater, M. Babaie, and F. Sebastiano, "A cryo-cmos sar adc with fia sampling driver enabled by cryogenic-aware back-biasing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–11, 2023. DOI: 10.1109/TCSI.2023.3336566.

[28] B. Razavi, "The strongarm latch [a circuit for all seasons]," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 2, pp. 12–17, 2015. DOI: 10.1109/MSSC.2015.2418155.

[29] C.-H. Chan, Y. Zhu, U.-F. Chio, S.-W. Sin, S.-P. U., and R. Martins, "A voltage-controlled capacitance offset calibration technique for high resolution dynamic comparator," in *2009 International SoC Design Conference (ISOCC)*, 2009, pp. 392–395. DOI: 10.1109/SOCDC.2009.5423836.

[30] J. Yuan and C. Svensson, "High-speed cmos circuit technique," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, pp. 62–70, 1989. DOI: 10.1109/4.16303.

[31] L. Kong, Y. Chang, and B. Razavi, "An inductorless 20-gb/s cdr with high jitter tolerance," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 10, pp. 2857–2866, 2019. DOI: 10.1109/JSSC.2019.2930899.

[32] B. Razavi, "The current-steering dac [a circuit for all seasons]," *IEEE Solid-State Circuits Magazine*, vol. 10, no. 1, pp. 11–15, 2018. DOI: 10.1109/MSSC.2017.2771102.

[33] A. van den Bosch, M. Borremans, M. Steyaert, and W. Sansen, "A 10-bit 1-gsample/s nyquist current-steering cmos d/a converter," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 3, pp. 315–324, 2001. DOI: 10.1109/4.910469.

[34] B. Razavi, "The ring oscillator [a circuit for all seasons]," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 4, pp. 10–81, 2019. DOI: 10.1109/MSSC.2019.2939771.

[35] A. Hajimiri, S. Limotyrakis, and T. Lee, "Jitter and phase noise in ring oscillators," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 6, pp. 790–804, 1999. DOI: 10.1109/4.766813.

[36] X. Zhao, Y. Chen, P.-I. Mak, and R. P. Martins, “A 0.0285mm2 0.68pj/bit single-loop full-rate bang-bang cdr without reference and separate frequency detector achieving an 8.2(gb/s)/µs acquisition speed of pam-4 data in 28nm cmos,” in *2020 IEEE Custom Integrated Circuits Conference (CICC)*, 2020, pp. 1–4. DOI: 10.1109/CICC48029.2020.9075885.

A

MATLAB Code

```
clear;
close all;
fig = 1;            % Figure multiplier
%% System variables
samp = 1e8;       % Amount of time steps
samp_d = 1;       % Samples to neglect in PN calculations
f_dat = 12e9;     % Data frequency
t_dat = 1/f_dat; % Data period
t_trans_f = 41.667e-12; % Fall time of transistion
t_trans_r = 41.667e-12; % Rise time of data
t_trans_av = t_trans_f/2+t_trans_r/2; % Average transition time
n = 0.75;         % Transisiton density

% Frequency vector for phase domain model
f_min = 1e3;
f_step = f_min;
f_max = 1e10;
f = f_min:f_step:f_max;

% Variables for calibration loop
cal_th=255; % Threshold of calibration counter
cal_f = 100; % Calibration frequency (f_dat/f_dal)
calibration = 0; % 1 to turn on calibration loop
lock = 1; % Locked transition in calibration loop

%Simulate feedtrough
Feedtrough = 0.01;


%System parameters
delay=11;     %Delay of the system
bits=5;       %DAC bits
k_dco = 9e6; %Dco step in frequency (also fbb)
alpha = 1;    %Gain of proportional path
rho = 0.001;  %Gain of integral path

%volatage comparison
Vcomp_min = 0e-3;          % Comparator minimum input difference
Vmax = 0.5;                % Data voltage amplitude
comp_matrix = zeros(12,bits); % Matrix for comparison voltages

%Create data and data voltage
data = randi([0 3],1,samp+2); % Data
```

```
datav = data/3*Vmax;        % Data voltage

%% Initialize comparison matrix
transition_code([1 2]) = round(0.5*2^bits); % Major
transition_code([3 4]) = round((2/3)*2^bits); % Middle High
transition_code([5 6]) = round((1/3)*2^bits); % Middle Low
transition_code([7 8]) = round((5/6)*2^bits); % Minor High
transition_code([9 10]) = round((3/6)*2^bits); % Minor Middle
transition_code([11 12]) = round((1/6)*2^bits); % Minor Low

for i =1:12
    comp_matrix(i,:) = int2bit(transition_code(i),bits,1);
end

%% Inittialize KDCO Table
% Variables
int_bits = 13;
int_bits_ign = 7;
KDCO_PROP = zeros(3,1);
KDCO_INT = zeros(int_bits,1);

% Proportional path
KDCO_PROP(1) = 0; %BBPD = -1
KDCO_PROP(2) = 1.125e6;
KDCO_PROP(3) = 1.125e6+1.125e6;

% Integral path, first bits are not applied to DCO
KDCO_INT(1) = 0;%1.125e3;
KDCO_INT(2) = 0;%2.25e3;
KDCO_INT(3) = 0;%4.5e3;
KDCO_INT(4) = 0;%9e3;
KDCO_INT(5) = 0;%18e3;
KDCO_INT(6) = 0;%36e3;
KDCO_INT(7) = 0;%72e3;
KDCO_INT(8) = 144e3;
KDCO_INT(9) = 288e3;
KDCO_INT(10) = 576e3;
KDCO_INT(11) = 1152e3;
KDCO_INT(12) = 2304e3;
KDCO_INT(13) = 4608e3;

f_base = f_dat/8 - sum(KDCO_INT(int_bits)) - KDCO_PROP(2);
f_base = 8*f_base;
KDCO_INT = 8*KDCO_INT;
KDCO_PROP = 8*KDCO_PROP;


%% Observation variables
Tck_dco = zeros(1,samp); % DCO timestamps
T_error = zeros(1,samp); % Timing errors
Vsamp = zeros(1,samp);  % Voltage Samples
Cal_reg = zeros(12,samp); % Register codes
counter = zeros(1,12);  % Sample counter
BBPD_out = zeros(1,samp); % PD output
f_obs = zeros(1,samp);  % DCO frequency
f_int_obs = zeros(1,samp); % Integral path frequency

%% Noise
sig_dat = 250e-15; % Input jitter std

% DCO noise power
```

```
PN_20dB = -79.77; % dBc per Hz at 1MHz
f_offset = 1e6; % Frequency for the thermal noise
PN_30dB = 9.5; % dBc per Hz at 1KHz
f_offsetf = 1e3; % Frequency for the flicker noise
a = 10^(PN_20dB/10); % Thermal noise
b = 10^(PN_30dB/10); % Flicker noise
pn_osc = 2*(f_offset^2*a)./(f.^2) + 2*(f_offsetf^3*b)./(f.^3);

%DCO noise std
sig_dco = (f_offset/f_dat)*sqrt(10^(PN_20dB/10)/f_dat);

%Flicker noise is not added to time domain model. It is only plotted in the
%phase domain model to show it has negligible influence
%% Time model
Tck_data = (1:samp)*t_dat;                      %ideal data timings
Tck_data = randn([size(Tck_data)])*sig_dat + Tck_data; %data timings with jitter

% Initialize
t_ck = 0;
reg_lpf_rho = 2^(int_bits-1)+2^(int_bits_ign-1);
reg_lpf = zeros(1,delay);


for count = 1:samp
   select = mod(count-1,delay)+1; %select correct control word for given delay
   if reg_lpf(select)==-1 %Proportional path frequency
      f_prop = KDCO_PROP(1);
   elseif reg_lpf(select)==0
      f_prop = KDCO_PROP(2);
   elseif reg_lpf(select)==1
      f_prop = KDCO_PROP(3);
   end
   f_int = sum(flip(int2bit(reg_lpf_rho,int_bits,1)).*KDCO_INT); % Integral path frequency
   f_inst = f_base + f_prop + f_int; % DCO frequency
   t_ck = t_ck + 1/f_inst +(sig_dco*randn); %Create new DCO timing
   Tck_dco(count) = t_ck; % Record DCO timing

   T_error(count) = (Tck_data(count) - Tck_dco(count)); %Record timing error
   % Calucalte voltage sample
   if (data(count)>=data(count+1)) % Fall transition
   Vsamp(count) =
       (datav(count)+(datav(count+1)-datav(count))*((0.5*t_trans_f+T_error(count))/t_trans_f));
   Vsamp(count) = Vsamp(count)+(datav(count+2)-Vsamp(count))*Feedtrough;
   else % Rise Transition
   Vsamp(count) =
       (datav(count)+(datav(count+1)-datav(count))*((0.5*t_trans_f+T_error(count))/t_trans_r));
   Vsamp(count) = Vsamp(count)+(datav(count+2)-Vsamp(count))*Feedtrough;
   end

   if (data(count)~=data(count+1)) % Caclculate transition type
      if (data(count) < data(count+1))
      rf = 1;
      else
      rf = 0;
      end

      if (data(count)==0&&data(count+1)==3)
      trans=1; % Major rise
      elseif (data(count)==3&&data(count+1)==0)
      trans=2; % Major fall
      elseif (data(count)==1&&data(count+1)==3)
```

```
        trans=3; % MiddleH rise
        elseif (data(count)==3&&data(count+1)==1)
        trans=4; % MiddleH fall
        elseif (data(count)==0&&data(count+1)==2)
        trans=5; % MiddleL rise
        elseif (data(count)==2&&data(count+1)==0)
        trans=6; % MiddleL fall
        elseif (data(count)==2&&data(count+1)==3)
        trans=7; % MinorH rise
        elseif (data(count)==3&&data(count+1)==2)
        trans=8; % MinorH fall
        elseif (data(count)==1&&data(count+1)==2)
        trans=9; % MinorM rise
        elseif (data(count)==2&&data(count+1)==1)
        trans=10; % MinorM fall
        elseif (data(count)==0&&data(count+1)==1)
        trans=11; % MinorL rise
        elseif (data(count)==1&&data(count+1)==0)
        trans=12; % MinorL fall
        else
        return
        end
    else
        trans=0;
    end

    % Calculate BBPD output
    if trans==0
        BBPD = 0;
    elseif abs((bit2int(comp_matrix(trans,:).',bits)*(Vmax/(2^bits)) -
         Vsamp(count)))<Vcomp_min % Voltage difference too small for comparator to settle
        BBPD = 0;
    elseif (bit2int(comp_matrix(trans,:).',bits)*(Vmax/(2^bits)) >=Vsamp(count))&&(rf==1) %
         Lower sample rise -> lag
        BBPD = 1;
    elseif (bit2int(comp_matrix(trans,:).',bits)*(Vmax/(2^bits)) < Vsamp(count))&&(rf==1) %
         Higher sample rise -> lead
        BBPD = -1;
    elseif (bit2int(comp_matrix(trans,:).',bits)*(Vmax/(2^bits)) >=Vsamp(count))&&(rf==0) %
         Lower sample fall -> lead
        BBPD = -1;
    elseif (bit2int(comp_matrix(trans,:).',bits)*(Vmax/(2^bits)) < Vsamp(count))&&(rf==0) %
         Higher sample fall -> lag
        BBPD = 1;
    else
        return
    end


    %% Calibration loop
    if (calibration==1) % Check if calibration loop is active
        if (mod(count,cal_f)==0)&&(trans~=0)&&(rf==1)&&(trans~=lock) %Update rising transition
            counter(1,trans) = counter(1,trans)+BBPD;
            if abs(counter(1,trans))>cal_th
                comp_matrix(trans,:) = int2bit((bit2int(comp_matrix(trans,:).',bits)-BBPD),bits);
                counter(1,trans) = 0;
            end
        elseif (mod(count,cal_f)==0)&&(trans~=0)&&(rf==0)&&(trans~=lock) %Update falling
             transition
            counter(1,trans) = counter(1,trans)+BBPD;
            if abs(counter(1,trans))>cal_th
```

```
            comp_matrix(trans,:) = int2bit((bit2int(comp_matrix(trans,:).',bits)+BBPD),bits);
            counter(1,trans) = 0;
        end
    end

    for i=1:12 %Record change of calibration loop
        Cal_reg(i,count) = bit2int(comp_matrix(i,:).',bits);
    end
end

if select == delay % Update integral path at lower frequency
  reg_lpf_rho = reg_lpf_rho + sum(reg_lpf);
end

reg_lpf(select) = (alpha*BBPD); % Update loop filter
BBPD_out(count) = BBPD; % Record BBPD output
f_obs(count) = f_inst; % Record frequency
f_int_obs(count) = f_int; %Record integral path frequency
end

%% Calculate PN from time model
jitter = Tck_dco((samp_d+1):(samp)) - ((samp_d+1):samp)*t_dat; %caluclate jitter
figure(fig)
[psd1,freq] =
    pwelch(2*pi*jitter/t_dat,blackmanharris(length(jitter)/8),[],[],1/t_dat,'onesided');
semilogx(freq(10:end),10*log10(psd1(10:end))); grid on;
xlabel('Frequency (Hz)');ylabel('Phase Noise L(f) (dBc/Hz)');
title('One sided phase noise')

%% Phase domain model
%% Calculate KPD
e_BBPD = sum(BBPD_out(samp_d+1:end).*T_error(samp_d+1:end))/(samp-samp_d); % Expected value
    BBPD
e_TERR = sum(T_error(samp_d+1:end).*T_error(samp_d+1:end))/(samp-samp_d); % Expected value
    Terr
k_exp = -e_BBPD/e_TERR; %Brute force Kpd

sig_terror =
    sqrt(sum((T_error(samp_d+1:end)-mean(T_error(samp_d+1:end))).^2)/length(T_error(samp_d+1:end)));
    % Std Terr
sig_terror_min = ((1+delay)*alpha*(1/n)*k_dco)/(sqrt(3)*f_dat^2); % Minimal noise for limit
    cycling
t_error_margin = sig_terror-sig_terror_min; % Margin
k_calc = sqrt(2/pi)*(n/(sig_terror)); % Linearized model Kpd



%% Quantization Noise BBPD
q = BBPD_out(samp_d+1:end) + k_exp*T_error(samp_d+1:end);
var_q = sum((q-mean(q)).^2)/length(q); %Brute force quantization error
var_q_calc = n-(2/pi)*n^2; % Linearized model

%% DAC Quantization noise linear model
Vspe(5) = Vmax/t_trans_r; % Speed major transition rise
Vspe(6) = Vmax/t_trans_f; % Speed major transition fall
Vspe([3 9]) = (2/3)*(Vmax/t_trans_r); % Speed middle transition rise
Vspe([4 10]) = (2/3)*(Vmax/t_trans_f); % Speed middle transition fall
Vspe([1 7 11]) = (1/3)*(Vmax/t_trans_r); % Speed minor transition rise
Vspe([2 8 12]) = (1/3)*(Vmax/t_trans_f); % Speed minor transition fall
values = (0:2^bits-1)./(2^bits); %Dac voltage
x=zeros(1,16); % Lock offsets due to DAC noise
```

```
n = [5/6 5/6 4/6 4/6 3/6 3/6 3/6 3/6 2/6 2/6 1/6 1/6]; %Vector with middle voltages
for i=1:12
    [~,idx]=min(abs(values-n(i)));
    x(i) = Vmax*(n(i)- values(idx))/Vspe(i);
end
sig_qdac=std(x);

%% PN Contributions
%quantization nosie
pn_q = var_q./(0.5*f_dat).*(2*pi*f_dat).^2; % Quantization PN brute force
pn_qcalc = var_q_calc./(0.5*f_dat).*(2*pi*f_dat).^2; % Quantization PN linear model
pn_qdac = sig_qdac^2./(0.5*f_dat).*(2*pi*f_dat).^2; % Quantization PN DAC
pn_data = sig_dat^2./(0.5*f_dat).*(2*pi*f_dat).^2; % Data PN


%% Transfer Functions
TF_KPD = k_exp; %Phase detector
TF_DLF = alpha+rho./(1i*2*pi*f*t_dat); %DLF
TF_delay = exp(-1i*2*pi*f*t_dat*(delay)); %Delay
TF_DCO = (k_dco./(1i*2*pi*f*t_dat))*1/(f_dat^2); %DCO
TF_ZOH = sinc(f/f_dat); %Zero order hold
H_ol = TF_KPD.*TF_DLF.*TF_delay.*TF_DCO.*TF_ZOH; %Open Loop
H_cl = H_ol./(1+H_ol); % Closed loop

% Jitter Tolerance
w = 2*pi*f;
h=((1-t_trans_f/t_dat-t_trans_r/t_dat)/2 -(7*sig_terror/(t_dat)));
c=k_dco*n*rho*f_dat*pi^2;
G_jt2 = (sqrt(h^2.*(w).^2+pi^4*k_dco^2*alpha^2))./w; %20dB/dec region
G_jt4 = (sqrt(10121*c^2 - 48800*c*h*w.^2 + 160000*h^2.*w.^4)./(400.*w.^2)); %40dB/dec region
figure(2*fig)
loglog(w,abs(G_jt2))
hold on
loglog(w,abs(G_jt4))
legend('Estimated jitter tolerance -20dB/dec region', 'Estimated jitter tolerance -40dB/dec
    region')
hold on
title('Jitter tolerance')
xlabel('Frequency [rad/s]');
ylabel('Jitter amplitude [UIpp]');
yline(1,'HandleVisibility','off')

Jitter_tolerance = f(find(G_jt2<1,1))% Estimated 1UI tolerance


% Noise contributions to output
H_data = H_ol./(1+H_ol);
H_q = H_ol/TF_KPD./(1+H_ol);
H_ndco = 1./(1+H_ol);

pll_osc = (abs(H_ndco)).^2.*(pn_osc);
pll_q = (abs(H_q)).^2.*(pn_q);
pll_qcalc = (abs(H_q)).^2.*(pn_qcalc);
pll_qdac =(abs(H_data)).^2.*(pn_qdac);
pll_data = (abs(H_data)).^2.*(pn_data);
total = pll_osc+pll_qcalc+pll_data+pll_qdac;

figure(fig);
hold on;
semilogx(f,10*log10(pll_osc),'LineWidth',1);
hold on;
```

```
%semilogx(f,10*log10(pll_q),'LineWidth',1);
semilogx(f,10*log10(pll_qdac),'LineWidth',1);
semilogx(f,10*log10(pll_qcalc),'LineWidth',1);
semilogx(f,10*log10(pll_data),'LineWidth',1);
semilogx(f,10*log10(total),'LineWidth',2);
legend('Simulation','DCO Noise','DAC Q Noise','Q Noise', 'Data Noise','Total')
axis([1e3 1e11 -180 -50])

sig_jitter_calc = sqrt(fstep)*sqrt(sum(abs(total(1:end))))/(2*pi*f_dat) % Linear model jitter
sig_jitter = rms(jitter) % Time domain model jitter

%% Bandwidth calculation
mag_cl = (abs(H_cl)).^2;
A=find(mag_cl<0.5);
Bwindex = A(1);
BW = f(Bwindex);

%% Bode Plots
s=tf('s');
TF_KPD = k_exp;
TF_DLF = alpha+rho/(s*t_dat);
TF_delay = exp(-s*t_dat*(delay));
TF_DCO = (k_dco/(s*t_dat))*1/(f_dat^2);
TF_ZOH = 1;%sinc(1i*2*pi*s/f_dat); %not necessary since model fails when ZOH has effect
H_ol = TF_KPD*TF_DLF*TF_delay*TF_DCO*TF_ZOH;
H_cl = H_ol/(1+H_ol);

%Plotting open loop tf
figure(3*fig);
margin(H_ol,f);
title('Open-loop');
p=pole(H_ol);
z=zero(H_ol);
xline(abs(p),'b');
xline(abs(z),'r');
axis([1e4 1e12 -190 -90])

%Plotting closed loop tf
figure(4*fig);
margin(H_cl,f);
title('Closed-loop');
p=pole(H_cl);
z=zero(H_cl);
xline(abs(p),'b');
xline(abs(z),'r');
axis([1e4 1e12 -190 0])

%Plotting comparison voltages
if calibration==1
figure(5*fig)
plot(Cal_reg.')
title('Comparison voltages');
xlabel('Sample')
ylabel('Code')
axis([0 samp 0 2^bits])
end
```

B

Verilog Code

```
module register_cal(
i_CKs,
i_enable,
i_rst,

i_major,
i_middleH,
i_middleL,
i_minorH,
i_minorM,
i_minorL,
i_rise,
i_fall,
i_pd,

i_thr,
i_lock,

i_regi0,i_regi1,i_regi2,i_regi3,i_regi4,i_regi5,i_regi6,i_regi7,i_regi8,i_regi9,i_regi10,i_regi11,
o_regi0,o_regi1,o_regi2,o_regi3,o_regi4,o_regi5,o_regi6,o_regi7,o_regi8,o_regi9,o_regi10,o_regi11,

);

//Define Parameters
parameter BITS = 5;
parameter COUNT_BITS = 5;
parameter TH_BITS = 10;
integer i;

//Define Inputs
input i_CKs;
input i_enable;
input i_rst;
input i_major;
input i_middleH;
input i_middleL;
input i_minorH;
input i_minorM;
input i_minorL;
input i_rise;
input i_fall;
input [1:0] i_pd;
input [11:0] i_lock;
```

```
input [BITS-1:0]
    i_regi0,i_regi1,i_regi2,i_regi3,i_regi4,i_regi5,i_regi6,i_regi7,i_regi8,i_regi9,i_regi10,i_regi11;
input [TH_BITS-1:0] i_thr;

//Define outputs
output reg [BITS-1:0]
    o_regi0,o_regi1,o_regi2,o_regi3,o_regi4,o_regi5,o_regi6,o_regi7,o_regi8,o_regi9,o_regi10,o_regi11;
//Signal registers
reg s_major;
reg s_middleH;
reg s_middleL;
reg s_minorH;
reg s_minorM;
reg s_minorL;
reg s_rise;
reg s_fall;
reg [1:0] s_pd;


//Counters for each register
reg signed [COUNT_BITS-1:0] c_regi[0:11];
// threshold and lock register
reg signed [TH_BITS-1:0] THRESHOLD;
reg [11:0] LOCK;




//////////////////////
// Reset
// Load input register
// Load threshold value
// Reset counters
//////////////////////

always @ (posedge i_rst) begin
       o_regi0 <= i_regi0;
       o_regi1 <= i_regi1;
       o_regi2 <= i_regi2;
       o_regi3 <= i_regi3;
       o_regi4 <= i_regi4;
       o_regi5 <= i_regi5;
       o_regi6 <= i_regi6;
       o_regi7 <= i_regi7;
       o_regi8 <= i_regi8;
       o_regi9 <= i_regi9;
       o_regi10 <= i_regi10;
       o_regi11 <= i_regi11;
       THRESHOLD <= i_thr;
       LOCK <= i_lock;

   for (i=0; i<12; i=i+1) begin
          c_regi[i]<= 0;
          end
       end

// Load input states
always @ (negedge i_CKs && i_enable) begin
       s_major <= i_major;
       s_middleH <= i_middleH;
       s_middleL <= i_middleL;
```

```
        s_minorH <= i_minorH;
        s_minorM <= i_minorM;
        s_minorL <= i_minorL;
        s_rise <= i_rise;
        s_fall <= i_fall;
        s_pd[1:0] <= i_pd[1:0];
        end

////////////////////////////
// Check phase information
// Check transition loop
// Check if locked
// Check counter full
// update register or counter
////////////////////////////

always @ (posedge i_CKs && i_enable) begin
        if  (s_pd==0) begin //PD says SLOWER
                if ((s_major && s_rise) && (LOCK[0]==0)) begin
                        if (c_regi[0] < -THRESHOLD) begin
                                o_regi0 <= o_regi0+1;
                                c_regi[0] <= 0;
                                end
                        else begin
                                c_regi[0]     <= c_regi[0]-1;
                                end
                        end

                else if ((s_major && s_fall) && (LOCK[6]==0)) begin
                        if (c_regi[6] < -THRESHOLD) begin
                                o_regi6 <= o_regi6-1;
                                c_regi[6] <= 0;
                                end
                        else begin
                                c_regi[6]     <= c_regi[6]-1;
                                end
                        end

                else if ((s_middleH && s_rise) && (LOCK[1]==0)) begin
                        if (c_regi[1] < -THRESHOLD) begin
                                o_regi1 <= o_regi1+1;
                                c_regi[1] <= 0;
                                end
                        else begin
                                c_regi[1]     <= c_regi[1]-1;
                                end
                        end

                else if ((s_middleH && s_fall) && (LOCK[7]==0)) begin
                        if (c_regi[7] < -THRESHOLD) begin
                                o_regi7 <= o_regi7-1;
                                c_regi[7] <= 0;
                                end
                        else begin
                                c_regi[7]     <= c_regi[7]-1;
                                end
                        end

                else if ((s_middleL && s_rise) && (LOCK[2]==0)) begin
                        if (c_regi[2] < -THRESHOLD) begin
                                o_regi2 <= o_regi2+1;
```

```
                        c_regi[2] <= 0;
                        end
                else begin
                        c_regi[2]     <= c_regi[2]-1;
                        end
                end

        else if ((s_middleL && s_fall) && (LOCK[8]==0)) begin
                if (c_regi[8] < -THRESHOLD) begin
                        o_regi8 <= o_regi8-1;
                        c_regi[8] <= 0;
                        end
                else begin
                        c_regi[8]     <= c_regi[8]-1;
                        end
                end

        else if ((s_minorH && s_rise) && (LOCK[3]==0)) begin
                if (c_regi[3] < -THRESHOLD) begin
                        o_regi3 <= o_regi3+1;
                        c_regi[3] <= 0;
                        end
                else begin
                        c_regi[3]     <= c_regi[3]-1;
                        end
                end

        else if ((s_minorH && s_fall) && (LOCK[9]==0)) begin
                if (c_regi[9] < -THRESHOLD) begin
                        o_regi9 <= o_regi9-1;
                        c_regi[9] <= 0;
                        end
                else begin
                        c_regi[9]     <= c_regi[9]-1;
                        end
                end

        else if ((s_minorM && s_rise) && (LOCK[4]==0)) begin
                if (c_regi[4] < -THRESHOLD) begin
                        o_regi4 <= o_regi4+1;
                        c_regi[4] <= 0;
                        end
                else begin
                        c_regi[4]     <= c_regi[4]-1;
                        end
                end

        else if ((s_minorM && s_fall) && (LOCK[10]==0)) begin
                if (c_regi[10] < -THRESHOLD) begin
                        o_regi10 <= o_regi10-1;
                        c_regi[10] <= 0;
                        end
                else begin
                        c_regi[10]    <= c_regi[10]-1;
                        end
                end

        else if ((s_minorL && s_rise) && (LOCK[5]==0)) begin
                if (c_regi[5] < -THRESHOLD) begin
                        o_regi5 <= o_regi5+1;
                        c_regi[5] <= 0;
```

```
                        end
                    else begin
                        c_regi[5]     <= c_regi[5]-1;
                        end
                    end

            else if ((s_minorL && s_fall) && (LOCK[11]==0)) begin
                    if (c_regi[11] < -THRESHOLD) begin
                        o_regi11 <= o_regi11-1;
                        c_regi[11] <= 0;
                        end
                    else begin
                        c_regi[11]    <= c_regi[11]-1;
                        end
                    end
            end

        else if (s_pd==3) begin //PD SAYS FASTER
            if ((s_major && s_rise) && (LOCK[0]==0)) begin
                    if (c_regi[0] > THRESHOLD) begin
                        o_regi0 <= o_regi0 -1;
                        c_regi[0] <= 0;
                        end
                    else begin
                        c_regi[0]     <= c_regi[0]+1;
                        end
                    end

            else if ((s_major && s_fall) && (LOCK[6]==0)) begin
                    if (c_regi[6] > THRESHOLD) begin
                        o_regi6 <= o_regi6+1;
                        c_regi[6] <= 0;
                        end
                    else begin
                        c_regi[6]     <= c_regi[6]+1;
                        end
                    end

            else if ((s_middleH && s_rise) && (LOCK[1]==0)) begin
                    if (c_regi[1] > THRESHOLD) begin
                        o_regi1 <= o_regi1 -1;
                        c_regi[1] <= 0;
                        end
                    else begin
                        c_regi[1]     <= c_regi[1]+1;
                        end
                    end

            else if ((s_middleH && s_fall) && (LOCK[7]==0)) begin
                    if (c_regi[7] > THRESHOLD) begin
                        o_regi7 <= o_regi7+1;
                        c_regi[7] <= 0;
                        end
                    else begin
                        c_regi[7]     <= c_regi[7]+1;
                        end
                    end

            else if ((s_middleL && s_rise) && (LOCK[2]==0)) begin
                    if (c_regi[2] > THRESHOLD) begin
                        o_regi2 <= o_regi2 -1;
```

```
                        c_regi[2] <= 0;
                        end
                else begin
                        c_regi[2]     <= c_regi[2]+1;
                        end
                end

        else if ((s_middleL && s_fall) && (LOCK[8]==0)) begin
                if (c_regi[8] > THRESHOLD) begin
                        o_regi8 <= o_regi8+1;
                        c_regi[8] <= 0;
                        end
                else begin
                        c_regi[8]     <= c_regi[8]+1;
                        end
                end

        else if ((s_minorH && s_rise) && (LOCK[3]==0)) begin
                if (c_regi[3] > THRESHOLD) begin
                        o_regi3 <= o_regi3 -1;
                        c_regi[3] <= 0;
                        end
                else begin
                        c_regi[3]     <= c_regi[3]+1;
                        end
                end

        else if ((s_minorH && s_fall) && (LOCK[9]==0)) begin
                if (c_regi[9] > THRESHOLD) begin
                        o_regi9 <= o_regi9+1;
                        c_regi[9] <= 0;
                        end
                else begin
                        c_regi[9]     <= c_regi[9]+1;
                        end
                end

        else if ((s_minorM && s_rise) && (LOCK[4]==0)) begin
                if (c_regi[4] > THRESHOLD) begin
                        o_regi4 <= o_regi4 -1;
                        c_regi[4] <= 0;
                        end
                else begin
                        c_regi[4]     <= c_regi[4]+1;
                        end
                end

        else if ((s_minorM && s_fall) && (LOCK[10]==0)) begin
                if (c_regi[10] > THRESHOLD) begin
                        o_regi10 <= o_regi10+1;
                        c_regi[10] <= 0;
                        end
                else begin
                        c_regi[10]    <= c_regi[10]+1;
                        end
                end

        else if ((s_minorL && s_rise) && (LOCK[5]==0)) begin
                if (c_regi[5] > THRESHOLD) begin
                        o_regi5 <= o_regi5 -1;
                        c_regi[5] <= 0;
```

```
                        end
                    else begin
                        c_regi[5]     <= c_regi[5]+1;
                        end
                    end

            else if ((s_minorL && s_fall) && (LOCK[11]==0)) begin
                    if (c_regi[11] > THRESHOLD) begin
                        o_regi11 <= o_regi11+1;
                        c_regi[11] <= 0;
                        end
                    else begin
                        c_regi[11]    <= c_regi[11]+1;
                        end
                    end
            end

      end
endmodule
```